
Obsługa bazy za pomocą PDO

Nawiązywanie połączenia

W przypadku PDO nawiązywanie połączenia z bazą polega na utworzeniu obiektu klasy PDO. Wywołanie konstruktora ma schematyczną postać:

```
PDO(źródło_danych[, użytkownik[, hasło[, opcje]]]):
```

gdzie *źródło_danych* (DSN, z ang. *Data Source Name*) to ciąg znaków określający rodzaj bazy danych i sposób połączenia, *użytkownik* to nazwa użytkownika, *hasło* to hasło, a *opcje* to tablica zawierająca dodatkowe opcje związane z połączeniem (w typowych sytuacjach jego używanie nie jest konieczne). Jedynym wymaganym argumentem jest *źródło_danych*, pozostałe są opcjonalne.

Uproszczona postać ciągu *źródło_danych* dla bazy MySQL ma postać:

```
mysql:host=nazwa_serwera;port=numer_portu;dbname=nazwa_bazy
```

Poszczególne składowe mają następujące znaczenie:

- ◆ *nazwa_serwera* — określenie nazwy lub adresu serwera baz danych (np. localhost, 127.0.0.1).
- ◆ *numer_portu* — port, na którym ma nastąpić połączenie z bazą. Może być pominięty, zostanie wtedy zastosowana wartość standardowa.
- ◆ *nazwa_bazy* — nazwa bazy danych, z którą ma nastąpić połączenie.

Tak więc ciąg *źródło_danych*, pozwalający na nawiązanie połączenia z bazą testphp, która znajduje się na serwerze MySQL pracującym na komputerze lokalnym *localhost* na porcie 3306, będzie miał postać:

```
mysql:host=localhost;port=3306;dbname=testphp
```

a jeśli określenie portu zostanie pominięte, wówczas:

```
mysql:host=localhost;dbname=testphp
```

Jeżeli wywołanie konstruktora zakończy się sukcesem (uda się nawiązać połączenie z daną bazą), zostanie zwrócony obiekt, który będzie służył do dalszej komunikacji, dlatego należy przypisać go jakiejś zmiennej. W przypadku gdy połączenia nie uda się nawiązać, zostanie wygenerowany wyjątek (wyjątki były opisane w rozdziale 5.) typu `PDOException`, który będzie zawierał opis przyczyny powstania błędu.

Zamykanie połączenia

Połączenie nawiązane za pomocą obiektu PDO pozostaje aktywne przez cały czas życia obiektu i jest kończone przy usuwaniu obiektu z pamięci. Zostanie to wykonane automatycznie po zakończeniu pracy skryptu lub po przypisaniu wartości `null` zmien-

nej obiektowej przechowującej odwołanie do obiektu. Przy założeniu, że jest to zmienna o nazwie `$dbo`, efekt ten zostanie osiągnięty po użyciu przypisania:

```
$dbo = null;
```

Testowanie połączenia z bazą

Na listingu 14.3 został przedstawiony fragment kodu ustalający, czy udało się nawiązać połączenie z bazą danych. Podobnie jak we wcześniejszych przykładach, połączenie jest nawiązywane z bazą `testphp` znajdującą się na serwerze lokalnym `localhost`, gdzie nazwa użytkownika to `php`, a jego hasło to `test`. Jeżeli w trakcie tworzenia obiektu (nawiązywania połączenia) wystąpi błąd, zostanie wygenerowany wyjątek (typu `PDOException`), który następnie będzie przechwycony w bloku `try...catch`. Komunikat związany z tym wyjątkiem będzie pobierany przez wywołanie metody `getMessage` obiektu `$e` i użyty jako część informacji wyświetlanej na ekranie.

Listing 14.3. Testowanie połączenia z bazą przy użyciu PDO

```
<?php
$dsn = "mysql:host=localhost;dbname=testphp";
$uzytkownik = "php";
$haslo = "test";
try{
    $dbo = new PDO($dsn, $uzytkownik, $haslo);
}
catch (PDOException $e){
    echo 'Błąd podczas otwierania połączenia: ' . $e->getMessage();
    exit;
}
echo 'Połączenie z bazą danych zostało nawiązane...<br />';

//tutaj instrukcje wykonujące operacje na bazie danych

$dbo = null;
echo 'Połączenie z bazą danych zostało zamknięte...<br />';
?>
```

Wykonywanie zapytań pobierających dane

Styl proceduralny — `mysqli`

W stylu proceduralnym (przy użyciu `mysqli`) zapytania są wysyłane do bazy za pomocą funkcji `mysqli_query`, której w postaci argumentów należy podać identyfikator połączenia (zwrócony przez wywołanie `mysqli_connect`) oraz treść zapytania. Schematycznie takie wywołanie ma postać:

```
mysqli_query(identyfikator, zapytanie[, tryb])
```

Opcjonalny argument *tryb* pozwala ustalić tryb, w jakim będą przetwarzane wyniki zapytania. Może przyjmować wartości `MYSQLI_STORE_RESULT` lub `MYSQLI_USE_RESULT`. Trybem domyślnym jest `MYSQLI_STORE_RESULT`. Oznacza to, że cały wynik zapytania będzie buforowany i przesłany do klienta, a po zakończeniu działania funkcji będzie można wykonywać kolejne zapytania. To powoduje znacząco większe zużycie zasobów systemowych, w tym — pamięci. W trybie drugim (`MYSQLI_USE_RESULT`) wyniki zapytania nie są buforowane, ale przesyłane do klienta na żądanie wiersz po wierszu. Zmniejsza to zużycie zasobów, jednak blokuje też możliwość wykonywania jakichkolwiek innych operacji, dopóki wynik nie zostanie zwolniony za pomocą funkcji `mysqli_free_result`. Wszystkie prezentowane dalej przykłady będą pracowały w trybie domyślnym `MYSQLI_STORE_RESULT`.

Wartość zwracana przez funkcję jest zależna od typu zapytania. Jeśli było to zapytanie typu `SELECT`, `SHOW`, `EXPLAIN` lub `DESCRIBE`, wartością zwracaną jest identyfikator zasobów (pozwalający na dalsze przetwarzanie danych), o ile wykonanie zapytania zakończyło się sukcesem, lub wartość `false`, jeżeli wykonanie zapytania nie powiodło się. W przypadku pozostałych typów zapytań zwracaną wartością jest `true`, jeśli zapytanie było poprawne, lub `false` — w przeciwnym razie.

W przypadku zapytań typu `SELECT` funkcja `mysql_query` zwraca identyfikator zasobów (ściślej rzecz ujmując: obiekt typu `mysqli_result`), który może zostać następnie użyty do odczytu danych zwróconych przez zapytanie. Istnieje kilka funkcji odczytujących takie dane. Są to:

- ◆ `mysqli_fetch_all` — zwraca wszystkie wiersze wynikowe w postaci tablicy (asocjacyjnej, numerycznej lub jednocześnie asocjacyjnej i numerycznej),
- ◆ `mysqli_fetch_array` — zwraca pojedynczy wiersz wynikowy w postaci tablicy (asocjacyjnej, numerycznej lub jednocześnie asocjacyjnej i numerycznej),
- ◆ `mysqli_fetch_assoc` — zwraca pojedynczy wiersz wynikowy w postaci tablicy asocjacyjnej,
- ◆ `mysqli_fetch_field_direct` — pobiera metadane z pojedynczego pola,
- ◆ `mysqli_fetch_field` — zwraca wartość kolejnego pola danych,
- ◆ `mysqli_fetch_fields` — zwraca tablicę obiektów reprezentujących pola w wynikach zapytania,
- ◆ `mysqli_fetch_object` — zwraca kolejny wiersz wynikowy w postaci obiektu,
- ◆ `mysqli_fetch_row` — zwraca kolejny wiersz wynikowy w postaci tablicy indeksowanej numerycznie.

Pomocna może być także funkcja `mysql_num_rows`, która pozwala określić, ile wierszy znajduje się w wynikach zapytania.

Każde wywołanie funkcji `mysqli_fetch_array`, `mysqli_fetch_assoc`, `mysqli_fetch_row` zwraca kolejny wiersz z tabeli będącej wynikiem zapytania. Dane zwracane są w tablicy, w której kolejne komórki zawierają dane z kolejnych kolumn tabeli wynikowej. Jeśli zostaną odczytane wszystkie wiersze, funkcja zwraca wartość `false`. Oznacza to, że

wszystkie wyniki zapytania (dla funkcji `mysqli_fetch_row`; w przypadku pozostałych schemat jest identyczny) mogą zostać odczytane w pętli `while` o schematycznej postaci:

```
while($arr = mysqli_fetch_row($result)){
    //instrukcje przetwarzające wyniki
}
```

gdzie `$arr` to tablica, do której będą zapisywane dane z kolejnych wierszy, a `$result` — zmienna zawierająca identyfikator zasobów zwrócony przez funkcję `mysqli_query`. Drugi wariant to użycie funkcji `mysqli_num_rows` do pobrania liczby wierszy z tabeli wynikowej i pętli typu `for`. Taka konstrukcja miałaby schematyczną postać:

```
$count = mysqli_num_rows($result);
for($i = 0; $i < $count; $i++){
    $arr = mysqli_fetch_row($result);
    //instrukcje przetwarzające wyniki
}
```

Sprawdźmy zatem, jak w praktyce odczytać dane z wybranej tabeli, wykorzystując do tego celu funkcję `mysqli_fetch_row`. Niech będzie to wykorzystywana w poprzednim rozdziale tabela *Książki* z bazy *ksiegarnia*. Odpowiedni kod jest widoczny na listingu 14.4.

Listing 14.4. Odczytanie zawartości tabeli

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Pobieranie danych z bazy</title>
</head>
<body>

<?php
if ($db_lnk = mysqli_connect("localhost", "php", "test", "ksiegarnia")){
    //Udało się nawiązać połączenie z bazą

    $query = 'SELECT * FROM Książki';

    if($result = mysqli_query($db_lnk, $query)){
        //Udało się wykonać zapytanie
    }?>

<table>
<tr>
<th>Id</th>
<th>Tytuł</th>
<th>Rok wydania</th>
<th>Cena</th>
</tr>

<?php
//Odczytanie wyników zapytania i umieszczenie ich w tabeli
while($row = mysqli_fetch_row($result)){
    echo "<tr>";
    echo "<td>$row[0]</td>";
    echo "<td>$row[2]</td>";
```

```

        echo "<td>$row[4]</td>";
        echo "<td>$row[6]</td>";
        echo "</tr>";
    }
?>

</table>

<?php
}
else{
    //Wystąpił błąd przy wykonywaniu zapytania
    echo 'Wystąpił błąd: nieprawidłowe zapytanie... ' ;
}
mysqli_close($db_lnk);
}
else{
    //Wystąpił błąd przy próbie połączenia z bazą (serwerem MySQL)
    echo 'Wystąpił błąd podczas próby połączenia z serwerem MySQL... ' ;
}
?>
</body>
</html>

```

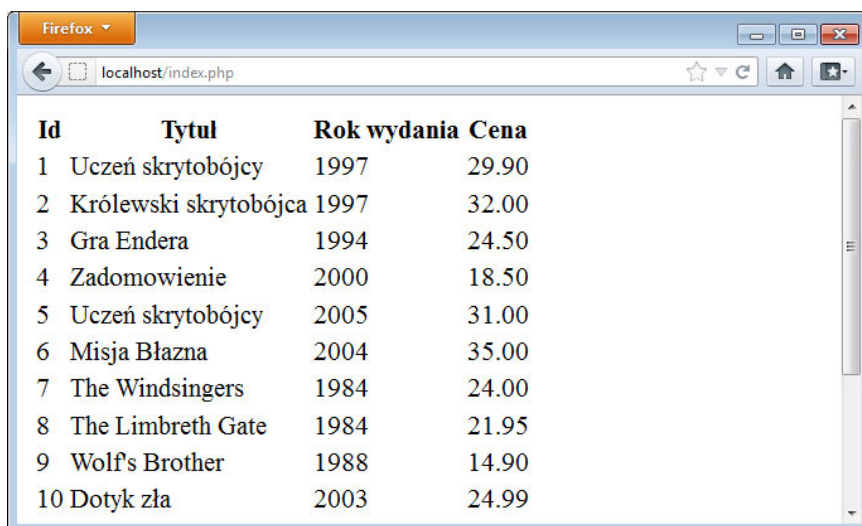
Na listingu kod HTML jest przeplatany z kodem PHP. Połączenie z bazą jest nawiązywane w sposób analogiczny do przedstawionego w przykładzie z listingu 14.1. Jeżeli uda się je nawiązać (funkcja `mysqli_connect` zwróci wartość różną od `null`), zmiennej `$query` jest przypisywany ciąg znaków tworzący treść zapytania SQL, które pobiera wszystkie wiersze z tabeli `Ksiazki`. Zapytanie jest wysyłane do serwera przez wywołanie funkcji `mysqli_query`, a rezultat działania funkcji przypisuje się zmiennej pomocniczej `$result`. Stan tej zmiennej badany jest za pomocą instrukcji warunkowej `if`. Cały kod umieszczony w bloku tej instrukcji (pomiędzy znakami nawiasu klamrowego) będzie wykonany tylko wtedy, gdy zapytanie zostanie wykonane poprawnie, czyli gdy zmienna `$result` zawiera identyfikator (dokładniej: obiekt) wyniku. Dotyczy to również kodu HTML! A zatem tabela tworzona za pomocą znaczników `<table>`, `<tr>`, `<th>` i `<td>` pojawi się na stronie wyłącznie w sytuacji poprawnego przetworzenia danych.

Wyniki zapytania są pobierane w kolejnym bloku PHP — w pętli `while`. Jest w niej wywoływana funkcja `mysqli_fetch_row`, która pobiera kolejne wiersze tabeli wynikowej i zwraca je w postaci tablicy indeksowanej numerycznie. Pod indeksem 0 tej tablicy znajduje się wartość z kolumny `Id`, pod indeksem 1 — wartość z kolumny `WydawnictwoId`, pod indeksem 2 — wartość z kolumny `Tytuł` itd. Innymi słowy, kolejność kolumn w tablicy `row` będzie taka sama jak zwrócona przez wydane zapytanie. Skrypt wyświetla jedynie zawartość wybranych kolumn o indeksach:

- ◆ 0 — kolumna `Id`,
- ◆ 2 — kolumna `Tytuł`,
- ◆ 4 — kolumna `Rok wydania`,
- ◆ 6 — kolumna `Cena`.

Za znacznikiem zamykającym tablicę (znacznik `</table>` poza blokiem PHP) znajduje się ostatnia część skryptu, a w niej bloki `else`. Pierwszy blok dotyczy instrukcji `if` obsługującej wyniki działania funkcji `mysqli_query`, natomiast drugi — instrukcji `if` obsługującej wyniki działania funkcji `mysqli_connect`. W obu przypadkach działanie sprowadza się do wysłania do przeglądarki stosownego komunikatu oraz do zamknięcia połączenia z bazą.

Po uruchomieniu skryptu, o ile nie wystąpią żadne błędy, w przeglądarce pojawi się widok taki jak zaprezentowany na rysunku 14.3. Gdyby w wynikach nie pojawiły się poprawne polskie litery, wskazówki dotyczące rozwiązania tego problemu można znaleźć w podrozdziale „Problem polskich liter”.



Id	Tytuł	Rok wydania	Cena
1	Uczeń skrytobójcy	1997	29.90
2	Królewski skrytobójca	1997	32.00
3	Gra Endera	1994	24.50
4	Zadomowienie	2000	18.50
5	Uczeń skrytobójcy	2005	31.00
6	Misja Błazna	2004	35.00
7	The Windsingers	1984	24.00
8	The Limbreth Gate	1984	21.95
9	Wolf's Brother	1988	14.90
10	Dotyk zła	2003	24.99

Rysunek 14.3. Zawartość tabeli *Książki* wyświetlona w przeglądarce

Jeżeli w miejsce funkcji `mysqli_fetch_row` do pobierania danych zostanie użyta `mysqli_fetch_array`, zamiast indeksów kolumn będzie można wykorzystać ich nazwy. Zakładając zatem, że zmienna `$row` zawiera wynik działania funkcji `mysqli_fetch_array`, dane da się uzyskać za pomocą konstrukcji o schematycznej postaci:

```
$row['nazwa kolumny']
```

Istnieje więc możliwość, aby pętla `while` pobierająca wyniki zapytania miała postać przedstawioną na listingu 14.5. Pozostała część skryptu pozostanie bez zmian (tak jak na listingu 14.4).

Listing 14.5. Wykorzystanie funkcji `mysqli_fetch_array` do pobrania danych

```
<?php
//Odczytanie wyników zapytania i umieszczenie ich w tabeli
while($row = mysqli_fetch_array($result)){
    echo '<tr>';
    echo '<td>' . $row['Id'] . '</td>';
    echo '<td>' . $row['Tytuł'] . '</td>';
```

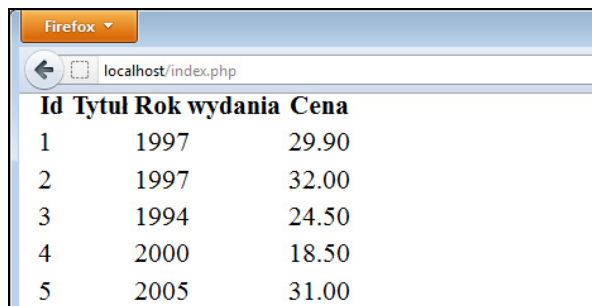
```

    echo '<td>' . $row['Rok wydania'] . '</td>';
    echo '<td>' . $row['Cena'] . '</td>';
    echo '</tr>';
}
?>

```

Taki sposób jest znacznie bardziej czytelny, może jednak również prowadzić do powstania błędów. Zamiast spodziewanych wyników zapytania możemy zobaczyć na ekranie komunikaty o nieprawidłowym indeksie (o ile włączona jest opcja powiadamiania o błędach) oraz pustą kolumnę Tytuł (rysunek 14.4). Jak można się domyślić, problem spowodowany został użyciem w nazwie kolumny polskich liter. Do takiej sytuacji dojdzie wówczas, gdy w kodzie skryptu polskie znaki zostaną zapisane w innym kodowaniu niż w wynikach zapytania zwróconych przez serwer baz danych. Trzeba więc zawsze pamiętać, aby kodowanie liter w skrypcie było takie samo jak kodowanie wyników zapytania.

Rysunek 14.4.
Błędne kodowanie polskich znaków spowodowało brak danych w kolumnie Tytuł



Id	Tytuł	Rok wydania	Cena
1		1997	29.90
2		1997	32.00
3		1994	24.50
4		2000	18.50
5		2005	31.00

Zamiast stosowania do odczytu wyników pętli `while` można również użyć pętli `for`. Schemat postępowania w takim wypadku został przedstawiony na poprzednich stronach. W praktyce wyglądałoby to tak jak na listingu 14.6. Fragment ten należy wstawić w odpowiednie miejsce do kodu z listingu 14.4. Do pobrania liczby wierszy zwróconych przez zapytanie (co jest potrzebne do skonstruowania warunku zakończenia pętli) służy funkcja `mysqli_num_rows`. Efekt działania kodu będzie taki sam jak w dwóch poprzednich przypadkach.

Listing 14.6. *Wykorzystanie pętli `for` do pobrania wyników zapytania*

```

<?php
//Odczytanie wyników zapytania i umieszczenie ich w tabeli
$count = mysqli_num_rows($result);
for($i = 0; $i < $count; $i++){
    $row = mysqli_fetch_row($result);
    echo "<tr>";
    echo "<td>$row[0]</td>";
    echo "<td>$row[2]</td>";
    echo "<td>$row[4]</td>";
    echo "<td>$row[6]</td>";
    echo "</tr>";
}
?>

```

Wykonajmy teraz nieco bardziej zaawansowany przykład, w którym będzie możliwość nie tylko wyświetlania zawartości czterech wybranych kolumn tablicy *Książki*, ale również sortowania wyników względem wybranej kolumny. Indeks kolumny, względem której powinno być wykonywane sortowanie, będzie przekazywany do skryptu w parametrze o nazwie `sortId` za pomocą metody GET. Sortowanie będzie mogło się odbywać względem kolumn o indeksach 0, 2, 4 i 6, czyli *Id*, *Tytuł*, *Rok wydania* lub *Cena*. Przyjmujemy, że jeżeli do skryptu nie zostanie przekazany parametr o nazwie `sortId` lub jego wartość będzie różna od 2, 4 lub 6, sortowanie odbędzie się względem kolumny o indeksie 0, czyli kolumny *Id*. Kod skryptu działającego w taki sposób został przedstawiony na listingu 14.7.

Listing 14.7. *Wyświetlenie zawartości tabeli z możliwością sortowania względem wybranych kolumn*

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Pobieranie danych z bazy</title>
</head>
<body>

<?php
if ($db_lnk = mysqli_connect("localhost", "php", "test", "ksiegarnia")){
    //Udało się nawiązać połączenie z bazą

    $sort = 'Id';

    //Odczytanie wartości parametru sortId i ustalenie
    //kolumny, względem której ma się odbyć sortowanie

    if(isset($_GET['sortId'])){
        switch($_GET['sortId']){
            case 2 : $sort = 'Tytuł'; break;
            case 4 : $sort = 'Rok wydania'; break;
            case 6 : $sort = 'Cena'; break;
        }
    }

    //Konstrukcja zapytania
    $query = 'SELECT * FROM Książki ORDER BY ``. $sort. ``';

    if($result = mysqli_query($db_lnk, $query)){
        //Udało się wykonać zapytanie
    ?>

<table>
<tr>
<th><a href="index.php?sortId=0">Id</a></th>
<th><a href="index.php?sortId=2">Tytuł</a></th>
<th><a href="index.php?sortId=4">Rok wydania</a></th>
<th><a href="index.php?sortId=6">Cena</a></th>
</tr>

<?php
    //Odczytanie wyników zapytania i umieszczenie ich w tabeli
    while($row = mysqli_fetch_row($result)){
```

```

        echo "<tr>";
        echo "<td>$row[0]</td>";
        echo "<td>$row[2]</td>";
        echo "<td>$row[4]</td>";
        echo "<td>$row[6]</td>";
        echo "</tr>";
    }
?>

</table>

<?php
}
else{
    //Wystąpił błąd przy wykonywaniu zapytania
    echo 'Wystąpił błąd: nieprawidłowe zapytanie...';
}
mysqli_close($db_lnk);
}
else{
    //Wystąpił błąd przy próbie połączenia z bazą (serwerem MySQL)
    echo 'Wystąpił błąd podczas próby połączenia z serwerem MySQL...';
}
?>
</body>
</html>

```

Część skryptu nawiązująca połączenie z serwerem MySQL oraz dokonująca wyboru bazy pozostała bez zmian. Po wykonaniu tych czynności tworzona jest pomocnicza zmienna \$sort o początkowej wartości Id. Następnie jest badane, czy do skryptu został przekazany parametr sortId. Jeśli nie został przekazany lub też jego wartość jest różna od 2, 4 bądź 6, wartością zmiennej \$sort pozostaje ciąg znaków Id. W przeciwnym razie zmiennej jest przypisywany jeden z ciągów:

- ◆ Tytuł — gdy parametr sortId jest równy 2,
- ◆ Rok wydania — gdy parametr sortId jest równy 4,
- ◆ Cena — gdy parametr sortId jest równy 6.

Tak określona wartość zmiennej sort jest następnie dołączana do zapytania SQL w postaci:

```
SELECT * FROM Ksiazki ORDER BY
```

a tym samym pozwala na określenie sposobu sortowania. Należy zwrócić uwagę na to, że ponieważ nazwa kolumny, względem której ma się odbywać sortowanie, może zawierać spacje, inne znaki niestandardowe bądź mieć nazwę będącą słowem kluczowym języka SQL, musi być ujęta w znaki ` . Jeśli zatem sortowanie ma się odbywać względem kolumny Rok wydania, konstrukcja zapytania musi być następująca:

```
SELECT * FROM Ksiazki ORDER BY `Rok wydania`
```

Odpowiednio powinny być także skonstruowane nagłówki kolumn w tabeli HTML. Każdy z nich jest odnośnikiem do skryptu, który zawiera parametr sortId o określonej

wartości. Odnośniki są konstruowane w sposób klasyczny za pomocą znacznika `<a>`. Kliknięcie takiego odnośnika spowoduje wykonanie skryptu i wyświetlenie w przeglądarce zawartości tabeli *Książki* posortowanej względem wskazanej kolumny (rysunek 14.5).



<u>Id</u>	<u>Tytuł</u>	<u>Rok wydania</u>	<u>Cena</u>
7	The Windsingers	1984	24.00
8	The Limbreth Gate	1984	21.95
9	Wolf's Brother	1988	14.90
3	Gra Endera	1994	24.50
1	Uczeń skrytobójcy	1997	29.90
2	Królewski skrytobójca	1997	32.00
15	Lovelock	1997	18.50
4	Zadomowienie	2000	18.50
10	Dotyk zła	2003	24.99
6	Misja Błazna	2004	35.00

Rysunek 14.5. Zawartość tabeli *Książki* posortowana względem kolumny *Rok wydania*

Styl obiektowy — `mysqli`

W stylu obiektowym (przy korzystaniu z `mysqli`) zapytania są wysyłane do bazy za pomocą metody `query`, której w postaci argumentu należy podać treść zapytania. Schematycznie takie wywołanie ma postać:

```
$db_obj->query(zapytanie[, tryb])
```

przy założeniu, że zmienna `$db_obj` zawiera obiekt reprezentujący połączenie z bazą danych (utworzony przy pomocy konstruktora klasy `mysqli`). Opcjonalny parametr `tryb` ma takie samo znaczenie jak w przypadku omówionej wyżej metody `mysqli_query`.

Również wartość zwracana przez funkcję, podobnie jak w stylu proceduralnym, jest zależna od typu zapytania. Jeśli było to zapytanie typu `SELECT`, `SHOW`, `EXPLAIN` lub `DESCRIBE`, wartością zwracaną jest obiekt typu `mysqli_result`, o ile wykonanie zapytania zakończyło się sukcesem, lub wartość `false`, jeżeli wykonanie zapytania nie powiodło się. W przypadku pozostałych typów zapytań zwracaną wartością jest `true`, jeśli zapytanie było poprawne, lub `false` — w przeciwnym razie.

Obiekt typu `mysqli_result` uzyskany w rezultacie wywołania metody `query` może zostać następnie użyty do odczytu danych zwróconych przez zapytanie. Używane są w tym celu metody będące odpowiednikami funkcji przedstawionych w poprzednim punkcie. Są to:

-
- ◆ `fetch_all` — zwraca wszystkie wiersze wynikowe w postaci tablicy (asocjacyjnej, numerycznej lub jednocześnie asocjacyjnej i numerycznej),
 - ◆ `fetch_array` — zwraca pojedynczy wiersz wynikowy w postaci tablicy (asocjacyjnej, numerycznej lub jednocześnie asocjacyjnej i numerycznej),
 - ◆ `fetch_assoc` — zwraca pojedynczy wiersz wynikowy w postaci tablicy asocjacyjnej,
 - ◆ `fetch_field_direct` — pobiera metadane z pojedynczego pola,
 - ◆ `fetch_field` — zwraca wartość kolejnego pola danych,
 - ◆ `fetch_fields` — zwraca tablicę obiektów reprezentujących pola w wynikach zapytania,
 - ◆ `fetch_object` — zwraca kolejny wiersz wynikowy w postaci obiektu,
 - ◆ `fetch_row` — zwraca kolejny wiersz wynikowy w postaci tablicy indeksowanej numerycznie.

Przy pracy z zapytaniami pomocna może być także właściwość `num_rows` obiektu typu `mysqli_result`, która zawiera liczbę wierszy zwróconych przez zapytanie (jest to odpowiednik metody `mysqli_num_rows`).

Zasada działania przedstawionych wyżej metod jest taka sama jak odpowiadających im funkcji. Oznacza to, że np. przy użyciu metody `fetch_row` wszystkie wyniki zapytania mogą zostać odczytane w pętli `while` o schematycznej postaci:

```
while($arr = $result->fetch_row()){  
    //instrukcje przetwarzające wyniki  
}
```

gdzie `$result` to obiekt zwrócony przez wywołanie metody `query`.

Przykładowy skrypt (odpowiednik skryptu z listingu 14.4) odczytujący wszystkie dane z tabeli `Ksiazki` przy użyciu interfejsu obiektowego został przedstawiony na listingu 14.8.

Listing 14.8. *Odczyt danych przy użyciu interfejsu obiektowego*

```
<!DOCTYPE html>  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">  
<title>Pobieranie danych z bazy</title>  
</head>  
<body>  
  
<?php  
$db_obj = new mysqli("localhost", "php", "test", "ksiegarnia");  
if(!$db_obj->connect_errno){  
    //Udało się nawiązać połączenie z bazą  
  
    $query = 'SELECT * FROM Ksiazki';  
  
    if($result = $db_obj->query($query)){  
        //Udało się wykonać zapytanie
```

```

?>

<table>
<tr>
<th>Id</th>
<th>Tytuł</th>
<th>Rok wydania</th>
<th>Cena</th>
</tr>

<?php
    //Odczytanie wyników zapytania i umieszczenie ich w tabeli
    while($row = $result->fetch_row()){
        echo "<tr>";
        echo "<td>$row[0]</td>";
        echo "<td>$row[2]</td>";
        echo "<td>$row[4]</td>";
        echo "<td>$row[6]</td>";
        echo "</tr>";
    }
?>

</table>

<?php
    }
    else{
        //Wystąpił błąd przy wykonywaniu zapytania
        echo 'Wystąpił błąd: nieprawidłowe zapytanie...';
    }
    $db_obj->close();
}
else{
    //Wystąpił błąd przy próbie połączenia z bazą (serwerem MySQL)
    echo 'Wystąpił błąd podczas próby połączenia z serwerem MySQL...';
}
?>
</body>
</html>

```

Ogólna struktura kodu jest taka sama jak w przykładzie z listingu 14.4, używana jest tu jednak inna technika dostępu do bazy danych. Połączenie z serwerem MySQL nawiązuje się przez utworzenie nowego obiektu typu `mysqli`, który dalej reprezentowany jest przez zmienną `$db_obj`. Sprawdzenie, czy ta operacja zakończyła się sukcesem, jest przeprowadzane poprzez zbadanie stanu właściwości `connect_errno`, podobnie jak miało to miejsce w przykładzie z listingu 14.2.

Zapytanie pobierające dane jest wysyłane do bazy za pomocą metody `query` (`$db_obj->query($query)`), a obiekt wynikowy (typu `mysqli_result`) trafia do zmiennej `$result`. Poszczególne wiersze wyniku są pobierane w pętli `while` za pomocą wywołań metody `fetch_row` (`$result->fetch_row()`). Ponieważ metoda ta jest dokładnym odpowiednikiem funkcji `mysqli_fetch_row`, przetwarzanie danych odbywa się w taki sam sposób jak w przykładzie z listingu 14.4. Na końcu kodu połączenie z bazą jest zamykane dzięki metodzie `close` (`$db_obj->close()`).

Gdyby do pobierania wierszy wynikowych miała być użyta metoda `fetch_array`, dzięki której uzyskane dane mają postać tablicy asocjacyjnej, pętla `while` z listingu 14.8 przyjąłaby postać widoczną na listingu 14.9. To odpowiednik kodu z listingu 14.5, w którym używana była funkcja `mysqli_fetch_array`.

Listing 14.9. *Użycie metody `fetch_array`*

```
<?php
//Odczytanie wyników zapytania i umieszczenie ich w tabeli
while($row = $result->fetch_array()){
    echo '<tr>';
    echo '<td>' . $row['Id'] . '</td>';
    echo '<td>' . $row['Tytuł'] . '</td>';
    echo '<td>' . $row['Rok wydania'] . '</td>';
    echo '<td>' . $row['Cena'] . '</td>';
    echo '</tr>';
}
?>
```

Odczyt przy użyciu pętli `for` również jest możliwy. W celu uzyskania liczby wierszy w wynikach zapytania należy skorzystać z właściwości `num_rows` (zamiast używanej w poprzednim punkcie funkcji `mysqli_num_rows`). Taka pętla mogłaby mieć postać widoczną na listingu 14.10.

Listing 14.10. *Użycie pętli `for` i właściwości `num_rows`*

```
<?php
//Odczytanie wyników zapytania i umieszczenie ich w tabeli
for($i = 0; $i < $result->num_rows; $i++){
    $row = $result->fetch_array();
    echo '<tr>';
    echo '<td>' . $row['Id'] . '</td>';
    echo '<td>' . $row['Tytuł'] . '</td>';
    echo '<td>' . $row['Rok wydania'] . '</td>';
    echo '<td>' . $row['Cena'] . '</td>';
    echo '</tr>';
}
?>
```

Styl obiektowy — PDO

Przy stosowaniu PDO zapytania pobierające dane są wysyłane do bazy za pomocą metody `query`. Treść zapytania należy przekazać w postaci argumentu. Zakładając zatem, że istnieje obiekt `$db`, który reprezentuje nawiązane połączenie z bazą danych, schematyczne wywołanie tej metody będzie miało postać:

```
$db->query(zapytanie);
```

Jeżeli zapytanie udało się wykonać, metoda query zwraca obiekt typu PDOStatement pozwalający na odczyt danych, jeśli natomiast wystąpił błąd, zwracaną wartością jest false. Obiekt ten udostępnia metody pozwalające na odczyt poszczególnych wierszy zwróconych przez zapytanie. Są to:

- ◆ fetch — zwraca kolejny wiersz wynikowy. Postać zwróconych danych jest zależna od wartości parametru przekazanego metodzie.
- ◆ fetchAll — zwraca tablicę zawierającą wszystkie wiersze zwrócone przez zapytanie.
- ◆ fetchColumn — zwraca wartość wybranej kolumny.
- ◆ fetchObject — zwraca kolejny wiersz wynikowy w postaci obiektu.

Argument metody fetch wskazujący typ zwracanego wyniku może przyjmować następujące wartości:

- ◆ PDO::FETCH_ASSOC — zwraca tablicę asocjacyjną, w której nazwy kolumn wynikowych są kluczami.
- ◆ PDO::FETCH_BOTH — zwraca tablicę indeksowaną zarówno numerycznie, jak i asocjacyjnie (jest to wartość domyślna).
- ◆ PDO::FETCH_BOUND — zwraca wartość true oraz przypisuje wartość z kolumn wyniku do zmiennych PHP ustalonych wcześniej za pomocą wywołania metody bindColumn().
- ◆ PDO::FETCH_CLASS — zwraca nową instancję klasy, dokonując mapowania kolumn wynikowych na właściwości klasy.
- ◆ PDO::FETCH_INTO — uaktualnia istniejącą instancję klasy, dokonując mapowania kolumn wynikowych na właściwości klasy.
- ◆ PDO::FETCH_LAZY — kombinacja PDO::FETCH_BOTH i PDO::FETCH_OBJ.
- ◆ PDO::FETCH_NUM — zwraca tablicę indeksowaną numerycznie.
- ◆ PDO::FETCH_OBJ — zwraca obiekt z właściwościami o nazwach i wartościach odpowiadającym kolumnom wynikowym zapytania.

Aby zmienić domyślny tryb obowiązujący dla wszystkich zapytań (czyli standardowe PDO::FETCH_BOTH), należy wykorzystać metodę setFetchMode, której wywołanie ma postać:

```
setFetchMode([domyślny_typ_wyniku])
```

gdzie *domyślny_typ_wyniku* to jedna z wartości wymienionych wyżej. Jeśli domyślnym typem wyniku ma być tablica indeksowana numerycznie, to zakładając, że zmienna \$result zawiera obiekt PDOStatement, należy zastosować wywołanie:

```
$result->setFetchMode(PDO::FETCH_NUM);
```

Po jego wykonaniu wszystkie wywołania metody fetch będą zwracały tablice numeryczne.

Ponieważ każde wywołanie `fetch` powoduje zwrócenie kolejnego wiersza wyniku lub wartości `false`, jeśli zostały odczytane wszystkie wiersze, metodę tę można wywołać w pętli `while` o schematycznej postaci:

```
while($row = $result->fetchRow()){
    //instrukcje przetwarzające wiersz tabeli
}
```

Aby otrzymać liczbę wierszy wynikowych, należy zastosować metodę `rowCount`, np.:

```
$liczba_wierszy = $result->rowCount();
```

To pozwala do odczytu wyników użyć również pętli `for`:

```
$count = $result->rowCount();
for($i = 0; $i < $count; $i++){
    $row = $result->fetchRow();
    //instrukcje przetwarzające wyniki
}
```

Przykładowy skrypt (odpowiednik skryptów z listingów 14.4 i 14.8) odczytujący wszystkie dane z tabeli *Książki* przy użyciu interfejsu obiektowego PDO został przedstawiony na listingu 14.11.

Listing 14.11. *Odczyt danych z tabeli przy użyciu PDO*

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Pobieranie danych z bazy</title>
</head>
<body>

<?php
$dsn = "mysql:host=localhost;dbname=ksiegarnia";
$uzytkownik = "php";
$haslo = "test";

try{
    $dbo = new PDO($dsn, $uzytkownik, $haslo);
}
catch (PDOException $e){
    //Wystąpił błąd przy próbie połączenia z bazą (serwerem MySQL)
    echo 'Błąd podczas otwierania połączenia: ' . $e->getMessage();
    echo '</body></html>';
    exit;
}

$query = 'SELECT * FROM Książki';

if($result = $dbo->query($query)){
    //Udało się wykonać zapytanie
?>

<table>
<tr>
```

```

<th>Id</th>
<th>Tytuł</th>
<th>Rok wydania</th>
<th>Cena</th>
</tr>

<?php
    //Odczytanie wyników zapytania i umieszczenie ich w tabeli
    while($row = $result->fetch(PDO::FETCH_NUM)){
        echo "<tr>";
        echo "<td>$row[0]</td>";
        echo "<td>$row[2]</td>";
        echo "<td>$row[4]</td>";
        echo "<td>$row[6]</td>";
        echo "</tr>";
    }
?>

</table>

<?php
}
else{
    //Wystąpił błąd przy wykonywaniu zapytania
    echo 'Wystąpił błąd: nieprawidłowe zapytanie...';
}
$dbo = null;
?>
</body>
</html>

```

Połączenie z bazą jest nawiązywane przy tworzeniu obiektu typu PDO, tak jak zostało to opisane wyżej. Dane określające parametry połączenia zostały umieszczone w trzech zmiennych pomocniczych: \$dsn, \$uzytkownik, \$haslo. Gdyby połączenia nie udało się nawiązać, czyli nie powstałby obiekt \$dbo, wygenerowany będzie wyjątek PDOException, który jest przechwytywany za pomocą instrukcji try...catch. Wtedy w bloku catch (za pomocą instrukcji echo) będą wysyłane informacje o przyczynie błędu i skrypt skończy działanie.

Jeżeli jednak wyjątek nie nastąpił, czyli połączenie udało się nawiązać, do bazy za pomocą metody query jest wysyłane zapytanie SQL (zapytanie to znajduje się w pomocniczej zmiennej \$query) pobierające zawartość tabeli Książki. Wynik jego działania jest przypisywany zmiennej \$result. W przypadku gdy wynikiem tym jest false (jest to badane za pomocą instrukcji warunkowej if), pojawia się komunikat o błędzie i skrypt kończy działanie (wykonywany jest blok else). W przeciwnym wypadku (gdy zmienna \$result jest różna od false, a więc zawiera obiekt typu PDOStatement) jest wykonywana dalsza część skryptu (wykonywany jest blok if).

Dane z tabeli wynikowej są pobierane w pętli while przez metodę fetch, a każdy wiersz jest zwracany w postaci tablicy indeksowanej numerycznie (jest to osiągnięte za pomocą argumentu PDO::FETCH_NUM). Tak więc w każdym przebiegu pętli zmienna \$row zawiera dane z pojedynczego wiersza tabeli wynikowej i są one umieszczane w kolejnych komórkach tablicy HTML. Na ekranie pojawi się więc taki sam widok jak zaprezentowany

na rysunku 14.4. Na końcu skryptu połączenie z bazą jest zamykane przez przypisanie zmiennej `$dbo` wartości `null` (warto stosować takie rozwiązanie, choć nie jest to formalnie konieczne, ponieważ połączenie zostałoby i tak zamknięte po zakończeniu pracy skryptu).

Gdyby przy wydawaniu zapytania metodzie `query` został przekazany parametr `PDO::FETCH_ASSOC` (lub `PDO::FETCH_BOTH`), dzięki któremu uzyskane dane miałyby postać tablicy asocjacyjnej, możliwe byłoby używanie nazw kolumn przy odczycie danych z wyników zapytania. Pętla `while` z listingu 14.11 przyjęłaby wtedy postać widoczną na listingu 14.12. To odpowiednik kodu z listingów 14.5 i 14.9, w którym używana była funkcja `mysqli_fetch_array` i metoda `fetch_array`.

Listing 14.12. *Odczyt danych z tablicy asocjacyjnej*

```
<?php
//Odczytanie wyników zapytania i umieszczenie ich w tabeli
while($row = $result->fetch(PDO::FETCH_ASSOC)){
    echo '<tr>';
    echo '<td>' . $row['Id'] . '</td>';
    echo '<td>' . $row['Tytuł'] . '</td>';
    echo '<td>' . $row['Rok wydania'] . '</td>';
    echo '<td>' . $row['Cena'] . '</td>';
    echo '</tr>';
}
?>
```

Z kolei użycie opisanej wyżej metody `rowCount` pozwala zastosować przy odczycie pętlę `for`. Ponieważ metoda ta pozwala na uzyskanie liczby wierszy wynikowych zapytania, uzyskana wartość może być użyta w warunku zakończenia pętli. Taka konstrukcja mogłaby mieć postać widoczną na listingu 14.13. To odpowiednik skryptów z listingów 14.6 i 14.10.

Listing 14.13. *Odczyt danych za pomocą pętli `for`*

```
<?php
//Odczytanie wyników zapytania i umieszczenie ich w tabeli
$count = $result->rowCount();
for($i = 0; $i < $count; $i++){
    $row = $result->fetch(PDO::FETCH_NUM);
    echo "<tr>";
    echo "<td>$row[0]</td>";
    echo "<td>$row[2]</td>";
    echo "<td>$row[4]</td>";
    echo "<td>$row[6]</td>";
    echo "</tr>";
}
?>
```

Dodatkowo warto zobaczyć, jak wyglądałoby przetwarzanie wyników zapytania, gdyby miały być one pobierane jako obiekty (użyty zostanie parametr `PDO::FETCH_OBJ` metody `fetch`). Taką technikę można również spotkać w praktycznych projektach. W takiej sytuacji wartością zwracaną przez metodę `fetch` jest obiekt, którego właściwości odpo-

wiadają kolumnom wynikowym zapytania SQL. To znaczy nazwami właściwości są nazwy kolumn, a ich wartościami — wartości zapisane w tych kolumnach. A zatem pętla `while` generująca kolejne komórki tabeli HTML o zawartości pobieranej z wyników zapytania miałyby w tej technice postać przedstawioną na listingu 14.14.

Listing 14.14. *Pobieranie wyników zapytania w postaci obiektów*

```
<?php
//Odczytanie wyników zapytania i umieszczenie ich w tabeli
while($row = $result->fetch(PDO::FETCH_OBJ)){
    echo "<tr>";
    echo "<td>" . $row->Id . "</td>";
    echo "<td>" . $row->Tytuł . "</td>";
    echo "<td>" . $row->{'Rok wydania'} . "</td>";
    echo "<td>" . $row->Cena . "</td>";
    echo "</tr>";
}
?>
```

Należy tu zwrócić uwagę na sposób odwołania do danych z kolumny Rok wydania. Ponieważ w tej nazwie występuje spacja, bezpośredni zapis uniemożliwiłby poprawną interpretację skryptu. Dlatego za pomocą znaków apostrofu (można też użyć znaków cudzysłowu) z nazwy został utworzony ciąg znaków, ujęty następnie w nawias klamrowy. To oznacza w tym przypadku, że ciąg znaków ma zostać potraktowany jak nazwa właściwości.

Zapytania typu INSERT, UPDATE, DELETE

Styl proceduralny — mysqli

W przypadku zapytań modyfikujących dane w bazie podczas programowania w stylu proceduralnym za pomocą `mysqli` funkcja `mysqli_query` zwraca jedynie wartość `true`, jeśli serwer przyjął zapytanie, lub wartość `false`, jeśli zapytanie zostało odrzucone. Liczbę wierszy, na które zapytanie miało wpływ, można odczytać przez wywołanie funkcji `mysqli_affected_rows`. Wywołanie to ma postać:

```
mysqli_affected_rows(identyfikator)
```

gdzie *identyfikator* jest identyfikatorem połączenia z bazą zwróconym przez funkcję `mysqli_connect`. Na listingu 14.15 został przedstawiony skrypt, którego zadaniem jest dodanie do używanej w poprzednim rozdziale tabeli *Autorzy* nowego wiersza danych.

Listing 14.15. *Skrypt dodający pojedynczy wiersz do tabeli *Autorzy**

```
<!DOCTYPE html>
<html>
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Dodawanie danych do bazy</title>
</head>
<body>

<?php
if ($db_lnk = mysqli_connect("localhost", "php", "test", "ksiegarnia")){
    //Udało się nawiązać połączenie z bazą

    $query = "INSERT INTO Autorzy VALUES(";
    $query .= "7, 'Zbigniew Nienacki'";
    $query .= ")";

    if(mysqli_query($db_lnk, $query)){
        //Udało się wykonać zapytanie

        $rowsNo = mysqli_affected_rows($db_lnk);
        echo "Liczba dodanych rekordów: $rowsNo";
    }
    else{
        //Wystąpił błąd przy wykonywaniu zapytania
        echo 'Wystąpił błąd: nieprawidłowe zapytanie...';
    }
    mysqli_close($db_lnk);
}
else{
    //Wystąpił błąd przy próbie połączenia z bazą (serwerem MySQL)
    echo 'Wystąpił błąd podczas próby połączenia z serwerem MySQL...';
}
?>
</body>
</html>
```

Za pomocą funkcji `mysqli_connect` jest nawiązywane połączenie z serwerem oraz wybierana baza `ksiegarnia`. Ten fragment kodu ma postać analogiczną do przedstawionej w poprzednich przykładach (listingi 14.1 i 14.4). Następnie zmiennej `$query` jest przypisywana treść zapytania SQL w postaci:

```
INSERT INTO Autorzy VALUES(7, 'Zbigniew Nienacki')
```

które dodaje do tabeli `Autorzy` nowego autora. Zapytanie to jest wysyłane do serwera przy użyciu funkcji `mysqli_query`. Następnie za pomocą instrukcji `if` jest sprawdzane, czy wywołanie funkcji zwróciło wartość `true`, czy `false`. Jeśli jest to wartość `false`, czyli zapytanie zostało odrzucone, do przeglądarki jest wysyłany stosowny komunikat (w bloku `else`).

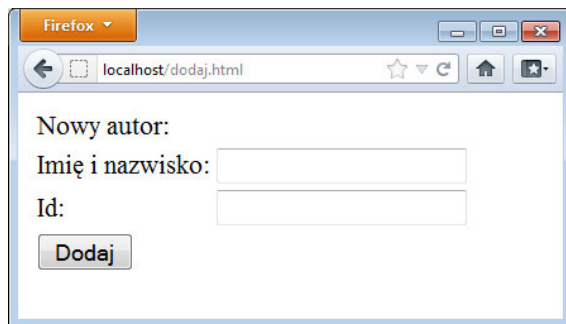
Jeśli jednak zwrócona wartość to `true`, wykonywana jest dalsza część skryptu (w bloku `if`). Zmiennej `$rowsNo` jest przypisywana wartość zwrócona przez wywołanie funkcji `mysqli_affected_rows`, która określa, na ile wierszy w bazie miało wpływ ostatnio wykonane zapytanie. Wartość ta jest następnie wysyłana do przeglądarki, tak aby można było stwierdzić, czy na pewno został dodany jeden wiersz. Na zakończenie połączenie z bazą jest zamykane za pomocą funkcji `mysqli_close`.

Skoro już wiadomo, w jaki sposób dodawać rekord do bazy z poziomu PHP, warto napisać bardziej złożony skrypt, który umożliwi dodawanie dowolnych danych do tabeli Autorzy. Będzie on współpracował z formularzem HTML umożliwiającym wprowadzanie identyfikatora oraz imienia i nazwiska autora. Kod takiego formularza został przedstawiony na listingu 14.16 (należy go zapisać w pliku *dodaj.html*), natomiast jego wygląd zobrazowano na rysunku 14.6 (do formatowania elementów została użyta tabela HTML). Dane wprowadzone do tego formularza będą po kliknięciu przycisku *Dodaj* wysyłane do skryptu o nazwie *dodaj.php* za pomocą metody GET.

Listing 14.16. Formularz HTML umożliwiający wprowadzanie danych

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Dodawanie autora do bazy</title>
</head>
<body>
<form method="get" action="dodaj.php">
<table>
<tr>
<td colspan="2">Nowy autor:</td>
</tr>
<tr>
<td>Imię i nazwisko:</td>
<td><input type="text" name="nazwa"></td>
</tr>
<tr>
<td>Id:</td>
<td><input type="text" name="id"></td>
</tr>
<tr>
<td colspan="2"><input type="submit" value="Dodaj"></td>
</tr>
</table>
</form>
</body>
</html>
```

Rysunek 14.6.
Formularz umożliwiający wprowadzanie danych dotyczących autorów



Treść skryptu *dodaj.php* została przedstawiona na listingu 14.17. Wykonywanie kodu rozpoczyna się od instrukcji *if* sprawdzającej, czy w tablicy *\$_GET* znajdują się indeksy

nazwa i id oraz czy wartość parametru nazwa jest różna od pustego ciągu znaków (w polu nazwa formularza były jakieś dane). Wykorzystywana jest w tym celu znana już funkcja `isset`. Jeżeli wymienione indeksy znajdują się w tablicy `$_GET`, czyli jeśli do skryptu zostały przekazane parametry nazwa oraz id, wywoływana jest funkcja `addToDBTable`, która wykonuje dalsze czynności. W przeciwnym wypadku do przeglądarki wysyłany jest komunikat o braku danych.

Listing 14.17. *Skrypt odbierający dane z formularza i wprowadzający je do bazy danych*

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Dodawanie autora</title>
</head>
<?php

function addToDBTable($baza, $tabela, $kolumny, $wartosci){
    if ($db_lnk = mysqli_connect("localhost", "php", "test", $baza)){
        //Udało się nawiązać połączenie z bazą

        //Zabezpieczenie znaków specjalnych
        foreach ($wartosci as $key => $val){
            $wartosci[$key] = mysqli_real_escape_string($db_lnk, $val);
        }

        //Utworzenie zapytania
        $query = "INSERT INTO `".$tabela.` (`". implode("`", `", $kolumny) .`)";
        $query .= " VALUES('" . implode("'", '", $wartosci) . "')";

        //Próba wykonania zapytania
        if($result = mysqli_query($db_lnk, $query)){
            //Udało się wykonać zapytanie

            $rowsNo = mysqli_affected_rows($db_lnk);
            echo "Zapytanie zostało wykonane.<br />";
            echo "Liczba dodanych rekordów: $rowsNo";
        }
        else{
            //Wystąpił błąd przy wykonywaniu zapytania
            echo 'Wystąpił błąd. Dane nie zostały dodane do tabeli ' . $tabela;
        }
        mysqli_close($db_lnk);
    }
    else{
        //Wystąpił błąd przy próbie połączenia z bazą (serwerem MySQL)
        echo 'Wystąpił błąd podczas próby połączenia z serwerem...';
    }
}

//Sprawdzenie, czy są dostępne parametry oraz wywołanie funkcji dodającej dane
if(isset($_GET['nazwa']) &&
    isset($_GET['id']) && $_GET['nazwa'] != ''){
    $id = strval(intval($_GET['id']));
    $nazwa = $_GET['nazwa'];
```

```
    addToDBTable(
        'ksiegarnia',
        'Autorzy',
        array('Id', 'Nazwa'),
        array($id, $nazwa)
    );
}
else{
    echo 'Niepoprawne dane!';
}
?>
<br /><br />
<a href="dodaj.html">Powrót</a>
</body>
</html>
```

Przed przekazaniem funkcji `addToDBTable` dane otrzymane metodą GET przypisywane są zmiennym pomocniczym `$id` (wartość parametru `id`) i `$nazwa` (wartość parametru `nazwa`). W pierwszym przypadku najpierw wykonywana jest podwójna konwersja wartości. Ciąg znaków zamieniany jest na wartość liczbową za pomocą funkcji `intval` (w ten sposób można pozbyć się nieprawidłowych danych wprowadzonych przez użytkownika), a następnie ponownie zamieniany na ciąg znaków niezbędny do dalszego działania.

Funkcja `addToDBTable` została napisana w taki sposób, aby można było za jej pomocą dodawać dane do rozmaitych tabel w różnych bazach danych. Na stałe zapisane są w niej jedynie: nazwa hosta z serwerem MySQL oraz nazwa i hasło użytkownika (te dane można oczywiście przekazywać również w postaci argumentów bądź też umieścić je w zmiennych globalnych). Funkcja przyjmuje cztery argumenty:

- ◆ `$baza` — nazwa bazy danych.
- ◆ `$tabela` — nazwa tabeli w bazie.
- ◆ `$kolumny` — tablica z listą kolumn, do których mają być wprowadzone dane.
- ◆ `$wartosci` — wartości, które mają być wprowadzone do kolumn wskazanych przez argument `$kolumny`.

Dlatego też wywołanie w skrypcie ma postać:

```
addToDBTable(
    'ksiegarnia',
    'Autorzy',
    array('Id', 'Nazwa'),
    array($id, $nazwa)
);
```

Oznacza ono modyfikację znajdującą się w bazie `ksiegarnia` tabeli `Autorzy`, do której w kolumnach `Id` i `Nazwa` mają być wprowadzone wartości wskazywane przez zmienne `$id` i `$nazwa`. Dzięki utworzonym w locie tablicom będzie można automatycznie skonstruować zapytanie wprowadzające dane do bazy.

W funkcji `addToDBTable` połączenie z serwerem MySQL jest nawiązywane analogicznie do przedstawionego w poprzednich przykładach. Następnie w pętli `for` cała zawartość

tablicy \$wartosci (otrzymanej w postaci argumentu) jest poddawana działaniu funkcji `mysqli_real_escape_string`. To konieczne, ponieważ w otrzymanych danych mogą znajdować się znaki, które nie mogłyby być bezpośrednio użyte w poprawnym zapytaniu SQL¹. Funkcja `mysqli_real_escape_string` przed znakami specjalnymi (NUL, \n, \r, \, ', ", Ctrl-Z) dodaje znak \.



W realnie działającym przykładzie należy rozważyć dodatkową weryfikację danych i ewentualne usuwanie niepożądanych znaków. W przedstawionym skrypcie wszelkie otrzymane dane są wstawiane do bazy niezależnie od ich zawartości i długości. Dotyczy to także potencjalnie niebezpiecznych sekwencji składających się np. na skrypty, które z kolei mogłyby spowodować problemy przy późniejszym wyświetlaniu zawartości na stronie WWW.

Kolejnym krokiem jest utworzenie zapytania SQL typu `INSERT INTO`. Jest ono przypisywane zmiennej \$query i ma schematyczną postać:

```
"INSERT INTO nazwa_tabeli (kolumny) VALUES(wartosci)"
```

Przy tym nazwa tabeli oraz nazwy kolumn powinny być ujęte w znaki lewego apostrofu (``), natomiast wartości kolumn w znaki zwykłego apostrofu prostego ('). Dlatego też w treść zapytania wplecione są dwa wywołania funkcji `implode` (rozdział 4.). Pierwsze tworzy ciąg znaków z wartości pobranych z tablicy \$kolumny:

```
implode("`", `", $kolumny)
```

Jeżeli na przykład w tablicy \$kolumny będą zapisane dwa ciągi: Id i Nazwa, to powyższe wywołanie spowoduje powstanie ciągu:

```
Id`, `Nazwa
```

Drugie wywołanie funkcji `implode` tworzy ciąg znaków z wartości pobranych z tablicy \$wartosci:

```
implode("'", "'", $wartosci)
```

Dla przykładu, jeżeli w tablicy \$wartosci będą zapisane dwa ciągi: 7 i Jan Nowak, to powyższe wywołanie spowoduje powstanie ciągu:

```
7', 'Jan Nowak
```

W rezultacie w zmiennej \$query pojawiłoby się zapytanie w postaci:

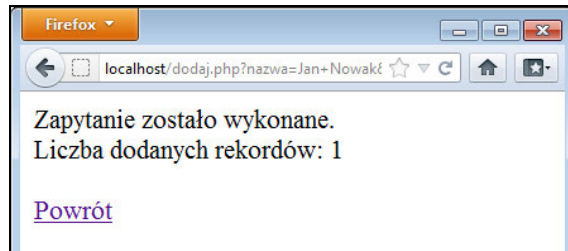
```
INSERT INTO `Autorzy` (`Id`, `Nazwa`) VALUES('7', 'Jan Nowak')
```

Tak skonstruowane zapytanie jest wysyłane do serwera za pomocą funkcji `mysqli_↵query`. W przypadku gdyby zostało ono odrzucone, czyli gdyby wartością zwróconą przez `mysqli_query` było `false`, do przeglądarki jest wysyłany stosowny komunikat i funkcja `addToDBTable` kończy działanie. Jeśli jednak zapytanie zostało przyjęte przez

¹ W tym przykładzie nie dotyczy to co prawda wartości dla kolumny Id (gdyż została ona przetworzona wcześniej w kodzie skryptu i nie może zawierać łańcucha znaków niewskazującego liczby całkowitej), jednak w funkcji trzeba przetworzyć całą tablicę, gdyż nie można mieć pewności, czy i które dane z kolumn były wcześniej przetwarzane.

serwer, wykonywana jest funkcja `mysqli_affected_rows`, która pozwala na stwierdzenie, czy faktycznie do bazy został dodany jeden rekord (rysunek 14.7). Na zakończenie połączenie z serwerem MySQL jest zamykane przez wywołanie funkcji `mysqli_close`.

Rysunek 14.7.
*Ekran potwierdzający
dodanie nowego
rekordu do bazy*



Warto zwrócić uwagę, że przedstawiona realizacja pozwala na automatyczne generowanie identyfikatorów autorów (wartości w kolumnie `Id` tabeli `Autorzy`). Jest to możliwe, ponieważ w bazie `ksiegarnia`, która powstała w rozdziale 13., kolumna `Id` tabeli `Autorzy` ma atrybut `AUTO_INCREMENT` (atrybuty kolumn były opisane w rozdziale 11.). Wystarczy zatem nie wypełniać pola `Id` w formularzu (parametr `id` zostanie wtedy przekazany do skryptu, ale będzie zawierał pusty ciąg znaków) lub też wpisać tam wartość `0`, aby identyfikator został wygenerowany automatycznie.

Należy też pamiętać, że w przypadku parametru `id` weryfikowane jest jedynie to, czy ma on wartość całkowitą. A zatem skrypt dopuści dowolne wartości całkowite, w tym — ujemne. Możliwe więc będzie wprowadzenie do bazy ujemnych identyfikatorów. Takie zachowanie (przechowywanie w tabeli ujemnych identyfikatorów) może być czasem pożądanym, z reguły jednak najczęściej się do tego nie dopuszcza. Trzeba więc wykonać w skrypcie dodatkową weryfikację lub zmodyfikować konstrukcję tabeli, zmieniając typ danych w kolumnie `Id` tak, aby zawierał modyfikator `UNSIGNED`.

Styl obiektowy — `mysqli`

Gdy używamy `mysqli` z interfejsem obiektowym, wykonywanie zapytań modyfikujących dane jest wykonywane za pomocą metody `query`, która zwraca wartość `true`, jeśli serwer przyjął zapytanie, lub wartość `false`, jeśli zapytanie zostało odrzucone. Liczbę wierszy, na które zapytanie miało wpływ, można odczytać dzięki właściwości `affected_rows`.

Przykładowy skrypt dodający do tabeli `autorzy` pojedynczy wiersz danych i wyświetlający informację o tym, czy operacja zakończyła się sukcesem, został przedstawiony na listingu 14.18 (to odpowiednik skryptu z listingu 14.15; część HTML została pominięta).

Listing 14.18. *Dodawanie wiersza danych z użyciem interfejsu obiektowego `mysqli`*

```
<?php
$db_obj = new mysqli("localhost", "php", "test", "ksiegarnia");
if(!$db_obj->connect_errno){
    //Udało się nawiązać połączenie z bazą

    $query = "INSERT INTO Autorzy VALUES(";
```

```

$query .= "7, 'Zbigniew Nienacki'";
$query .= "));

    if($db_obj->query($query)){
        //Udało się wykonać zapytanie

        echo "Liczba dodanych rekordów: " . $db_obj->affected_rows;
    }
    else{
        //Wystąpił błąd przy wykonywaniu zapytania
        echo 'Wystąpił błąd: nieprawidłowe zapytanie...';
    }
    $db_obj->close();
}
else{
    //Wystąpił błąd przy próbie połączenia z bazą (serwerem MySQL)
    echo 'Wystąpił błąd podczas próby połączenia z serwerem MySQL...';
}
?>

```

Nawiązywanie połączenia jest wykonywane w taki sam sposób, jak we wcześniejszych przykładach — tworzony jest obiekt typu `mysqli`. Gdy czynność ta zakończy się sukcesem zmiennej `$query`, zapisywane jest zapytanie SQL:

```
INSERT INTO Autorzy VALUES(7, 'Zbigniew Nienacki')
```

wysyłane następnie do bazy za pomocą metody `query`.

Następnie używa się instrukcji `if` do sprawdzania wartości zwróconej przez `query`. Jeśli jest to wartość `false` (czyli zapytanie zostało odrzucone), do przeglądarki wysyłany jest komunikat o błędzie. Jeżeli zwrócona wartość to `true`, wysyłana jest informacja o liczbie dodanych rekordów, która pobierana jest przez odczytanie właściwości `affected_rows`. Na zakończenie połączenie z bazą jest zamykane za pomocą metody `close`.

Na listingu 14.19 została przedstawiona zmodyfikowana wersja funkcji `addToDBTable` ze skryptu z listingu 14.17. Wykorzystuje ona interfejs obiektowy `mysqli` i pozwala na dodawanie do tabeli autorzy danych otrzymanych za pomocą metody `GET`. Część HTML oraz kod PHP spoza funkcji zostały pominięte, ponieważ mają identyczną postać jak na listingu 14.17. Cały skrypt może współpracować z formularzem HTML z listingu 14.16.

Listing 14.19. Zmodyfikowana wersja funkcji `addToDBTable`

```

function addToDBTable($baza, $tabela, $kolumny, $wartosci){
    $db_obj = new mysqli("localhost", "php", "test", "ksiegarnia");
    if(!$db_obj->connect_errno){
        //Udało się nawiązać połączenie z bazą

        //Zabezpieczenie znaków specjalnych
        foreach ($wartosci as $key => $val){
            $wartosci[$key] = $db_obj->real_escape_string($val);
        }

        //Utworzenie zapytania
        $query = "INSERT INTO `".$tabela.` (`". implode("`", `", $kolumny) .`)";

```

```

$query .= " VALUES('" . implode("'", $wartosci) . "')";

//Próba wykonania zapytania
if($result = $db_obj->query($query)){
    //Udało się wykonać zapytanie

    echo "Zapytanie zostało wykonane.<br />";
    echo "Liczba dodanych rekordów: " . $db_obj->affected_rows;
}
else{
    //Wystąpił błąd przy wykonywaniu zapytania
    echo 'Wystąpił błąd. Dane nie zostały dodane do tabeli ';
    echo $tabela;
}
$db_obj->close();
}
else{
    //Wystąpił błąd przy próbie połączenia z bazą (serwerem MySQL)
    echo 'Wystąpił błąd podczas próby połączenia z serwerem...';
}
}
}

```

Zasada działania funkcji pozostała tu taka sama jak w przykładzie z listingu 14.17. Różnice dotyczą jedynie sposobu odwoływania się do bazy danych. Najpierw tworzony jest obiekt `$db_obj` nawiązujący połączenie z bazą, a następnie wywoływane są jego metody:

- ◆ `real_escape_string` — przetwarza znaki specjalne,
- ◆ `query` — wysyła zapytanie do bazy,
- ◆ `close` — zamyka połączenie z bazą.

W celu uzyskania liczby rekordów, na które miało wpływ zapytanie (liczby rekordów dodanych do bazy), zamiast metody `mysqli_affected_rows` używana jest właściwość `affected_rows`.

Styl obiektowy — PDO

Podczas korzystania z PDO do wykonywania zapytań modyfikujących dane używa się metody `exec`. Wysyła ona zapytanie do serwera oraz zwraca wartość całkowitą określającą liczbę rekordów, na które to zapytanie miało wpływ (czyli np. liczbę zmodyfikowanych, usuniętych lub dodanych wierszy tabeli). Jeżeli wykonanie zapytania nie powiodło się (np. miało złą składnię), metoda zwraca wartość logiczną `false`. Zapytanie należy przekazać w postaci argumentu, zatem schematyczne wywołanie (przy założeniu, że zmienna `$db` zawiera obiekt typu PDO) wygląda następująco:

```
$db->exec("Treść zapytania");
```

Na listingu 14.20 został przedstawiony skrypt dodający do tabeli Autorzy jeden wiersz danych. Jest to odpowiednik skryptów 14.15 i 14.18. Część HTML została pominięta.

Listing 14.20. Dodanie do tabeli wiersza danych przy użyciu PDO

```
<?php
$dsn = "mysql:host=localhost;dbname=ksiegarnia";
$uzytkownik = "php";
$haslo = "test";

try{
    $dbo = new PDO($dsn, $uzytkownik, $haslo);
}
catch (PDOException $e){
    //Wystąpił błąd przy próbie połączenia z bazą (serwerem MySQL)
    echo 'Błąd podczas otwierania połączenia: ' . $e->getMessage();
    echo '</body></html>';
    exit;
}

$query = "INSERT INTO Autorzy VALUES(";
$query .= "7, 'Zbigniew Nienacki'";
$query .= ")";

if(($rowsNo = $dbo->exec($query)) !== false){
    //Udało się wykonać zapytanie
    echo "Liczba dodanych rekordów: $rowsNo";
}
else{
    //Wystąpił błąd przy wykonywaniu zapytania
    echo 'Wystąpił błąd: nieprawidłowe zapytanie...';
}
$dbo = null;
?>
```

Ogólny schemat postępowania pozostał taki sam jak w przykładzie z listingu 14.3. Zapytanie SQL jest natomiast takie samo jak w przykładach 14.15 i 14.18. To zapytanie jest wysyłane do bazy za pomocą metody `exec`, której wynik działania przypisuje się zmiennej `$rowsNo`. Następnie wynik ten jest porównywany z wartością `false`. Dzieje się to w złożonym wyrażeniu warunkowym instrukcji `if`:

```
if(($rowsNo = $dbo->exec($query)) !== false){
```

Jeżeli zapytanie zostanie wykonane prawidłowo, wartość zapisana w `$rowsNo` (zwrócona przez metodę `exec`) będzie różna od `false` i zostanie użyta w komunikacie dla użytkownika. W przeciwnym razie (`$rows` równe `false`) wyświetlana jest informacja o błędnym zapytaniu. Należy zwrócić uwagę, że zwrócenie przez metodę `exec` wartości różnej od `false` nie oznacza, że wiersz na pewno został dodany do tabeli, ale że zapytanie zostało przyjęte przez serwer. Wiersz będzie dodany tylko wtedy, gdy w wyniku zostanie uzyskana wartość 1.

Na listingu 14.21 została przedstawiona wersja skryptu z listingu 14.17, która wykorzystuje interfejs obiektowy PDO w celu dodania wiersza danych do tabeli `Autorzy`. Część HTML została pominięta. Struktura kodu PHP (w przeciwieństwie do przykładu z listingu 14.20) została natomiast nieco zmodyfikowana; do sygnalizacji błędów zostały użyte wyjątki.

Listing 14.21. *Wprowadzanie danych do tabeli Autorzy za pomocą interfejsu PDO*

```
<?php
function addToDBTable($baza, $tabela, $kolumny, $wartosci){
    //Parametry połączenia
    $dsn = "mysql:host=localhost;dbname=ksiegarnia";
    $uzytkownik = "php";
    $haslo = "test";

    $dbo = new PDO($dsn, $uzytkownik, $haslo);

    //Zabezpieczenie danych
    foreach ($wartosci as $key => $val){
        $wartosci[$key] = $dbo->quote($val);
    }

    //Utworzenie zapytania
    $query = "INSERT INTO `{$tabela}` (`". implode("`", `", $kolumny) .`)";
    $query .= " VALUES(" . implode(", ", $wartosci) . ")";

    if(($rowsNo = $dbo->exec($query)) === false){
        //Wystąpił błąd przy wykonywaniu zapytania
        throw new PDOException("Treść zapytania: $query");
    }

    $dbo = null;

    echo "Zapytanie zostało wykonane.<br />";
    echo "Liczba dodanych rekordów: $rowsNo";
}

//Odczytanie parametrów i wywołanie funkcji dodającej dane
if(isset($_GET['nazwa']) &&
    isset($_GET['id']) && $_GET['nazwa'] != ''){
    $id = strval(intval($_GET['id']));
    $nazwa = $_GET['nazwa'];
    try{
        addToDBTable(
            'ksiegarnia',
            'Autorzy',
            array('Id', 'Nazwa'),
            array($id, $nazwa)
        );
    }
    catch(Exception $e){
        echo 'Wystąpił błąd. Szczegóły: ' . $e;
    }
}
else{
    echo 'Niepoprawne dane!';
}
?>
```

Funkcja `addToDBTable` przyjmuje takie same argumenty jak jej wersje z przykładów 14.17 i 14.19. W jej wnętrzu tworzony jest obiekt typu PDO obsługujący połączenie

z bazą danych. Nie jest przy tym przechwytywany wyjątek typu `PDOException`, który może powstać, gdy połączenie nie będzie mogło być nawiązane. Przechwycenie będzie się musiało odbywać podczas wywoływania funkcji.

W pętli `foreach` przetwarzającej dane w celu zabezpieczenia znaków specjalnych, które mogą występować w tabeli `$wartosci`, używana jest metoda `quote`. Dzięki niej przed znakami specjalnymi zostanie umieszczony znak ucieczki odpowiedni dla używanej bazy danych. Dodatkowo przetwarzane ciągi zostaną ujęte w znaki apostrofu prostego. Konsekwencją takiego zachowania jest konieczność zmiany sposobu tworzenia zapytania w części dotyczącej wartości z tablicy `$wartosci`. Obecnie nie należy dodawać do ciągu znaków apostrofu — pomiędzy wartości wystarczy wstawić przecinek. Odpowiedni fragment ma zatem postać:

```
$query .= " VALUES(" . implode(", ", $wartosci) . ")";
```

Zapytanie jest wysyłane do bazy za pomocą metody `exec` umieszczonej w złożonym warunku instrukcji `if`. Wynik działania metody jest przypisywany zmiennej `$rowsNo`. Jest to konstrukcja analogiczna do przedstawionej na listingu 14.20. Różnica jest taka, że w przypadku niepowodzenia (gdy `exec` zwróci wartość `false`) generowany jest wyjątek typu `PDOException`. W komunikacie zawarta jest treść zapytania, tak aby łatwo można było stwierdzić, czy składnia polecenia SQL była poprawna. Jeżeli wykonanie zapytania zakończyło się sukcesem, obiekt `$dbo` jest zerowany, a do przeglądarki wysyłany odpowiedni komunikat.

Ponieważ zmieniło się zachowanie funkcji `addToDBTable` i generuje ona wyjątki, zmodyfikowany został również fragment kodu wywołujący tę funkcję. Wywołanie jest ujęte w blok `try...catch`, a w bloku `catch` znajduje się instrukcja `echo` wyświetlająca informacje o błędzie, w tym komunikat diagnostyczny zawarty w obiekcie wyjątku `$e`.

Wybór sposobu obsługi

W bieżącym rozdziale zostały przedstawione trzy sposoby obsługi baz MySQL przy użyciu PHP. Który z nich wybrać? To oczywiście zależy od indywidualnych preferencji oraz ewentualnie od wymagań i specyfiki projektu, nad którym się aktualnie pracuje. Jeżeli obsługa ma się odbywać w stylu proceduralnym, odpowiedź jest w zasadzie jedna — `mysqli` i interfejs proceduralny. Jeśli preferowany jest styl obiektowy, do wyboru będzie `mysqli` z interfejsem obiektowym lub `PDO`. To drugie rozwiązanie warto zastosować, gdy przewidujemy, że w przyszłości może zaistnieć konieczność zmiany bazy danych. Wtedy użycie `PDO` ułatwi przyszłą migrację, a zmiany w kodzie skryptów będą niewielkie. Z kolei gdy wiadomo, że projekt będzie pracował wyłącznie z MySQL, można zastosować `mysqli`, które jest dedykowane dla tej bazy danych. Decyzję każdy musi podjąć samodzielnie, dostosowując ją do konkretnego zadania. W dalszej części książki będzie używane `mysqli` z interfejsem obiektowym.