Łukasz Sosna

CodeIgniter

Zaawansowane tworzenie stron w PHP

Wolisz proste rozwiązania?

Możesz liczyć na CodeIgniter!

Helion

Redaktor prowadzący: Ewelina Burska
Projekt okładki: Michał Wójcik
Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

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

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

Materiały do książki znajdują się pod adresem:
ftp://ftp.helion.pl/przyklady/codeig.zip

ISBN: 978-83-246-4964-8

Printed in Poland.

Spis treści

Rozdział 4.

Sterowniki systemu do obsługi bazy danych

Sterownik przeznaczony do obsługi bazy danych umożliwia korzystanie z jej zasobów. CodeIgniter pozwala na wykorzystywanie takich baz danych jak na przykład MySQL i PostgreSQL. Dzięki temu jest dogodną platformą do tworzenia zaawansowanych aplikacji — umożliwia przeprowadzenie różnych operacji na rekordach, takich jak umieszczanie ich w tabeli, aktualizowanie czy usuwanie.

Praca z bazą danych

Zanim zaczniemy przygodę z bazą danych, musimy utworzyć dwie bazy: pierwszą o nazwie *codeigniter* oraz drugą — *codeigniter2*. Najprawdopodobniej pracujesz na oprogramowaniu XAMPP, więc w celu ich utworzenia możesz skorzystać z narzędzia phpMyAdmin, do którego dostęp uzyskasz, wpisując adres URL w pasku adresu przeglądarki internetowej: *http://localhost/phpMyAdmin*.

Następnie w pierwszej bazie o nazwie *codeigniter* należy wykonać poniższy kod zapytań SQL:

```
CREATE TABLE IF NOT EXISTS 'tabela' (
  'id' int(11) NOT NULL AUTO_INCREMENT,
  'tytul' varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY ('id')
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci AUTO_INCREMENT=3 ;

INSERT INTO 'tabela' ('id', 'tytul') VALUES
(1, 'Treść 1'),
(2, 'Treść 2');

CREATE TABLE IF NOT EXISTS 'tabela_post' (
  'post_id' int(11) NOT NULL AUTO_INCREMENT,
  'post_tytul' varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  'post_tresc' text COLLATE utf8_unicode_ci NOT NULL,
```

```
  'post_aktywny' char(1) COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY ('post_id')
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci AUTO_INCREMENT=3 ;

INSERT INTO 'tabela_post' ('post_id', 'post_tytul', 'post_tresc', 'post_aktywny') VALUES
(1, 'Temat wiadomości', '', 't'),
(2, 'Kolejny temat wiadomości', '', 't');

CREATE TABLE IF NOT EXISTS 'uzytkownicy' (
  'uzytkownik_id' int(11) NOT NULL AUTO_INCREMENT,
  'uzytkownik_nazwa' varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY ('uzytkownik_id')
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci AUTO_INCREMENT=1 ;
```

W przypadku zainstalowania oprogramowania XAMPP użytkownik mający pełen dostęp do baz danych to *root*, a jego hasło jest puste.

Wczytanie sterownika (load->database)

Sterownik służący do obsługi bazy danych jest uruchamiany poprzez metodę `database()` z obiektu `load` (listing 4.1). Wszystkie dane potrzebne do nawiązania połączenia z bazą danych definiuje się w pliku konfiguracyjnym.

Listing 4.1. *Ładowanie sterownika bazy danych*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
      $this->load->database();
   }
}
?>
```

Połączenie z bazą danych — konfiguracja danych dostępowych

Aby zapewnić sobie możliwość wykorzystania bazy danych, najpierw należy skonfigurować dane dostępowe: określić nazwę hosta, nazwę użytkownika i jego hasło, wskazać nazwę bazy danych oraz zdefiniować inne potrzebne właściwości. W przypadku CodeIgnitera dane te są przechowywane w pliku *application/config/database.php*.

Plik ten zawiera tablicę, w której ustawia się różnego rodzaju dane dostępowe do bazy danych (listing 4.2).

Listing 4.2. *Tablica konfiguracji danych dostępowych do bazy danych*

```
$db['default']['hostname'] = "localhost";
$db['default']['username'] = "root";
$db['default']['password'] = "";
$db['default']['database'] = "codeigniter";
$db['default']['dbdriver'] = "mysql";
$db['default']['dbprefix'] = "";
$db['default']['pconnect'] = TRUE;
$db['default']['db_debug'] = FALSE;
$db['default']['cache_on'] = FALSE;
$db['default']['cachedir'] = "";
$db['default']['char_set'] = "utf8";
$db['default']['dbcollat'] = "utf8_general_ci";
$db['default']['swap_pre'] = "";
$db['default']['autoinit'] = TRUE;
$db['default']['stricton'] = FALSE;
$db['default']['port'] = 3306;
```

Poniżej zostały objaśnione dostępne parametry:

- `hostname` — nazwa hosta bazy danych,
- `username` — nazwa użytkownika,
- `password` — hasło użytkownika,
- `database` — nazwa bazy danych,
- `dbdriver` — typ sterownika bazy danych, przykładowe wartości tego parametru to: *mysql*, *postgres*, *odbc* itd.,
- `dbprefix` — prefiks nazw tabel w bazie danych, który będzie automatycznie dodawany w przypadku korzystania z interfejsu *Active Record*,
- `pconnect` (`TRUE`/`FALSE`) — definiuje, czy chcemy używać stałego połączenia,
- `db_debug` (`TRUE`/`FALSE`) — wyświetlanie komunikatów o błędach bazy danych,
- `cache_on` (`TRUE`/`FALSE`) — włączenie pamięci podręcznej do przechowywania wyników zapytań,
- `cachedir` — katalog na serwerze do tymczasowego przechowywania wyników zapytań do bazy danych,
- `char_set` — zestaw znaków wykorzystywany do pracy z bazą danych,
- `dbcollat` — ustawienie kodowania pól w bazie danych,
- `swap_pre` — prefiks nazw tabel, stosowany podczas pracy nad wersją rozwojową aplikacji,
- `autoinit` — określenie automatycznego łączenia się z bazą danych po włączeniu interfejsu,

- `stricton` (`TRUE`/`FALSE`) — wymuszenie połączenia z bazą danych w trybie *Strict Mode*[1],
- `port` — port wykorzystany do nawiązania połączenia z bazą danych.

Nie wszystkie opcje konfiguracyjne dostępne w pliku *database.php* są zawsze potrzebne. To, jakie dane będziesz musiał zdefiniować, zależy od rodzaju bazy danych. Przedstawiona powyżej konfiguracja jest odpowiednia dla bazy MySQL.

Połączenie z bazą danych (database)

CodeIgniter umożliwia dwa sposoby łączenia się z bazą danych. Pierwszy z nich polega na każdorazowym załadowaniu odpowiedniego sterownika podczas uruchamiania aplikacji, co powoduje automatyczne nawiązanie połączenia z bazą danych. Opcja ta jest bardzo przydatna, gdy mamy zamiar utworzyć aplikację korzystającą z zasobów tylko jednej bazy danych.

Drugim sposobem jest ręczne zestawianie połączenia za pomocą metody `database()`. Pierwszym parametrem tej metody jest nazwa klucza tablicy, w której znajdują się dane potrzebne do połączenia z bazą danych. Jak pamiętasz, tablicę tę deklarujemy w pliku *application/config/database.php*. Kopiujemy już obecną w tym pliku tablicę `$db['default']` i zmieniamy jej klucz na dowolny inny, ważne jednak, by był on zgodny z nazwami kluczy w języku PHP (listing 4.3). Może się on nazywać na przykład *drugiepolaczenie* (listing 4.4).

Listing 4.3. *Fragment zmodyfikowanego pliku database.php*

```
$db['default']['hostname'] = "localhost";
$db['default']['username'] = "root";
$db['default']['password'] = "";
$db['default']['database'] = "codeigniter";
$db['default']['dbdriver'] = "mysql";
$db['default']['dbprefix'] = "";
$db['default']['pconnect'] = TRUE;
$db['default']['db_debug'] = FALSE;
$db['default']['cache_on'] = FALSE;
$db['default']['cachedir'] = "";
$db['default']['char_set'] = "utf8";
$db['default']['dbcollat'] = "utf8_general_ci";
$db['default']['swap_pre'] = "";
$db['default']['autoinit'] = TRUE;
$db['default']['stricton'] = FALSE;
$db['default']['port'] = 3306

$db['drugiepolaczenie']['hostname'] = "localhost";
$db['drugiepolaczenie']['username'] = "root";
$db['drugiepolaczenie']['password'] = "";
```

[1] *Strict Mode* — tzw. tryb ścisły pracy, w którym baza danych nie będzie podejmowała prób obsłużenia danych przekazanych w nieprawidłowym formacie — *przyp. red.*

```
$db['drugiepolaczenie']['database'] = "codeigniter2";
$db['drugiepolaczenie']['dbdriver'] = "mysql";
$db['drugiepolaczenie']['dbprefix'] = "";
$db['drugiepolaczenie']['pconnect'] = TRUE;
$db['drugiepolaczenie']['db_debug'] = FALSE;
$db['drugiepolaczenie']['cache_on'] = FALSE;
$db['drugiepolaczenie']['cachedir'] = "";
$db['drugiepolaczenie']['char_set'] = "utf8";
$db['drugiepolaczenie']['dbcollat'] = "utf8_general_ci";
$db['drugiepolaczenie']['swap_pre'] = "";
$db['drugiepolaczenie']['autoinit'] = TRUE;
$db['drugiepolaczenie']['stricton'] = FALSE;
$db['drugiepolaczenie']['port'] = 3306;
```

Listing 4.4. *Ręczne łączenie się z bazą danych*

```
$this->load->database('drugiepolaczenie');
```

Na etapie tworzenia projektu aplikacji należy się dobrze zastanowić, z ilu baz danych będzie się korzystać. W zależności od tego, czy będzie to jedna baza, czy większa ich liczba, konstrukcja poleceń w metodzie będzie nieco inna (listingi 4.5 i 4.6).

Listing 4.5. *Wykonywanie zapytań w przypadku łączenia się z tylko jedną bazą danych*

```
$this->db->query();
$this->db->result();
```

Listing 4.6. *Wykonywanie zapytań w przypadku łączenia się z dwiema lub większą liczbą baz danych*

```
$DB1->query();
$DB1->result();

$DB2->query();
$DB2->result();
```

Pamiętaj, że poszczególne obiekty obsługujące bazy danych zawierają w swojej nazwie liczby odpowiadające kolejności dokonywania połączeń. Znajomość tych numerów umożliwi prawidłowe odwoływanie się do tych obiektów.

Metoda reconnect()

Podczas pracy aplikacji niekiedy dochodzi do utraty połączenia z bazą danych, na przykład na skutek przekroczenia limitu bezczynności serwera bazy danych. Aby uniknąć ponownego definiowania wszystkich argumentów, w celu ponownego połączenia się z bazą wystarczy wywołać metodę `reconnect()`, która automatycznie podtrzyma połączenie lub — w przypadku jego utraty — ponownie je nawiąże (listing 4.7).

Listing 4.7. *Podtrzymanie lub powtórne nawiązanie połączenia z bazą danych*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
      $this->load->database();

      $Informacje = $this->db->query('SELECT * FROM tabela');

      echo '<pre>';
      print_r($Informacje);
      echo '</pre>';

      $this->db->reconnect();

      $Informacje = $this->db->query('SELECT * FROM tabela');

      echo '<pre>';
      print_r($Informacje);
      echo '</pre>';
   }
}
?>
```

Metoda close()

CodeIgniter zapewnia automatyczne zamykanie połączenia z bazą danych, jednak jeżeli chcesz zamknąć je ręcznie, możesz wywołać metodę `close()` (listing 4.8).

Listing 4.8. *Zakończenie połączenia z bazą danych*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
      $this->load->database();

      $this->db->close();
   }
}
?>
```

Wykonywanie zapytania (query)

Zapytanie, które chcemy wykonać w bazie danych, musi spełniać wymagania standardu języka SQL zgodnego z bazą danych, którą będziemy wykorzystywać. Zapytanie SQL podajemy jako parametr metody `query()` z obiektu `db` (listing 4.9). Przykładowy rezultat wykonania skryptu obrazuje rysunek 4.1.

Listing 4.9. *Wykonanie zapytania SQL*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
      $this->load->database();

      $Informacje = $this->db->query('SELECT * FROM tabela');

      echo '<pre>';
      print_r($Informacje);
      echo '</pre>';
   }
}
?>
```

Rysunek 4.1.
Wynik zapytania

```
CI_DB_mysql_result Object
(
    [conn_id] => Resource id #44
    [result_id] => Resource id #45
    [result_array] => Array
        (
        )

    [result_object] => Array
        (
        )

    [custom_result_object] => Array
        (
        )

    [current_row] => 0
    [num_rows] => 2
    [row_data] =>
)
```

Metoda query()

Same zapytania wysyłane do bazy danych konstruuje się w języku SQL w implementacji odpowiadającej zastosowanej bazie danych. Jednak ze względu na obowiązujące standardy w większości przypadków składnia SQL jest bardzo podobna.

W języku PHP do uzyskiwania wyników zapytań SQL wykorzystuje się metodę `query()`. Parametrem tej metody jest zapytanie w języku SQL. Na listingu 4.10 przedstawiono przykład prostego zapytania do bazy danych.

Listing 4.10. *Zapytanie do bazy danych*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
      $this->load->database();

      $this->db->query('SELECT * FROM tabela');
   }
}
?>
```

W przypadku zapytania SQL pobierającego dane z bazy danych, takiego jak `SELECT`, metoda `query()` zwraca obiekt zbioru wyników odpowiadający wykonanemu zapytaniu. W dalszej części rozdziału pokażemy, w jaki sposób można przetworzyć ten obiekt. W pozostałych przypadkach (kiedy w wyniku zapytania nie otrzymujemy zbioru danych) wartością przekazaną przez metodę `query()` jest `TRUE`, gdy zapytanie zostanie poprawnie wykonane, lub `FALSE`, gdy nastąpi błąd.

Zabezpieczanie danych (escape, escape_str i escape_like_str)

Aby uchronić swoją aplikację przed atakami typu *SQL Injection*, trzeba pamiętać o przefiltrowaniu danych przed ich wprowadzeniem do zapytania SQL i wysłaniem do bazy danych. Jest to bardzo ważne ze względu na bezpieczeństwo naszej aplikacji. W tabeli 4.1 przedstawiono dostępne w CodeIgniterze metody umożliwiające zabezpieczanie danych wpisywanych przez użytkownika poprzez wstawianie lewych ukośników przed znakami specjalnymi, takimi jak apostrof czy cudzysłów.

Tabela 4.1. *Metody zabezpieczające dane*

Metoda	Opis	Przykład
`$this->db->escape()`	Odpowiedni tylko dla ciągów znakowych	`$Sql = "INSERT INTO tabela (tytul) VALUES (".$this->db->escape($Tytul).")";`
`$this->db->escape_str()`	Odpowiedni dla wszystkich danych, bez względu na ich typ	`$Sql = "INSERT INTO tabela (tytul) VALUES('".$this->db->escape_str($Tytul)."')";`
`$this->db->escape_like_str()`	Odpowiedni dla ciągu znakowego wykorzystanego jako parametr w zapytaniach z użyciem słowa kluczowego `LIKE`	`$Tytul = '20% upustu';` `$Sql = "SELECT id FROM tabela WHERE tytul LIKE '%".$this->db->escape_like_str ($Tytul)."%'";`

Bezpieczne zapytania (query)

Zastosowanie tego typu zapytań pozwala na zapewnienie większego bezpieczeństwa aplikacji, ponieważ programista nie musi pamiętać o sprawdzaniu danych wejściowych. Sterownik bazy danych CodeIgnitera zapewni odpowiednie filtrowanie danych (listing 4.11).

Listing 4.11. *Tworzenie zapytania wiązanego*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
      $this->load->database();

      $Zapytanie = "SELECT * FROM tabela_post WHERE post_id = ? AND post_tytul = ?
      ↪AND post_aktywny = ?";
      $Parametry = array(1, 'Temat wiadomości', 't');

      $this->db->query($Zapytanie, $Parametry);
   }
}
?>
```

Metoda `query()` przyjmuje w tym przypadku dwa parametry. Pierwszym jest zapytanie do bazy danych wraz ze znakami zapytania (`?`) w miejscach, w których powinny się znaleźć konkretne wartości. Drugim parametrem jest tablica z tymi właśnie wartościami.

Musimy pamiętać o jeszcze jednej bardzo ważnej sprawie: liczba wartości w tablicy musi być równa liczbie znaków zapytania umieszczonych w pierwszym parametrze.

Generowanie rezultatów zapytania

Istnieje kilka sposobów generowania rezultatu wykonanego zapytania. Wybór konkretnego sposobu zależy od tego, jaką wartość chcemy uzyskać, w jaki sposób ma zostać zaprezentowana użytkownikowi i do czego ma nam służyć.

Metoda result()

Metoda `result()` daje dostęp do wyniku zapytania uzyskanego z wcześniej omówionej metody `query()`. Wynik ten jest udostępniany jako obiekt, który można przetworzyć w pętli `foreach`. Na listingu 4.12 pokazano przykładowy skrypt. Rezultatem jego wykonania będzie:

```
1 Temat wiadomości t
2 Kolejny temat wiadomości t
```

Listing 4.12. *Odbieranie wyniku zapytania za pomocą metody result()*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
      $this->load->database();

      $Rezultat = $this->db->query('SELECT * FROM tabela_post');

      foreach($Rezultat->result() as $dane)
      {
         echo $dane->post_id;
         echo ' ';
         echo $dane->post_tytul;
         echo ' ';
         echo $dane->post_aktywny;
         echo '<br />';
      }
   }
}
?>
```

Metoda result_array()

Metoda `result_array()` umożliwia otrzymanie wyniku zapytania w postaci zwykłej tablicy asocjacyjnej, w której kluczami są nazwy kolumn z tabeli (listing 4.13).

Listing 4.13. *Odbieranie wyników zapytań za pomocą metody result_array()*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
      $this->load->database();

      $Rezultat = $this->db->query('SELECT * FROM tabela_post');

      foreach ($Rezultat->result_array() as $dane)
      {
         echo $dane['post_id'];
         echo ' ';
         echo $dane['post_tytul'];
         echo ' ';
         echo $dane['post_aktywny'];
         echo '<br />';
      }
   }
}
?>
```

Metoda num_rows()

Aby otrzymać liczbę rekordów uzyskanych w wyniku zapytania, można się posłużyć metodą `num_rows()`. Wywołujemy ją jako metodę obiektu zawierającego rezultat tego zapytania. Na listingu 4.14 pokazano przykładowy skrypt. Wynikiem jego wykonania będzie liczba `2`.

Listing 4.14. *Przekazywanie liczby wybranych rekordów*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
```

```
        $this->load->database();

        $Rezultat = $this->db->query('SELECT * FROM tabela_post');

        echo $Rezultat->num_rows();
    }
}
?>
```

Metoda free_result()

Zwykle PHP automatycznie zwalnia pamięć przy zakończeniu przetwarzania skryptu. Jeśli jednak tworzysz duży projekt i chcesz uniknąć ryzyka wystąpienia błędu PHP spowodowanego wyczerpaniem się przydzielonych aplikacji zasobów pamięci RAM serwera, warto rozważyć ręczne, a nie automatyczne zwalnianie pamięci po każdym wykonaniu zapytania. Do tego celu służy metoda `free_result()`. Na listingu 4.15 znajduje się przykład zastosowania tej metody.

Listing 4.15. *Czyszczenie zawartości zapytania z pamięci serwera*

```
<?php
if(!defined('BASEPATH'))
{
    exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
    public function index()
    {
        $this->load->database();

        $Rezultat = $this->db->query('SELECT * FROM tabela_post');

        foreach ($Rezultat->result_array() as $dane)
        {
            echo $dane['post_id'];
            echo ' ';
            echo $dane['post_tytul'];
            echo ' ';
            echo $dane['post_aktywny'];
            echo '<br />';
        }

        $Rezultat->free_result();
    }
}
?>
```

Metoda insert_id()

Metoda `insert_id()` przekazuje identyfikator wartości, która została ostatnio dodana do tabeli za pomocą zapytania `INSERT`. Jest on pobierany z pola typu `auto_increment`. Rezultatem wykonania skryptu z listingu 4.16 będzie liczba 3.

Listing 4.16. *Wyświetlanie ostatniego dodanego numeru w polu auto_increment*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
      $this->load->database();

      $Rezultat = $this->db->query('INSERT INTO tabela_post
      ↪(post_tytul,post_aktywny) VALUES ("Inny post", "t")');

      echo $this->db->insert_id();
   }
}
?>
```

Metoda affected_rows()

Metoda `affected_rows()` przekazuje liczbę rekordów zmienionych podczas wykonywania ostatniego zapytania `UPDATE`. Rezultatem wykonania skryptu z listingu 4.17 będzie liczba 1.

Listing 4.17. *Wyświetlanie liczby rekordów zmienionych podczas ostatniego zapytania*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
      $this->load->database();

      $Rezultat = $this->db->query('UPDATE tabela_post SET post_tytul = "Nowy post"
      ↪WHERE post_id = 3');

      echo $this->db->affected_rows();
```

```
    }
}
?>
```

Metoda count_all()

Metoda ta zlicza wszystkie rekordy tabeli, której nazwę podano w parametrze. Na listingu 4.18 znajduje się przykładowy skrypt. Rezultatem jego wykonania będzie liczba 3.

Listing 4.18. *Wyświetlanie liczby rekordów w tabeli*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
      $this->load->database();

      echo $this->db->count_all('tabela_post');
   }
}
?>
```

Metoda platform()

Metoda `platform()` pozwala na wyświetlenie nazwy typu bazy danych. Rezultatem wykonania skryptu z listingu 4.19 może być na przykład `mysql`.

Listing 4.19. *Przekazywanie informacji o typie bazy danych*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
      $this->load->database();

      echo $this->db->platform();
   }
}
?>
```

Metoda version()

Metoda `version()` przekazuje numer wersji aktualnie wykorzystywanej bazy danych. Na listingu 4.20 znajduje się odpowiedni przykład. Rezultatem wykonania tego skryptu będzie `5.5.8` (wersja bazy danych może się różnić).

Listing 4.20. *Numer wersji bazy danych*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
      $this->load->database();

      echo $this->db->version();
   }
}
?>
```

Metoda last_query()

Metoda `last_query()` pozwala na wyświetlenie ostatniego zapytania, jakie wykonano na bazie danych. Na listingu 4.21 pokazano przykład wykorzystania tej metody. Rezultatem wykonania tego skryptu będzie `UPDATE tabela_post SET post_tytul = "Nowy post" WHERE post_id = 3`.

Listing 4.21. *Wyświetlanie ostatniego wykonanego zapytania*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
      $this->load->database();

      $Rezultat = $this->db->query('UPDATE tabela_post SET post_tytul = "Nowy post"
      ↪WHERE post_id = 3');

      echo $this->db->last_query();
   }
}
?>
```

Informacje o tabelach

W ramach obsługi baz danych CodeIgniter udostępnia również metody pozwalające na uzyskanie pewnych informacji o tabelach w bazie danych.

Metoda list_tables()

Za pomocą metody `list_tables()` możemy uzyskać nazwy wszystkich tabel z aktualnie wykorzystywanej bazy danych. Na listingu 4.22 pokazano odpowiedni przykład. Rezultatem wykonania tego skryptu będzie wynik:

```
tabela
tabela_post
uzytkownicy
```

Listing 4.22. *Wyświetlanie nazw tabel z obecnie wykorzystywanej bazy danych*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
      $this->load->database();

      $Tabele = $this->db->list_tables();

      foreach ($Tabele as $Tabela)
      {
         echo $Tabela;
         echo '<br />';
      }

   }
}
?>
```

Metoda table_exists()

Metoda `table_exists()` umożliwia sprawdzenie, czy tabela o nazwie podanej jako parametr tej metody istnieje w bazie danych (listing 4.23). Rezultatem wykonania skryptu będzie: `Tabela istnieje w bazie danych`, jeżeli się tam rzeczywiście znajduje, lub komunikat: `Tabela NIE istnieje w bazie danych` w przypadku jej braku.

Listing 4.23. *Sprawdzenie, czy dana tabela istnieje w bazie danych*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
      $this->load->database();

      if($this->db->table_exists('tabela'))
      {
         echo 'Tabela istnieje w bazie danych';
      }
      else
      {
         echo 'Tabela NIE istnieje w bazie danych';
      }
   }
}
?>
```

Informacje o polach tabeli

Twórcy CodeIgnitera udostępnili programistom metody pomocne w uzyskiwaniu dodatkowych informacji na temat pól (kolumn) w tabelach bazy danych.

Metoda list_fields()

Metoda `list_fields()` przekazuje listę wszystkich pól w tabeli, której nazwa została podana jako parametr tej metody. Na listingu 4.24 znajduje się przykładowy kod. Rezultatem jego wykonania będzie:

```
post_id
post_tytul
post_tresc
post_aktywny
```

Listing 4.24. *Informacje o polach tabeli*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
```

```
{
   public function index()
   {
      $this->load->database();

      $Pola = $this->db->list_fields('tabela_post');

      foreach ($Pola as $Pole)
      {
         echo $Pole;
         echo '<br />';
      }

   }
}
?>
```

Metoda field_exists()

Metoda `field_exists()` umożliwia sprawdzenie, czy w tabeli istnieje dane pole. Pierwszym parametrem metody jest nazwa kolumny, a drugim — nazwa tabeli (listing 4.25). Rezultatem wykonania skryptu będzie `Kolumna post_tytul istnieje w tabeli` w przypadku stwierdzenia jej obecności.

Listing 4.25. *Informacja o określonym polu w tabeli*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
      $this->load->database();

      if($this->db->field_exists('post_tytul', 'tabela_post'))
      {
         echo 'Kolumna post_tytul istnieje w tabeli';
      }
      else
      {
         echo 'Kolumna post_tytul NIE istnieje w tabeli';
      }
   }
}
?>
```

Metoda field_data()

Metoda `field_data()` przekazuje podstawowe informacje na temat pola w tabeli, którego nazwę podano jako parametr tej metody. Odpowiedni przykład znajduje się na listingu 4.26. Rezultatem jego wykonania będzie:

```
post_id int 11 1
post_tytul varchar 255 0
post_tresc text 0
post_aktywny char 1 0
```

Listing 4.26. *Wyświetlanie informacji o polach tabeli*

```
<?php
if(!defined('BASEPATH'))
{
   exit('No direct script access allowed');
}

class Mvc extends CI_Controller
{
   public function index()
   {
      $this->load->database();

      $Pola = $this->db->field_data('tabela_post');

      foreach($Pola as $Pole)
      {
         echo $Pole->name;
         echo ' ';
         echo $Pole->type;
         echo ' ';
         echo $Pole->max_length;
         echo ' ';
         echo $Pole->primary_key;
         echo '<br />';
      }
   }
}
?>
```

Poniżej omówiono przekazywane wartości:

- `name` — nazwa pola,
- `type` — typ pola,
- `max_length` — maksymalna długość danych w polu (wielkość przechowanych informacji),
- `primary_key` — jeżeli dane pole stanowi klucz główny tabeli, wartość tego parametru wyniesie 1.

Skorowidz

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

V

W

X

Z

CodeIgniter to ni mniej, ni więcej, tylko doskonały framework, który przyda się wszystkim programistom języka PHP. Pozwala on na szybkie i efektywne zaimplementowanie tych elementów stron czy aplikacji internetowych, których samodzielne przygotowanie jest trudne i czasochłonne. Jego główne zalety to niewielkie wymagania co do serwera, na którym ma działać oprogramowanie, zapewnienie logicznego i wygodnego rozdziału elementów strony, brak potrzeby konfiguracji oraz świetna dokumentacja i życzliwa społeczność użytkowników, chętnie dzielących się z innymi swoim doświadczeniem w pracy z programem.

Ta książka, przeznaczona zarówno dla początkujących, jak i zaawansowanych programistów PHP, pozwoli Ci zapoznać się z wieloma aspektami używania CodeIgniter. Znajdziesz tu informacje o instalacji frameworka i jego strukturze. Zobaczysz, jak działają klasy systemu, jakie funkcje oraz metody warto wykorzystać w różnych sytuacjach i jak działają sterowniki systemu do bazy danych. Będziesz mógł także przećwiczyć tworzenie aplikacji z użyciem tego zestawu bibliotek. Programowanie z frameworkiem CodeIgniter to czysta przyjemność!

- Instalacja CodeIgniter na serwerze
- Używanie frameworka (model, widok, kontroler)
- Klasy systemu
- Sterowniki systemu dla bazy danych
- Pomocnicy (funkcje)
- Tworzenie pierwszej aplikacji

Zyskaj czas z CodeIgniter!

helion.pl
księgarnia internetowa
Nr katalogowy: 8668
Księgarnia internetowa
http://helion.pl
Zamówienia telefoniczne:
0 801 339900
0 601 339900
Informatyka w najlepszym wydaniu

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

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

ISBN 978-83-246-4964-8
9 788324 649648
Cena: 47,00 zł