

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE
O NOWOŚCIACH

ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

PHP i MySQL. Tworzenie aplikacji WWW

Autor: Marc Wandschneider

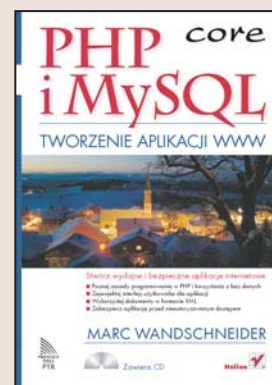
Tłumaczenie: Jarosław Dobrzański, Radosław Meryk

ISBN: 83-246-0323-9

Tytuł oryginału: [Core Web Application](#)

[Development with PHP and MySQL](#)

Format: B5, stron: 856



Stwórz wydajne i bezpieczne aplikacje internetowe

- Poznaj zasady programowania w PHP i korzystania z baz danych
- Zaprojektuj interfejs użytkownika dla aplikacji
- Wykorzystaj dokumenty w formacie XML
- Zabezpiecz aplikację przed nieautoryzowanym dostępem

Sieć WWW już dawno przestała być jedynie zbiorem setek statycznych witryn. Dziś w sieci można znaleźć dziesiątki aplikacji – sklepów, katalogów, systemów bankowych, blogów i wielu innych. Do tworzenia takich aplikacji wykorzystuje się kilka technologii, z których największą popularnością cieszy się język PHP w połączeniu z bazą danych MySQL. Nieodpłatny dostęp, łatwość obsługi oraz potężne możliwości sprawiły, że ta platforma jest stosowana przez ogromne rzesze programistów aplikacji WWW na całym świecie.

„PHP i MySQL. Tworzenie aplikacji WWW” to książka, dzięki której poznasz możliwości tej technologii i nauczysz się z nich korzystać, pisząc aplikacje internetowe. Dowiesz się, jak tworzyć programy w języku PHP, manipulować danymi zgromadzonymi w bazie, projektować rozbudowane aplikacje i wdrażać je. Przeczytasz tu o zabezpieczaniu aplikacji, usuwaniu błędów, korzystaniu z plików XML i usług sieciowych oraz projektowaniu ergonomicznych interfejsów użytkownika. Dzięki praktycznym przykładom szybko nauczysz się stosować PHP i MySQL w swoich projektach.

- Programowanie w języku PHP
- Organizacja kodu
- Zasady programowania obiektowego
- Projektowanie i tworzenie baz danych
- Stosowanie języka SQL
- Przygotowywanie projektu aplikacji
- Budowanie interfejsu użytkownika
- Usuwanie błędów z kodu
- Metody uwierzytelniania użytkowników
- Wyrażenia regularne
- Usługi sieciowe i protokół SOAP
- Wdrażanie aplikacji

Rozpocznij przygodę z programowaniem w PHP



Spis treści

Wprowadzenie	17
Dla kogo przeznaczona jest ta książka?	17
O PHP	18
Układ książki	19
Zanim zaczniesz	20
Podziękowania	20
Część I Podstawy PHP	21
Rozdział 1. Wprowadzenie w tematykę PHP	23
Pierwsze programy w PHP	23
Wpisywanie kodu PHP	25
Oznaczanie sekcji kodu PHP	26
Łączenie PHP i HTML	27
Instrukcje i komentarze	28
W jaki sposób można zapamiętywać dane?	29
Podstawowe typy danych w PHP	30
Liczby	30
Ciągi znaków	32
Wartości typu boolean	35
Przydatne funkcje	36
nl2br	36
var_dump	36
print_r	37
var_export	37
Podsumowanie	38
Rozdział 2. Język PHP	39
Więcej informacji o wprowadzaniu ciągów znaków	39
Więcej informacji o typach danych	41
Tablice	41
Obiekty	42
Specjalne typy i wartości	43

Konwersja typów	44
Podstawy	44
Specyficzne rodzaje konwersji typów	45
Przydatne funkcje do konwersji typów	49
Zmienne i stałe	51
Definiowanie stałych	51
Odwołania do zmiennych przez wartość lub przez referencję	51
Zasięg zmiennych	53
Czas istnienia zmiennych	53
Zmienne predefiniowane	54
Wyrażenia i operatory	55
Operatory: łączenie wyrażeń	55
Łączenie wyrażeń i priorytety operatorów	62
Struktury sterujące	63
Instrukcje if	63
Instrukcja switch	64
Pętle while (do...while)	66
Pętle for	67
Pętle foreach	67
Przerwanie działania pętli	67
Podsumowanie	69
Rozdział 3. Organizacja kodu i jego wielokrotne wykorzystywanie	71
Podstawowy mechanizm wielokrotnego wykorzystywania kodu: funkcje	71
Definiowanie i wywoływanie funkcji	72
Przekazywanie parametrów do funkcji	74
Zwracanie wartości przez funkcje	78
Zasięg zmiennych wewnątrz funkcji	79
Zasięg i dostępność funkcji	83
Funkcje jako zmienne	84
Sposób wielokrotnego wykorzystywania kodu dla średnio zaawansowanych:	
zastosowanie plików włączanych	85
Organizacja kodu w plikach	86
Wybór nazw plików i folderów	88
Włączanie plików bibliotecznych w skryptach	90
Zastosowanie mechanizmu włączania do ładowania szablonów stron	95
Podsumowanie	96
Rozdział 4. Programowanie obiektowe	97
Nie tylko biblioteki	97
Programowanie obiektowe	100
Podstawowa terminologia	100
Podstawowe informacje o programowaniu obiektowym w PHP	101
Inicjalizowanie i usuwanie obiektów	105
Dostępność klasy: kto może oglądać informacje	107
Definiowanie statycznych danych wewnątrz klas	109
Rozszerzanie obiektów	113
Rozszerzanie zdefiniowanych klas	114
Więcej o widoczności	114
Ponowna implementacja metod klasy bazowej	115
Definiowanie klas w taki sposób, aby działały tak samo: polimorfizm	116

Inne własności	123
Porównywanie obiektów	124
Klonowanie obiektów	124
Wyświetlanie wartości obiektów w sposób przyjazny dla użytkownika	125
Przekazywanie wskazówek dotyczących typów	126
Automatyczne ładowanie	127
Podsumowanie	127
Rozdział 5. Tablice	129
Więcej informacji o tablicach	129
Tworzenie tablic i umieszczanie w nich danych	130
Dostęp do elementów tablic	132
Usuwanie elementów i całych tablic	134
Zliczanie elementów tablicy	135
Przetwarzanie elementów tablic w pętli	135
Pętle foreach	135
Pętle standardowe	136
Wewnętrzne liczniki pętli oraz metody each, next, prev, pos i reset	137
Metoda array_walk	139
Tablice wielowymiarowe	140
Operacje na tablicach	141
Sortowanie tablic	142
Inne operacje na tablicach	145
Podsumowanie	147
Rozdział 6. Ciągi znaków i znaki alfabetów narodowych	149
Ciągi znaków w PHP	149
W jaki sposób PHP interpretuje ciągi znaków?	150
Zestawy znaków i kodowanie Unicode	150
Kod ASCII	150
Zestawy znaków ISO 8859	151
Zestawy znaków języków dalekowschodnich	151
Unicode	152
Zestawy znaków Unicode	153
Zestawy znaków w PHP	153
Obsługa innych zestawów znaków	153
Kłopoty, kłopoty...	155
W jaki sposób postępować ze znakami?	156
Konfiguracja PHP w celu wykorzystania kodowania Unicode	157
Instalacja i konfiguracja rozszerzeń mbstring i mbrex	157
Przeciążanie funkcji	157
Operacje na ciągach znaków	158
Pobieranie informacji o ciągach znaków	159
Porządkowanie ciągów znaków	160
Wyszukiwanie i porównywanie	162
Wydzielanie podciągów	165
Zmiana wielkości liter	166
Konwersje kodowania	167
Podsumowanie	167

Rozdział 7. Interakcje z serwerem: formularze	169
Przykłady formularzy	169
Przetwarzanie formularzy HTML	172
Wprowadzanie formularzy na stronie	172
W jaki sposób są przesyłane dane?	174
Dostęp do danych formularzy z poziomu skryptu	176
Formularze a zestawy znaków	178
Praca z serwerem	179
Środowisko serwera	180
Zmienne serwera	180
Zmienne środowiskowe	185
Przekierowania	186
Podsumowanie	188

Część II Podstawowe wiadomości o bazach danych 189

Rozdział 8. Wprowadzenie do baz danych	191
Terminologia	191
Podstawy	191
Relacyjne bazy danych	193
Powody, dla których warto korzystać z systemów zarządzania bazami danych	194
Czynniki przemawiające przeciwko plikom tekstowym bądź arkuszom kalkulacyjnym ...	194
Serwery baz danych	195
Popularne serwery baz danych	196
MySQL	196
PostgreSQL	197
Oracle	197
Microsoft SQL Server	197
Inne	198
W jaki sposób dokonać wyboru serwera bazy danych?	198
Analiza danych	198
Możliwości	199
Wydajność	199
Dostępność z poziomu PHP	199
Koszty	199
Nasz wybór	200
Wspólne interfejsy	200
Podsumowanie	201
Rozdział 9. Projektowanie i tworzenie baz danych	203
Co należy umieścić w bazie danych?	203
Organizacja danych	205
Klucze główne	205
Wybór typów danych	206
Organizacja danych w tabelę	210
Indeksy ułatwiające wyszukiwanie	213
Wprowadzenie do języka SQL	213
Tworzenie baz danych	215
Komunikacja z serwerem bazy danych	215
Nawiązywanie połączenia i uwierzytelnianie	215
Tworzenie bazy danych	216

Nadawanie uprawnień użytkownikom	217
Przygotowania do tworzenia użytkowników	218
Tworzenie użytkowników	219
Usuwanie użytkowników lub uprawnień	220
Tworzenie tabel	221
Typy danych w języku SQL	221
Cykl istnienia klienta bazy danych	225
Tworzenie tabeli	225
Mechanizmy zapisu tabel	227
Tworzenie indeksów	228
Klucze obce i kaskadowe usuwanie rekordów	228
Usuwanie tabel i baz danych	230
Podsumowanie	231
Rozdział 10. Wykorzystanie baz danych: przechowywanie i pobieranie informacji	233
Zanim zacniemy	233
Wprowadzanie danych do tabel	234
Instrukcja INSERT INTO	234
Wprowadzanie danych w trybie masowym	235
Wprowadzanie danych innych typów niż tekstowe	237
Pobieranie danych z tabel	238
Podstawowa składnia	239
Łączenie danych z tabel przy pobieraniu	241
Sortowanie pobieranych danych	244
Pobieranie po kilka wierszy naraz	245
Modyfikowanie danych w tabelach	246
Usuwanie danych z tabel	247
Podsumowanie	248
Rozdział 11. Wykorzystanie baz danych: zaawansowany dostęp do danych	249
Transakcje	249
Problem	250
Rozwiązanie	251
Pisanie transakcji	252
Bardziej złożony problem	253
Bardziej zaawansowane zapytania	255
Łączenie wyrażeń	255
Określanie zbiorów i zakresów wartości	256
Pobieranie niepowtarzalnych wartości	257
Wykorzystanie funkcji SQL w zapytaniach	257
Grupowanie danych dla funkcji agregacji	262
Modyfikowanie schematu tabel	263
Podsumowanie	264
Rozdział 12. PHP a dostęp do danych	265
Przygotowania	265
Nawiązywanie połączenia i uwierzytelnianie	267
Kolejność zdarzeń	267
Nawiązywanie połączenia	268
Ustawianie zestawu znaków obowiązującego dla połączenia	270

Wykonywanie zapytań	270
Pobieranie danych	270
Sprawdzanie poprawności danych wprowadzanych przez użytkownika	273
Wprowadzanie, usuwanie i aktualizowanie danych	275
Transakcje	276
Obsługa błędów rozszerzenia mysqli	277
Wielokrotne wykorzystywanie zapytań	279
Wiązanie parametrów	280
Wiązanie wyników	281
Stara szkoła: interfejsy proceduralne	282
Podstawowe wiadomości dotyczące proceduralnych interfejsów obsługi baz danych	283
Trwałe połączenia	285
Podsumowanie	286

Część III Planowanie aplikacji internetowych **287**

Rozdział 13. Aplikacje internetowe i internet **289**

Światowa pajęczyna „od kuchni”	289
Internet jest prostszy, niż przypuszczasz	290
Komputery komunikujące się z innymi komputerami	290
Protokół HTTP	292
Typy MIME	295
Protokół SSL	295
Inne ważne protokoły	299
Projektowanie aplikacji internetowych	299
Terminologia	300
Prosty układ	300
Interfejs użytkownika	302
Implementacja reguł biznesu	303
Warstwa tylna — serwer	306
Architektury n-warstwowe	307
Wydajność i skalowalność	307
Podsumowanie	309

Rozdział 14. Implementacja interfejsu użytkownika **311**

Elementy interfejsu użytkownika	311
Czym jest interfejs użytkownika?	312
Planowanie interfejsu użytkownika	313
Udzielanie użytkownikom pomocy w rozwiązywaniu problemów	316
Wskazówki i strategie projektowania	317
Implementacja interfejsu użytkownika	319
Kaskadowe arkusze stylów (CSS)	319
Włączanie plików	322
Biblioteki kodu do generowania interfejsu użytkownika	323
Implementacja sterowania interfejsem użytkownika	327
Podsumowanie	329

Rozdział 15. Zarządzanie użytkownikami **331**

W jaki sposób użytkownicy łączą się z aplikacjami internetowymi?	331
Goście a zarejestrowani użytkownicy	334
Uwierzelnianie użytkowników	335

Logowanie częściowe i pełne	335
Gdzie można zapisać informacje użytkowników?	336
Co można zapisać, a czego zapisywać nie należy	337
Podsumowanie	339
Rozdział 16. Zabezpieczenia aplikacji internetowych: planowanie i bezpieczeństwo kodu	341
Strategie zabezpieczeń	341
Co jest najważniejsze?	342
Równowaga pomiędzy bezpieczeństwem a wygodą użytkownika	343
Po zakończeniu fazy tworzenia aplikacji	344
Podstawowe podejście	344
Identyfikacja zagrożeń	344
Zagrożenia	344
Czarne charaktery	347
Zabezpieczenia kodu	349
Złota zasada	349
Filtrowanie danych wejściowych	349
Organizacja kodu	355
Co można umieścić w kodzie?	356
Zagadnienia dotyczące systemu plików	357
Stabilność kodu i błędy	357
Operator wykonywania poleceń powłoki i instrukcja exec	358
Podsumowanie	359
Rozdział 17. Zabezpieczenia aplikacji internetowych: bezpieczeństwo sprzętu i oprogramowania	361
Zabezpieczenia serwerów WWW i PHP	361
Używaj aktualnego oprogramowania	362
php.ini	363
Konfiguracja serwera WWW	364
Serwery wirtualne	365
Aplikacje internetowe umieszczone na serwerach komercyjnych	367
Protokół SSL	368
Zastosowanie SSL w PHP	368
Bezpieczeństwo bazy danych	370
Użytkownicy i system uprawnień	370
Wysyłanie danych do serwera	371
Nawiązywanie połączenia z serwerem	371
Uruchamianie serwera	372
Zabezpieczenia sieci	372
Zapory firewall	373
Strefy DMZ	373
Więcej o sieciowych atakach DoS i DDoS	374
Zabezpieczenia komputera i systemu operacyjnego	374
Pamiętaj o aktualizacjach systemu operacyjnego	374
Uruchamiaj tylko te programy, które są potrzebne	375
Pamiętaj o fizycznych zabezpieczeniach serwera	375
Plany awaryjne	376
Podsumowanie	377

Część IV Implementacja aplikacji internetowych**379**

Rozdział 18. Obsługa błędów i debugowanie	381
W jaki sposób powstają błędy?	381
Błędy PHP	381
Błędy w kodzie	383
Błędy zewnętrzne	384
W jaki sposób PHP zarządza błędami?	385
W jaki sposób PHP wyświetla komunikaty o błędach?	385
Jakie błędy generuje mechanizm obsługi PHP?	386
Obsługa błędów	387
Konfiguracja systemu obsługi błędów w PHP	392
Wyjątki	392
Podstawowe wiadomości o wyjątkach	393
Wykorzystanie wyjątków	395
Nieobsłużone wyjątki	398
Rozszerzanie wyjątków	399
Debugowanie	401
Wprowadzenie instrukcji diagnostycznych	401
Debugery kodu źródłowego	403
Podsumowanie	404
Rozdział 19. Pliki cookie i sesje	405
Pliki cookie: niewielkie i przydatne	405
Podstawowe operacje	405
Jak działają pliki cookie?	409
Zarządzanie poprawnością plików cookie	410
Usuwanie plików cookie	411
Tablice plików cookie	411
Co można umieszczać w plikach cookie?	412
Sposoby na „odchudzających się”	412
Sesje	413
Podstawowe zastosowanie	414
Konfiguracja PHP w celu obsługi sesji	416
W jaki sposób jest przesyłany identyfikator sesji?	417
Zapisywanie danych w sesji	418
Buforowanie stron	421
Niszczanie sesji	423
W jaki sposób działa pamięć sesji	424
Bezpieczeństwo sesji	428
Uzyskanie identyfikatora sesji	428
Ograniczanie możliwych zniszczeń spowodowanych przejęciem identyfikatora sesji	429
Podsumowanie	431
Rozdział 20. Uwierzytelnianie	433
Planowanie logowania	433
Uwierzytelnianie realizowane przez serwer WWW	436
Proste uwierzytelnianie HTTP	436
Schematy uwierzytelniania stosowane w systemie Windows	441
Implementacja własnego mechanizmu uwierzytelniania	442
Konfiguracja bazy danych pod kątem obsługi logowania	443
Dodawanie nowych użytkowników	444

Logowanie użytkowników	453
Aktualizacja stron, które wymagają zalogowania się	460
Wylogowywanie użytkowników	463
Usuwanie użytkowników	465
Podsumowanie	466
Rozdział 21. Zaawansowane techniki wysyłania treści do przeglądarki oraz buforowanie wysyłanej treści	467
Globalizacja i parametry regionalne	467
Parametry regionalne i ich właściwości	468
Ustalanie lokalizacji użytkownika	468
Ustawianie parametrów regionalnych dla bieżącej strony (Unix)	470
Ustawianie parametrów regionalnych dla bieżącej strony (Windows)	472
Poznanie bieżących parametrów lokalnych	474
Wysyłanie sformatowanej treści	474
Formatowanie liczb	475
Waluty	475
Inne funkcje formatujące	478
Buforowanie treści	481
Jak to działa	482
Korzystanie z buforowania treści	482
Pisanie własnego programu obsługi	484
Podsumowanie	485
Rozdział 22. Kontrola poprawności danych za pomocą wyrażeń regularnych	487
Używanie wyrażeń regularnych	487
Czym są wyrażenia regularne?	488
Instalacja	489
Testowanie wyrażeń	489
Proste wyszukiwanie	490
Klasy znaków	491
Granice	493
Kropka	494
Repetycja wzorców	494
Grupowanie i warianty	495
Sztuczki i pułapki	495
Kontrola poprawności danych za pomocą wyrażeń regularnych	496
Kontrola poprawności nazwy użytkownika	496
Wzorce pasujące do numerów telefonów	497
Wzorce pasujące do kodów pocztowych	498
Wzorce pasujące do adresów e-mail	499
Inne funkcje operujące na wyrażeniach regularnych	500
Funkcja <code>ereg_replace</code>	500
Funkcja <code>split</code>	502
Podsumowanie	503
Rozdział 23. XML i XHTML	505
XML	505
Czym jest XML?	506
Kiedy korzystać z XML?	507
Podstawowa terminologia	508

Struktura dokumentu XML	509
Przestrzenie nazw	513
Kontrola poprawności XML	515
Technologie pochodne	517
Praca z XML w PHP	518
Wybór pomiędzy SAX i DOM	519
Korzystanie z modelu DOM	519
XHTML	530
Po co używać XHTML?	530
Jak używać XHTML?	531
Konwersja na XHTML	533
Podsumowanie	533
Rozdział 24. Pliki i katalogi	535
Dostęp do plików	535
Otwieranie plików	535
Zamykanie plików	538
Odczyt plików	538
Zapis do plików	540
Prawa dostępu do plików i inne informacje	542
Usuwanie i zmiana nazw plików	544
Dostęp do katalogów	544
Operowanie ścieżkami dostępu	545
Przeglądanie zawartości katalogów przy użyciu klas	546
Przeglądanie zawartości katalogów za pomocą funkcji	546
Zmiana bieżącego katalogu	547
Tworzenie i usuwanie katalogów	547
Względy bezpieczeństwa	547
Kwestie związane z dostępem do plików	548
Podsumowanie	550
Rozdział 25. Wysyłanie plików do serwera	551
Wysyłanie do serwera plików użytkownika	551
Na czym polega wysyłanie plików do serwera	551
Konfigurowanie PHP pod kątem wysyłania plików do serwera	552
Formularz klienta	553
Kod po stronie serwera	555
Ograniczanie rozmiaru wysyłanego pliku	557
Wysyłanie większej liczby plików	558
Wysyłanie plików do serwera — przykład	560
Przygotowania	560
Formularz nowego konta	561
Tworzenie nowego konta	561
Podgląd danych użytkownika	564
Pobieranie emblematu z bazy danych	566
Względy bezpieczeństwa	567
Tylko zaufani użytkownicy	567
Ataki Denial of Service	568
Kontrola poprawności plików	568
„Złośliwe” nazwy plików	568
Podsumowanie	569

Rozdział 26. Operowanie datami i czasem	571
Źródła dat i czasu	571
PHP	571
System operacyjny	572
Serwer bazy danych	572
Strony WWW i użytkownicy	573
Daty i czas w PHP	574
Datowniki w PHP	574
Pobieranie daty i czasu	575
Kontrola poprawności daty i czasu	579
Porównywanie dat i czasu	580
Wyświetlanie sformatowanych dat i czasu	583
Problem z datownikami	587
Data i czas w serwerach baz danych	587
Zakresy dat i czasu w popularnych serwerach baz danych	587
Dodawanie i odejmowanie interwałów czasowych	588
Interpretacja dat w bazach danych	588
MySQL i datowniki	589
Podsumowanie	589
Rozdział 27. Usługi XML Web Services i SOAP	591
Usługi XML Web Services	591
Tworzenie platformy	592
Wchodzimy do świata usług XML Web Services	592
Szukanie usług Web Services	593
Jak działają usługi Web Services	594
SOAP	594
WSDL	595
HTTP	601
XML-RPC	602
Wykorzystywanie usług Web Services w PHP	602
Wybieranie usługi	602
Konfiguracja PHP	604
Korzystanie z usługi	605
Przykład — korzystanie z API wyszukiwarki Google	609
Przygotowanie do korzystania z interfejsów API wyszukiwarki Google	609
Dalsze poznawanie usługi	610
Jak działa wyszukiwanie	611
Wyszukiwanie słów kluczowych	612
Podsumowanie	616
Rozdział 28. Korzystanie z PEAR	617
Wprowadzenie do PEAR	617
Biblioteka kodu	618
Klasy podstawowe PEAR	618
Społeczność dająca wsparcie	618
PECL	619
Instalacja i konfiguracja	619
Unix	619
Windows	620

Podstawowe polecenia	621
Udostępnianie pomocy	622
Wyświetlanie listy pakietów	622
Pobieranie i instalowanie pakietów	623
Pobieranie informacji	624
Aktualizacja istniejących pakietów	625
Odinstalowywanie pakietów	625
Opcje konfiguracyjne PEAR	626
Przykład — zastosowanie klasy Date	626
Instalacja	627
Podstawowy sposób użycia	627
Dalsze przykłady	628
Podsumowanie	629

Rozdział 29. Tworzenie i wdrażanie aplikacji 631

Standardy pisania kodu	631
Troska o styl	631
Opracowywanie dokumentu ze standardami pisania kodu	633
Święte wojny	635
Inne kwestie do rozważenia	636
Kontrola kodu źródłowego	637
Co nami kieruje?	637
Jak to działa	638
Wybieranie systemu kontroli kodu źródłowego	641
Praca z systemem kontroli kodu źródłowego	642
Testowanie	644
Po co testować?	644
Testowanie modułów	645
Testowanie wydajności i pracy przy obciążeniu	647
Kontrola błędów	650
Wdrażanie	651
Serwery testowe	651
Pisanie skryptów i automatyzacja procesu	652
Wdrażanie na serwery docelowe	652
Podsumowanie	653

Część V Przykładowe projekty i dalsze pomysły 655

Rozdział 30. Strategie tworzenia udanych aplikacji internetowych 657

Obiekty typu singleton	657
Zarządzanie sesjami	660
Konfiguracja	660
Bezpieczeństwo	661
Połączenie elementów w całość	662
Holistyczny sposób obsługi błędów	664
Błędy użytkownika a błędy aplikacji	664
Zastępowanie domyślnych programów obsługi	667
Wyświetlanie komunikatów o błędach	669
Tworzenie nowych klas wyjątków	671

Zarządzanie połączeniami z bazą danych	672
Lepsze rozwiązanie	674
Najlepsze rozwiązanie	675
Nowa, poprawiona funkcja unieszkodliwiająca znaki specjalne	676
Ustawienia konfiguracyjne PHP	677
Ustawienia ogólne	677
Ustawienia związane ze znakami kodowanymi wielobajtowo	678
Ustawienia związane z błędami	679
Ustawienia dotyczące bazy danych	679
Podsumowanie	679
Rozdział 31. System zarządzania terminami	681
Przegląd	681
Instalacja i uruchamianie przykładu	683
Struktura aplikacji i nawigowanie po stronach	683
Struktura strony	684
Układ bazy danych	686
Strategia interfejsu użytkownika	688
Pełna lista plików	689
Analiza kodu	691
Klasa AppointmentManager	691
Obsługa dat i czasu	695
Przetwarzanie formularzy i nawigowanie między stronami	698
Prezentowanie tygodnia i miesiąca	703
Ćwiczenia i sugestie	707
Zmiana widoku tygodniowego i dziennego	707
Tygodnie od poniedziałku do niedzieli	707
Usuwanie lub przenoszenie terminów	708
Konwersja do klasy Date z PEAR	708
Dopuszczenie nakładających się terminów	708
Udostępnienie systemu wielu użytkownikom	708
Podsumowanie	709
Rozdział 32. Blog	711
Przegląd	711
Instalacja i uruchamianie przykładu	713
Struktura aplikacji i nawigowanie między stronami	713
Układ stron	714
Struktura bazy danych i uwagi dotyczące bazy	716
Strategia budowy interfejsu użytkownika	718
Pełna lista plików	718
Analiza kodu	720
Generowanie interfejsu użytkownika	720
Zarządzanie użytkownikami	724
Śledzenie zalogowanych użytkowników	729
Zarządzanie wpisami i komentarzami	734
Sugestie i ćwiczenia	740
Ulepszenie listy użytkowników na stronie głównej	740
Dopuszczenie anonimowych komentarzy	741
Komentarze hierarchiczne	741
Zastosowanie mechanizmu transakcji przy tworzeniu kont użytkowników	741
Implementacja nowej funkcji strip_tags	742
Podsumowanie	742

Rozdział 33. Sklep internetowy	743
Przegląd	743
Instalacja i uruchamianie przykładu	744
Struktura aplikacji i nawigowanie między stronami	746
Podstawowa struktura aplikacji	746
Struktura witryny	748
Struktura bazy danych	750
Strategia budowy interfejsu użytkownika	753
Pełna lista plików	754
Analiza kodu	757
Przeglądanie produktów	757
Implementacja koszyka z zakupami	759
Przetwarzanie w ramach sekwencji kasowej	762
Przesyłanie zamówień	770
Bezpieczeństwo	776
Przetwarzanie płatności	776
Sugestie i ćwiczenia	777
Pominięcie informacji o dostawie	777
Przetwarzanie po złożeniu zamówienia	777
Strony administracyjne	777
Status zamówienia i anulowanie zamówienia	777
Podsumowanie	778
 Dodatki	 779
Dodatek A Instalacja i konfiguracja	781
Dodatek B Odpowiedniki funkcji obsługi baz danych	805
Dodatek C Zalecane materiały źródłowe	813
Skorowidz	815

11

Wykorzystanie baz danych: zaawansowany dostęp do danych

W rozdziale 10. „Wykorzystanie baz danych: przechowywanie i pobieranie informacji” omówiłem język SQL w zakresie wykonywania podstawowych operacji na tabelach: wprowadzania danych, pobierania rekordów, ich aktualizowania i usuwania. W tym rozdziale wykorzystamy poznane podstawy i przejdziemy do bardziej zaawansowanych zagadnień i konstrukcji języka SQL.

W tym rozdziale:

- nauczymy się, gdzie, kiedy i jak korzystać z transakcji;
- dowiemy się, w jaki sposób wykonuje się zaawansowane zadania języka SQL, takie jak korzystanie z wyrażeń i funkcji;
- poznamy sposoby modyfikowania schematów tabeli w celu dodawania nowych kolumn lub zmiany nazw tabel.

Transakcje

Podczas wykonywania operacji na danych zawsze trzeba dbać o zachowanie spójności i integralności danych. O ile system zarządzania bazą danych zapewnia bezpieczne wprowadzanie danych do tabel i pobieranie z nich informacji, nie może zagwarantować, że dane zapisane w bazie zawsze będą miały sens. Przykładowo, jeśli zaznaczyliśmy w tabeli zawierającej informacje o książkach, że sprzedaliśmy trzy książki klientowi, ale przed zapisaniem zamówienia w odpowiedniej tabeli nastąpiła awaria zasilania, baza danych będzie zawierała niespójne informacje. Może również wystąpić problem rywalizacji, kiedy dwóch użytkowników jednocześnie próbuje kupić ostatni egzemplarz książki. W najgorszym przypadku może się zdarzyć, że zamówienie złożą obaj użytkownicy.

Wspomniane problemy można rozwiązać za pomocą odpowiedniego kodu, opracowując mechanizmy „blokowania” tabel w celu uniemożliwienia korzystania z nich innym użytkownikom. Można również utworzyć mechanizmy wykrywające niespójności i niedokończone operacje. Jest to jednak sposób kosztowny, skomplikowany i podatny na błędy. Lepsze efekty można osiągnąć powierzając te działania serwerowi bazy danych za pomocą **transakcji**.

Problem

Przeanalizujemy dokładniej przykład księgarni online. Wyobraźmy sobie, że utworzono prymitywną tabelę zawierającą dane o wszystkich produktach przeznaczonych na sprzedaż oraz tabelę opisującą zamówienia. Dla uproszczenia zakładamy, że można zakupić tylko jeden typ książek. W przykładzie wykorzystamy zdefiniowane poniżej table i pominiemy wiele szczegółów zamówień, takich jak informacje o wysyłce, cenie oraz płatnościach.

```
CREATE TABLE Products
(
  pid INTEGER AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(200) NOT NULL,
  isbn VARCHAR(200) NOT NULL,
  price NUMERIC(10,2) NOT NULL,
  number_in_stock INTEGER NOT NULL
)
ENGINE = InnoDB;

CREATE TABLE Orders
(
  order_id INTEGER AUTO_INCREMENT PRIMARY KEY,
  order_date DATETIME NOT NULL,
  user_id INTEGER NOT NULL,
  product INTEGER NOT NULL,
  num_units INTEGER NOT NULL,
  FOREIGN KEY(user_id) REFERENCES Users(user_id),
  FOREIGN KEY(product) REFERENCES Products(pid)
)
ENGINE = InnoDB;
```

W przypadku sprzedaży książki klientowi powinniśmy wykonać dwa zapytania:

```
UPDATE Products SET number_in_stock = 10 WHERE pid = 343;
INSERT INTO Orders (order_date, user_id, product, num_units)
VALUES (NOW(), 4358, 343, 1);
```

Problem powstanie w przypadku, gdy pierwsze zapytanie wykona się pomyślnie, ale drugie z jakichkolwiek przyczyn nie zostanie wykonane. Taka sytuacja może się zdarzyć z któremkolwiek z następujących powodów:

- utracono połączenie z bazą danych;
- nastąpiła awaria serwera bazy danych bądź brakło miejsca na dysku twardym, co przyczyniło się do niepowodzenia drugiego zapytania;
- z powodu awarii zasilania nastąpiło wyłączenie serwera.

Wielu Czytelników zapewne wzrusza w tej chwili ramionami, mówiąc „taka sytuacja nie może mi się **przytrafić**”. Jeśli jednak weźmiemy za przykład duże aplikacje internetowe, przetwarzające tysiące, jeśli nie miliony transakcji dziennie, przekonamy się, że takie sytuacje zdarzają się, a klienci oczekują, że powstałe problemy zostaną rozwiązane w rozsądny i niekłopotliwy sposób. W powyższym przykładzie na szczęście klient nie został obciążony za coś, czego nie otrzyma, ale byłoby lepiej, gdyby aplikacja mogła automatycznie wykrywać awarie i odtwarzać brakujące pozycje w polu `number_in_stock` tabeli `Products`.

Rozwiązanie

Mając na uwadze możliwe problemy, wprowadzimy pojęcie transakcji. Jest to mechanizm pozwalający na grupowanie wielu zapytań i instrukcji SQL w jedną niepodzielną operację w bazie danych. Albo jest wykonywana i akceptowana (**zatwierdzana**) cała grupa instrukcji, albo żadna z nich i wtedy skutki operacji są **wycofywane**. W rzeczywistości taki mechanizm jest bardziej skomplikowany niż się wydaje na pierwszy rzut oka. System zarządzania relacyjną bazą danych obsługujący transakcje musi spełniać tzw. reguły ACID:

- **Niepodzielność** (ang. *Atomicity*) — transakcje rządzą się zasadą **wszystko albo nic**. Cała grupa działań w obrębie transakcji musi być traktowana jako niepodzielna jednostka. W przypadku niepowodzenia żadna z operacji nie może się wykonać.
- **Spójność** (ang. *Consistency*) — po wykonaniu transakcji baza danych musi znajdować się w spójnym stanie, a wszystkie ograniczenia i reguły integralności danych muszą zostać spełnione.
- **Izolacja** (ang. *Isolation*) — zmiany dokonywane przez transakcję w trakcie jej przeprowadzania nie mogą być dostępne dla innych transakcji. Wewnętrzne operacje różnych transakcji powinny być od siebie odseparowane.
- **Trwałość** (ang. *Durability*) — serwer musi mieć możliwość zakończenia transakcji w trybie nadzwyczajnym. Może to oznaczać, że po pomyślnie zakończonej transakcji pozostaną zaległe operacje lub — jeśli transakcja zostanie przerwana lub anulowana — tylko część operacji zostanie wykonana. W każdym przypadku serwer musi mieć możliwość natychmiastowego zakończenia działania i rekonstrukcji spójnego stanu przy ponownym uruchomieniu (niezależnie od tego, czy mamy do czynienia z zatwierdzoną transakcją, czy z transakcją anulowaną).

Wszystkie bazy danych omawiane w tej książce obsługują transakcje. W systemie MySQL trzeba jednak zwrócić uwagę na wykorzystywany typ tabeli. Jeśli tabele zostały utworzone z wykorzystaniem mechanizmu zapisu `MyISAM`, transakcje nie będą obsługiwane. W tej książce wszędzie tam, gdzie będziemy wykorzystywać transakcje, zastosujemy mechanizm zapisu `InnoDB`.

Niektóre serwery obsługują różne **poziomy izolacji transakcji**. Większość obsługuje cztery poziomy. Na najniższym poziomie (który jednocześnie jest najszybszy) transakcje mają możliwość odczytywania postępu i bieżącego stanu innych transakcji. Na najwyższym poziomie transakcje są całkowicie od siebie odseparowane (poziom najwolniejszy). Istnienie tych poziomów można potraktować jako przyznanie się do potrzeby kompromisu pomiędzy poziomem izolacji transakcji a wydajnością aplikacji. Dla naszych potrzeb odpowiedni poziom izolacji zapewnia poziom domyślny dla większości serwerów baz danych omawianych

w tej książce (**powtarzalny odczyt** — ang. *repeatable read*). Wspomnę jeszcze, że w systemie Oracle domyślnie obowiązuje nieco niższy poziom izolacji, ale to nie stanowi dla nas problemu. Czytelników zainteresowanych bardziej szczegółowym zapoznaniem się z obsługą transakcji polecam sięgnąć do książki, która w bardziej obszerny sposób opisuje ten temat.

Pisanie transakcji

Sposób oznaczania początku transakcji jest różny dla różnych serwerów baz danych, jednak w przypadku większości z nich transakcje rozpoczyna się prostą instrukcją:

```
BEGIN;
```

W niektórych systemach baz danych jest dostępna opcja **automatycznego zatwierdzenia** (ang. *autocommit*) zarządzająca sposobem wykonywania zapytań. Jeśli ustawi się ją na wartość TRUE (lub 1, co jest wartością domyślną), wszystkie wprowadzane zapytania będą automatycznie zatwierdzone w bazie danych. W przypadku ustawienia jej na FALSE (0), zapytania będą rejestrowane tak, jakby wchodziły w skład transakcji. W tym przypadku zatwierdzenie wykonuje się za pomocą jawnej instrukcji. Tak więc w systemie MySQL można rozpocząć transakcję za pomocą instrukcji:

```
SET AUTOCOMMIT =0;
```

Po zakończeniu wykonywania wszystkich zapytań w transakcji, jej wyniki zatwierdza się w bazie danych za pomocą następującej instrukcji:

```
COMMIT;
```

Aby anulować transakcję i cofnąć wykonane w niej operacje, należy zastosować następującą instrukcję:

```
ROLLBACK;
```

W przypadku zakończenia połączenia z serwerem, jego awarii lub przerwania transakcji w inny sposób przed wykonaniem instrukcji COMMIT, automatycznie wykonuje się instrukcja ROLLBACK i transakcja jest anulowana.

Wróćmy do naszego problemu sprzedaży książek. Jeśli skorzystamy z transakcji, sekwencja instrukcji SQL przyjmie następującą postać:

```
BEGIN;
```

```
UPDATE Products SET number_in_stock = 10 WHERE pid = 343;
```

```
INSERT INTO Orders (order_date, user_id, product, num_units)
VALUES(NOW(), 4538, 343, 1);
```

Jeśli doszliśmy do tego miejsca bez błędów:

```
COMMIT;
```

W innym przypadku, jeśli wystąpiły błędy:

```
ROLLBACK;
```

Bardziej złożony problem

Trudno się do tego przyznać, ale w naszym systemie ciągle jeszcze występuje pewien kłopot. Rozwiązaliśmy problem spójności bazy danych w przypadku awarii serwera podczas wykonywania programu sprzedaży, ale może się zdarzyć inna sytuacja — kilku użytkowników będzie chciało zakupić jedną książkę w tym samym czasie.

Spróbujmy przeanalizować sytuację, w której dwóch użytkowników spróbuje zakupić tę samą książkę, podczas gdy w magazynie pozostał tylko jeden egzemplarz.

```
+-----+-----+-----+-----+-----+
| pid | title      | isbn | price | number_in_stock |
+-----+-----+-----+-----+-----+
| 343 | 'Szczęście' | 'xx' | 19.99 | 1 |
+-----+-----+-----+-----+-----+
```

Kod obsługi sprzedaży jest następujący:

```
BEGIN
```

```
SELECT number_in_stock FROM Products WHERE pid = 343;
```

Odjęcie 1 od wartości number_in_stock i ustawienie jej na nową wartość (tu oznaczymy ją jako "nowa")

```
UPDATE Products SET number_in_stock = nowa WHERE pid = 343;
```

```
INSERT INTO Orders(order_date, user_id, product, num_units)
VALUES (NOW(), 4538, 343, 1);
```

Jeśli doszliśmy do tego miejsca bez błędów:

```
COMMIT;
```

W innym przypadku, jeśli wystąpiły błędy:

```
ROLLBACK;
```

Nasz nowy problem wystąpi w przypadku, gdy dwóch użytkowników naszej aplikacji próbuje zakupić tę samą książkę w tym samym czasie. Poniżej zaprezentuję sekwencję operacji wykonywanych przez obu użytkowników w przybliżeniu w tym samym czasie.

Użytkownik 1. rozpoczyna proces zakupu. Wykonują się następujące instrukcje:

```
[Użytkownik 1]
```

```
BEGIN
```

```
SELECT number_in_stock FROM Products WHERE pid = 343;
```

Odjęcie 1 od wartości number_in_stock i ustawienie jej na nową wartość (tu oznaczymy ją jako "nowa")

Program wykonywany przez pierwszego użytkownika stwierdza, że w magazynie jest jeden egzemplarz książki i jest gotowy do obsługi zakupu. Jednak w tym samym czasie robi zakupy drugi użytkownik i wykonuje następujący kod:

```
[Użytkownik 2]
```

```
BEGIN
```

```
SELECT number_in_stock FROM Products WHERE pid = 343;
```

Odjęcie 1 od wartości number_in_stock i ustawienie jej na nową wartość (tu oznaczymy ją jako "nowa")

Zatem program wykonywany przez drugiego użytkownika również stwierdza, że w magazynie pozostał jeden egzemplarz i jest gotowy do obsługi zakupu. Domyślny poziom izolacji transakcji dla naszego serwera bazy danych nie jest najbardziej restrykcyjny, zatem równoległe wykonujące się transakcje mogą odczytać wartości tego samego wiersza. Teraz jednak kod obsługujący pierwszego użytkownika wykona następujące instrukcje:

[Użytkownik 1.]

```
UPDATE Products SET number_in_stock = nowa WHERE pid = 343;

INSERT INTO Orders(order_date, user_id, product, num_units)
VALUES (NOW(), 4538, 343, 1);
COMMIT;
```

Użytkownik numer 1 pomyślnie zakupił książkę. Kiedy drugi użytkownik spróbuje ją zakupić za pomocą następującego kodu:

[Użytkownik 2.]

```
UPDATE Products SET number_in_stock = nowa WHERE pid = 343;

INSERT INTO Orders(order_date, user_id, product, num_units)
VALUES (NOW(), 4538, 343, 1);
COMMIT;
```

wykonanie zapytania UPDATE powiedzie się, ale **nie spowoduje aktualizacji wierszy w tabeli!** Wynika to z faktu, iż program obsługujący transakcję serwera bazy danych stwierdzi, że dane uległy zmianie i nie pozwoli drugiemu procesowi ich zmodyfikować. To jednak nie powoduje zgłoszenia błędu. W kodzie musimy teraz wprowadzić dodatkowe instrukcje, które wykryją, czy interesujący nas wiersz został zmodyfikowany i w takiej sytuacji anulują transakcję lub ponowią próbę aktualizacji, jeśli modyfikacja nie nastąpiła.

Istnieje jednak o wiele bardziej eleganckie rozwiązanie, polegające na zastosowaniu zmodyfikowanej instrukcji SELECT: SELECT... FOR UPDATE. W przypadku pobrania wartości wiersza za pomocą tego zapytania jednocześnie wskazujemy, że mamy zamiar zmodyfikować w nim dane. W tej sytuacji inne transakcje lub użytkownicy próbujący uzyskać dostęp do danych zostaną zablokowani do czasu zakończenia transakcji. Teraz możemy przepisać nasz kod w następujący sposób:

```
BEGIN

SELECT number_in_stock FROM Products
WHERE pid = 343 FOR UPDATE;
```

Odjęcie 1 od wartości number_in_stock i ustawienie jej na nową wartość (tu oznaczymy ją jako "nowa")

```
UPDATE Products SET number_in_stock = nowa WHERE pid = 343;

INSERT INTO Orders(order_date, user_id, product, num_units)
VALUES (NOW(), 4538, 343, 1);
```

Jeśli doszliśmy do tego miejsca bez błędów:

```
COMMIT;
```

W innym przypadku, jeśli wystąpiły błędy:

```
ROLLBACK;
```

Dzięki wprowadzeniu klauzuli FOR UPDATE wszystkie inne procesy próbujące uzyskać dostęp do tej samej wartości będą musiały czekać na wywołanie instrukcji COMMIT lub ROLLBACK. To eliminuje problem rywalizacji o zakup ostatniej książki.

Dla transakcji istnieją bardziej zaawansowane mechanizmy blokowania, ale w tej książce nie będziemy pisali na tyle skomplikowanych programów, aby mogły być potrzebne. Przykłady transakcji zaprezentuję w części V „Przykładowe projekty i dalsze pomysły”.

Bardziej zaawansowane zapytania

Do tej pory omówiłem podstawowe zapytania SQL, jednak możliwości języka są o wiele większe. W tym podrozdziale opiszę niektóre bardziej zaawansowane własności języka SQL.

Łączenie wyrażeń

Słowo kluczowe WHERE umożliwia wprowadzenie warunków ograniczających zakres zwracanych wierszy przez niektóre zapytania, w tym SELECT, UPDATE i DELETE. Wcześniej używaliśmy prostych wyrażeń o postaci:

NazwaKolumny operator wartość

Wyrażenia można jednak łączyć ze sobą i w ten sposób formułować bardziej złożone warunki. Do tego celu służą słowa kluczowe AND i OR. Ich działanie jest podobne do działania operatorów AND i OR w innych językach programowania. Wynik operacji AND jest prawdziwy, jeśli są spełnione warunki po obu stronach operatora, natomiast wynik operacji OR jest prawdziwy, jeśli jest spełniony którykolwiek warunek po dowolnej stronie operatora. Dzięki zastosowaniu operatorów AND i OR można dowolnie formułować zapytania. Na przykład, aby znaleźć wszystkich użytkowników o imieniu Grażyna, którzy urodzili się po roku 1980, można wykonać następujące zapytanie:

```
SELECT user_name, user_email FROM Users
WHERE full_name LIKE 'Grażyna%'
AND birthdate >= '1980-01-01';
```

W podobny sposób można znaleźć użytkowników o imieniu Bogdan lub Bogusław:

```
SELECT user_name, user_email FROM Users
WHERE full_name LIKE 'Bogdan%'
OR full_name LIKE 'Bogusław%';
```

W połączeniach wielu wyrażeń tego typu można i należy stosować nawiasy, które objaśniają kolejność sprawdzania warunków:

```
SELECT user_name, full_name, birthdate FROM Users
WHERE (full_name LIKE 'Bogdan%' OR full_name LIKE 'Bogusław%')
AND birthdate >= '1980-01-01';
```

Określanie zbiorów i zakresów wartości

Do określania zakresów wartości w wyrażeniach w języku SQL są dostępne dwa przydatne słowa kluczowe: IN i BETWEEN.

Słowo kluczowe IN umożliwia określenie zbioru wartości skalarnych (nie można stosować symboli wieloznacznych), do którego ma należeć wartość w kolumnie:

```
SELECT * FROM Messages
WHERE forum_id IN (1, 3, 7, 4);
```

Słowo kluczowe BETWEEN umożliwia zdefiniowanie zakresu wartości. Można je wykorzystać zarówno dla liczb, jak i dla dat:

```
SELECT * FROM Users
WHERE birthdate BETWEEN '1970-01-01' AND '1970-12-31';
SELECT * FROM Messages
WHERE message_id BETWEEN 1000 AND 5000;
```

W przypadku ciągów znaków jest nieco gorzej. Jak wspominałem wcześniej, do określenia zakresu ciągów znaków serwer bazy danych wykorzystuje zdefiniowany porządek sortowania. Dzięki zdefiniowaniu porządku sortowania można określić sposób porządkowania nie tylko danych w języku angielskim, ale także w innych językach. Tak więc następujące zapytanie:

```
SELECT * FROM Users
WHERE user_name BETWEEN 'a' AND 'm';
```

zwraca nazwy użytkowników pomiędzy *a* a *m*. Przy wprowadzeniu tego zapytania może wyświetlić się następujący komunikat o błędzie:

```
ERROR 1270 (HY000): Illegal mix of collations
(utf8_general_ci, IMPLICIT), (latin1_swedish_ci, COERCIBLE),
(latin1_swedish_ci, COERCIBLE) for operation 'BETWEEN';
```

Najbardziej prawdopodobną przyczyną tego błędu jest wykorzystanie w programie klienckim używanym do połączenia z serwerem innego zestawu znaków i porządku sortowania niż ustawiony na serwerze (patrz rozdział 9. „Projektowanie i tworzenie baz danych”). Zazwyczaj na serwerze ustawia się kodowanie Unicode (utf8). Aby wyeliminować problem, należy ustawić w programie klienckim ten sam zestaw znaków i porządek sortowania, jak w bazie danych. W przypadku klienta mysql bazy danych MySQL robi się to za pomocą następującej instrukcji:

```
mysql> set character_set_connection = @@character_set_database;
```

Powyższa instrukcja ustawia także porządek sortowania na domyślny dla wybranego zestawu znaków. Porządek sortowania można również wprowadzić w osobnej instrukcji w następujący sposób:

```
mysql> set collation_connection = @@collation_database;
```

Dotychczasową komplikacją jest fakt, iż niektóre serwery baz danych interpretują zakresy wartości zdefiniowane w klauzuli BETWEEN włącznie z wartościami granicznymi, natomiast inne nie uwzględniają tych wartości. Przed skorzystaniem ze słowa kluczowego BETWEEN należy upewnić się, w jaki sposób są interpretowane granice zakresu (we wszystkich bazach danych omawianych w tej książce granice przedziałów wchodzą w skład zakresu).

Pobieranie неповtarzalnych wartości

Czasami zamiast wszystkich wartości w kolumnie (gdzie mogą występować duplikaty) interesuje nas tylko zbiór możliwości. W przypadku naszej tabeli Messages wykonanie zapytania:

```
SELECT forum_id FROM Messages;
```

może spowodować wyświetlenie wielu powtórzeń nazw forów, na których publikowano więcej niż jedną wiadomość. Jeśli jednak interesuje nas lista forów, na których opublikowano co najmniej jedną wiadomość, możemy w języku SQL zastosować zapytanie `SELECT DISTINCT`:

```
SELECT DISTINCT forum_id FROM Messages;
```

Powyższe zapytanie zwróci pojedyncze wartości pola `forum_id` i wyeliminuje wszystkie duplikaty.

Wykorzystanie funkcji SQL w zapytaniach

Oprócz wartości skalarnych i kolumn, w zapytaniach języka SQL można wykorzystać szereg funkcji. Mogą to być zarówno wbudowane funkcje serwera, jak i funkcje zdefiniowane przez użytkownika w systemach obsługujących tzw. **procedury składowane** (ang. *stored procedures*) — funkcje zdefiniowane przez użytkownika i przechowywane na serwerze w postaci skompilowanej.

W języku SQL występują dwie klasy funkcji:

- **Funkcje agregacji** — funkcje, które przetwarzają zbiór wartości (na przykład wartości danych w kolumnie tabeli) i zwracają pojedynczą wartość skalarną. Przykładem mogą być funkcje obliczające sumę wartości w kolumnie lub ich średnią arytmetyczną.
- **Funkcje skalarne** — funkcje przetwarzające pojedynczą wartość skalarną (na przykład wartość określonej kolumny w określonym wierszu) i zwracające pojedynczą wartość skalarną. W przypadku podania jako parametru nazwy kolumny, funkcja zwraca kolumnę, której wartości są jej wynikami dla wartości kolumny źródłowej. Przykładami mogą być funkcje zamiany ciągów znaków w polu na małe litery, konwersji walut oraz funkcje formatujące dane przed wyświetleniem.

W wyrażeniach można stosować wyłącznie funkcje skalarne. Funkcje agregacji można wykorzystać do przetwarzania wyników zapytania przed ich zwróceniem do użytkownika. Najczęściej stosuje się je w odniesieniu do wyników instrukcji `SELECT`.

Niestety, funkcje stanowią kolejny obszar, w którym poszczególne serwery baz danych bardzo się różnią pomiędzy sobą. Choć zestaw własności funkcjonalnych jest podobny, nazwy funkcji i sposób ich używania bardzo się różnią pomiędzy poszczególnymi serwerami. W tym rozdziale postaram się opisać najpopularniejsze funkcje dla najpopularniejszych serwerów. W przypadku, gdy rozbieżności będą zbyt wielkie, przedstawię wersję dla bazy danych MySQL (zwięzły opis odpowiedników dla innych serwerów baz danych można znaleźć w dodatku B „Odpowiedniki funkcji obsługi baz danych”).

W przypadku wywoływania funkcji w zapytaniach SQL, pomiędzy nazwą funkcji a otwierającym znakiem nawiasu przed listą argumentów nie może być spacji.

```
AVG(daily_precip)      Dobrze
AVG (daily_precip)    Źle
```

Funkcje numeryczne

Rozpocznię od opisanie kilku funkcji numerycznych, ponieważ są one najbardziej zrozumiałe i najłatwiejsze w użyciu. Wszystkie opisane poniżej funkcje to funkcje agregacji, które wykorzystamy do przetwarzania zbioru wierszy w zapytaniu w celu uzyskania pojedynczej wartości.

COUNT

Funkcję COUNT można wykorzystać na dwa sposoby: `COUNT(NazwaKolumny)` lub `COUNT(*)`. Pierwszy sposób wywołania zlicza w zestawie wyników liczbę wartości określonej kolumny różnych od NULL. W drugiej wersji funkcja COUNT zlicza liczbę wierszy w zestawie wyników. Na przykład instrukcja:

```
SELECT COUNT(*) FROM Users;
```

zlicza użytkowników w tabeli Users. Aby policzyć użytkowników w tabeli Users, których nazwiska są różne od NULL (co przy naszym schemacie tabeli jest dozwolone), można wykorzystać następujące zapytanie:

```
mysql> SELECT COUNT(full_name) from Users;
+-----+
| count(full_name) |
+-----+
|                4 |
+-----+
1 row in set (1.26 sec)
```

SUM

Aby w zestawie wyników uzyskać sumę wartości zawartych w określonej kolumnie, można skorzystać z funkcji SUM. Gdybyśmy mieli tabelę z informacjami o pogodzie w określonym miejscu i chcieli obliczyć całkowitą wartość opadów w 2002 roku, moglibyśmy wykonać następujące zapytanie:

```
SELECT SUM(daily_precip) FROM DailyWeatherReports
WHERE date BETWEEN '2002-01-01' AND '2002-12-31';
```

MAX i MIN

Aby obliczyć maksymalną lub minimalną wartość kolumny w zestawie wyników, można skorzystać z funkcji MAX i MIN. Kontynuując nasz przykład bazy danych o pogodzie z poprzedniego punktu, aby znaleźć dni w 2002 roku, w których wystąpiły największe i najmniejsze opady, możemy skorzystać z następujących zapytań:

```
SELECT MAX(daily_precip) FROM DailyWeatherReports
WHERE date BETWEEN '2002-01-01' AND '2002-12-31';
```

```
SELECT MIN(daily_precip) FROM DailyWeatherReports
WHERE date BETWEEN '2002-01-01' AND '2002-12-31';
```

Powyższe dwa zapytania można połączyć w jedno. Zestaw wyników takiego zapytania będzie składał się z dwóch kolumn:

```
SELECT MAX(daily_precip), MIN(daily_precip)
FROM DailyWeatherReports
WHERE date BETWEEN '2002-01-01' AND '2002-12-31';
```

AVG

Aby obliczyć średnią arytmetyczną wartości w kolumnie, można skorzystać z funkcji AVG. Wykorzystuje się ją w następujący sposób:

```
SELECT AVG(daily_precip) FROM DailyWeatherReports
WHERE date BETWEEN '2002-01-01' AND '2002-12-31';
```

Funkcje znakowe

W języku SQL występuje wiele przydatnych funkcji przetwarzania ciągów znaków. Większość z nich to funkcje skalarne, które można wykorzystać w wielu przypadkach.

Wydzielanie podciągów

Aby wydzielić część ciągu znaków w języku SQL, można skorzystać z funkcji SUBSTRING (w systemie Oracle — SUBSTR). Funkcja pobiera trzy argumenty: wartość (nazwę kolumny), z której ma być wydzielony podciąg, indeks pierwszego znaku podciągu oraz liczbę znaków do wydzielenia.

Uwaga

W odróżnieniu od języka PHP, gdzie indeksy mają wartości począwszy od 0, indeksy w ciągach znaków w języku SQL liczy się od 1. Oznacza to, że pierwszy znak w ciągu ma indeks 1.

Aby pobrać pierwszych 5 znaków z każdego wiersza tabeli zawierającej nazwy wszystkich stanów w USA i prowincji w Kanadzie, można wykonać następujące zapytanie:

```
SELECT SUBSTRING(name, 1, 5) FROM states_provinces;
```

Aby znaleźć stany i prowincje, których nazwy zaczynają się od sekwencji New, można wykonać następujące zapytanie:

```
SELECT * FROM states_provinces
WHERE SUBSTRING(name, 1, 3) = 'New';
```

Konwersja wielkości liter

Aby przekształcić ciągi znaków na małe bądź wielkie litery, można skorzystać z funkcji LOWER lub UPPER.

```
SELECT LOWER(user_name) FROM Users;  
SELECT UPPER(last_name) FROM customers;
```

Funkcji LOWER i UPPER rzadko używa się w wyrażeniach formułowanych w klauzuli WHERE, ponieważ — jak wspominałem w poprzednim rozdziale — w przypadku większości operacji sortowania i porównań wielkość liter nie ma znaczenia.

Wyszukiwanie ciągu znaków

Często przydaje się funkcja, która wyszukuje pozycję określonego ciągu znaków w tekście (lub kolumnie). Chociaż we wszystkich serwerach baz danych, które będziemy wykorzystywać, taka funkcja jest dostępna, nazwy i sposoby korzystania z niej mogą być różne. Zarówno w systemie MySQL, jak i Oracle jest dostępna funkcja INSTR:

```
INSTR(szukaj_tu, znajdź_mnie)
```

Na przykład, poniższe zapytanie można wykorzystać do pobrania imion użytkowników naszej internetowej tablicy ogłoszeń (przy założeniu, że w polu name imię występuje jako pierwsze):

```
SELECT user_name  
       SUBSTRING(full_name, 1, INSTR(full_name, ' '))  
FROM Users;
```

Jeśli szukany ciąg nie zostanie odnaleziony, funkcja INSTR zwraca wartość 0.

Konkatenacja ciągów znaków

W języku SQL istnieje funkcja służąca do konkatenacji ciągów znaków. W różnych serwerach baz danych funkcja ta może nazywać się inaczej. W tym miejscu opiszę składnię polecenia w systemie MySQL:

```
CONCAT(wartość1, wartość2, ..., wartośćn)
```

Aby na podstawie zawartości tabeli Users uzyskać sformatowany ciąg znaków składający się z nazwy użytkownika i adresu e-mail, można wykonać następujące zapytanie:

```
SELECT CONCAT('Nazwa użytkownika: ',  
             user_name,  
             '\tAdres e-mail:',  
             user_email)  
FROM Users;
```

Przycinanie ciągów znaków

Funkcja TRIM usuwa wiodące i końcowe spacje z ciągów znaków. Działa tak samo jak odpowiadająca jej funkcja w PHP:

```
SELECT user_name, TRIM(full_name), user_email
FROM Users
WHERE user_name LIKE 'F%';
```

Funkcje przetwarzania dat i godzin

W języku SQL występuje kilka przydatnych funkcji przetwarzających daty i godziny. Nie są one standardowe i w różnych serwerach występują pod innymi nazwami, ale większość podstawowych funkcji jest dostępna we wszystkich najpopularniejszych systemach baz danych.

Now

Aby odczytać bieżącą datę i godzinę w systemie MySQL i PostgreSQL, można skorzystać z funkcji NOW.

```
SELECT NOW();

INSERT INTO Orders (prodid, user_id, when)
VALUES(445455423, 32345, Now());
```

Rok, miesiąc, dzień

W języku SQL występuje grupa funkcji służących do pobierania części wartości z dat. Funkcje te znacznie się różnią w różnych systemach baz danych. W bazie danych MySQL funkcja YEAR pobiera datę i na jej podstawie zwraca czterocyfrowy numer roku. Funkcja MONTH zwraca numer miesiąca daty, natomiast funkcje DAYOFMONTH, DAYOFWEEK i DAYNAME zwracają informacje na temat dnia w różnym formacie.

Formatowanie dat i godzin

Chociaż we wszystkich systemach baz danych występują funkcje służące do formatowania dat i godzin, poszczególne implementacje bardzo się różnią pomiędzy sobą. W systemie MySQL do formatowania dat służy funkcja DATE_FORMAT, która pobiera wartość daty do sformatowania oraz ciąg formatu.

```
DATE_FORMAT(sformatuj_mnie, ciąg_formatu)
```

Parametr *ciąg_formatu* to sekwencja znaków, które są zastępowane odpowiednimi wartościami dotyczącymi daty. Na przykład:

```
mysql> SELECT full_name, DATE_FORMAT(birthdate, '%W %D %M %Y')
-> FROM Users
-> WHERE birthdate <> '000-00-00';
```

full_name	DATE_FORMAT(birthdate, '%W %D %M %Y')
Krzysztof Malinowski	Tuesday 16th December 1980
Akira Tanaka	Wednesday 13th September 0000
Patrycja Dominikowska	Monday 31st March 1975

```
3 rows in set (0.00 sec)
```

Najpopularniejsze i najbardziej interesujące kody formatów zestawiono w tabeli 11.1. Pełną listę wartości można znaleźć w dokumentacji bazy danych MySQL.

Tabela 11.1. Kody formatów funkcji `DATE_FORMAT`

Kod	Znaczenie
%W	Dzień tygodnia w języku serwera bazy danych (często jest nim język angielski).
%w	Dzień tygodnia w formacie liczbowym (0= niedziela, 6= sobota).
%Y	Rok według kalendarza juliańskiego przedstawiony za pomocą 4 cyfr (na przykład 1999).
%y	Rok według kalendarza juliańskiego przedstawiony za pomocą 2 cyfr (na przykład 33).
%M	Nazwa miesiąca w języku serwera bazy danych (często jest nim język angielski).
%m	Numer miesiąca w formacie liczbowym (01 – 12).
%D	Dzień miesiąca z przyrostkiem liczebnika porządkowego dla języka angielskiego (1st, 2nd, 3rd itd.).
%d	Dzień miesiąca w formacie liczbowym (01 – 31)
%H	Godzina w formacie 24-godzinnym (00 – 23).
%h	Godzina w formacie 12-godzinnym (01 – 12).
%p	Przyrostek AM bądź PM.
%i	Minuta w formacie liczbowym (00 – 59).
%S lub %s	Sekunda w formacie liczbowym (00 – 59).

Grupowanie danych dla funkcji agregacji

Funkcje agregacji w języku SQL są jeszcze bardziej przydatne, jeśli przed wykonaniem funkcji pogrupuje się wartości danych w tabeli. W zaprezentowanym poprzednio przykładzie bazy danych z informacjami o pogodzie pokazałem, w jaki sposób można wykorzystać funkcję `AVG` do obliczenia średniej wartości opadów w roku. Aby uzyskać średnią wartość opadów dla kilku lat, trzeba by było wykonać kilka zapytań i podać w nich różne wartości lat.

Byłoby o wiele wygodniej, gdyby można to było zrobić za pomocą języka SQL. Klauzula `GROUP BY` umożliwia grupowanie wartości w określonej kolumnie przed wykonaniem funkcji agregacji dla każdej z grup:

```
SELECT YEAR(date). AVG(daily_precip)
FROM DailyWeatherReports
GROUP BY YEAR(date);
```

Klauzula `GROUP BY` razem ze skalarną funkcją `YEAR` umożliwia pogrupowanie tabeli według wspólnych wartości roku, a następnie wykonanie funkcji `AVG` dla wszystkich wierszy z poszczególnych grup. Zapytanie można jeszcze bardziej uściślić za pomocą klauzuli `HAVING`, która jest nieco podobna do klauzuli `WHERE`, ale dotyczy warunków ograniczających dla podklauzuli `GROUP BY`. Przykładowo, aby uzyskać listę lat, w których średnia wartość opadów przekroczyła 50 mm, można wykorzystać następujące zapytanie:

```
SELECT YEAR(date). AVG(daily_precip)
FROM DailyWeatherReports
GROUP BY YEAR(date)
HAVING AVG(daily_precip) > 50;
```

Dzięki możliwości łączenia powyższych własności można tworzyć niezwykle skomplikowane zapytania. Na przykład, aby uzyskać średnią wartość opadów dla lat 1990 – 2000 dla wszystkich miast o rocznej średniej wartości opadów przekraczającej 50 mm, można wykonać następujące zapytanie:

```
SELECT YEAR(date). AVG(daily_precip)
FROM DailyWeatherReports
WHERE YEAR(date) BETWEEN 1990 AND 2000
GROUP BY YEAR(date)
HAVING AVG(daily_precip) > 50;
```

Modyfikowanie schematu tabel

W pewnych sytuacjach trzeba dodać bądź usunąć kolumnę z tabeli. Można to zrobić za pomocą instrukcji `ALTER TABLE`, która ma wiele możliwych zastosowań. W tym podrozdziale zdemonstrujemy najczęściej stosowane:

```
ALTER TABLE NazwaTabeli
ADD NazwaKolumny TypKolumny atrybuty...;
```

```
ALTER TABLE NazwaTabeli
DROP COLUMN NazwaKolumny;
```

```
ALTER TABLE NazwaTabeli
CHANGE COLUMN NazwaKolumny Nowe_szczegóły;
```

```
ALTER TABLE NazwaTabeli
RENAME AS NowaNazwaTabeli;
```

Modyfikowanie schematu tabeli jest operacją, której nie należy wykonywać często. Trzeba mieć pewność, że wykonywane działania są prawidłowe. Celem projektowania jest utworzenie struktury bazy danych, która jest wydajna, elastyczna, skalowalna i zapewni spełnienie potrzeb użytkowników w przyszłości. Jeśli zbyt często musimy modyfikować schematy tabel, może to oznaczać konieczność powtórnego przeprowadzenia procesu projektowania tabel. Poza tym, modyfikowanie schematu tabel jest operacją kosztowną. W niektórych systemach baz danych usunięcie kolumny wymaga dużej ilości miejsca na dysku. Często tabela jest zablokowana przez cały czas usuwania niepotrzebnych danych. Podobne, mniej lub bardziej przejściowe problemy z wydajnością, mogą się zdarzyć w przypadku dodawania nowych kolumn.

Aby dodać kolumnę, należy skorzystać z klauzuli `ADD` i wprowadzić nazwę nowej kolumny, jej typ danych oraz inne atrybuty. Na przykład, aby w tabeli `Users` dodać pole `password` służące do zapisywania haseł, można wykorzystać następujące zapytanie:

```
ALTER TABLE Users
ADD password VARCHAR(50) NOT NULL;
```

Powyższe zapytanie dodaje nową kolumnę tekstową do tabeli Users z atrybutem NOT NULL wykluczającym użycie NULL jako wartości w dowolnym z wierszy. Dla istniejących wierszy w tabeli kolumna password zostanie ustawiona na pusty ciąg znaków ''. W języku SQL występują dwa opcjonalne słowa kluczowe — FIRST i AFTER, pozwalające na określenie miejsca w tabeli, w którym mają zostać umieszczone nowe kolumny:

```
ALTER TABLE Users
  ADD password VARCHAR(50) NOT NULL
  AFTER user_name;
```

Aby usunąć kolumnę z tabeli, należy skorzystać z klauzuli DROP COLUMN. Na przykład, aby usunąć kolumnę password, którą dodaliśmy przed chwilą, można wykorzystać następujące zapytanie:

```
ALTER TABLE Users
  DROP COLUMN password;
```

Podobnie jak inne operacje, które powodują usuwanie informacji z bazy danych, usunięcie kolumny jest trwałe i nie można go cofnąć. Dlatego właśnie takie polecenia muszą być używane ze szczególną ostrożnością (i raczej nie należy zezwalać na ich wykonywanie zwykłym użytkownikom bazy danych).

Do zmiany definicji kolumny służy klauzula CHANGE (klauzulą równoważną jest MODIFY). W przypadku wprowadzenia nazwy w nowej definicji kolumny, kolumna zostanie przemianowana. Klauzulę CHANGE można również wykorzystać do zmiany typu oraz atrybutów kolumny. Przykładowo, aby zmienić typ pola user_name w tabeli Users na ciąg 100-znakowy, można wykorzystać następujące zapytanie:

```
ALTER TABLE Users
  CHANGE COLUMN user_name VARCHAR(100) NOT NULL;
```

Aby zmienić nazwę tabeli User, można skorzystać z klauzuli RENAME:

```
ALTER TABLE Users
  RENAME AS MessageBoardUsers;
```

Instrukcję ALTER TABLE mogą wykonywać tylko użytkownicy posiadający uprawnienie ALTER.

Podsumowanie

Po lekturze tego rozdziału Czytelnik rozszerzył swoją wiedzę na temat języka SQL. Wykorzystaliśmy go do wykonywania coraz bardziej skomplikowanych zapytań i operacji na danych. Dzięki zastosowaniu transakcji można wykonać wiele instrukcji SQL w formie niepodzielnej operacji. Podczas obsługi transakcji można stosować różne poziomy izolacji i blokowania. Po lekturze tego rozdziału Czytelnik powinien wiedzieć, w jaki sposób można kwalifikować i sortować dane, wykorzystywać funkcje, a także modyfikować schemat tabel (pamiętając o kosztach tej operacji).

Po lekturze ostatnich czterech rozdziałów dysponujemy wystarczającą wiedzą na temat baz danych, aby zacząć z nich korzystać z poziomu kodu PHP. W następnym rozdziale przedstawimy sposoby nawiązywania połączenia z serwerem, wykonywania zapytań i przeglądania wyników z poziomu naszej aplikacji internetowej.