helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

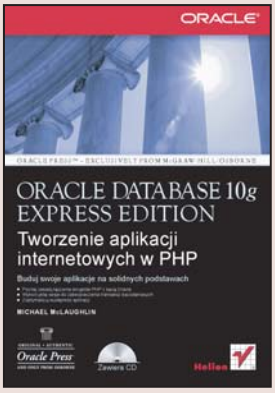
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

Oracle Database 10g Express Edition. Tworzenie aplikacji internetowych w PHP



Autor: Michael McLaughlin
Tłumaczenie: Rafał Jońca
ISBN: 978-83-246-1204-8
Tytuł oryginału: Oracle Database 10g Express Edition PHP Web Programming
Format: B5, stron: 656
Zawiera CD-ROM

Buduj swoje aplikacje na solidnych podstawach

- Poznaj zasady łączenia skryptów PHP z bazą Oracle
- Wykorzystaj sesje do zabezpieczania transakcji bazodanowych
- Zoptymalizuj wydajność aplikacji

Baza danych Oracle od dawna słynie z wyjątkowej stabilności, wydajności i... ceny! Jednak po wydaniu jej najnowszej wersji, oznaczonej symbolem 10g, producent zdecydował się na zaskakujący krok, udostępnił jedną z nich – Express Edition – bezpłatnie. Otworzyło to przed twórcami aplikacji sieciowych możliwości wykorzystania tego doskonałego produktu w projektach niskobudżetowych. Wszędzie tam, gdzie klienci oczekują zaplecza bazodanowego zbudowanego w oparciu o sprawdzony system i nieprzychylnie reagują na rozwiązania open-source, można dziś zaproponować Oracle Database 10g Express Edition. Aby jednak w aplikacjach internetowych w pełni wykorzystać możliwości tego rozwiązania, trzeba poznać specyficzne dla niego techniki programistyczne.

Książka „Oracle Database 10g Express Edition. Tworzenie aplikacji internetowych w PHP" to podręcznik, w którym znajdziesz omówienie wszystkich aspektów korzystania z bazy Oracle 10g w połączeniu ze skryptami PHP. Dowiesz się, jak zainstalować i skonfigurować środowisko robocze i poznasz najważniejsze elementy języka PHP. Nauczysz się tworzyć bezpieczne i wydajne aplikacje internetowe wykorzystujące ogromne możliwości bazy Oracle 10g Express Edition. Przeczytasz o uwierzytelnianiu użytkowników, zabezpieczaniu transakcji, stosowaniu procedur składowanych i obsłudze dużych obiektów. Znajdziesz tu również informacje o administrowaniu bazą Oracle, formułowaniu zapytań SQL oraz optymalizowaniu aplikacji pod kątem wydajności i szybkości działania.

- Instalacja i konfiguracja Oracle Express, Apache i PHP
- Typy danych w PHP
- Instrukcje i polecenia języka PHP
- Programowanie obiektowe
- Obsługa błędów
- Praca z systemem plików
- Korzystanie z plików cookie i mechanizmów sesji
- Połączenie skryptu z bazą danych Oracle
- Operacje na danych w bazie
- Korzystanie z procedur składowanych
- Obsługa obiektów typu BLOB

Wykorzystaj pełnię możliwości Oracle 10g Express Edition, budując wydajne i bezpieczne aplikacje internetowe

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

O autorze **11**

Wprowadzenie **13**

Część I Podstawy języka PHP **17**

Rozdział 1. Omówienie PHP i programowania stron WWW z użyciem baz danych Oracle **19**

Historia i tło 20
Czym jest PHP? 20
Czym jest Oracle? 21
Czym jest Zend? 21
Tworzenie rozwiązań dla aplikacji internetowych 22
Co i gdzie się umieszcza i dlaczego właśnie tak? 22
Co oferuje Oracle dla języka PHP? 24
Dlaczego PHP 5 jest istotny? 24
Podsumowanie 24

Rozdział 2. Instalacja i konfiguracja Oracle Express, Apache i PHP **25**

Instalacja na platformie Linux 26
Apache 26
Oracle Database Express Edition 27
PHP 31
Instalacja na platformie Windows XP 37
Apache 38
Oracle Database Express Edition 42
PHP 46
Podsumowanie 58

Rozdział 3. Tworzenie stron WWW **59**

Określanie fragmentów z kodem PHP 59
Wysyłanie danych jako części strony WWW 62
Umieszczanie komentarzy w skryptach PHP 67
Jednowierszowe komentarze o składni znanej z języków C++, C#, Java lub JavaScript 67
Wielowierszowe komentarze o składni znanej z języków C++, C#, Java lub JavaScript 68
Jednowierszowe komentarze znane ze skryptów powłoki systemu Unix 68
Podsumowanie 68

Rozdział 4. Zmienne, operatory, typy danych i dołączanie plików 69
Zmienne 69
Zmienne definiowane przez użytkownika 70
Operatory 73
Typy danych 80
Zmienne globalne 85
Zmienne predefiniowane 86
Dołączanie plików bibliotecznych 87
Słowa kluczowe i stałe systemowe 89
Podsumowanie 91

Rozdział 5. Struktury sterujące 93
Instrukcje warunkowe 93
Instrukcja if 94
Instrukcja switch 97
Pętle 100
Pętla do-while 100
Pętla for 101
Pętla foreach 103
Pętla while 105
Podsumowanie 107

Część II Programowanie w języku PHP 109

Rozdział 6. Tablice 111
Definiowanie tablic 113
Zarządzanie tablicami 121
Funkcje identyfikujące i zliczające 121
Funkcje tworzące tablice o zadanych wymiarach 123
Funkcje zarządzające dostępem w stylu kolejkowym 124
Funkcje wyszukujące 126
Funkcje przejścia 131
Sortowanie tablic 133
Funkcje sort() i rsort() 134
Funkcje asort() i arsort() 136
Funkcje ksort() i krsort() 136
Funkcje natsort() i natcasesort() 137
Funkcja array_multisort() 139
Funkcje usort() i uksort() 140
Łączenie i dzielenie tablic 141
Składnie tablic 146
Łączenie tablic 147
Wydobywanie fragmentów tablic 148
Zastępowanie fragmentu tablicy 149
Znajdowanie części wspólnej tablic 151
Poznawanie różnic między tablicami 152
Podsumowanie 152

Rozdział 7. Funkcje 153
Definiowanie funkcji 154
Zasięg zmiennych w funkcjach 157
Parametry funkcji 160
Przekazywanie parametrów przez wartość lub referencję 160
Domyślne wartości parametrów 163
Listy parametrów o zmiennej długości 164

Funkcje zwracające wartość ... 165
Zarządzanie dynamicznymi wywołaniami funkcji ... 168
Funkcje rekurencyjne ... 171
Podsumowanie ... 174

Rozdział 8. Obiekty ... 175
Definiowanie klas i używanie obiektów ... 177
Definiowanie klasy i tworzenie jej egzemplarza ... 179
Definiowanie składowych, metod i stałych ... 182
Metody ustawiające i pobierające ... 187
Definiowanie klas używających dziedziczenia i polimorfizmu ... 188
Tworzenie podklas i przesłanianie operacji ... 190
Wykorzystanie interfejsów i klas abstrakcyjnych ... 193
Implementacja obiektów ... 196
Klonowanie ... 198
Porównywanie, wyświetlanie zawartości i podpowiadanie typów ... 201
Introspekcja klas i obiektów ... 205
Podsumowanie ... 210

Rozdział 9. Zarządzanie błędami i obsługa wyjątków ... 211
Zarządzanie błędami ... 213
Konfiguracja zrządzania błędami ... 213
Obsługa błędów ... 215
Obsługa wyjątków ... 219
Blok try-catch ... 220
Klasa Exception ... 222
Obsługa wyjątków w praktyce ... 226
Podsumowanie ... 229

Rozdział 10. Obsługa plików ... 231
Pliki w systemie plików ... 232
Funkcje informujące o parametrach pliku ... 235
Funkcje lokalizujące pliki ... 242
Funkcje zarządzania plikami ... 244
Funkcje stanu pliku ... 252
Funkcje obsługi systemu plików ... 254
Odczyt i zapis plików ... 256
Odczyt plików ... 262
Zapis plików ... 268
Uzyskiwanie plików od użytkowników ... 272
Podsumowanie ... 275

Część III Tworzenie aplikacji w języku PHP ... 277

Rozdział 11. Proste uwierzytelnianie PHP i formularze ... 279
Opis zdalnego wykonywania procedur ... 280
Proste uwierzytelnianie HTTP ... 281
Tworzenie formularzy uwierzytelniania ... 286
Podsumowanie ... 300

Rozdział 12. Cookies i sesje ... 301
Definiowanie i wykorzystywanie danych cookies ... 303
Definiowanie i wykorzystywanie danych sesji ... 307
Dane cookies i sesje w praktyce ... 313
Podsumowanie ... 324

Część IV Aplikacje internetowe wykorzystujące bazę danych Oracle Express 325

Rozdział 13. Zapytania i transakcje w Oracle SQL 327
Połączenie z bazą Oracle za pomocą biblioteki OCI8 330
Wykorzystanie biblioteki funkcji OCI8 340
Podsumowanie funkcji z biblioteki OCI8 340
Wprowadzenie do modeli bazodanowych 347
Przykład tworzenia zapytania 348
Przykład uruchamiania transakcji 349
Przykład zmiany hasła użytkownika 353
Zapytania i transakcje wykorzystujące instrukcje SQL 354
Podsumowanie 363

Rozdział 14. Transakcje Oracle PL/SQL 365
Wyjaśnienie sposobu działania procedur zapamiętanych PL/SQL 366
Wykorzystanie kursorów referencyjnych i procedur zapamiętanych PL/SQL 367
Użycie tablic asocjacyjnych z wartościami skalarnymi i procedur zapamiętanych PL/SQL 372
Użycie kolekcji skalarnych i procedur zapamiętanych PL/SQL 377
Użycie typu danych NESTED TABLE 381
Użycie typu danych VARRAY 385
Podsumowanie 387

Rozdział 15. Transakcje Oracle dotyczące dużych obiektów 389
Definicje funkcji dużych obiektów i klasa OCI-Lob 390
Wykorzystanie typów danych BLOB, CLOB i NCLOB 395
Jak odczytać kolumny CLOB? 396
Aktualizacja kolumn CLOB 404
Wykorzystanie typów danych CFILE i BFILE 410
Konfiguracja środowiska dla plików BFILE 411
Pobieranie kolumny BFILE za pomocą polecenia SQL 413
Pobieranie kolumny BFILE za pomocą polecenia PL/SQL 416
Przesył plików BFILE na serwer 418
Podsumowanie 422

Dodatki 423

Dodatek A Znaczniki HTML 425

Dodatek B Teksty, narzędzia i techniki 431

Dodatek C Stałe środowiska PHP 439

Dodatek D Interfejsy środowiska i typy obiektów 457

Dodatek E Funkcje obsługi plików w standardzie POSIX 463

Dodatek F Funkcje dotyczące daty i czasu 473

Dodatek G Wprowadzenie do baz danych Oracle 491
Architektura baz danych Oracle 491
Uruchamianie i wyłączanie bazy danych Oracle 496
Operacje w systemie Linux 497
Operacje w systemie Windows 500
Uruchamianie i wyłączanie procesu nasłuchującego 503

Wykorzystywanie programu SQL*Plus ... 508
Interfejs wiersza poleceń ... 509
Interfejs WWW ... 514
Podsumowanie ... 517

Dodatek H Wprowadzenie do SQL ... 519
Typy danych kolumn tabel ... 520
Język definicji danych (DDL) ... 523
Zarządzanie tabelami i ograniczeniami ... 524
Zarządzanie widokami ... 527
Zarządzanie procedurami zapamiętanymi ... 529
Zarządzanie sekwencjami ... 529
Zarządzanie własnymi typami ... 532
Język pobierania danych (DQL) ... 533
Zapytania ... 534
Język modyfikacji danych (DML) ... 536
Polecenia INSERT ... 536
Polecenia UPDATE ... 538
Polecenia DELETE ... 539
Język sterowania danymi (DCL) ... 540
Podsumowanie ... 540

Dodatek I Wprowadzenie do PL/SQL ... 541
Struktura bloków PL/SQL ... 542
Zmienne, przypisania i operatory ... 545
Struktury sterujące ... 549
Instrukcje warunkowe ... 550
Pętle ... 552
Procedury zapamiętane, funkcje zapamiętane oraz pakiety ... 555
Funkcje zapamiętane ... 556
Procedury ... 559
Pakiety ... 561
Wyzwalacze bazodanowe ... 564
Wyzwalacze DDL ... 565
Wyzwalacze DML ... 565
Wyzwalacze „zamiast” ... 567
Wyzwalacze systemowe lub bazodanowe ... 567
Kolekcje ... 568
Typ danych VARRAY ... 569
Typ danych NESTED TABLE ... 570
Tablica asocjacyjna ... 571
Interfejs kolekcji ... 574
Wykorzystanie pakietu DBMS_LOB ... 574
Konfiguracja i weryfikacja środowiska dla typów LOB ... 576
Zapis i odczyt typu danych CLOB ... 577
Lokalizacja i odczyt typu danych BFILE ... 582
Podsumowanie ... 584

Dodatek J Skrypty przykładowych baz danych ... 585
Utworzenie wyzwalaczy zapamiętujących czas łączenia i rozłączania ... 586
Utworzenie struktury danych dla wypożyczalni filmów ... 588
Wypełnienie bazy danych wypożyczalni filmów danymi ... 599
Utworzenie bazy danych prezydentów, zdefiniowanie kodu i wypełnienie tablicy danymi ... 614
Podsumowanie ... 625

Skorowidz ... 627

Część IV

Aplikacje internetowe wykorzystujące bazę danych Oracle Express

Rozdział 13.

Zapytania i transakcje w Oracle SQL

Niniejszy rozdział omawia połączenie się z bazą danych Oracle Database 10*g* XE w trybie zapytań lub trybie transakcyjnym i wykorzystanie obiektów schematu bazy danych: tabel, widoków i procedur zapamiętanych. Choć baza danych udostępnia standardowy język zapytań SQL, obsługuje dodatkowo wzbogacony język PL/SQL. Rozwiązania dotyczące standardowego SQL pojawią się w tym rozdziale, natomiast opis obsługi wartości skalarnych, kolekcji i kursorów referencyjnych za pomocą PL/SQL zostanie opisany w rozdziale 14.

Rozdział 2. wyjaśnia proces instalacji bazy danych Oracle Database 10*g* XE, jej konfigurację i weryfikację połączenia. Aby wykonać przykłady prezentowane w tym rozdziale, trzeba posiadać poprawnie skonfigurowane aplikacje Apache, PHP i Oracle. Wszystkie prezentowane przykłady były testowane w PHP 5.1.4, ale powinny działać we wszystkich wersjach PHP od 5.1.2.

Przed wersją 5.1.2 trzeba korzystać z innego zestawu bibliotek. Co więcej, liczba dostępnych funkcji jest uboższa. W nowszej wersji PHP starsze funkcje działają jako aliasy dla nowszych funkcji, ale warto w trakcie migracji zmienić nazwy wywoływanych funkcji na nowsze. Warto zainstalować nowszą wersję PHP, by skorzystać z nowej biblioteki OCI8.

Przed rozpoczęciem czytania tego rozdziału warto przeczytać wprowadzenia do języków SQL i PL/SQL zawarte w dodatkach H i I. Dodatkowo, podstawowy kod wykorzystywany w rozdziałach od 13. do 15. został bardziej szczegółowo opisany w dodatku J.

Rozdział został podzielony na trzy części:

- Połączenie z bazą Oracle za pomocą biblioteki OCI8
- Wykorzystanie biblioteki funkcji OCI8
- Zapytania i transakcje wykorzystujące instrukcje SQL

Podrozdziały warto czytać po kolei. Pierwszy podrozdział opisuje trzy sposoby łączenia się z bazą danych Oracle. Różnią się one zachowaniem i funkcjonalnością. Drugi podrozdział stanowi podsumowanie funkcji OCI8 i prezentuje wykonywanie podstawowych operacji SQL — `INSERT`, `SELECT`, `UPDATE` i `DELETE` — z poziomu kodu PHP. Podrozdział opisuje użycie instrukcji

INSERT i UPDATE do wstawiania i aktualizacji pojedynczych wierszy. Pojedyncze polecenie INSERT lub UPDATE może aktualizować wiele wierszy danych, jeśli wykorzystamy tzw. **dowiązania**. Sposób ich działania szczegółowo omawia trzeci z podrozdziałów. Zmienne dowiązania odwzorowują wartości skalarne i kolekcje w poleceniach SQL.

Obecnie biblioteka OCI8 obsługuje jedynie ograniczony zestaw typów danych dostępnych w bazie danych Oracle Database 10*g* XE. Zespół programistów odpowiedzialny za rozwój biblioteki OCI8 planuje dodanie obsługi odczytu i zapisu obiektów PL/SQL bezpośrednio z poziomu aplikacji PHP. Data **wprowadzenia** tej funkcji nie jest jeszcze znana. Aktualnych informacji na ten temat warto szukać w witrynie *http://otn.oracle.com*.

W razie napotkania problemów w trakcie korzystania z bazy danych lub biblioteki OCI8 można skorzystać z wielu narzędzi wspomagających poszukiwanie przyczyny ich powstawiania. Dodatek G opisuje narzędzie tnspring, które ułatwia testowanie nasłuchiwania serwera bazy danych na konkretnym porcie. Ten sam dodatek wyjaśnia, w jaki sposób testować, czy baza danych została uruchomiona, i jak wykonać proste zapytania SQL.

Istnieje pięć skryptów SQL, z których warto skorzystać w trakcie wykonywania przykładów z niniejszego rozdziału. Skrypt *create_user.sql* został opisany w dodatku G. Należy go uruchomić jako użytkownik SYSTEM, by utworzyć schemat bazy danych i użytkownika dla prezentowanych przykładów. Skrypty przedstawiane w dodatku J zakładają wcześniejsze uruchomienie wspomnianego skryptu. Jeśli chcesz skorzystać z innego użytkownika niż proponowany, należy zmodyfikować prezentowane przykłady.

W jaki sposób zainstalować Oracle Instant Client?

Biblioteki OCI8 można zainstalować na komputerze lokalnym pracującym pod kontrolą systemu Windows lub Linux. Przedstawione kroki należy wykonać tylko wtedy, gdy zamierza się testować programy PHP na innym komputerze niż ten, na którym zainstalowano serwer Apache i bazę danych.

Instalacja w systemie Linux

Z witryny *http://otn.oracle.com* pobierz dwa pliki zawarte w archiwum TAR o nazwie *oracle-client-10103.tar.bz2*. Zapisz plik w dowolnej lokalizacji, a następnie rozpakuj go, wykonując poniższe polecenie z poziomu konsoli systemowej:

```
tar -xvjf oracle-client-10103.tar.bz2
```

Po rozpakowaniu w folderze pojawią się dwa pliki o nazwach:

```
oracle-istantclient-basic-10.1.0.3-1.i386.rpm
oracle-istantclient-devel-10.1.0.3-1.i386.rpm
```

Plik basic zawiera bibliotekę OCI8, która zostanie zainstalowana w folderze */usr/lib/oracle/10.1.0.3/client/lib*. Plik devel zawiera pliki nagłówków i pliki instalacyjne, które umieszcza w folderze */usr/include/oracle/10.1.0.3/client*. Instalację można przeprowadzić ręcznie lub za pomocą dołączonego skryptu, wywoływanego poniższym poleceniem:

```
./runmefirst.sh
```

Ponowna konfiguracja PHP wymaga wywołania polecenia:

```
./configure \
--with-oci8-instantclient=/usr/lib/oracle/10.1.0.3/client/lib \
--prefix=$HOME/php --with-apxs=$HOME/apache/bin/apxs \
--enable-sigchild --with-config-file-path=$HOME/apache/conf
```

Ponowną kompilację rozpoczyna polecenie `make`. Następnie należy ustawić zmienną środowiskową `LD_LIBRARY_PATH` na folder */usr/lib/oracle/10.1.0.3/client/lib* i ponownie uruchomić serwer Apache. Do wykrycia przyczyny ewentualnych błędów użyj pliku dziennika błędów serwera Apache.

Dodatkowo należy pamiętać o ustawieniu zmiennej środowiskowej `TNS_ADMIN`, by wskazywała na lokalizację pliku *tnsnames.ora*, jeśli korzysta się tylko i wyłącznie z Oracle Instant Client. Więcej informacji na temat połączeń sieciowych w Oracle zawiera dodatek G. Więcej informacji na temat sposobu instalacji bibliotek można znaleźć w ogólnodostępnej dokumentacji *Oracle Database Client Quick Installation Guide*.

Instalacja w systemie Windows

Z witryny *http://otn.oracle.com* pobierz trzy pliki zawarte w pliku ZIP dostępnym do pobrania na stronie dotyczącej Oracle Database 10*g* Release 1. Rozpakuj pliki zawarte w archiwum. W folderze pojawią się trzy pliki:

```
oraociei10.dll
orannzsbb10.dll
oci.dll
```

Pliki można umieścić w dowolnej lokalizacji, ale najczęściej przyjmuje się folder *C:\instantclient10_1*. Po utworzeniu folderu należy jego lokalizację dodać do zmiennej środowiskowej `PATH`. Warto dodać ją jako pierwszą ze ścieżek wymienianych w zmiennej środowiskowej.

Jeśli korzystasz z pliku *tnsnames.ora*, umieść go w tym samym folderze i ustaw zmienną środowiskową `TNS_ADMIN`, by wskazywała na ten folder. Domyślne ustawienia językowe zostaną pobrane z danych systemu operacyjnego. Aby je zmienić, wystarczy zdefiniować zmienną środowiskową `NLS_LANG`. Więcej informacji na ten temat znajduje się w książkach Steve'a Borowskiego *Hands-on Oracle Database 10g Expres Edition* for Windows i podręczniku *Oracle Database Express Edition 2 Day DBA*.

Po instalacji klienta ponownie uruchom serwer Apache. Do wykrycia przyczyny ewentualnych błędów użyj pliku dziennika błędów serwera Apache.



Analizator składniowy OCI8 uniemożliwia użycie znaku wieloznaczności SQL, którym jest znak `%`. Wprowadzenie wyrażeń wieloznacznych wymaga zastąpienia frazy `LIKE` wywołaniem `REGEXP_LIKE()`. Wspomniana funkcja jest dostępna w Oracle Database 10*g*R2 i może nie być dostępna we wcześniejszych wersjach.

Niniejszy rozdział wymaga skryptów *create_signon_triggers.sql*, *create_store.sql* i *seed_store.sql*. Zostały one przedstawione w dodatku J. Oto krótki opis znaczenia poszczególnych skryptów:

- Skrypt *create_signon_triggers.sql* wspomaga wykonanie połączenia z bazą danych Oracle za pomocą bibliotek OCI8 przedstawionych w pierwszym z podrozdziałów.
- Skrypty *create_store.sql* i *seed_store.sql* tworzą i wypełniają tabele bazy danych wykorzystywane w dwóch pozostałych podrozdziałach.

Niniejszy rozdział wymaga posiadania lokalnej bazy danych Oracle 10*g*R2 lub przynajmniej zainstalowanego klienta Oracle. W architekturze wielowarstwowej jako całkowite minimum trzeba posiadać zainstalowany serwer Apache z PHP i biblioteki Oracle Client. W ten sposób system uzyskuje możliwość łączenia się ze zdalną bazą danych nasłuchującą połączeń nadchodzących z innych komputerów.

Pierwszy z podrozdziałów opisuje trzy możliwe sposoby łączenia się z bazą danych Oracle oraz prezentuje przykłady tych połączeń.

Połączenie z bazą Oracle za pomocą biblioteki OCI8

Biblioteka OCI8 udostępnia trzy rodzaje połączeń z bazą danych Oracle.

- **Połączenie standardowe** — tworzy połączenie bazodanowe dobre na czas wykonywania skryptu lub do momentu jawnego zamknięcia połączenia przez skrypt. Wszystkie wywołania do bazy danych w skrypcie korzystają z tego samego połączenia, dopóki jawnie nie otworzymy nowego połączenia, wywołując funkcję `oci_new_connect()`. Standardowe połączenie dodaje narzut łączenia się z serwerem i przekazywania danych uwierzytelniających przy każdym uruchomieniu skryptu — między żądaniami HTTP nie jest pamiętany żaden stan połączenia bazodanowego.
- **Połączenie unikatowe** — tworzy połączenie bazodanowe dobre na czas wykonywania skryptu lub do momentu jawnego zamknięcia połączenia przez skrypt. Połączenie to zapewnia, iż jeden skrypt może posiadać wiele niezależnych połączeń do bazy danych, o ile stosują one autonomiczne transakcje. Transakcje autonomiczne działają symultanicznie zamiast sekwencyjnie, bo wyniki ich działań nie zależą od innych transakcji. Połączenia unikatowe również dodają narzut łączenia się z serwerem i przekazywania danych uwierzytelniających przy każdym uruchomieniu skryptu — między żądaniami HTTP nie jest pamiętany żaden stan połączenia bazodanowego.
- **Połączenie trwałe** — tworzy połączenie bazodanowe dobre na czas wykonywania skryptu lub do momentu jawnego zamknięcia połączenia przez skrypt. Wszystkie wywołania do bazy danych w skrypcie korzystają z tego samego połączenia, dopóki jawnie nie otworzymy nowego połączenia, wywołując funkcję `oci_new_connect()`. Połączenie trwałe dodaje narzut łączenia się z serwerem i przekazywania danych uwierzytelniających jedynie przy pierwszym żądaniu (do momentu kolejnego wywołania wspomnianej funkcji) — między żądaniami HTTP serwer pamięta stan połączenia bazodanowego z wcześniejszego skryptu. Połączenie trwałe zostaje zamknięte dopiero po określonym czasie bezczynności (a nie po zakończeniu skryptu), więc trzeba bardzo uważnie dobrać dopuszczalny czas nieaktywności, by nie marnować cennych zasobów.

Plik *php.ini* zawiera kilka opcji konfiguracyjnych dla biblioteki OCI8. Określają one sposób działania połączeń w środowisku PHP. Środowisko zastosuje wartości domyślne dla tych ustawień, jeśli nie określimy ich jawnie. Z tego powodu warto upewnić się, czy ustawienia domyślne są odpowiednie. Tabela 13.1 przedstawia dostępne dyrektywy konfiguracyjne.

Bardzo szczegółowa analiza działania systemu powinna poprzedzać jakiekolwiek zmiany w domyślnych wartościach dyrektyw OCI8 w pliku *php.ini*. Połączenia trwałe po czasie bezczynności może zamykać Apache lub Oracle. Jeśli chcemy odpowiedzialność za to zrzucić na Apache, warto przyjrzeć się opcjom `MaxRequestPerChild`, `MaxSpareServers` i `KeepAlive`. W bazie danych Oracle przerywanie bezczynnych połączeń odbywa się po ustawieniu opcji `IDLE_TIMEOUT` w ustawieniach profilu użytkownika.

Niezależnie od wyboru sposobu realizacji połączeń należy przeprowadzić wiele testów wydajnościowych przed wprowadzeniem danego rozwiązania do systemu produkcyjnego. Dzięki testom można uniknąć nieprzyjemnych efektów w trakcie obsługi istotnych klientów.

Tabela 13.1. *Dyrektywy OCI8 w pliku php.ini*

Nazwa	Wartość domyślna	Opis
oci8.privileged_connect	0	Opcja umożliwia wykonywanie uprzywilejowanych połączeń dla ról SYSOPER i SYSDBA. Domyślnie połączenia te są wyłączone. By je włączyć, ustaw dyrektywę na wartość 1.
oci8.max_persistent	-1	Opcja określa maksymalną liczbę trwałych połączeń. Domyślnie nie jest nakładane żądne ograniczenie. Ustawienie wartości dodatniej nakłada ograniczenie na liczbę trwałych połączeń. Wartość 0 wyłącza trwałe połączenia.
oci8.persistent_timeout	-1	Określa maksymalny czas bezczynności dla trwałych połączeń. Domyślnie połączenia trwałe mogą pozostawać otwarte dowolnie długo.
oci8.ping_interval	60	Opcja określa czas sprawdzania poprawności trwałych połączeń. Wyłączenie opcji zwiększa szybkość działania trwałych połączeń, ale nie wykrywa błędów w komunikacji, które mogą zdarzyć się na dalszym etapie realizacji skryptu. Ustawienie wartości 0 wyłącza sprawdzanie trwałych połączeń.
oci8.statement_cache_size	20	Określa liczbę buforowanych poleceń SQL i jest równoznaczne zapamiętywaniu poleceń w SGA. By wyłączyć buforowanie, ustaw dyrektywę na wartość 0.
oci8.default_prefetch	10	Ustawia domyślną liczbę wierszy, która zostaje pobrana zaraz po wykonaniu polecenia SQL. Zwiększenie wartości poprawia czas reakcji dla skryptów przetwarzających wiele wierszy danych. Warto pozostawić tę dyrektywę na domyślnej wartości, a gdy to konieczne, używać funkcji oci_set_prefetch() w skryptach wymagających przetwarzania dużej ilości danych.
oci8.old_oci_close_semantics	0	Włącza zgodność wstecz i wyłącza wykonywanie jakichkolwiek działań przez funkcję oci_close(). Oracle **zaleca** usunięcie wywołań tej funkcji przed włączeniem tej dyrektywy.



> Warto ustawić odpowiedni bufor dla sekwencji SYS.AUDSES$, jeśli do bazy danych w ciągu sekundy będzie wykonywanych kilkaset połączeń. Można początkowo ustawić wartość na 10 000, a następnie monitorować, czy jest wystarczająca. Sekwencja SYS.AUDSES$ jest wykonywana jako część zestawiania połączenia.

Istnieją tylko cztery funkcje OCI8 związane z otwieraniem i zamykaniem połączeń. Istnieje również funkcja oci_error(), która zwraca informację o rodzaju błędu, jeśli połączenie z bazą danych nie powiodło się. Wszystkich pięć funkcji zostało wymienionych w tabeli 13.2. Nazwy funkcji pisane kursywą zostały wycofane z użycia i pochodzą ze starszych wersji OCI. Obecnie **stanowią aliasy** dla nowszych nazw funkcji.

Tabela 13.2. *Funkcje łączenia i rozłączania z bazą danych Oracle*

Funkcja	Opis
`oci_close()` *`ocilogoff()`*	Funkcja jawnie zamyka otwarte połączenie bazodanowe w trakcie działania skryptu. To nowe zachowanie biblioteki OCI8 począwszy od PHP 5.1.2. Funkcja przyjmuje jeden parametr, którym jest zasób połączenia. Zwraca wartość logiczną `true` po udanym rozłączeniu lub wartość `false` w sytuacji przeciwnej. **Nie trzeba jawnie wywoływać funkcji, ponieważ jest ona wykonywana niejawnie po zakończeniu skryptu**. Składnia funkcji: `bool oci_close(resource` *`połączenie`*`)`
`oci_error()` *`ocierror()`*	Funkcja zwraca tablicę wartości. Przyjmuje jeden opcjonalny parametr, którym jest zasób połączenia. Bez podania parametru zwraca błąd dotyczący ostatnio otwartego połączenia. Zwrócona tablica zawiera elementy `code`, `message`, `offset` i `sqltext`. Element `code` określa kod błędu z bazy danych Oracle, element `message` w krótki sposób opisuje rodzaj błędu, element `offset` zawiera numer wiersza z błędem, a element `sqltext` zawiera fragment polecenia SQL z błędem. Składnia funkcji: `array oci_error([resource` *`połączenie`*`])`
`oci_connect()` *`ocilogin()`*	Funkcja zwraca zasób połączenia po poprawnym połączeniu z bazą danych lub wartość `false`, jeśli połączenie nie powiodło się. Przyjmuje pięć parametrów, z których pierwsze dwa są wymagane. W rzeczywistości najczęściej w trakcie połączenia podaje się trzy parametry. Trzeci parametr dotyczący nazwy bazy danych w wielu sytuacjach jest odwzorowywany automatycznie przez plik *tnsnames.ora*, ale można również wskazać bazę danych ręcznie, podając ją w kodzie PHP. Czwarty parametr (kodowanie znaków) jest pobierany z systemu operacyjnego, jeśli nie przekaże się go jawnie. Piąty parametr określa rodzaj połączenia i przyjmuje domyślną wartość `OCI_DEFAULT`, co powoduje niemożność połączenia się jako użytkownik uprzywilejowany. Użycie innego trybu (*`OCI_SYSOPER`* lub *`OCI_SYSDBA`*) wymaga dodatkowo zmiany dyrektywy `oci.privileged_connect` na wartość 1. Składnia funkcji: `resource oci_connect(string` *`nazwa_użytkownika`*`, string` *`hasło`* `[, string` *`nazwa_bazy`* `[, string` *`kodowanie_znaków`* `[, string` *`tryb_sesji`*`]]])`
`oci_new_connect()` *`ocinlogin()`*	Funkcja zwraca zasób połączenia po poprawnym połączeniu z bazą danych lub wartość `false`, jeśli połączenie nie powiodło się. Jeśli skrypt użył wcześniej funkcji `oci_connect()`, tworzy nowe połączenie bazodanowe. Niedokończone (niezatwierdzone) transakcje z jednego połączenia są niedostępne w drugim połączeniu i mogą się nawet wzajemnie blokować. Funkcja przyjmuje tych samych pięć parametrów, co funkcja `oci_connect()`. Ich znaczenie również jest identyczne. Składnia funkcji: `resource oci_new_connect(string` *`nazwa_użytkownika`*`, string` *`hasło`* `[, string` *`nazwa_bazy`* `[, string` *`kodowanie_znaków`* `[, string` *`tryb_sesji`*`]]])`
`oci_pconnect()` *`ociplogin()`*	Funkcja zwraca zasób połączenia po poprawnym połączeniu z bazą danych lub wartość `false`, jeśli połączenie nie powiodło się. Kilkukrotne wywołanie funkcji z tymi samymi parametrami zwraca to samo połączenie, jeśli nie zostało ono wcześniej przerwane. Połączenie trwa dłużej niż wykonywanie pojedynczego skryptu, ale na końcu każdego skryptu niezatwierdzone transakcje zostają wycofane. Funkcja przyjmuje tych samych pięć parametrów, co funkcja `oci_connect()`. Ich znaczenie również jest identyczne. Składnia funkcji: `resource oci_pconnect(string` *`nazwa_użytkownika`*`, string` *`hasło`* `[, string` *`nazwa_bazy`* `[, string` *`kodowanie_znaków`* `[, string` *`tryb_sesji`*`]]])`

Po przedstawieniu rodzajów połączeń możemy rozpocząć przykłady. Zgodnie z wcześniejszymi wyjaśnieniami uruchom skrypty *create_user.sql* i *create_signon_trigger.sql*, by stworzyć odpowiednie środowisko dla skryptów w bazie danych. Dokładniejszy opis tych skryptów znajduje się w dodatku J.

Program *OracleStandardConnection.php* spróbuje połączyć się z bazą danych. Jeśli połączenie będzie udane, program wyświetli stosowną informację. W przeciwnym przypadku wyświetli tablicę błędów wygenerowaną przez funkcję `oci_error()`.

```
-- Przedstawiony kod znajduje się w pliku OracleStandardConnection.php, na dołączonej płycie CD-ROM.

<?php
  // Spróbuj połączyć się z bazą danych, używając podanego loginu i hasła oraz aliasu TNS.
  if ($c = @oci_connect("php","php","xe"))
  {
    // Wyświetl komunikat o udanym połączeniu.
    echo "Udane połączenie z bazą danych Oracle.<br />";

    // Rozłącz się.
    oci_close($c);
  }
  else
  {
    // Pobierz informacje o błędzie połączenia.
    $errorMessage = oci_error();

    // Otwórz tabelę HTML.
    print '<table border="1" cellpadding="0" cellspacing="0">';

    // Wyświetl elementy tablicy.
    foreach ($errorMessage as $name => $value)
      print '<tr><td>'.$name.'</td><td>'.$value.'</td></tr>';

    // Zamknij tabelę HTML.
    print '</table>';
  }
?>
```

Wywołanie funkcji `oci_connect()` określa nazwę użytkownika, hasło i alias sieciowy (lub nazwę bazy danych). Należy zmodyfikować ten i wszystkie kolejne skrypty, jeśli używa się innej nazwy użytkownika, innego hasła lub innej nazwy bazy danych. Jeśli skrypt poprawnie połączy się z bazą danych, pojawi się następujący komunikat:

```
Udane połączenie z bazą danych Oracle
```

Można również sprawdzić, czy połączenie zostało poprawnie rozpoznane po stronie serwera bazy danych. Połącz się z bazą danych jako użytkownik administracyjny i wykonaj skrypt *get_connection_results.sql*, który pobiera i wyświetla dane z tabeli `CONNECTION_LOG`.

```
-- Przedstawiony kod SQL znajduje się w pliku get_connection_results.sql, na dołączonej płycie CD-ROM.

SELECT   event_id
,        event_user_name
,        event_type
,        TO_CHAR(event_date,'DD-MON-YYYY HH24:MI:SS') time
FROM     system.connection_log;
```

Oczywiście, prezentowane daty będą różniły się od tych przedstawianych poniżej. Uzyskanych wyników również może być więcej — wszystko zależy od liczby przeprowadzanych prób. Ogólny wynik powinien odpowiadać poniższemu:

```
EVENT_ID   EVENT_USER_NAME   EVENT_TYPE   TIME
--------   ---------------   ----------   ----------------------------
      45   PHP               CONNECT      23-CZE-2007 09:52:06
      46   PHP               DISCONNECT   23-CZE-2007 09:52:06
```

Wyniki dobitnie pokazują, że skrypt *OracleStandardConnection.php* połączył się z bazą danych, by po chwili znowu się rozłączyć. Aby przekonać się, jak wygląda zgłoszenie informacji o błędzie, można chwilowo wyłączyć proces nasłuchujący połączeń w sposób opisany w dodatku G.

Po wyłączeniu nasłuchiwania połączeń i uruchomieniu skryptu pojawi się w przeglądarce następujący tekst:

```
code        12541
message     ORA-12541: TNS:no listener
offset      0
sqltext
```

Błąd ORA-12541 informuje, że bibliotece nie udało się uzyskać połączenia z procesem nasłuchującym. Numer linii i fragment polecenia SQL nie są w tym przypadku pomocne, ponieważ błąd wystąpił na etapie połączenia, a nie na etapie wykonywania konkretnych poleceń. W aplikacji produkcyjnej warto zapamiętać przynajmniej treść elementu `message` po wykryciu błędu, bo dostarcza on największą liczbę informacji na temat tego, co było przyczyną niepowodzenia.

Program *OracleNewConnection.php* ilustruje, że można uzyskać dwa połączenia do bazy danych Oracle w obrębie jednego skryptu PHP. Funkcja `oci_connect()` wykonuje pierwsze połączenie, a funkcja `oci_new_connect()` drugie. **Choć oba połączenia zostałyby niejawnie zamknięte po zakończeniu skryptu, kod zamyka je jawnie, wywołując funkcję** `oci_close()`. Kod tego programu jest przedstawiony poniżej.

```
-- Przedstawiony kod znajduje się w pliku OracleNewConnection.php, na dołączonej płycie CD-ROM.

<?php
  // Spróbuj połączyć się z bazą danych, używając podanego loginu i hasła oraz aliasu TNS.
  if ($c = @oci_connect("php","php","xe"))
  {
    // Wyświetl komunikat o udanym połączeniu.
    echo "Udane połączenie z bazą danych Oracle.<br />";

    // Opóźnij wykonanie drugiego połączenia o 5 sekund.
    if (sleep(5));

    if ($c2 = oci_new_connect("php","php","xe"))
    {
      // Wyświetl komunikat, jeśli połączenie było udane.
      echo "Udane dodatkowe połączenie z bazą danych Oracle.<br />";

      // Zamknij dodatkowe połączenie.
      oci_close($c2);
    }
    else
    {
      // Pobierz informację o błędzie połączenia.
      $errorMessage = oci_error();
      print $errorMessage['message'];
    }
```

```
    // Opóźnij wykonanie drugiego zamknięcia o 5 sekund.
    if (sleep(5));

    // Zamknij połączenie.
    oci_close($c1);
  }
  else
  {
    // Pobierz informację o błędzie połączenia.
    $errorMessage = oci_error();
    print $errorMessage['message'];
  }
?>
```

Skrypt generuje następujący wynik, jeśli nie napotkał żadnych błędów:

```
Udane połączenie z bazą danych Oracle.
Udane dodatkowe połączenie z bazą danych Oracle.
```

Skrypt używa funkcji `sleep()` do uzyskania opóźnień między połączeniami. Dzięki temu widać różnicę w czasie wykonywania poszczególnych działań po uruchomieniu skryptu *get_connection_results.sql*. Pamiętaj, że czas i identyfikatory zależą od liczby wcześniej wykonywanych prób połączeń.

```
EVENT_ID   EVENT_USER_NAME   EVENT_TYPE   TIME
--------   ---------------   ----------   ----------------------------
      47   PHP               CONNECT      23-CZE-2007 09:55:06
      48   PHP               CONNECT      23-CZE-2007 09:55:12
      49   PHP               DISCONNECT   23-CZE-2007 09:55:12
      50   PHP               DISCONNECT   23-CZE-2007 09:55:18
```

Po zapoznaniu się z kodem wykonującym połączenia pojedyncze i połączenia wielokrotne przejdźmy do kodu tworzącego połączenia trwałe. Połączenie trwałe pozostaje otwarte aż do momentu wykonania innego połączenia lub przekroczenia czasu bezczynności połączenia. W ten sposób wszystkie informacje statusowe pozostają zapamiętane między uruchomieniami skryptu, a sam proces odczytu danych z bazy nie staje się szybszy (nie trzeba zestawiać nowego połączenia).

Aby zademonstrować działanie połączeń trwałych od strony bazy danych, potrzebujemy dodatkowego kodu SQL i PL/SQL. Aby dokładniej zrozumieć wykonywane przez ten kod operacje, warto wcześniej przeczytać dodatki H i I. Niniejszy podrozdział wykorzystuje również zmienne dowiązane, szczegółowo opisywane w dalszej części tego rozdziału oraz w rozdziale 14.

Oracle udostępnia pakiet `DBMS_APPLICATION_INFO`, który umożliwia zapis i odczyt danych z kolumny `CLIENT_INFO` widoku `V$SESSION`. Procedura `SET_CLIENT_INFO` umożliwia zapisanie informacji, która pozostaje taka sama przez całą sesję. Sesja to czas od połączenia z bazą danych do momentu rozłączenia. Przed ustawieniem wartości procedurą `SET_CLIENT_INFO` kolumna zawiera wartość `null`. Po ustawieniu kolumny jej wartość odczytuje się procedurą `READ_CLIENT_INFO`.

Skrypt *create_session_structures.sql* tworzy tabelę `SESSION_LOG` i pakiet `SESSION_MANAGER`. Pakiety bazodanowe przypominają pliki biblioteczne języka PHP. Język PL/SQL obsługuje dwa rodzaje zestawów operacji: funkcje i procedury. Funkcje zwracają wartość i mogą zostać użyte w instrukcji SQL lub jako prawa część operatora przypisania. Procedury nie zwracają

wartości w tradycyjnym pojęciu i dodatkowo obsługują **przekazywanie parametrów przez referencję**. Przekazywanie przez referencję przypomina ten sam mechanizm z języka PHP (patrz rozdział 7.), tyle że w PHP włączamy go, umieszczając przed nazwą parametru symbol ampersand. Procedury PL/SQL nie mogą być wykorzystane po prawej stronie operacji przypisania.

Skrypt *create_session_structures.sql* tworzy tabelę i sekwencję. Sekwencja umożliwia automatyczną numerację wierszy tabeli.

```
-- Przedstawiony kod SQL znajduje się w pliku create_session_structures.sql, na dołączonej płycie CD-ROM.

CREATE TABLE session_log
( session_id        NUMBER
, session_activity  VARCHAR2(3)
, session_name      VARCHAR2(64)
, session_date      DATE);

CREATE SEQUENCE session_log_s1;
```

Podprocedury jako czarne skrzynki

Istnieją trzy sposoby pisania podprocedur nazywane funkcjami, procedurami i metodami.

- Klasyczna funkcja przekazująca parametry przez wartość operuje na kopiach danych przekazywanych przez użytkownika. Parametry to zmienne o zasięgu lokalnym. Funkcja zwraca tylko jedną wartość skalarną lub złożoną. Zmienna złożona to często po prostu adres w pamięci powiązany z tablicą danych. Ten sposób działania odpowiada funkcjom PL/SQL i jednemu z dwóch stylów tworzenia funkcji w PHP.
- Klasyczna funkcja przekazująca parametry przez referencję może działać w trybie tylko do odczytu lub w trybie odczytu i zapisu. Poszczególne parametry są przekazywane przez referencję, co oznacza, że nie powstaje ich kopia, a funkcja może zmienić ich oryginalną zawartość. Oznacza to, że funkcje tego typu mogą zwracać wartości w dwojaki sposób: jako wynik swego działania lub jako modyfikację zawartości parametrów do nich przekazanych. Zastosowanie typu `void` jako zwracanej wartości powoduje, że funkcja może przekazywać wartości tylko dzięki zmianom zawartości parametrów. Choć klasyczne przekazywanie przez referencję działa w języku PHP, nie jest dostępne w funkcjach PL/SQL.
- Klasyczna procedura jest nieco zmodyfikowaną wersją funkcji przekazującej parametry przez referencję, ponieważ nie może jawnie zwracać wartości. W większości sytuacji liczy się efekt działania procedury, a nie zwracane przez nią wartości. Warto jednak pamiętać, że procedura może zwracać wartości, tyle że musi w tym celu użyć parametrów. Klasyczna procedura to jedyny model przekazywania danych przez referencję obsługiwany przez bibliotekę OCI8.

Skrypt tworzy pakiet z dwoma procedurami otaczającymi procedury `DBMS_APPLICATION_INFO`. Procedury umożliwiają sprawdzenie, czy połączenia trwałe działają zgodnie z oczekiwaniami.

```
-- Przedstawiony kod SQL znajduje się w pliku create_session_structures.sql, na dołączonej płycie CD-ROM.

CREATE OR REPLACE PACKAGE session_manager IS
  PROCEDURE set_session
  (session_name IN VARCHAR2);
  PROCEDURE get_session
  (session_name IN OUT VARCHAR2);
END session_manager;
/
CREATE OR REPLACE PACKAGE BODY session_manager IS
```

```
  PROCEDURE set_session
  (session_name IN VARCHAR2) IS
  BEGIN
    -- Ustaw kolumnę V$SESSION.CLIENT_INFO dla sesji.
    dbms_application_info.set_client_info(session_name);
    -- Zapamiętaj aktywność.
    INSERT INTO session_log VALUES
    ( session_log_s1.nextval
    ,'SET'
    , session_name
    , SYSDATE );
    COMMIT;
  END set_session;
  PROCEDURE get_session
  (session_name IN OUT VARCHAR2) IS
  BEGIN
    dbms_application_info.read_client_info(session_name);
    -- Warunkowo odczytaj wartość.
    IF session_name IS NOT NULL THEN
      -- Zapamiętaj aktywność.
      INSERT INTO session_log VALUES
      ( session_log_s1.nextval
      ,'GET'
      , session_name
      , SYSDATE );
      COMMIT;
    END IF;
  END get_session;
END session_manager;
```

Program *OraclePersistentConnection.php* wykorzystuje kilka funkcji i rozwiązań opisywanych dokładniej w dalszej części rozdziału. Program współdzieli jedno połączenie bazodanowe między dwa wykonania poleceń, które przekazują zmienną dowiązaną między skryptem PHP i bazą danych. Zmienne **dowiązane** stanowią tymczasowe zastępniki dla rzeczywistych wartości. Dzięki nim w programie PHP bardzo łatwo przekazać zawartość zmiennej PHP do zmiennej Oracle i na odwrót.

Przed uruchomieniem programu nie istnieje trwałe połączenie bazodanowe. Program tworzy nowe połączenie, zapisuje dane do sesji i potwierdza wykonanie tego zadania, zapisując informację o stanie do tabeli `SESSION_LOG`. Ponownie uruchomienie programu wykorzysta istniejące połączenie bazodanowe, odczyta wartość z poziomu sesji i zapisze ją do tabeli `SESSION_LOG`.

Program *OraclePersistentConnection.php* ilustruje tworzenie i wykorzystywanie trwałego połączenia do bazy danych Oracle. Pamiętaj o tym, by wcześniej poprawnie ustawić wszystkie potrzebne dyrektywy z plików *php.ini* i *httpd.conf*; **dotyczy to również dyrektywy `KeepAlive`, której wartość nie powinna być większa niż kilkanaście sekund. W przeciwnym razie znacząco wzrasta prawdopodobieństwo ataku typu *DOS* (*Denial Of Service*).**

Kod programu *OraclePersistentConnection.php* jest następujący:

```
-- Przedstawiony kod znajduje się w pliku OraclePersistentConnection.php, na dołączonej płycie CD-ROM.

<?php
  // Spróbuj połączyć się z bazą danych, używając podanego loginu i hasła oraz aliasu TNS.
  if ($c = @oci_pconnect("php","php","xe"))
```

```
{
  // Ustaw zmienną śledzenia sesji.
  $session_name_in = "Sesja z [".date('d-M-y H:i:s')."]";
  $session_name_out = "";

  // Polecenia SQL.
  $stmt1 = "BEGIN session_manager.get_session(:s_name); END;";
  $stmt2 = "BEGIN session_manager.set_session(:s_name); END;";

  // Przetwórz polecenia.
  $s1 = oci_parse($c,$stmt1);
  $s2 = oci_parse($c,$stmt2);

  // Dołącz zmienną $s1 w trybie IN/OUT, a zmienną $s2 w trybie IN.
  oci_bind_by_name($s1,":s_name",$session_name_out,64,SQLT_CHR);
  oci_bind_by_name($s2,":s_name",$session_name_in,64,SQLT_CHR);

  // Wykonaj procedurę GET_SESSION.
  oci_execute($s1);

  // Odczytaj wynik z GET_SESSION.
  if (is_null($session_name_out))
  {
    // Wykonaj procedurę SET_SESSION i ponownie wykonaj procedurę GET_SESSION.
    oci_execute($s2);

    // Wyświetl komunikat.
    print "Udane utworzenie trwałego połączenia do bazy Oracle.<br />";
    print "Ustawione [$session_name_in]<br>";
  }
  else
  {
    // Wyświetl komunikat.
    print "Udane pobranie trwałego połączenia do bazy Oracle.<br />";
    print "Pobrane [$session_name_out]<br>";
  }
}
else
{
  // Przypisz informację o błędzie do zmiennej.
  $errorMessage = oci_error();
}
?>
```

Podobnie jak wcześniej wyzwalacz dotyczący logowania automatycznie utworzy nowy wiersz w tabeli CONNECTION_LOG. Dzieje się tak przy każdym uruchamianiu skryptu. Sytuacja ta jest podobna do chwilowego przełączenia na innego użytkownika w obrębie tej samej sesji Oracle. Kontynuację sesji można wykazać, sprawdzając zawartość tabeli SESSION_LOG. Zauważ, że skrypt nie używa funkcji oci_connect(), gdyż spowodowałoby to utworzenie nowej sesji zamiast kontynuacji aktualnej.

Pierwsze uruchomienie programu spowoduje wyświetlenie następującego wyniku w przeglądarce internetowej:

```
Udane utworzenie trwałego połączenia do bazy Oracle.
Ustawione [Sesja z [23-Jun-07 12:20:23]]
```

Kolejne uruchomienia spowodują uzyskanie następującego wyniku:

```
Udane pobranie trwałego połączenia do bazy Oracle.
Pobrane [Sesja z [23-Jun-07 12:20:23]]
```

Zawartość tabeli SESSION_LOG łatwo sprawdzić, używając skryptu *get_session_logs.sql* za pomocą aplikacji SQL*Plus lub interfejsu internetowego bazy danych. Dokładne wyjaśnienie działania interfejsu wykonywania skryptów SQL znajduje się w dodatku G. Skrypt *get_session_logs.sql* wyświetla informacje zapamiętywane przez procedury z DBMS_APPLICATION_INFO.

```
SELECT   session_id
,        session_activity
,        session_name
,        TO_CHAR(session_date,'DD-MON-YYYY HH24:MI:SS') connection
FROM     session_log
```

Skrypt wyświetla następujące dane w interaktywnej sesji SQL*Plus.

```
                                                Połączenie
   #  Akcja  Nazwa sesji                        Data
----  -----  -------------------------------  ---------------------------------
   1  SET    Sesja z [23-Jun-07 12:20:23]      23-CZE-2007 12:20:23
   2  GET    Sesja z [23-Jun-07 12:20:23]      23-CZE-2007 12:20:29
   3  GET    Sesja z [23-Jun-07 12:20:23]      23-CZE-2007 12:20:33
   4  GET    Sesja z [23-Jun-07 12:20:23]      23-CZE-2007 12:20:53
```

Wynik pokazuje wykorzystanie akcji SET w pierwszym przypadku i akcji GET we wszystkich pozostałych wywołaniach. Akcja SET sygnalizuje uruchomienie nowej sesji, co oznacza, że kolejne wywołania oci_pconnect() zwracały tę samą sesję. Dodatkową informacją potwierdzającą tę tezę jest ten sam czas podawany w nazwie sesji.

Zbyt wiele trwałych połączeń może uniemożliwić przyjmowanie nowych połączeń lub zdalne wyłączenie serwera bazy danych. Gdy zostaje zgłoszony błąd ORA-12520, oznacza to, że system nie może znaleźć uchwytu dla połączenia. Błąd ORA-12514 oznacza niemożność uruchomienia usługi na podstawie przekazanego deskryptora. Błędy ORA-24323 i ORA-23324 wskazują na zajętość zasobów związaną z otwarciem zbyt wielu połączeń. Najlepszym sposobem rozwiązania problemu okazuje się ponowne uruchomienie serwera Apache.



> Można zacząć zastanawiać się, czy połączenia trwałe naprawdę są potrzebne. Odpowiedź na to pytanie wymaga wyliczenia potencjalnego ryzyka związanego z trwałymi połączeniami, a z drugiej strony zalet takich połączeń w witrynach stosujących AJAX, wirtualne prywatne bazy danych itp.

Choć w tym rozdziale zostały zaprezentowane połączenia trwałe, we wszystkich kolejnych przykładach kod wykorzystuje połączenia standardowe. Następny podrozdział opisuje podstawowe funkcje OCI8 i sposoby dowiązywania zmiennych.

Wykorzystanie biblioteki funkcji OCI8

Biblioteka funkcji OCI8 zawiera funkcje dotyczące łączenia, wykonywania działań i rozłączania z bazą danych Oracle. Dodatkowo zawiera funkcje obsługi kolekcji i dużych obiektów. W poprzednim podrozdziale zajęliśmy się trzema funkcjami łączącymi i jedną rozłączającą. Ten podrozdział opisuje większość pozostałych funkcji biblioteki OCI8 z wyłączeniem obsługi kolekcji i dużych obiektów, które zostaną opisane szczegółowo w rozdziałach 14. i 15.

Tabela 13.3 zawiera listę funkcji OCI8 umożliwiających realizację poleceń SQL i PL/SQL. Przedstawione funkcje działają poprawnie tylko w połączeniu z bazą danych Oracle Database 10*g* i PHP w wersji 5.1.2 lub nowszej. Istnieją również aliasy starszych funkcji dla osób, które migrują ze starszych wydań — wycofywane nazwy funkcji zostały zapisane kursywą w tabeli 13.3.

Podrozdział został podzielony na 5 punktów:

- Podsumowanie funkcji z biblioteki OCI8
- Wprowadzenie do modeli bazodanowych
- Przykład tworzenia zapytania
- Przykład uruchamiania transakcji
- Przykład zmiany nazwy użytkownika

Podsumowanie funkcji z biblioteki OCI8

Biblioteka OCI8 zawiera funkcje dotyczące wykonywania statycznych i dynamicznych operacji SQL związanych z DDL (*Data Definition Language*), DML (*Data Manipulation Language*), DQL (*Data Query Language*) i DCL (*Data Control Language*). Polecenia DDL umożliwiają **tworzenie**, **modyfikację** i **usuwanie** obiektów bazy danych (tabel, widoków itp.). Nazwy funkcji z tabeli 13.3 pisane kursywą są obecnie wycofywane z użycia. Nowe funkcje, np. `oci_bind_array_by_name()`, nie posiadają swoich starszych odpowiedników.

Tabela 13.3. *Funkcje biblioteki OCI8 związane z zapytaniami i dostępne w języku PHP*

Funkcja	Opis
`oci_bind_array_by_name()`	Funkcja dowiązuje indeksowaną numerycznie tablicę PHP do tablicy asocjacyjnej PL/SQL (dawniej nazywaną tabelą PL/SQL). Funkcja zwraca wartość `true` po udanym dowiązaniu lub `false` w przypadku przeciwnym. Obecnie funkcja może jedynie dowiązywać tablicę z typami skalarnymi, na przykład `VARCHAR2`, `NUMBER` i `DATE`. Zespół programistyczny Oracle planuje dodanie obsługi tablic, ale dokładna data wprowadzenia tej funkcji nie jest jeszcze znana. Funkcja przyjmuje sześć parametrów, z czego cztery są wymagane. Pierwszy i drugi parametr jest przekazywany przez wartość. Pierwszy określa zasób polecenia, a drugi nazwę zmiennej dowiązania użytej w poleceniu SQL. Trzeci parametr zostaje przekazany z przez referencję, co oznacza, że może ulec zmianie, ale **tylko jeśli** włączony zostanie tryb `IN/OUT`. Kolejne parametry są ponownie przekazywane przez wartość. Czwarty parametr określa liczbę elementów na liście

Tabela 13.3. *Funkcje biblioteki OCI8 związane z zapytaniami i dostępne w języku PHP — ciąg dalszy*

Funkcja	Opis

	i musi przyjmować wartość dodatnią lub 0. Piąty parametr określa maksymalny rozmiar wartości skalarnej w tablicy. **Parametr musi być równy fizycznemu rozmiarowi kolumny lub dla kolumn dynamicznych ich maksymalnemu rozmiarowi**. Kolumny dynamiczne powstają m.in. przy złączaniu kilku wartości tekstowych. Szósty parametr określa typ danych:
	♦ SQLT_AFC — typ danych CHAR,
	♦ SQLT_AVC — typ danych CHAR2,
	♦ SQLT_CHR — typ danych VARCHAR2,
	♦ SQLT_FLT — typ danych FLOAT,
	♦ SQLT_INT — typ danych INTEGER,
	♦ SQLT_LVC — typ danych LONG,
	♦ SQLT_NUM — typ danych NUMBER,
	♦ SQLT_ODT — typ danych DATE,
	♦ SQLT_STR — typ danych STRING,
	♦ SQLT_VCS — typ danych VARCHAR.
	Składnia funkcji:
	`bool oci_bind_array_by_name(resource` *instrukcja*`,`
	`string` *nazwa_dowiązania*`, array &`*tablica_referencyjna*`,`
	`int` *maks_elementów* `[, int` *maks_długość_pola* `[, int` *odwzorowany_typ*`]])`

`oci_bind_by_name()` *ocibindbyname*`()`	Funkcja odwzorowuje typ Oracle na zmienną PHP. Zmienna może być skalarem lub kolekcją wartości skalarnych, ale nie tablicą asocjacyjną z Oracle 10g, bo w takiej sytuacji należy zastosować funkcję `oci_bind_array_by_name()`. Kolekcja wartości skalarnych może mieć przypisany typ VARRAY lub **zagnieżdżony** typ TABLE. Dokładne omówienie tych typów znajduje się w dodatku I. Funkcja zwraca wartość `true` po udanym dowiązaniu lub `false` w przypadku przeciwnym. Funkcja przyjmuje pięć parametrów, z czego trzy są wymagane. Pierwszy i drugi parametr jest przekazywany przez wartość. Pierwszy określa zasób polecenia, a drugi nazwę zmiennej dowiązania użytej w poleceniu SQL. Trzeci parametr zostaje przekazany z przez referencję, co oznacza, że może ulec zmianie, ale **tylko jeśli** włączony zostanie tryb IN/OUT. Kolejne parametry są ponownie przekazywane przez wartość. Czwarty parametr określa maksymalną liczbę elementów na liście. Piąty parametr określa maksymalny rozmiar wartości skalarnej w tablicy. Parametr można ustawić na wartość –1, by funkcja niejawnie określiła rozmiar pola. Szósty parametr określa typ danych: ♦ SQLT_B_CURSOR — typ danych będący kursorem referencyjnym, ♦ SQLT_BIN — typ danych RAW, ♦ SQLT_BLOB — typ danych BLOB, ♦ SQLT_CFILE — typ danych CFILE, ♦ SQLT_CHR — typ danych VARCHAR, ♦ SQLT_CLOB — typ danych CLOB, ♦ SQLT_FILE — typ danych BFILE, ♦ SQLT_INT — typ danych INTEGER lub NUMBER, ♦ SQLT_LBI — typ danych LONG RAW, ♦ SQLT_LNG — typ danych LONG, ♦ SQLT_NTY — typ danych zdefiniowany przez użytkownika lub kolekcja wartości skalarnych (albo zagnieżdżony typ TABLE), ♦ SQLT_RDD — typ danych ROWID.

Tabela 13.3. *Funkcje biblioteki OCI8 związane z zapytaniami i dostępne w języku PHP — ciąg dalszy*

Funkcja	Opis
	Typy abstrakcyjne alokuje się, wywołując funkcję `oci_new_descriptor()` przed ich dowiązaniem. Typami abstrakcyjnymi są: `LOB`, `ROWID` i `BFILE`. Przed dowiązaniem kursora referencyjnego trzeba wywołać funkcję `oci_new_cursor()`. Składnia funkcji: `bool oci_bind_by_name(resource` *instrukcja*`, string nazwa_`*dowiązania*`, mixed &`*wartość*` [, int` *maks_długość_pola* `[, int` *odwzorowany_typ*`]])`
`oci_cancel()` *ocicancel*`()`	Funkcja zamyka kursor i zwalnia wszystkie związane za nim zasoby. Zwraca wartość `true` po udanym wykonaniu lub `false` w przypadku przeciwnym. Przyjmuje jeden parametr, którym jest zasób polecenia. Składnia funkcji: `bool oci_cancel(resource` *polecenie*`)`
`oci_commit()` *ocicommit*`()`	Funkcja wysyła polecenie sterujące `COMMIT` do aktualnej sesji, co utrwala wszystkie zmiany dokonane w danych. Zwraca wartość `true` po udanym wykonaniu lub `false` w przypadku przeciwnym. Przyjmuje jeden parametr, którym jest zasób połączenia. Składnia funkcji: `bool oci_commit(resource` *połączenie*`)`
`oci_define_by_name()` *ocidefinebyname*`()`	Funkcja definiuje zmienną PHP i odwzorowuje ją na kolumnę zwracaną przez zapytanie SQL. Funkcję wywołuje się przed wykonaniem funkcji `oci_execute()` lub gdy jeszcze nie występuje powiązanie z lokalną zmienną. Funkcja zwraca wartość `true` po udanym wykonaniu lub `false` w przypadku przeciwnym. Przyjmuje cztery parametry, z których trzy są wymagane. Pierwszy i drugi parametr jest przekazywany przez wartość. Pierwszy określa zasób polecenia, a drugi nazwę kolumny z polecenia SQL **pisaną dużymi literami**. Trzeci parametr zostaje przekazany z przez referencję i określa zmienną PHP, do której zostanie wstawiony wynik. Czwarty parametr określa typ danych, który jest taki sam jak typy wymienione dla funkcji `oci_bind_by_name()`. Składnia funkcji: `bool oci_define_by_name(resource` *instrukcja*`, string` *nazwa_dowiązania*`, mixed &`*zmienna* `[, int` *odwzorowany_typ*`])`
`oci_execute()` *ociexecute*`()`	Funkcja wykonuje przetworzone polecenie SQL lub PL/SQL. Zwraca wartość `true` po udanym wykonaniu lub `false` w przypadku przeciwnym. Przyjmuje dwa parametry: wymagany i opcjonalny. Pierwszy parametr określa zasób polecenia, który został zwrócony jako wynik funkcji `oci_parse()`. Parametr opcjonalny określa tryb wykonywania polecenia SQL. Domyślnie PHP przyjmuje tryb `OCI_COMMIT_ON_SUCCESS`, który automatycznie zatwierdza wykonane polecenie. Tryb `OCI_DEFAULT` wyłącza automatyczne zatwierdzanie. Składnia funkcji: `bool oci_execute(resource` *polecenie* `[, int` *tryb*`])`
`oci_fetch()` *ocifetch*`()`	Funkcja pobiera wiersz danych z wykonanego kursora. Zwraca wartość `true` po udanym wykonaniu lub `false` w przypadku przeciwnym. Przyjmuje jeden parametr, którym jest zasób połączenia. Składnia funkcji: `bool oci_fetch(resource` *połączenie*`)`
`oci_fetch_all()` `ocifetchstatement()`	Funkcja pobiera wszystkie wiersze z wykonanego kursora do jednej tablicy wyników. Funkcja zwraca liczbę odczytanych wierszy lub wartość `false` w przypadku niepowodzenia. Przyjmuje pięć argumentów, z których dwa są wymagane. Pierwszy parametr to zasób połączenia. Drugi to nazwa docelowej tablicy, w której zostaną umieszczone dane. Trzeci parametr (pierwszy z opcjonalnych) określa liczbę początkowych wierszy do pominięcia w trakcie odczytu. Domyślnie przyjmuje wartość 0. Czwarty parametr określa maksymalną liczbę wierszy do odczytania. Przyjmuje domyślną wartość –1, która oznacza

Tabela 13.3. *Funkcje biblioteki OCI8 związane z zapytaniami i dostępne w języku PHP — ciąg dalszy*

Funkcja	Opis
	odczytywanie wszystkich wierszy. Piąty parametr określa sposób umieszczania danych w tablicy. Można użyć jednej z podanych opcji lub połączyć je wszystkie operatorem bitowym: ♦ OCI_FETCHSTATEMENT_BY_ROW lub ♦ OCI_FETCHSTATEMENT_BY_COLUMN i ♦ OCI_ASSOC lub ♦ OCI_NUM. Kombinacja OCI_FETCHSTATEMENT_BY_COLUMN i OCI_ASSOC zostaje użyta domyślnie, jeśli nie podamy tego parametru. Opcja OCI_ASSOC ma jedną zaletę — nazwy pól wewnętrznych podtablic są nazwami kolumn. Składnia funkcji: `int oci_fetch_all(resource polecenie, array &tablica_danych [, int pomiń_wiersze [, int maksymalnie_wierszy [, int format_wyników]]])`
oci_fetch_array() ocifetchinto()	Funkcja pobiera następny wiersz danych z wykonanego kursora. Zwraca dane z wiersza w postaci tablicy asocjacyjnej lub wartość false, jeśli nie ma już danych do pobrania. Przyjmuje dwa parametry, z których pierwszy jest wymagany. Pierwszy parametr to zasób połączenia. Drugi parametr (opcjonalny) określa sposób pobierania danych. Dostępne wartości: ♦ OCI_BOTH — zwraca wiersz jako tablicę z wartościami **numerycznymi** i **tekstowymi**. Jest to tryb domyślny; ♦ OCI_ASSOC — zwraca tablicę z danymi wiersza w postaci tablicy asocjacyjnej z **nazwami kolumn**. W tym trybie funkcja działa podobnie do funkcji oci_fetch_assoc(); ♦ OCI_NUM — zwraca tablicę z danymi wiersza w postaci tablicy **numerycznej**. W tym trybie funkcja działa podobnie do funkcji oci_fetch_row(); ♦ OCI_RETURN_NULLS — zwraca dane podobnie jak rozwiązanie OCI_ASSOC, ale dodatkowo zamienia wartości null z bazy danych na wartości null języka PHP; ♦ OCI_RETURN_LOBS — zwraca kolumny z typem danych LOB jako wartości deskryptorów. **Jest to domyślne zachowanie**. Składnia funkcji: `array oci_fetch_array(resource polecenie [, int tryb]);`
oci_fetch_assoc()	Funkcja pobiera następny wiersz danych z wykonanego kursora. Zwraca dane z wiersza w postaci tablicy asocjacyjnej lub wartość false, jeśli nie ma już danych do pobrania. Przyjmuje jeden parametr, którym jest zasób połączenia. Zwrócona tablica jest tablicą asocjacyjną z kluczami będącymi **nazwami kolumn w bazie danych**. Funkcja działa dokładnie tak samo jak funkcja oci_fetch_array() w trybie OCI_ASSOC. Składnia funkcji: `array oci_fetch_assoc(resource polecenie);`
oci_fetch_object()	Funkcja pobiera następny wiersz danych z wykonanego kursora. Zwraca dane z wiersza w postaci **obiektu z kolumnami** będącymi elementami tegoż obiektu lub wartość false, jeśli nie ma już danych do pobrania. Przyjmuje jeden parametr, którym jest zasób połączenia. Od strony formalnej do pobrania nazw kolumn i uzyskania informacji o ich liczbie należy skorzystać z funkcji oci_num_fields() i oci_fetch_object(). Składnia funkcji: `object oci_fetch_object(resource polecenie);`

Tabela 13.3. *Funkcje biblioteki OCI8 związane z zapytaniami i dostępne w języku PHP — ciąg dalszy*

Funkcja	Opis
`oci_fetch_row()`	Funkcja pobiera następny wiersz danych z wykonanego kursora. Zwraca dane z wiersza w postaci tablicy lub wartość `false`, jeśli nie ma już danych do pobrania. Przyjmuje jeden parametr, którym jest zasób połączenia. Zwrócona tablica jest tablicą. Funkcja działa dokładnie tak samo jak funkcja `oci_fetch_array()` w trybie `OCI_NUM`. Składnia funkcji: `array oci_fetch_row(resource` *`polecenie`*`);`
`oci_field_is_null()` *`ocicolumnisnull`*`()`	Funkcja sprawdza, czy podane pole zawiera w bazie danych wartość `null`. Jeśli tak, zwraca wartość `true`. W przeciwnym razie zwraca `false`. Funkcja przyjmuje dwa parametry: zasób połączenia i nazwę pola. Składnia funkcji: `bool oci_field_is_null(resource` *`połączenie`*`, mixed` *`pole`*`);`
`oci_field_name()` *`ocicolumnname`*`()`	Funkcja zwraca nazwę pola pobranego wiersza danych. Zwraca nazwę pola lub wartość `false` w przypadku błędu. Funkcja przyjmuje dwa parametry: zasób połączenia i pozycję pola w pobranych danych. Numer pola powinien mieć wartość od 1 do 999. Składnia funkcji: `bool oci_field_name(resource` *`połączenie`*`, int` *`pozycja`*`);`
`oci_field_precision()` *`ocicolumnprecision`*`()`	Funkcja zwraca rozmiar danych pola *`NUMBER`* jako wartość dodatnią lub 0. Precyzja równa 0 oznacza, że nic nie może znajdować się w części ułamkowej. Wartość dodatnia określa maksymalną liczbę miejsc po przecinku. Funkcja przyjmuje dwa parametry: zasób połączenia i pozycję pola w pobranych danych. Numer pola powinien mieć wartość od 1 do 999. Składnia funkcji: `int oci_field_precision(resource` *`połączenie`*`, int` *`pozycja`*`);`
`oci_field_scale()` *`ocicolumnscale`*`()`	Funkcja zwraca rozmiar liczb (ich pojemność) po prawej stronie znaku przecinka. Informuje o liczbie znaków zastępczych użytych w wartości. Zwraca wartość dodatnią, 0 lub –127. Wartość równa –127 oznacza, że mamy do czynienia ze zmienną typu `FLOAT`. Funkcja przyjmuje dwa parametry: zasób połączenia i pozycję pola w pobranych danych. Numer pola powinien mieć wartość od 1 do 999. Składnia funkcji: `int oci_field_scale(resource` *`połączenie`*`, int` *`pozycja`*`);`
`oci_field_size()` *`ocicolumnsize`*`()`	Funkcja zwraca rozmiar pola tekstowego o zmiennej długości, na przykład pola `VARCHAR2`. Zwraca rozmiar w bajtach pola. Funkcja przyjmuje dwa parametry: zasób połączenia i pozycję pola w pobranych danych. Numer pola powinien mieć wartość od 1 do 999. Składnia funkcji: `int oci_field_size(resource` *`połączenie`*`, int` *`pozycja`*`);`
`oci_field_type()` *`ocicolumntype`*`()`	Funkcja zwraca typ danych pobranego pola. Informuje o typie dotyczącym daty, jeśli dane pole jest datą i w ten sposób możemy odpowiednio zareagować. **Listę możliwych wartości zawiera tabela H.2**. Funkcja przyjmuje dwa parametry: zasób połączenia i pozycję pola w pobranych danych. Numer pola powinien mieć wartość od 1 do 999. Składnia funkcji: `mixed oci_field_type(resource` *`połączenie`*`, int` *`pozycja`*`);`
`oci_field_type_raw()` *`ocicolumntyperaw`*`()`	Funkcja zwraca wewnętrzny liczbowy typ danych pobranego pola używany przez bazę Oracle. Wewnętrzna wartość typu stanowi cześć metadanych tabeli. **Listę możliwych wartości zawiera tabela H.2**. Funkcja przyjmuje dwa parametry: zasób połączenia i pozycję pola w pobranych danych. Numer pola powinien mieć wartość od 1 do 999. Składnia funkcji: `mixed oci_field_type_raw(resource` *`połączenie`*`, int` *`pozycja`*`);`

Tabela 13.3. *Funkcje biblioteki OCI8 związane z zapytaniami i dostępne w języku PHP — ciąg dalszy*

Funkcja	Opis
`oci_free_statement()` *`ocifreecursor()`*	Zwalnia wszystkie zasoby zaalokowanie przez polecenie SQL lub kursor. Zwraca wartość `true` po udanym wykonaniu lub `false` w przypadku przeciwnym. Przyjmuje jeden parametr, którym jest zasób polecenia. Składnia funkcji: `bool oci_free_statement(resource` *`polecenie`*`)`
`oci_internal_debug()` *`ociinternaldebug()`*	Funkcja włącza wewnętrzny tryb testowy Oracle. Funkcja nic nie zwraca, ponieważ tryb ten dotyczy testowania po stronie serwera. Zerowa wartość parametru wyłącza ten tryb, każda inna go włącza. Składnia funkcji: `void oci_internal_debug(int` *`włącz`*`)`

`oci_new_collection()`
`ocinewcollection()`
	Funkcja tworzy obiekt `OCI-Collection`, który odwzorowuje się na zmienną Oracle Collection. W przypadku sukcesu zwraca obiekt `OCI-Collection`, w przeciwnym razie zwraca wartość `false`. Dokładny opis biblioteki `OCI-Collection` zawiera tabela 14.1. W wersji PHP 5.1.4 typy ograniczone są do kolekcji wartości skalarnych. **Oracle może rozszerzyć obsługę o inne typy danych, ale obecnie *nie została jeszcze określona* żadna konkretna data wprowadzenia ich do biblioteki dostępnej w PHP**. Funkcja przyjmuje trzy parametry, z których dwa są wymagane. Pierwszy parametr to zasób połączenia. Drugi to nazwa schematu używana do wykonania kolekcji. Trzeci parametr (opcjonalny) umożliwia określenie dodatkowej nazwy schematu. Składnia funkcji:
	`OCI-Collection oci_new_collection(resource` *`połączenie`*`,`
	`string` *`nazwa_typu`* `[, string` *`schemat`*`])`

`oci_new_cursor()` *`ocinewcursor()`*	Funkcja tworzy i zwraca zasób kursora systemowego lub wartość `false` w przypadku niemożności jego wykonania. Funkcja przyjmuje jeden parametr — zasób połączenia. Składnia funkcji: `resource oci_new_cursor(resource` *`połączenie`*`)`
`oci_new_descriptor()` *`ocinewdescriptor()`*	Funkcja tworzy nowy obiekt `OCI-Lob`, który odwzorowuje zmienną `LOB`. Funkcja zwraca obiekt `OCI-Lob`. W przypadku błędu zwraca wartość `false`. Tabela 14.1 zawiera opis elementów obiektu `OCI-Lob`. Funkcja przyjmuje dwa parametry, ale tylko pierwszy z nich jest wymagany. Pierwszy parametr to zasób połączenia. Drugi parametr określa typ danych `LOB`. Oracle traktuje typ `LOB` jako typ abstrakcyjny, czyli w ten sam sposób jak typy `ROWID` i `FILE`. Możliwe wartości drugiego parametru: ♦ `OCI_D_FILE` — ustawia deskryptor w tryb obsługi plików binarnych lub znakowych, czyli odpowiednio `BFILE` i `CFILE`; ♦ `OCI_D_LOB` — ustawia deskryptor do obsługi dużych obiektów binarnych lub tekstowych, czyli odpowiednio `BLOB` i `CLOB`; ♦ `OCI_D_ROWID` — ustawia deskryptor w tryb zarządzania wartościami `ROWID`, które odwzorowują bloki systemu plików. Składnia funkcji: `OCI-Lob oci_new_descriptor(resource` *`połączenie`*`, int` *`typ_lob`*`)`
`oci_num_fields()` *`ocinumcols()`*	Funkcja zwraca liczbę pól pobranych w zapytaniu. Poprawne wykonanie zwraca liczbę dodatnią. Błąd powoduje zwrócenie wartości `false`. Przyjmuje jeden parametr, którym jest zasób polecenia. Składnia funkcji: `int oci_num_fields(resource` *`polecenie`*`)`
`oci_num_rows()` *`ocirowcount()`*	Funkcja informuje o liczbie wierszy zmodyfikowanych przez polecenie edycyjne SQL. Poprawne wykonanie zwraca liczbę dodatnią. Błąd powoduje zwrócenie wartości `false`. Funkcja działa poprawnie dla poleceń wstawiania, edycji i usuwania wierszy. Może również zwrócić poprawne wyniki dla skryptów PL/SQL.

Tabela 13.3. *Funkcje biblioteki OCI8 związane z zapytaniami i dostępne w języku PHP — ciąg dalszy*

Funkcja	Opis
	Można jej również użyć dla zapytań SQL, ale wtedy informuje nie o ogólnej liczbie wierszy, ale o liczbie wierszy pobranych funkcją `oci_fetch()`. Funkcja zwraca poprawne dane dopiero po wykonaniu funkcji `oci_execute()`. Składnia funkcji: `int oci_num_rows(resource polecenie)`
`oci_parse()` *`ociparse()`*	Funkcja umieszcza polecenie w Oracle SGA, który przygotowuje polecenie do wykonania. Zwraca zasób polecenia po udanym umieszczeniu zapytania lub wartość `false` po napotkania błędu. Funkcja nie sprawdza poprawności zapytania, więc by się przekonać o jego poprawności, trzeba je wykonać. Funkcja przyjmuje dwa parametry: zasób połączenia i treść polecenia SQL. Składnia funkcji: `resource oci_parse(resource połączenie, string polecenie_sql)`

`oci_password_change()`	Funkcja umożliwia zmianę hasła. Istnieją dwa sposoby jej użycia. Pierwsze zwraca wartość logiczną, a drugie zasób połączenia. W przypadku wartości logicznych uzyskanie wartości `true` oznacza sukces, natomiast uzyskanie wartości `false` błąd. Funkcja przyjmuje cztery parametry: zasób połączenia lub alias sieciowy zdefiniowany w pliku *tnsnames.ora*, nazwę użytkownika, stare hasło i nowe hasło. Jeśli pierwszy parametr był zasobem połączenia, zwraca wartość logiczną. W przeciwnym razie zwraca zasób połączenie. Składnia funkcji:
	`bool oci_password_change(resource połączenie, string użytkownik, string stare_hasło, string nowe_hasło)`
	`resource oci_password_change(string baza_danych, string użytkownik, string stare_hasło, string nowe_hasło)`

`oci_result()` *`ociresult()`*	Funkcja zwraca wartość pola z pobranego wiersza. Zwracana wartość pola jest tekstem za wyjątkiem typów abstrakcyjnych: `FILE`, `LOB` i `ROWID`. W przypadku błędu zwraca wartość `false`. Funkcja przyjmuje dwa parametry: zasób polecenia i nazwę lub pozycję pola. Składnia funkcji: `mixed oci_result(resource polecenie, mixed pole)`
`oci_rollback()` *`ocirollback()`*	Funkcja wycofuje wszystkie niezatwierdzone operacje w obrębie połączenia bazodanowego. Zwraca wartość `true` po udanym wycofaniu lub wartość `false` po napotkaniu błędu. Funkcja przyjmuje tylko jeden parametr — zasób połączenia. Składnia funkcji: `bool oci_rollback(resource połączenie)`
`oci_server_version()` *`ociserverversion()`*	Funkcja zwraca informacje na temat wersji serwera bazy danych w postaci tekstu. Po napotkaniu błędu zwraca wartość `false`. Funkcja zwraca jeden wymagany parametr, którym jest zasób połączenia. Składnia funkcji: `string oci_server_version(resource połączenie)`
`oci_set_prefetch()` *`ocisetperefetch()`*	Funkcja ustawia nową wartość liczby wierszy pobieranych z wyprzedzeniem z serwera bazy danych. Nadpisuje wartość zapisaną w dyrektywie `default_prefetech`, w pliku *php.ini*. Zwraca wartość `true` w przypadku sukcesu lub `false` po nieudanej zmianie wartości. Przyjmuje dwa parametry, z których tylko pierwszy jest wymagany: zasób połączenia i nową liczbę wierszy pobieranych z wyprzedzeniem. PHP domyślnie przyjmuje wartość 1, jeśli nie podamy drugiego parametru. Składnia funkcji: `bool oci_set_prefetech(resource połączenie [, int liczba_wierszy])`
`oci_statement_type()` *`ocistatementtype()`*	Funkcja zwraca typ polecenia SQL, np. `SELECT`, `UPDATE`, `DELETE`, `INSERT`, `CREATE`, `DROP`, `ALTER`, `BEGIN` i `DECLARE`. Przyjmuje jeden parametr, którym jest zasób polecenia. Składnia funkcji: `string oci_statement_type(resource polecenie)`

Instrukcje DML umożliwiają **tworzenie**, **modyfikację** i **usuwanie** informacji z obiektów bazodanowych. Instrukcje DQL **odpytują** obiekty bazy danych. Instrukcje DCL dotyczące zastawiania transakcji, które w większości sytuacji obejmują wiele obiektów bazodanowych i zapewniają, by zostały zapisane wszystkie zmiany lub też wszystkie zostały wycofane. Więcej informacji na ten temat znajduje się w dodatku H.

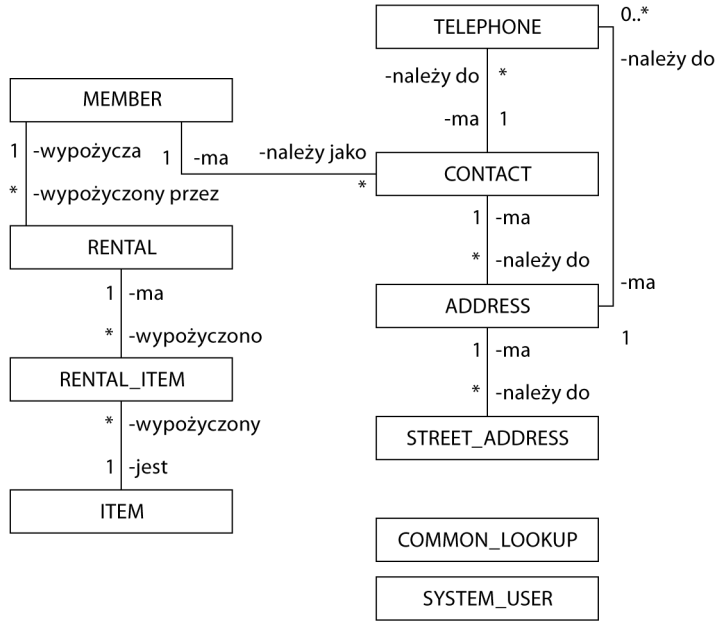
Jak wcześniej wspomniano, tabela 13.3 zawiera większość funkcji biblioteki OCI8, ale pomija te związane z kolekcjami i dużymi obiektami. Elementy te są dokładniej opisywane w rozdziale 14. i 15. Opis funkcji `oci_lob_copy()` i `oci_lob_is_equal()` znajdują się w tabeli 15.1 z rozdziału 15.

Wprowadzenie do modeli bazodanowych

Przykładowe programy bazują na dwóch modelach bazodanowych. Pierwszy to mała wypożyczalnia kaset wideo i gier, którego zadaniem jest przedstawienie obsługi transakcji obejmujących wiele tabel bazy danych. Drugi model bazy danych dotyczy prezydentów USA i ilustruje wykorzystanie tablic asocjacyjnych, skryptów PL/SQL, kursorów referencyjnych i kolekcji danych skalarnych (patrz rozdział 14.). Jak wcześniej wspomniano, pełny kod obu modeli znajduje się w dodatku J.

Rysunek 13.1 przedstawia uproszczony diagram klas dotyczący modelu bazodanowego dla aplikacji wypożyczalni kaset wideo. Ten diagram klas odpowiada **diagramowi związków encji** (**ERD**) wykorzystywanego przy zaawansowanym projektowaniu baz danych. Tabele odwzorowują klasy bez metod.

Rysunek 13.1. *Model bazy danych wypożyczalni kaset wideo*



Klasy (tabele) łączą linie związków. Każdy związek opisują dwa czasowniki oraz tzw. krotność związku. Często krotność podaje się z dwiema wartościami, co oznacza minimalną i maksymalną krotność związku. Minimalna krotność określa najmniejszą możliwą liczbę obiektów (wierszy) danego typu. Maksymalna krotność określa największą możliwą liczbę obiektów (wierszy) danego typu. Krotność czyta się od nazwy tabeli, przechodząc przez związek do wartości krotności i nazwy drugiej tabeli, czyli możemy przeczytać, że **wiersz tabeli CONTACT ma jeden lub więcej wierszy tabeli** `ADDRESS`. Można również czytać w drugą stronę: wiersz tabeli `ADDRESS` należy do jednego i tylko jednego wiersza tabeli `CONTACT`.

Jeśli jedna z tabel może występować w związku z drugą co najwyżej jeden raz, tabela ta przechowuje klucz obcy drugiej z tabel. Klucz obcy to wartość identyfikatora pochodząca (definiowana) z innej tabeli. Przykładowa baza danych stosuje się do prostej konwencji, w której pola klucz głównych zawierają przyrostek `_ID`. Klucz obcy z drugiej tabeli najczęściej (choć nie jest to regułą) ma taką samą nazwę jak pole klucza głównego. Odstępstwem od tej zasady są między innymi kolumny `CREATED_BY` i `LAST_UPDATED_BY`, które dotyczą klucza głównego `SYSTEM_USER_ID` tabeli `SYSTEM_USER`. Kolumny tworzenia i edycji są tak zwanymi **kolumnami audytowymi**, ponieważ informują o tym, kto ostatni tworzył lub edytował dany rekord.

Pozostała część tego rozdziału skupi się na prezentacji przykładów wykonywania typowych operacji SQL (**wstawiania**, **edycji**, **usuwania** i **pobierania**) z wykorzystaniem bazy danych wypożyczalni. Dalsze przykłady dotyczyć będą zmiany hasła i obsługi transakcji za pomocą funkcji `oci_connect()` i `oci_rollback()`.

Przykład tworzenia zapytania

Program *SelectItemRecords.php* ilustruje tworzenie zapytania pobierającego dane z dwóch tabel. Tworzy typowe połączenie, a następnie analizuje, wykonuje i pobiera dane na podstawie polecenia SQL zgodnego ze standardem SQL:2003.

```
-- Przedstawiony kod znajduje się w pliku SelectItemRecords.php, na dołączonej płycie CD-ROM.

<?php
  // Połącz się z bazą danych.
  if ($c = @oci_connect("php","php","xe"))
  {
    // Przekaż polecenie z danymi.
    $s = oci_parse($c,"SELECT   i.item_id id
                       ,        i.item_barcode as barcode
                       ,        c.common_lookup_type as type
                       ,        i.item_title as title
                       ,        i.item_rating as rating
                       ,        i.item_release_date release_date
                       FROM     item i inner join common_lookup c
                       ON       i.item_type = c.common_lookup_id
                       WHERE    c.common_lookup_type = 'XBOX'
                       ORDER BY i.item_title");

    // Wykonaj zapytanie bez jawnego zatwierdzania.
    oci_execute($s,OCI_DEFAULT);

    // Otwórz tabelę HTML.
    print '<table border="1" cellspacing="0" cellpadding="3">';
```

```
    // Odczytaj nazwy kolumn.
    print '<tr>';
    for ($i = 1;$i <= oci_num_fields($s);$i++)
      print '<td class="e">'.oci_field_name($s,$i).'</td>';
    print '</tr>';

    // Pobierz właściwe dane.
    while (oci_fetch($s))
    {
      // Dodaj dane wiersz po wierszu.
      print '<tr>';
      for ($i = 1;$i <= oci_num_fields($s);$i++)
        print '<td class="v">'.oci_result($s,$i).'</td>';
      print '</tr>';
    }

    // Zamknij tabelę HTML.
    print '</table>';

    // Odłącz się od bazy danych.
    oci_close($c);
  }
  else
  {
    // Wyświetl komunikat o błędzie.
    $errorMessage = oci_error();
    print htmlentities($errorMessage['message'])."<br />";
  }
?>
```

Program używa funkcji `oci_parse()` do umieszczenia instrukcji SQL w zasobie polecenia. Następnie wywołuje funkcję `oci_execute()` z dodatkowym parametrem `OCI_DEFAULT`, by uniknąć narzutu związanego z domyślną wartością `OCI_COMMIT_ON_SUCCESS`, ponieważ operacje zatwierdzania mają znaczenie tylko dla **wstawiania**, **edycji** i **usuwania** danych. Po wykonaniu polecenia, ale przed pobraniem właściwych kolumn, dostępna jest informacja na temat nazw kolumn. Program wykorzystuje funkcje `oci_num_fields()` i `oci_field_name()` do pobrania i wyświetlenia nagłówków tabeli HTML. Funkcja `oci_fetch()` zostaje wywołana wewnątrz instrukcji `while`, co powoduje pobranie wszystkich dostępnych wierszy. Wewnątrz pętli funkcje `oci_num_fields()` i `oci_result()` służą do wyświetlenia danych jako kolumn poszczególnych wierszy tabeli.

Przykład uruchamiania transakcji

Program *InsertItemRecord.php* ilustruje **wstawianie** danych do bazy danych wypożyczalni. Wykorzystuje podobne techniki jak program *SelectItemRecords.php*, czyli wyłącza automatyczne zatwierdzanie poleceń SQL. Dodatkowo obrazuje, że niezatwierdzone dane nie są widziane przez inne połączenia. Jest to tak zwana izolacja sesji. Rozwiązanie to zapewnia spójność danych widzianych przez poszczególnych użytkowników. Do zatwierdzenia wstawienia wiersza służy funkcja `oci_commit()`.

Program *InsertItemRecord.php* pokazuje efekt izolacji niezatwierdzonych danych między współbieżnymi sesjami.

```
-- Przedstawiony kod znajduje się w pliku InsertItemRecord.php, na dołączonej płycie CD-ROM.

<?php
  // Połącz się z bazą danych.
  if ($c = @oci_connect("php","php","xe"))
  {
    // Wstaw nowy wiersz z danymi.
    $s = oci_parse($c,"INSERT INTO item VALUES
                       ( item_s1.nextval
                       ,'9736-06125-4'
                       ,(SELECT   common_lookup_id
                         FROM     common_lookup
                         WHERE    common_lookup_type = 'DVD_WIDE_SCREEN')
                       ,'Przygody Indiany Jonesa'
                       ,''
                       ,'PG-13','21-LIS-03'
                       , 3, SYSDATE, 3, SYSDATE);");

    // Wykonaj polecenie bez jawnego zatwierdzania.
    oci_execute($s,OCI_DEFAULT);

    // Wykonaj zapytanie przed zatwierdzeniem i zatwierdź polecenie.
    query();
    oci_commit($c);
    print "Wstawiono i zatwierdzono [".oci_num_rows($s)."] wiersz(y).<br />";
    query();

    // Rozłącz się z bazą danych.
    oci_close($c);
  }
  else
  {
    // Wyświetl komunikat o błędzie.
    $errorMessage = oci_error();
    print htmlentities($errorMessage['message'])."<br />";
  }

  // Sprawdź wynik w innym połączeniu.
  function query()
  {
    // Otwórz połączenie.
    if ($nc = @oci_new_connect("php","php","xe"))
    {
      // Analizuj polecenie pobierania danych.
      $q = oci_parse($nc,"SELECT   item_id AS ID
                         ,         item_title AS TITLE
                         ,         item_release_date AS RELEASE_DATE
                         FROM      item
                         WHERE     REGEXP_LIKE(item_title,'Indiany Jonesa')");

      // Wykonaj polecenie.
      oci_execute($q);

      // Przypisz wynik.
      render_query($q);
```

```
      // Zamknij połączenie.
      oci_close($nc);
    }
    else
    {
      // Wyświetl komunikat o błędzie.
      $errorMessage = oci_error();
      print htmlentities($errorMessage['message'])."<br />";
    }
  }

  // Wyświetl wyniki.
  function render_query($rs)
  {
    // Zadeklaruj zmienną sterującą.
    $no_row_fetched = true;

    // Utwórz tabelę HTML.
    print '<table border="1" cellspacing="0" cellpadding="3">';

    // Odczytaj dane.
    while (oci_fetch($rs))
    {
      // Nagłówek wyświetl tylko raz.
      if ($no_row_fetched)
      {
        // Pobierz dane nagłówka.
        print '<tr>';
        for ($i = 1;$i <= oci_num_fields($rs);$i++)
          print '<td class="e">'.oci_field_name($rs,$i).'</td>';
        print '</tr>';

        // Wyłącz wyświetlanie nagłówka.
        $no_row_fetched = false;
      }

      // Wyświetl wszystkie dane z danego wiersza.
      print '<tr>';
      for ($i = 1;$i <= oci_num_fields($rs);$i++)
        print '<td class="v">'.oci_result($rs,$i).'</td>';
      print '</tr>';
    }

    // Zamknij tabelę HTML.
    print '</table>';
  }
?>
```

Polecenie `INSERT` **wstawia** wiersz do tabeli `ITEM` w momencie zakończenia wykonywania funkcji `oci_execute()`. Jak wcześniej wspomniano, funkcja działa w trybie bez zatwierdzania dzięki zastosowaniu opcji `OCI_DEFAULT`. **Informację o poprawnym wstawieniu wiersza danych zwraca funkcja** `oci_num_rows()`. Pierwsze wywołanie funkcji `query()` nie znajduje żadnych wyników, ponieważ dane nie zostały jeszcze zatwierdzone. Zatwierdzenie wstawienia funkcją `oci_commit()` powoduje **uwidocznienie danych innym sesjom**.

Przeglądarka internetowa wyświetli wynik zaprezentowany na poniższym rysunku.



Wyłączenie zatwierdzania kilku wstawień umożliwia w przyszłości zatwierdzenie wszystkich modyfikacji jako jednej całości. W takiej sytuacji, by dane zostały zapamiętane, **funkcję `oci_commit()` trzeba wykonać przed zamknięciem połączenia lub zakończeniem skryptu. W przeciwnym razie wszystkie zmiany zostaną wycofane**. Rozwiązanie alternatywne traktuje każde wywołanie `oci_execute()` jako autonomiczną transakcję, więc pomimo niepowodzenia niektórych operacji, pozostałe zostaną zapisane. Wybór rozwiązania zależy od tego, jak ważna jest integralność danych.

Rozważmy umieszczanie w bazie danych adresowych i numerów telefonów tej samej osoby za pomocą kilku autonomicznych transakcji. Jeśli uda się przeprowadzić zapis do tabel `ADDRESS` i `TELEPHONE`, ale nie do tabel `INDIVIDUAL` i `STREET_ADDRESS`, uzyskamy bezwartościowe wpisy w dwóch pierwszych tabelach. Tego rodzaju wiersze nazywa się **sierotami**, bo nie są one powiązane z żadnymi danymi nadrzędnymi. Autonomiczne transakcje wykorzystywane do zapisu kilku powiązanych ze sobą informacji rodzą pytanie: w jaki sposób poinformować użytkownika o takim połowicznym sukcesie?

Standardowy model „wszystko albo nic" ma wiele zalet, szczególnie w systemach aplikacji internetowych. Jeśli nie wszystkie wstawienia wierszy udało się wykonać, te które się powiodły, zostają wycofane, a użytkownik może rozpocząć edycję od początku (ewentualnie program może ponowić próbę wstawienia danych). Pamiętaj o tym, by zawsze wybierać rozwiązanie najlepiej pasujące do tworzonej aplikacji.


Wskazówka

Zapytanie w prezentowanym przykładzie wykorzystuje funkcję `REGEXP_LIKE()`, ponieważ nie można wykorzystywać standardowego operatora wieloznacznego `%` w środowisku zarządzanym przez OCI. Innymi słowy, nie można użyć operatora `LIKE`.


Uwaga

Polecenia SQL mogą zawierać dowolną liczbę przejść do nowego wiersza występujących w dowolnych miejscach. Ta elastyczność nie jest możliwa dla poleceń PL/SQL. Wynika to z odmiennej analizy składniowej obu systemów.

Rozwiązania dotyczące operacji UPDATE i DELETE są bardzo podobne do tych prezentowanych w programie *InsertItemRecord.php* dla polecenia INSERT. Następny program dotyczy zmiany hasła użytkownika.

Przykład zmiany hasła użytkownika

Zgodnie z wyjaśnieniem z tabeli 13.3 istnieją dwa tryby zmiany hasła wykorzystywanego przez użytkownika. Pierwsze wymaga istniejącego połączenia, a drugie automatycznie łączy się z bazą i zmienia hasło.

Program *ChangePassword.php* ilustruje łączenie się z bazą danych i zmianę hasła. Ten sposób zmiany jest znacznie bardziej prawdopodobny, jeśli stosujemy wirtualną, prywatną bazę danych Oracle. Tego rodzaju wirtualna baza danych dotyczy sytuacji, w której wielu użytkowników współdzieli ten sam schemat bazy; upraszcza to tworzenie aplikacji serwerowych.


Uwaga

Wirtualne, prywatne bazy danych to nowy element wprowadzony w Oracle Database 10*g* Release 2 i Oracle Database 10*g* XE.

Najważniejsze elementy programu znajdują się w funkcji change_password(). Dane dotyczące nazwy użytkownika, starego hasła oraz nowego hasła można przekazać jako parametry GET adresu URL. Program działa również bez podania tych parametrów, zmieniając hasło użytkownika plsql na oracle. Oto kod programu.

```
-- Przedstawiony kod znajduje się w pliku ChangePassword.php, na dołączonej płycie CD-ROM.

<?php
  // Ustawienie zmiennych wejściowych.
  (isset($_GET['uname'])) ? $u_name = $_GET['uname']
                          : $u_name = "plsql";
  (isset($_GET['opasswd'])) ? $o_passwd = $_GET['opasswd']
                            : $o_passwd = "plsql";
  (isset($_GET['npasswd'])) ? $n_passwd = $_GET['npasswd']
                            : $n_passwd = "oracle";

  // Zmiana hasła.
  change_password($u_name,$o_passwd,$n_passwd);

  function change_password($u_name,$o_passwd,$n_passwd)
  {
    // Połączenie z bazą danych.
    if ($c = @oci_connect($u_name,$o_passwd, "xe"))
    {
      print oci_server_version($c)."<br>";

      // Zmiana nazwy użytkownika.
      if (oci_password_change($c,$u_name,$o_passwd,$n_passwd))
        print get_params($u_name,$o_passwd,$n_passwd);
      else
        print "Hasło nie zostało zmienione.<br>";

      // Zamknij połączenie.
      oci_close($c);
```

```
    }
    else
    {
      // Poinformuj o błędzie.
      $errorMessage = oci_error();
      print htmlentities($errorMessage['message'])."<br />";
    }
  }

  // Zwróć tabelę HTML z nazwą użytkownika oraz starym i nowym hasłem.
  function get_params($u_name,$o_passwd,$n_passwd)
  {
    $out  = '<table border="1" cellpadding="0" cellspacing="0">';
    $out .= '<tr><td class="e" width="125">Użytkownik</td>';
    $out .= '<td class="v" width="125">'.$u_name.'</td></tr>';
    $out .= '<tr><td class="e">Stare hasło</td>';
    $out .= '<td class="v">'.$o_passwd.'</td></tr>';
    $out .= '<tr><td class="e">Nowe hasło</td>';
    $out .= '<td class="v">'.$n_passwd.'</td></tr>';
    $out .= '</table>';

    // Zwróć treść tabeli.
    return $out;
  }
?>
```

Program może wykorzystać adres URL bez parametrów. Podanie parametrów zmienia ustawienia domyślne. Decyzja o użyciu wartości domyślnej lub wartości przekazanej przez adres URL zostaje podjęta za pomocą operatora trójargumentowego. Domyślnie zmianie ulega hasło użytkownika `plsql`, które zmienia się z `plsql` na `oracle`.

Aby zmienić hasło innego użytkownika, na przykład użytkownika `php`, wystarczy użyć następującego adresu URL:

```
http://serwer/ChangePassword.php?uname=php&opasswd=php&npasswd=oracle
```

Po udanej zmianie hasła skrypt wyświetla tabelę HTML informującą o użytkowniku i nowych danych. By powrócić do poprzedniego hasła, wystarczy zamienić nowe i stare hasło miejscami w adresie URL.

Prezentowane do tej pory przykłady wykorzystywały statyczne polecenia SQL. Oracle obsługuje również dynamiczne polecenia SQL wykorzystujące zmiennie **dowiązywane**.

Zapytania i transakcje wykorzystujące instrukcje SQL

Statyczne polecenia SQL mają kilka ograniczeń. Trudno z wyprzedzeniem w programie zawrzeć wszystkie możliwe wartości, po których użytkownik będzie chciał wyszukiwać informacje. Z tego powodu warto zbierać dane wejściowe i dynamicznie wstawiać je do zapytań. Najlepszym sposobem osiągnięcia tego efektu jest mechanizm zastępowania.

Zmienne **dowiązywane** z Oracle działają mniej więcej tak jak zmienne tymczasowe. Umożliwiają przekazywanie danych skalarnych z i do poleceń SQL oraz PL/SQL. W programie PHP zmienne dowiązywane ułatwiają przekazywanie wartości ze zmiennych PHP do bazy danych Oracle i odwrotnie.

Niniejszy podrozdział zawiera kilka przykładów użycia zmiennych dowiązywanych w poleceniach SQL dotyczących pobierania danych i ich wstawiania. Zmienna dowiązana w Oracle przypomina zwykłą nazwę, ale jest dodatkowo poprzedzona znakiem dwukropka. Znak dwukropka przekazuje się również w funkcjach `oci_bind_by_name()` i `oci_bind_by_array_name()` wykorzystywanych do przeprowadzenia dowiązania.

Gdy statyczne polecenia SQL mają najczęściej trzy fazy: analiza, wykonanie i pobieranie, dynamiczne polecenia SQL mogą zawierać do pięciu faz: **analiza**, **dowiązanie**, **definiowanie**, **wykonanie** i **pobieranie**. Kursor referencyjny lub kolekcja wymaga **dowiązania** za pomocą fazy **definiowania**. Do **poleceń** SQL **przekazuje się** zmienne metodą przekazywania przez wartość. **Przekazywanie przez referencję** wymaga użycia procedur zapamiętanych języka PL/SQL, które są dokładniej opisane w rozdziale 14.

Programy wykorzystujące zapytania ilustrują, w jaki sposób użyć polecenia `SELECT` do zwrócenia danych wiersz po wierszu w kilku różnych trybach. Dodatkowo do ograniczenia liczby wyników programy stosują zmienne dowiązane.

Program *BindFetchSQL.php* ilustruje rozwiązanie, które zwraca wyniki wiersz po wierszu.

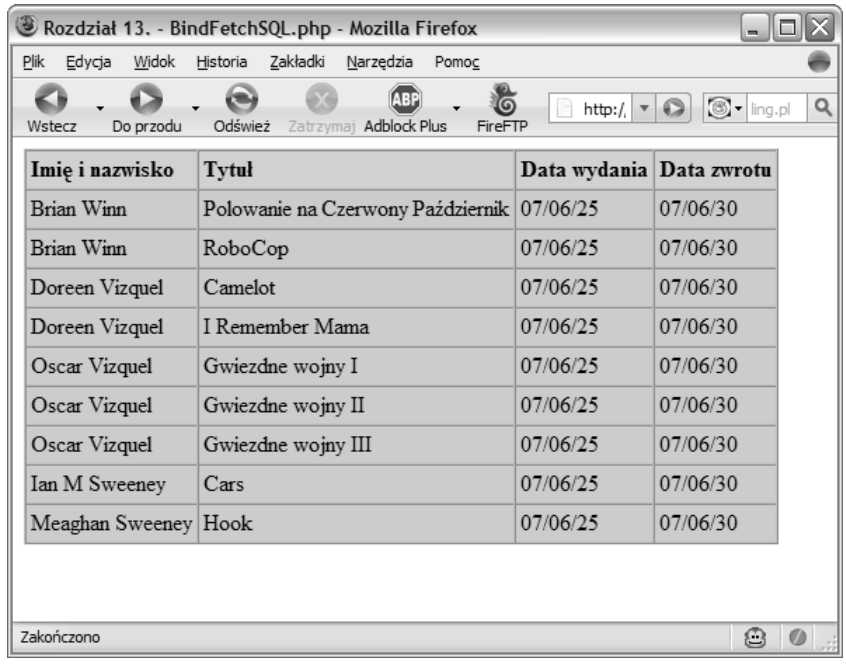
```
-- Przedstawiony kod znajduje się w pliku BindFetchSQL.php, na dołączonej płycie CD-ROM.

<?php
  // Tworzy połączenie z bazą danych.
  if ($c = @oci_connect("php","php","xe"))
  {
    // Deklaracja zmiennych wejściowych.
    (isset($_GET['lname'])) ? $lname = $_GET['lname']
                            : $lname = "[a-zA-Z]";

    // Deklaracja tablicy odwzorowującej rzeczywiste nazwy kolumn na przyjazne tytuły.
    $q_title = array("FULL_NAME"=>"Imię i nazwisko"
                    ,"TITLE"=>"Tytuł"
                    ,"CHECK_OUT_DATE"=>"Data wydania"
                    ,"RETURN_DATE"=>"Data zwrotu");

    // Przekazuje polecenie SQL do analizatora.
    $s = oci_parse($c,"SELECT   cr.full_name
                       ,        cr.title
                       ,        cr.check_out_date
                       ,        cr.return_date
                       FROM     current_rental cr
                       WHERE    REGEXP_LIKE(cr.full_name,:lname)");

    // Dowiązuje zmienne PHP do wartości z zapytania.
    oci_bind_by_name($s,":lname",$lname,-1,SQLT_CHR);

    // Wykonuje polecenie SQL bez automatycznego zatwierdzania.
    oci_execute($s,OCI_DEFAULT);
```

```
    // Rozpoczyna wyświetlanie tablicy.
    print '<table border="1" cellspacing="0" cellpadding="3">';

    // Wyświetla wiersz nagłówka.
    print '<tr>';
    for ($i = 1;$i <= oci_num_fields($s);$i++)
      print '<td class="e">'.$q_title[oci_field_name($s,$i)].'</td>';
    print "</tr>";

    // Odczytuje i wyświetla wiersze wyników.
    while (oci_fetch($s))
    {
      // Wyświetla pojedynczy wiersz wyników.
      print '<tr>';
      for ($i = 1;$i <= oci_num_fields($s);$i++)
        print '<td class="v">'.oci_result($s,$i).'</td>';
      print '</tr>';
    }

    // Zamknij tabelę.
    print '</table>';

    // Zamknij połączenie z bazą danych.
    oci_close($c);
  }
  else
  {
    // Wyświetla komunikat o błędzie.
    $errorMessage = oci_error();
    print htmlentities($errorMessage['message'])."<br />";
  }
?>
```

Program ilustruje kilka technik obsługi zbioru wyników. Zauważ, że jeśli w adresie URL nie pojawi się warunek ograniczający zbiór wyników, kod stosuje wyrażenie regularne znajdujące wszystkie tytuły rozpoczynające się lub zawierające w sobie literę. Polecenie SQL wykorzystuje funkcję `REGEXP_LIKE()` dostępną w Oracle. Tablica `$q_title` służy do odwzorowania nazw kolumn na nazwy bardziej przyjazne dla człowieka. Choć SQL*Plus obsługuje nazwy kolumn zawierające spacje po ich otoczeniu w cudzysłowach, to samo podejście nie jest możliwe w PHP. Przedstawiona sztuczka rozwiązuje problem.

Wewnątrz polecenia SQL znajduje się specjalna zmienna `:lname`. Ta sama nazwa pojawia się jako drugi argument wywołania funkcji `oci_bind_by_name()`. Wspomniana funkcja łączy wartość zmiennej w języku PHP z wartością używaną przez Oracle. Pozostałe argumenty funkcji określają rozmiar i typ zmiennej. Wartość –1 powoduje, że biblioteka OCI8 sama w sposób niejawny wyliczy rozmiar wartości. Stała `SQLT_CHR` oznacza, że przekazujemy wartość typu tekstowego o zmiennej długości (`VARCHAR2`). Więcej informacji na temat typów danych Oracle znajduje się w dodatku H. Bez podania parametru w adresie URL strona WWW renderuje się w sposób przedstawiony na rysunku na następnej stronie.

Pobieranie danych wiersz po wierszu nie jest trudny do zrozumienia. Co więcej, działa bardzo podobnie w wielu różnych systemach bazodanowych. Funkcja `oci_fetch_all()` pobiera wszystkie wiersze wyniku jako jedną całość i tworzy tablicę dwuwymiarową. Domyślnie pierwszy



Imię i nazwisko	Tytuł	Data wydania	Data zwrotu
Brian Winn	Polowanie na Czerwony Październik	07/06/25	07/06/30
Brian Winn	RoboCop	07/06/25	07/06/30
Doreen Vizquel	Camelot	07/06/25	07/06/30
Doreen Vizquel	I Remember Mama	07/06/25	07/06/30
Oscar Vizquel	Gwiezdne wojny I	07/06/25	07/06/30
Oscar Vizquel	Gwiezdne wojny II	07/06/25	07/06/30
Oscar Vizquel	Gwiezdne wojny III	07/06/25	07/06/30
Ian M Sweeney	Cars	07/06/25	07/06/30
Meaghan Sweeney	Hook	07/06/25	07/06/30

wymiar stanowią wiersze, a drugi wymiar to tablice asocjacyjne z nazwami kolumn i ich wartościami. Poszczególne wiersze są numerowane w ten sam sposób jak w zwykłej tablicy numerycznej, czyli pierwszy wiersz znajduje się w elemencie o indeksie 0.

Modyfikując domyślny tryb zwracania wyników, można wyłączyć zwracanie nazwy kolumn, by zamiast tego rozpocząć zwracanie numeru kolumny. Wyborem rozwiązania sterują wartości `OCI_NUM` i `OCI_ASSOC`. By uzyskać dwuwymiarową tablicę odpowiadającą wynikowi zwracanemu przez typowy kursor SQL, wystarczy użyć kombinacji `OCI_FETCHSTATEMENT_BY_ROW` i `OCI_NUM`.

Program *BindFetchAllSQL.php* umożliwia sprawdzenie wszystkich dostępnych kombinacji ustawień funkcji `oci_fetch_all()`.

```
-- Przedstawiony kod znajduje się w pliku BindFetchAllSQL.php, na dołączonej płycie CD-ROM.

<?php
  // Tworzy połączenie z bazą danych.
  if ($c = @oci_connect("php", "php", "xe"))
  {
    // Deklaracja zmiennych wejściowych.
    (isset($_GET['limit']))   ? $limit = (int) $_GET['limit']
                              : $limit = 5;
    (isset($_GET['param']))   ? $flag = (int) $_GET['param']
                              : $flag = 0;

    // Przekazuje polecenie SQL do analizatora.
    $s = oci_parse($c,"SELECT   cr.full_name
                       ,        cr.title
                       FROM     current_rental cr
                       ,       (SELECT   cr.account_number
```

```
                        FROM     current_rental cr
                        GROUP BY cr.account_number
                        HAVING   COUNT(account_number) >= :limit) lim
                WHERE   cr.account_number = lim.account_number");

// Dowiązuje zmienne PHP do wartości z zapytania.
oci_bind_by_name($s,":limit",$limit);

// Wykonuje polecenie SQL bez automatycznego zatwierdzania.
oci_execute($s,OCI_DEFAULT);

// Deklaruje zmienne sterujące formatowaniem tabeli HTML.
$dimensions = array("Kolumna","Wiersz");
$dimension1 = "";
$dimension2 = "";
$index = 0;

// Rozpoczyna wyświetlanie tablicy.
print '<table border="1" cellspacing="0" cellpadding="3">';

// Dynamicznie zmienia opcje pobierania danych.
switch($flag)
{
  case 0:
    // Pierwsza wartość to nazwa kolumny, a druga to numer wiersza.
    oci_fetch_all($s,$array_out,0,-1,OCI_FETCHSTATEMENT_BY_COLUMN|OCI_ASSOC);
    $dimension1 = $dimensions[0];
    $dimension2 = $dimensions[1];
    break;
  case 1:
    // Pierwsza wartość to numer kolumny, a druga to numer wiersza.
    oci_fetch_all($s,$array_out,0,-1,OCI_FETCHSTATEMENT_BY_COLUMN|OCI_NUM);
    $dimension1 = $dimensions[0];
    $dimension2 = $dimensions[1];
    break;
  case 2:
    // Pierwsza wartość to numer wiersza, a druga to nazwa kolumny.
    oci_fetch_all($s,$array_out,0,-1,OCI_FETCHSTATEMENT_BY_ROW|OCI_ASSOC);
    $dimension1 = $dimensions[1];
    $dimension2 = $dimensions[0];
    break;
  case 3:
    // Pierwsza wartość to numer wiersza, a druga to numer kolumny.
    oci_fetch_all($s,$array_out,0,-1,OCI_FETCHSTATEMENT_BY_ROW|OCI_NUM);
    $dimension1 = $dimensions[1];
    $dimension2 = $dimensions[0];
    break;
  default:
    // Pierwsza wartość to nazwa kolumny, a druga to numer wiersza.
    oci_fetch_all($s,$array_out,0,-1);
    $dimension1 = $dimensions[0];
    $dimension2 = $dimensions[1];
    break;
}

// Wyświetl nagłówek pierwszego wymiaru.
print '<tr>';
print '<td class="e">'.$dimension1.'<br />Indeks</td>';
```

```
    // Ustaw początkowy indeks.
    foreach ($array_out as $name => $value)
    {
      if (!is_numeric($name)) $index = $name;
      break;
    }

    // Wyświetl zagnieżdżone nagłówki tablicy.
    for ($i = 0;$i < count($array_out[$index]);$i++)
    {
      print '<td class="e">'.$dimension2.'<br />Indeks</td>';
      print '<td class="e">'.$dimension2.'<br />Wartość</td>';
    }

    // Zamknij wiersz nagłówka.
    print '</tr>';

    // Przejdź przez pierwszy wymiar wynikowych danych.
    foreach ($array_out as $name => $value)
    {
      print '<tr>';
      print '<td class="e">'.$name.'</td>';

      // Przejdź przez drugi wymiar wynikowych danych.
      foreach ($value as $subname => $subvalue)
      {
        print '<td class="e">'.$subname.'</td>';
        print '<td class="v">'.$subvalue.'</td>';
      }
      print '</tr>';
    }

    // Zamknij tabelę.
    print '</table>';

    // Zamknij połączenie z bazą danych.
    oci_close($c);
  }
  else
  {
    // Wyświetla komunikat o błędzie.
    $errorMessage = oci_error();
    print htmlentities($errorMessage['message'])."<br />";
  }
?>
```

Program przyjmuje parametr opcjonalny param w adresie URL. Przekazywanie różnych wartości parametru umożliwia testowanie różnych trybów działania funkcji `oci_fetch_all()`. Lokalna zmienna `$flag` przechowuje pobraną wartość parametru lub wartość domyślną, jeśli go nie przekazano. Zauważ, że domyślne ustawienia powodują zwrócenie takiego samego wyniku jak opcje `OCI_FETCHSTATEMENT_BY_COLUMN` i `OCI_ASSOC`.

Aby włączyć określony tryb działania funkcji `oci_fetch_all()`, użyj adresu URL podobnego do poniższego.

```
http://serwer/BindFetchAllSQL.php?param=2
```

Ten tryb spowoduje wyświetlenie wyników w postaci przypominającej tej dla pobierania tylko jednego wiersza, ale tym razem zwróci cały zestaw danych. Wynik działania programu prezentuje następujący rysunek:



Wiersz Indeks	Kolumna Indeks	Kolumna Wartość	Kolumna Indeks	Kolumna Wartość
0	FULL_NAME	Doreen Vizquel	TITLE	Camelot
1	FULL_NAME	Doreen Vizquel	TITLE	I Remember Mama
2	FULL_NAME	Oscar Vizquel	TITLE	Gwiezdne wojny I
3	FULL_NAME	Oscar Vizquel	TITLE	Gwiezdne wojny II
4	FULL_NAME	Oscar Vizquel	TITLE	Gwiezdne wojny III

Funkcja `oci_fetch_array()` zawiera jeden interesujący szczegół dotyczący zwracania danych pojedynczego wiersza. Podtablica zawierająca wiersz może być indeksowana na podstawie **numerów kolumn**, **nazw kolumn** lub **obu tych elementów naraz**. Dodatkowo domyślne zachowanie funkcji nie zwraca wartości `null` ani obiektów typu LOB. Aby zmienić to domyślne zachowanie, należy dołączyć do pozostałych opcji dwie dodatkowe, `OCI_RETURN_NULLS` i `OCI_RETURN_LOBS`, wykorzystując operator sumy binarnej `|`. Przykładowy program wykorzystujący tę opcję znajduje się na dołączonej płycie CD-ROM, w pliku *BindFetchArraySQL.php*.

Funkcja `oci_fetch_object()` działa podobnie do funkcji `oci_fetch`, ale zwraca całą zawartość wiersza jako obiekt PHP ze składowymi. W rzeczywistości jest to obiekt OCI8, który nie ma żadnych funkcji, stałych i składowych statycznych. Po obiekcie można poruszać się w ten sam sposób jak po tablicy asocjacyjnej, ale dostęp do poszczególnych elementów odbywa się w sposób obiektowy.


Uwaga

Choć Oracle 9*i*R2 wprowadził obiekty do języka PL/SQL, prezentowana funkcja PHP ich nie obsługuje. W obecnej wersji tworzy obiekty tylko i wyłącznie na postawie wyników zapytań SQL.

Program *BindFetchObjectSQL.php* ilustruje, w jaki sposób odczytać pierwszy wiersz wyników zapytania SQL.

```
-- Przedstawiony kod znajduje się w pliku BindFetchObjectSQL.php, na dołączonej płycie CD-ROM.

<?php
  // Tworzy połączenie z bazą danych.
  if ($c = @oci_connect("php", "php", "xe"))
  {
    // Deklaracja zmiennych wejściowych.
    (isset($_GET['limit']))   ? $limit = (int) $_GET['limit']
                              : $limit = 5;

    // Przekazuje polecenie SQL do analizatora.
    $s = oci_parse($c,"SELECT   cr.full_name
                       ,        cr.title
```

```
                        ,       cr.product
                        ,       cr.check_out_date
                        ,       cr.return_date
                        FROM    current_rental cr
                        ,      (SELECT   cr.account_number
                                FROM     current_rental cr
                                GROUP BY cr.account_number
                                HAVING   COUNT(account_number) >= :limit) l
                        WHERE   cr.account_number = l.account_number");

    // Dowiązuje zmienne PHP do wartości z zapytania.
    oci_bind_by_name($s,":limit",$limit);

    // Wykonuje polecenie SQL bez automatycznego zatwierdzania.
    oci_execute($s,OCI_DEFAULT);

    // Rozpoczyna wyświetlanie tablicy.
    print '<table border="1" cellspacing="0" cellpadding="3">';

    // Pobierz dane jednego wiersza.
    $object_out = oci_fetch_object($s);

    // Przejdź przez wartości wiersza.
    for ($i = 1;$i <= oci_num_fields($s);$i++)
    {
      $name = oci_field_name($s,$i);
      print '<tr>';
      print '<td class="v">'.$name.'</td>';
      if (oci_field_is_null($s,$i))
        print '<td class="v"> </td>';
      else
        print '<td class="v">'.$object_out->$name.'</td>';
      print '</tr>';
    }

    // Zamknij tabelę.
    print '</table>';

    // Zamknij połączenie z bazą danych.
    oci_close($c);
  }
  else
  {
    // Wyświetla komunikat o błędzie.
    $errorMessage = oci_error();
    print htmlentities($errorMessage['message'])."<br />";
  }
?>
```

Program używa funkcji `oci_fetch_object()` do wczytania pierwszego wiersza wyników i zwrócenia go jako obiektu. Następnie używa połączenia funkcji `oci_num_fields()` i `oci_field_name()` do przejścia przez wszystkie składowe i pobrania ich nazw. Nazwy składowych są takie same jak nazwy kolumn zwróconego polecenia SQL. Używając składni zaprezentowanej w rozdziale 8., możemy dynamicznie odwołać się do składowej obiektu za pomocą **operatora wskaźnika**.

```
$obiekt->$nazwa
```

Funkcja `oci_fetch_object()` nie oferuje żadnych dodatkowych opcji w porównaniu z pozostałymi metodami pobierania wiersza danych. Być może funkcja stanie się bardziej popularna po dodaniu możliwości zwracania przez bibliotekę OCI8 obiektów Oracle. Znacząco ułatwiłoby to przekazywanie obiektów między PHP i bazą Oracle.

Jak wcześniej wspomniano, użycie zmiennych dowiązanych nie jest ograniczone do zapytań `SELECT` — można go również używać w poleceniach `INSERT`, `UPDATE` i `DELETE`. Mechanizm przypisywania wartości pozostaje ten sam. Program *InsertTransaction.php* ilustruje działanie zmiennych dowiązanych przy wstawianiu nowego wiersza danych do tabeli.

-- Przedstawiony kod znajduje się w pliku InsertTransaction.php, na dołączonej płycie CD-ROM.

```
<?php
  // Tworzy połączenie z bazą danych.
  if ($c = @oci_connect("php","php","xe"))
  {
    // Deklaruje zmienne z danymi.
    $account_number = "B303-73740";
    $credit_card_no = "5555-5555-5555-5555";
    $credit_card_type = "DISCOVER_CARD";

    // Przekazuje polecenie SQL do analizatora.
    $s = oci_parse($c,"INSERT INTO member VALUES
                       ( member_s1.nextval
                       , :account_no
                       , :credit_card_no
                       ,(SELECT   common_lookup_id
                         FROM     common_lookup
                         WHERE    common_lookup_context = 'MEMBER'
                         AND      common_lookup_type = :credit_card_type)
                       , 3, SYSDATE, 3, SYSDATE )");

    // Dowiązuje zmienne PHP do wartości z zapytania.
    oci_bind_by_name($s,":account_no",$account_number,-1,SQLT_CHR);
    oci_bind_by_name($s,":credit_card_no",$credit_card_no,-1,SQLT_CHR);
    oci_bind_by_name($s,":credit_card_type",$credit_card_type,-1,SQLT_CHR);

    // Wykonuje polecenie SQL bez automatycznego zatwierdzania.
    oci_execute($s,OCI_DEFAULT);

    // Zatwierdza transakcję.
    oci_commit($c);

    // Zamknij połączenie z bazą danych.
    oci_close($c);
  }
  else
  {
    // Wyświetla komunikat o błędzie.
    $errorMessage = oci_error();
    print htmlentities($errorMessage['message'])."<br />";
  }
?>
```

Zauważ, że zmienne dowiązane można umieszczać w dowolnym miejscu polecenia SQL. Polecenie `INSERT` używa trzech zmiennych dowiązanych: dwóch jako dane wstawianego wiersza i trzecią w klauzuli `WHERE` podzapytania SQL. Zgodnie z wcześniejszymi wyjaśnieniami, jeśli funkcja `oci_execute()` nie zatwierdza automatycznie wykonanego polecenia, musimy wstawienie zatwierdzić ręcznie, wywołując funkcję `oci_commit()`. Usunięcie wywołania tej funkcji nie spowoduje dodania nowego wiersza do tabeli `MEMBER`.

Wydajność rozwiązania, które za jednym podejściem pobiera cały zbiór danych, jest większa niż rozwiązania, które pobiera dane wiersz po wierszu. Do obu rozwiązań powrócimy jeszcze w następnym rozdziale.

Podsumowanie

W niniejszym rozdziale przedstawione zostały techniki łączenia się z bazą danych Oracle Database 10*g* XE, tworzenie zapytań SQL i pobieranie wyników. Użyte polecenia SQL korzystały z informacji statycznych oraz dynamicznych. Na końcu zaprezentowano kilka technik pobierania wyników zapytania.