PEAR. Programowanie w PHP

Autorzy: Stephan Schmidt, Stoyan Stefanov, Carsten Lucke, Aaron Wormus
Tłumaczenie: Sławomir Dzieniszewski
ISBN: 978-83-246-0897-3
Tytuł oryginału: PHP Programming with PEAR
Format: B5, stron: 304

Przewodnik po najbardziej przydatnych pakietach PEAR

- Poznaj najpopularniejsze pakiety repozytorium PEAR
- Korzystaj z baz danych za pomocą MDB2
- Używaj gotowych komponentów do szybkiego tworzenia aplikacji w PHP

Jednym z głównych powodów popularności PHP jest szeroki dostęp do bibliotek i rozszerzeń tego języka. Najważniejszym ich źródłem jest PEAR — internetowe repozytorium komponentów i aplikacji języka PHP. Pakiety dostępne w PEAR zawierają gotowe rozszerzenia umożliwiające wykonanie niemal wszystkich standardowych operacji w PHP. Rozszerzenia te przechodzą przez ścisły system kontroli jakości, a ich autorzy muszą stosować się do określonych zaleceń. Dlatego pisanie programów z wykorzystaniem pakietów jest nie tylko szybsze, ale prowadzi też do powstawania lepszych i bardziej spójnych aplikacji.

Dzięki książce „PEAR. Programowanie w PHP" nauczysz się wykonywać codzienne zadania programistyczne przy użyciu klas z popularnych pakietów PEAR. Dowiesz się, jak obsługiwać bazy danych za pomocą pakietu MDB2, a także jak wyświetlać dane zapisane w różnych formatach. Poznasz sposoby tworzenia i analizowania dokumentów XML oraz przekształcania obiektów PHP na format XML i z powrotem. Zobaczysz, jak tworzyć własne usługi WWW oraz używać interfejsów udostępnianych w usługach autorstwa innych producentów.

- Praca z bazami danych
- Wyświetlanie informacji w różnych formatach
- Tworzenie i przetwarzanie plików XML
- Przygotowywanie i udostępnianie usług WWW
- Korzystanie z gotowych usług WWW
- Praca z datami

Zwiększ swoją produktywność, korzystając z gotowych komponentów

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

Spis treści

1

ROZDZIAŁ

MDB2

W ciągu ostatnich dziesięciu lat sieć WWW bardzo się rozrosła, jak również dojrzała i sprofesjonalizowała, w związku z czym pojawiło się zapotrzebowanie na coraz bardziej złożone i dynamiczne witryny WWW. Dawniej zupełnie wystarczało przechowywanie informacji w pliku tekstowym lub prostej bazie danych, obecnie jednak każdy programista piszący profesjonalną aplikację WWW musi posiadać rzetelną wiedzę na temat tego, jak komunikować się z profesjonalnymi relacyjnymi bazami danych.

Począwszy od najwcześniejszych wersji język PHP zawsze służył programistom solidnym wsparciem w kontaktach z bazami danych. Niemniej do czasu wprowadzenia rozszerzenia PDO (PHP Data Objects, obiekty danych PHP) nie istniał żaden standardowy sposób korzystania z najróżniejszych sterowników baz danych dodawanych do języka PHP. Brak ujednoliconego interfejsu API był oczywiście inspiracją dla kilku projektów mających na celu stworzenie jakiejś warstwy DBAL (Database Abstraction Layer), która oferowałaby uniwersalny poziom abstrakcji dla wszystkich baz danych. Głównym celem tych wysiłków był zamiar ułatwienia życia programistom, tak aby mogli oni pisać kod komunikujący się z bazą danych, który będzie niezależny od sytemu bazy danych wykorzystywanego przez aplikację. Dzięki temu klienci lub użytkownicy mogliby używać aplikacji w połączeniu z tym systemem zarządzania bazami danych, który im najbardziej odpowiada.

Trzy najważniejsze z rozpoczętych w tamtych latach prób stworzenia warstwy abstrakcji dla baz danych to: AdoDB, PEAR::DB i Metabase. W ostatnich latach kolejnym bardzo mocnym kandydatem na uniwersalną warstwę abstrakcji dla baz danych był pakiet PEAR::MDB. Niniejszy rozdział poświęcony będzie kolejnej inkarnacji MDB, a mianowicie MDB2.

Krótka historia MDB2

Wszystko zaczęło się, kiedy Lukas Smith, programista PEAR, opublikował kilka lat do istniejącej już warstwy abstrakcji bazy danych, Metabase. W którymś momencie między Lukasem a autorem Metabase wywiązała się dyskusja na temat, jakby opublikować Metabase w repozytorium PEAR jako nowy pakiet. Celem nowego pakietu byłoby połączenia funkcjonalności oferowanych przez Metabase z interfejsem API istniejącego już i bardzo popularnego pakietu PEAR::DB. Dzięki temu programiści otrzymaliby bogatą w funkcje i znakomicie spisującą się bibliotekę abstrakcji bazy danych, co byłoby z wielką korzyścią dla infrastruktury PEAR. W taki właśnie sposób narodził się przodek pakietu MDB2, czyli pakiet PEAR::MDB.

Po kilku latach pracy nad pakietem PEAR::MDB, dla autorów stało się oczywiste, że decyzja utrzymywania interfejsu API pakietu w takiej formie, aby był jak najbardziej zbliżony do interfejsów API Metabase i PEAR::DB, nieuchronnie stwarza pewne problemy, które utrudniają przekształcenie MDB w pełni profesjonalną warstwę abstrakcji bazy danych (DBAL). Ponieważ pakiet PEAR::MDB osiągnął już w repozytorium PEAR stabilną formę w pełni dojrzałego pakietu oprogramowania, niemożliwe było usunięcie pewnych wad bez rezygnacji z kompatybilności z tym starszym modułem, czego właśnie autorzy starali się za wszelką cenę uniknąć. Rozwiązaniem było wykorzystanie doświadczeń zdobytych podczas prac nad Metabase i MDB oraz zastosowanie ich w nowym pakiecie, który zawierać będzie profesjonalnie zaprojektowany i w pełni nowoczesny interfejs API. Nowy pakiet otrzymał nazwę MDB2.

Warstwy abstrakcji

Zanim przejdziemy do szczegółowego omawiania, w jaki sposób pakiet MDB2 radzi sobie z tworzeniem abstrakcji dla bazy danych, powinniśmy najpierw zapoznać się przynajmniej pobieżnie z teorią tworzenia warstw abstrakcji obudowujących systemy baz danych, by zrozumieć dokładnie, w jaki sposób się to robi. Na tworzenie warstw abstrakcji dla baz danych można spojrzeć z kilku perspektyw. Omówimy je teraz dokładniej po kolei i opowiemy, jakie mają wymagania.

Warstwa abstrakcji dla interfejsu bazy danych

Najważniejszym etapem w tworzeniu abstrakcji systemu bazy danych jest przygotowanie odpowiedniej abstrakcji dla interfejsu bazy danych. Dzięki temu programista będzie mógł sięgać do baz danych zarządzanych przez różne systemy, używając tych samych metod. Oznacza to, że zarówno tworzenie połączenia z bazą danych, wysyłanie zapytania, jak i pobieranie danych zawsze przebiegać będzie identycznie, niezależnie od tego, z jaką bazą danych będziemy współpracować.

Warstwa abstrakcji dla kodu SQL

Większość obecnie używanych systemów baz danych korzysta ze standardowego zestawu podstawowych instrukcji SQL, dlatego też większość kodu SQL pisanego przez programistów powinna działać zawsze, niezależnie od tego, z jakiej bazy danych będą korzystać. Niemniej wiele z systemów baz danych wprowadza własne, specyficzne tylko dla danego systemu instrukcje SQL i pomocnicze funkcje, dlatego też może się zdarzyć, że kod SQL napisany specjalnie dla jednej bazy danych nie będzie działał w innej. W miarę jak system zarządzania bazą danych (ang. *Relational Database Management System* — RDBMS) rozwija się, czasami implementuje funkcje, które nie są kompatybilne ze starszymi wersjami tej samej bazy danych. Dlatego też dla programisty pragnącego napisać kod, który byłby zgodny ze wszystkimi wersjami bazy danych (lub który mógłby współdziałać z kilkoma różnymi systemami baz danych), jedynym rozwiązaniem jest ograniczenie się tylko do tych instrukcji kodu SQL, o których wiadomo, że na pewno będą działać na wszystkich platformach baz danych. Lepszą opcją jest jednak skorzystanie ze specjalnej warstwy abstrakcji baz danych, która w razie potrzeby emuluje odpowiednie funkcje, jeśli nie będą dostępne na danej platformie.

Mimo iż nie jest wykonalne obudowanie każdej możliwej funkcji SQL, pakiet MDB2 obsługuje bardzo wiele powszechnie wykorzystywanych funkcji SQL. Funkcje te to między innymi obsługa zapytań LIMIT, subselektów (podzapytań select) i zapytań preparowanych. Korzystanie z mechanizmu abstrakcji kodu SQL, oferowanego przez MDB2, daje nam gwarancję, że będziemy mogli korzystać z tych zaawansowanych funkcji — nawet wtedy, gdy baza danych, z której korzystamy, samoistnie ich nie obsługuje. W dalszej części tego rozdziału opowiemy o różnych funkcjach oferujących abstrakcję dla zaawansowanych narzędzi SQL, które zapewnia pakiet MDB2.

Warstwa abstrakcji dla typów danych

Na koniec wreszcie konieczne jest przygotowanie abstrakcji dla typów danych stosowanych przez bazy. Wynika to z faktu, że różne systemy baz danych często obsługują typy danych w zupełnie inny sposób.

Uwarunkowania związane z prędkością

Zapewne czytelnikom cieknie już ślinka, by skosztować tych wspaniałych funkcji wkomponowanych w pakiet MDB2, niemniej najpierw należy powiedzieć kilka słów o zagadnieniach związanych z prędkością i wydajnością. Warto wiedzieć, że gdy korzystamy z warstwy abstrakcji bazy danych, często za bogactwo funkcji oferowanych przez pakiet dokonujący abstrakcji musimy zapłacić mniejszą wydajnością i szybkością działania bazy danych. Nie jest to

tylko ułomność pakietu MDB2 ani też warstw abstrakcji bazy danych, ale w ogóle wszelkiego rodzaju warstw abstrakcji i systemów wirtualizacji.

Na szczęście, inaczej niż w przypadku VMWare lub Miscrosoft Virtual PC, które dokonują abstrakcji każdego wykonywanego wywołania systemowego, pakiet MDB2 oferuje abstrakcję tylko wtedy, gdy dana funkcja nie jest dostępna w określonym systemie baz danych. Oznacza to, że wydajność zależeć będzie od platformy, na której skorzystamy z MDB2. Jeśli szczególnie zależy nam na szybkości i wydajności, to należy skorzystać z pamięci podręcznej dla kodów operacji (ang. *opcode cache*) lub włączyć mechanizm przechowywania zapytań w pamięci podręcznej w systemie baz danych, którego używamy. Dzięki wykorzystaniu wspomnianych możliwości języka PHP lub systemu baz danych będziemy mogli w znacznym stopniu ograniczyć negatywne efekty spowolnienia działania bazy danych, nieodłącznie związane z użyciem warstwy abstrakcji.

Konstrukcja pakietu MDB2

Interfejs API pakietu MDB2 został zaprojektowany w taki sposób, aby gwarantować maksimum wszechstronności i elastyczności. Poszczególnym obsługiwanym systemom baz danych i określonym zaawansowanym funkcjom przypisano określone moduły. Każdy z licznych *sterowników* (ang. *drivers*) dla baz danych jest osobnym i niezależnie rozwijanym modułem PEAR. Oznacza to, że każdy pakiet sterownika funkcjonuje niezależnie, a kolejne wersje i wersje stabilne publikowane są we własnych, niezależnych od innych sterowników cyklach. Dzięki temu programiści odpowiedzialni za przygotowywanie poszczególnych sterowników mogą je wypuszczać, ilekroć zachodzi taka potrzeba, bez konieczności czekania na publikację kolejnej wersji głównego pakietu MDB2. Pakiet MDB2 może zatem zachowywać stabilność, niezależnie od stanu prac nad pakietami obsługującymi poszczególne sterowniki. W efekcie zdarza się czasem, że stabilna wersja pakietu oferuje sterowniki dla niektórych systemów baz danych jedynie w wersji beta. Ponadto w chwili wypuszczenia nowego sterownika dla bazy danych oznaczany jest on jako wersja alfa i pakiet podlega procedurze sprawdzania zgodnie ze standardami repozytorium PEAR.

Drugi rodzaj modułów wbudowanych w pakiet MDB2 to moduły dodające specjalne, rozszerzone funkcje oferowane przez pakiet MDB2. Zamiast dołączać te funkcje do głównego pakietu MDB2 lub dodawać do niego nową klasę implementującą te funkcje, programista ma możliwość utworzenia nowej klasy w osobnym module i następnie załadowanie jej do pakietu MDB2 za pomocą metody `loadModule()`. Gdy już nowy moduł zostanie załadowany do pakietu MDB2, do jego metod będzie można sięgać w taki sam sposób, jakby były metodami wbudowanymi w pakiet MDB2. Pakiet MDB2 stosuje to rozwiązanie, aby jego wewnętrzne pakiety działały tak szybko, jak to tylko możliwe, a jednocześnie by pozostawić użytkownikom swobodę dołączania do pakietu MDB2 swoich własnych klas. Szczegółowe informacje o tym, jak we własnym zakresie rozwijać pakiet MDB2, można znaleźć w dalszej części tego rozdziału.

Zaczynamy pracę z MDB2

Poniżej omówimy podstawowe kroki, które trzeba wykonać, by zainstalować pakiet MDB2, utworzyć obiekt MDB2 oraz skonfigurować kilka opcji definiujących tryb pobierania danych. Na koniec powiemy, jak rozłączać się z bazą danych.

Instalowanie MDB2

Podczas instalowania pakietu MDB2 należy pamiętać, że nie zawiera on żadnych sterowników baz danych, dlatego trzeba je będzie zainstalować później osobno. Pakiet MDB2 jest rozprowadzany w wersji stabilnej, niemniej, jak już wspomnieliśmy, niektóre z wchodzących w jego skład modułów sterowników i rozszerzeń mogą być rozwijane w niezależnych cyklach, dlatego niektóre z wykorzystywanych przez nas modułów mogą być dopiero w wersji beta, alfa lub nawet jeszcze w fazie programowania. Należy o tym pamiętać podczas instalowania pakietów zawierających sterowniki poszczególnych baz danych.

Najprościej jest zainstalować MDB2 korzystając z programu instalacyjnego repozytorium PEAR:

```
> pear install MDB2
```

To polecenie zainstaluje klasy tworzące rdzeń MDB2, natomiast nie zainstaluje żadnego z dostępnych sterowników baz danych. Aby zainstalować sterownik właściwy dla systemu baz danych, którego używamy, należy skorzystać z polecenia:

```
> pear install MDB2_Driver_mysql
```

To akurat polecenie zainstaluje sterownik dla bazy MySQL. Aby zainstalować sterownik dla bazy SQLite, należy wpisać:

```
> pear install MDB2_Driver_sqlite
```

Oto pełna lista dostępnych aktualnie sterowników:

- `fbsql` — Front Base
- `ibase` — InterBase
- `mssql` — MS SQL Server
- `mysql` — MySQL
- `mysqli` — system MySQL korzystający z rozszerzenia mysqli PHP; więcej informacji pod adresem: *http://php.net/mysqli*
- `oci8` — Oracle
- `pgsql` — PostgreSQL
- `querysim` — Querysim
- `sqlite` — SQLite

Łączenie się z bazą danych

Aby połączyć się z wybraną bazą danych już po udanym zainstalowaniu pakietu MDB2 i modułu sterownika, konieczne będzie najpierw określenie nazwy źródła danych, DSN (Data Source Name). Nazwa DSN może mieć postać łańcucha lub tablicy i definiuje parametry połączenia z bazą danych takie jak: nazwa bazy danych, typ systemu RDBMS (systemu zarządzającego relacyjną bazą danych), nazwa użytkownika i hasło wykorzystywane do łączenia się z bazą danych itp.

Nazwa DSN jako tablica

Jeśli nazwa źródła danych, DSN, jest definiowana w formie tablicy, będzie wyglądać mniej więcej tak:

```
$dsn = array ( 'phptype' => 'mysql',
               'hostspec' => 'localhost:3306',
               'username' => 'user',
               'password' => 'pass',
               'database' => 'mdb2test'
             );
```

Oto lista różnych kluczy parametrów używanych w tablicy nazwy DSN:

- `phptype` — nazwa wykorzystywanego sterownika; innymi słowy: nazwa definiująca system RDBMS
- `hostspec` — (specyfikacja hosta) określa nazwę hosta, na którym działa baza danych; może przyjmować postać `host:port` lub też podawać tylko samą nazwę hosta, a port będzie wtedy definiowany osobno w kluczu `port`
- `database` — nazwa bazy danych, z którą się łączymy
- `dbsyntax` — jeśli używana składnia jest inna niż właściwa dla systemu `phptype`
- `protocol` — wykorzystywany protokół komunikacyjny, na przykład TCP
- `socket` — gniazdo, które należy określić, jeśli łączymy się za pośrednictwem gniazd
- `mode` — służy do definiowania trybu otwierania pliku bazy danych

Nazwa DSN jako łańcuch

Szybszym i bardziej przyjaznym dla człowieka sposobem (gdy już się do niego przyzwyczaimy) jest definiowanie nazw DSN za pomocą łańcucha tekstowego wyglądającego podobnie do adresu URL. Zasadniczo, jego składnia wygląda tak:

```
phptype://nazwa-użytkownika:hasło@specyfikacja-hosta/baza-danych
```

gdzie `phptype` to oczywiście typ systemu baz danych. Dla systemu MySQL łańcuch nazwy DSN może wyglądać na przykład tak:

```
$dsn = 'mysql://user:pass@localhost:3306/mdb2test';
```

Więcej informacji na temat nazw DSN oraz inne przykłady prawidłowych łańcuchów DSN można znaleźć w podręczniku repozytorium PEAR, dostępnym pod adresem: *http://pear.php.net/➥manual/en/package.database.mdb2.intro-dsn.php*.

Tworzenie instancji obiektu MDB2

Istnieją trzy metody umożliwiające tworzenie (instancjację) obiektu MDB2:

```
$mdb2 =& MDB2::connect($dsn);
$mdb2 =& MDB2::factory($dsn);
$mdb2 =& MDB2::singleton($dsn);
```

Metoda `connect()` tworzy obiekt i łączy się z bazą danych. Metoda `factory()` tworzy obiekt, natomiast połączenie utworzy dopiero, gdy będzie ono potrzebne. Wreszcie metoda `singleton()` działa podobnie jak metoda `factory()`, ale upewnia się, że istnieje tylko jeden obiekt MDB2 o danej nazwie źródła danych, DSN. Jeśli więc taki obiekt już istnieje, metoda zwraca ten obiekt, a jeśli nie, tworzy nowy.

Istnieje też sposób „zakłócania" działania metody `singleton()` za pomocą metody `setDatabase()`, która pozwala określić, że bieżąca baza danych ma być inna niż ta określona w nazwie DSN.

```
$dsn = 'mysql://root@localhost/mdb2test';
$mdb2_first =& MDB2::singleton($dsn);
$mdb2_first->setDatabase('inna_db');
$mdb2_second =& MDB2::singleton($dsn);
```

W tym przypadku będziemy mieli dwie różne instancje MDB2.

Wszystkie trzy wspomniane metody tworzą obiekt klasy sterownika bazy danych. Jeśli na przykład korzystamy ze sterownika bazy MySQL, zmienna `$mdb` zdefiniowana powyżej będzie instancją klasy `MDB2_Driver_mysql`.

Opcje

Pakiet MDB2 udostępnia kilka opcji, które można definiować przywołując metody `connect()`, `factory()` lub `singleton()` lub też później korzystając z metody `setOption()` (by zdefiniować jedną opcję) lub `setOptions()` (by zdefiniować kilka opcji na raz). Na przykład:

```
$options = array ( 'persistent' => true,
                   'ssl' => true,
                 );
$mdb2 =& $MDB2::factory($dsn, $options);
```

lub

```
$mdb2->setOption('portability', MDB2_PORTABILITY_NONE);
```

Pełną listę dostępnych opcji można znaleźć w dokumentacji interfejsu API pakietu MDB2, dostępnej pod adresem: *http://pear.php.net/package/MDB2/docs/*. Przyjrzyjmy się teraz dwóm najważniejszym opcjom.

Opcja „persistent"

Jest to opcja logiczna, która określa, czy utworzone połączenie powinno być połączeniem trwałym, czy też nie.

> W witrynie *mysql.com* można znaleźć bardzo dobry artykuł na temat zalet i wad korzystania z trwałych połączeń z bazą danych w systemie MySQL. Należy zajrzeć pod adres: *http://www.mysql.com/news-and- events/newsletter/2002-11/a0000000086.html*.

Domyślnie przypisywana jest jej wartość `false` (fałsz), określająca, że połączenie nie powinno być trwałe. Podczas tworzenia obiektu można zmienić to domyślne ustawienie:

```
$options = array ( 'persistent' => true
                 );
$mdb2 =& MDB2::factory($dsn, $options);
```

Natomiast metoda `setOption()` pozwala definiować opcje już po utworzeniu obiektu:

```
$mdb2->setOption('persistent', true);
```

Opcja „portability"

Pakiet MDB2 próbuje poradzić sobie jakoś z różnicami w sposobie implementowania pewnych funkcji baz danych przez różne systemy RDBMS. Opcja `portability` pozwala określić, w jakim zakresie warstwa bazy danych ma dbać o przenośność naszych skryptów.

Różne wartości opcji `portability` definiowane są jako stałe zaczynające się od `MDB2_PORTABILITY_*`, a domyślna wartość opcji to `MDB2_PORTABILITY_ALL` i oznacza „zrób wszystko, co tylko możliwe, by zagwarantować przenośność skryptów". Pełną listę stałych dla opcji `portability` oraz ich opis można znaleźć pod adresem: *http://pear.php.net/manual/en/package.database.* ➥*mdb2.intro-portability.php*.

Można definiować kilka wartości opcji `portability`, jak również definiować wyjątki za pomocą operatorów bitowych — dokładnie w taki sposób, w jaki definiuje się zasady raportowania błędów w języku PHP. To na przykład ustawienie poleca dbać o przenośność w pełnym zakresie, z wyjątkiem stosowania małych liter:

```
MDB2_PORTABILITY_ALL ^ MDB2_PORTABILITY_LOWERCASE
```

Jeśli natomiast nie interesują nas wszystkie funkcje przenośności oferowane przez MDB2, a chcielibyśmy tylko usunąć z wyniku spacje i zmienić puste wartości na łańcuchy `null`, to należy użyć opcji:

```
MDB2_PORTABILITY_RTRIM | MDB2_PORTABILITY_EMPTY_TO_NULL
```

Prawdopodobnie najlepszym rozwiązaniem będzie jednak pozostawienie domyślnego ustawienia `MDB2_PORTABILITY_ALL`. W ten sposób, w przypadku jakichś problemów z aplikacją, będziemy wiedzieli, że kod związany z sięganiem do bazy danych został dodatkowo sprawdzony pod kątem współpracy z różnymi systemami baz danych.

Definiowanie trybu pobierania danych

Kolejnym ustawieniem, które warto zdefiniować na początku, jest tryb pobierania danych (ang. *fetch mode*) lub też innymi słowy — sposób, w jaki dane te będą nam zwracane. Możemy otrzymywać dane w postaci uporządkowanej listy (ustawienie domyślne), w formie tablic asocjacyjnych lub w formie obiektów. Oto przykłady definiowania trybu pobierania danych:

```
$mdb2->setFetchMode(MDB2_FETCHMODE_ORDERED);
$mdb2->setFetchMode(MDB2_FETCHMODE_ASSOC);
$mdb2->setFetchMode(MDB2_FETCHMODE_OBJECT);
```

Oczywiście najbardziej przyjaznym dla człowieka i najczęściej stosowanym trybem pobierania danych będzie pobieranie ich w tablicach asocjacyjnych, ponieważ wyniki umieszczane są wtedy w tablicy, której klucze odpowiadają nazwom kolumn tabeli w bazie danych. Aby pokazać, na czym polega różnica, przyjrzyjmy się różnym sposobom pobierania danych zawartych w naszym zbiorze wyników:

```
echo $result[0]; // uporządkowana lub indeksowana tablica, domyślnie w MDB2
echo $result['name']; // tablica asocjacyjna
echo $result->name; // obiekt
```

Jest jeszcze jeden tryb pobierania danych, o nazwie `MDB2_FETCHMODE_FLIPPED` (tryb odwrócony). Jest on cokolwiek nietypowy i jego działanie zostało w dokumentacji API pakietu MDB2 opisane w następujący sposób:

„W przypadku wyników wielowymiarowych, zazwyczaj pierwszy poziom tablic jest numerem wiersza, podczas gdy drugi poziom jest indeksowany według nazwy bądź numeru kolumny. Tryb `MDB2_FETCHMODE_FLIPPED` odwraca ten porządek, w efekcie czego pierwszy poziom tablic będzie nazwą kolumny, a drugi poziom — numerem wiersza".

Rozłączanie się z bazą danych

Aby rozłączyć się z bazą danych, należy użyć następującego kodu:

```
$mdb2->disconnect();
```

Niemniej nawet jeśli sami nie zaznaczymy w kodzie, że chcemy rozłączyć się z bazą danych, pakiet MDB2 zrobi to za nas automatycznie w swoim destruktorze.

Korzystanie z MDB2

Gdy już połączymy się z bazą danych i określimy odpowiednie opcje połączenia oraz tryb pobierania danych, będzie można przystąpić do wykonywania zapytań. Na potrzeby przykładów prezentowanych w tym rozdziale założymy, że mamy tabelę o nazwie `people` (ludzie), z kolumnami `id` (identyfikator), `name` (imię), `family` (nazwisko) i `birth_date` (data_urodzenia):

id	name	family	birth_date
1	Eddie	Vedder	1964-12-23
2	Mike	McCready	1996-04-05
3	Stone	Gossard	1966-07-20

Przykład

Oto prosty przykład pokazujący, jak korzystać z MDB2. W dalszej części opowiemy o wszystkim szczegółowo, teraz jednak rzućmy okiem na kod, starając się zrozumieć w ogólnym zarysie, jak on działa.

```
<?php
require_once 'MDB2.php';
// przygotowania
$dsn = 'mysql://root:secret@localhost/mdb2test';
$options = array ('persistent' => true);
$mdb2 =& MDB2::factory($dsn, $options);
$mdb2->setFetchMode(MDB2_FETCHMODE_ASSOC);

// wykonujemy zapytanie
$sql = 'SELECT * FROM people';
$result = $mdb2->query($sql);

// wyświetlamy imiona
while ($row = $result->fetchRow())
{
echo $row['name'], '<br />';
}
// zwalniamy wykorzystywane zasoby
$result->free();

// wyłączamy zapytania
$mdb2->setOption('disable_query', true);

// usuwamy trzeci rekord
$id = 3;
$sql = 'DELETE FROM people WHERE id=%d';
```

```
$sql = sprintf($sql, $mdb2->quote($id, 'integer'));
echo '<hr />Zmienione wiersze: ';
echo $mdb2->exec($sql);

// zamykamy połączenie
$mdb2->disconnect();
?>
```

Wykonywanie zapytań

Aby wykonać zapytanie, można użyć metod `query()` lub `exec()`. Metoda `query()` zwraca obiekt wyniku `MDB2_Result`, natomiast metoda `exec()` zwraca liczbę wierszy w tabelach zmienionych przez zapytanie. Dlatego też metoda `exec()` będzie bardziej odpowiednia w przypadku zapytań, które modyfikują dane.

Mimo iż metoda `query()` pozwala wykonać praktycznie każdą operację na bazie danych, MDB2 oferuje również inne metody, które lepiej nadają się do określonych, często wykonywanych operacji.

Pobieranie danych

W przedstawionym wyżej przykładzie można znaleźć następujące wiersze:

```
$sql = 'SELECT * FROM people';
$result = $mdb2->query($sql);
```

Zmienna `$result` jest obiektem wyniku typu `MDB2_Result` lub też, ściślej mówiąc, zależną od konkretnego sterownika bazy danych klasą, która jest rozszerzeniem typu `MDB2_Result`, na przykład `MDB2_Result_mysql`. Do przeglądania zbioru wyników można natomiast użyć w pętli metody `fetchRow()`, która pobiera pojedyncze wiersze.

```
while ($row = $result->fetchRow())
{
   echo $row['name'], '<br />';
}
```

Za każdym razem gdy przywołujemy metodę `fetchRow()`, sprawdzi ona następny rekord i zwróci *odwołanie* (ang. *reference*) do zawartych w nim danych. Oprócz metody `fetchRow()` jest jeszcze parę innych metod z grupy `fetch*()`:

- `fetchAll()` zwraca od razu tablicę zawierającą wszystkie rekordy.
- `fetchOne()` jeśli zostanie przywołana bez żadnych parametrów, zwraca wartość pierwszego pola z bieżącego wiersza. Natomiast jeśli prześlemy jej odpowiednie parametry, będziemy mogli za jej pomocą pobrać dowolne pole z dowolnego wiersza. Na przykład wywołanie `fetchOne(1,1)` zwróci w naszym przykładzie imię **Mike**, czyli drugą kolumnę drugiego wiersza.

- `fetchCol($column)` zwraca pola w kolumnie o numerze `$column` dla wszystkich wierszy lub też pierwszą kolumnę, jeśli parametr `$column` nie zostanie określony.

Warto zwrócić uwagę, że metody `fetchRow()` i `fetchOne()` przenoszą wewnętrzny wskaźnik do bieżącego rekordu, podczas gdy metody `fetchAll()` i `fetchCol()` przeniosą go na koniec zbioru wyników. Można również skorzystać z wywołania `$result->nextResult()`, by z jego pomocą przenieść wskaźnik do następnego rekordu w zbiorze wyników lub z wywołania `$result->seek($rownum)`, by przenieść wskaźnik do wiersza określonego w parametrze `$rownum`. W razie wątpliwości można też skorzystać z wywołania `$result->rowCount()`, by sprawdzić, w którym miejscu zbioru wyników aktualnie znajduje się nasz wskaźnik.

Można również ustalić liczbę wierszy i liczbę kolumn w zbiorze wyników:

```
$sql = 'SELECT * FROM people';
$result = $mdb2->query($sql);
echo $result->numCols(); // wyświetla 4
echo $result->numRows(); // wyświetla 3
```

Skróty ułatwiające pobieranie danych

Bardzo często znacznie wygodniej jest pobierać dane w formie tablicy asocjacyjnej (lub zdefiniować je jako preferowany tryb pobierania danych) i nie kłopotać się szczegółami technicznymi związanymi z przeglądaniem zbioru wyników. Pakiet MDB2 oferuje dwa zestawy metod umożliwiających pobieranie danych „na skróty": metody z grupy `query*()` i metody z grupy `get*()`. Za ich pomocą następujące czynności wykonuje się przy użyciu pojedynczego wywołania metody:

1. Wykonywanie zapytania
2. Pobieranie zwracanych danych
3. Zwalnianie zasobów wykorzystywanych przez pobrany wynik

Skróty metod query*()

W tej grupie mamy do dyspozycji metody `queryAll()`, `queryRow()`, `queryOne()` i `queryCol()`, które odpowiadają analogicznym metodom z grupy `fetch*()`, omówionym wyżej. Oto przykład ilustrujący, czym różni się korzystanie z metod z grupy `query*()` od metod `fetch*()`:

```
// instrukcja SQL
$sql = 'SELECT * FROM people';
// jeden ze sposobów pobierania wszystkich danych
$result = $mdb2->query($sql);
$data = $result->fetchAll();
$result->free(); // nie wymagane
// krótszy sposób
$data = $mdb2->queryAll($sql);
```

W obu przypadkach, jeśli wyświetlimy za pomocą metody `print_r()` zawartość zmiennej `$data` i korzystamy z trybu pobierania używającego tablic asocjacyjnych, otrzymamy:

```
Array ( [0] => Array ( [id] => 1
                       [name] => Eddie
                       [family] => Vedder
                       [birth_date] => 1964-12-23
                     )
        [1] => Array ( [id] => 2
                       [name] => Mike
                       [family] => McCready
                       [birth_date] => 1966-04-05
                     )
...
)
```

Skróty metod get*()

Oprócz metod z grupy `query*()` można jeszcze korzystać ze skrótów oferowanych przez metody `get*()`. Metody z grupy `get*()` generalnie zachowują się w taki sam sposób jak metody z grupy `query*()`, niemniej pozwalają również na stosowanie w zapytaniach parametrów. Rozważmy następujący przykład:

```
$sql = 'SELECT * FROM people WHERE id=?';
$mdb2->loadModule('Extended');
$data = $mdb2->getRow($sql, null, array(1));
```

W tym przykładzie znak zapytania pojawiający się w instrukcji jest zmienną, która zostanie zastąpiona wartością przesłaną w trzecim parametrze metody `getRow()`.

Można również korzystać z parametrów posiadających własne nazwy:

```
$sql = 'SELECT * FROM people WHERE id=:the_id';
$mdb2->loadModule('Extended');
$data = $mdb2->getRow( $sql,
                       null,
                       array('the_id' => 1)
                      );
```

Warto zwrócić uwagę na to, że metody `get*()` są częścią modułu Extended pakietu MDB2, co oznacza, że aby były dostępne, należy je najpierw załadować używając polecenia `$mdb2->loadModule('Extended')`.

Dzięki ładowaniu modułów mamy możliwość przeciążania obiektów, czego nie można było robić przed pojawieniem się PHP5. Dlatego w wersji PHP4 języka, aby sięgnąć do metod modułu Extended, trzeba je było przywoływać w następujący sposób:

```
$mdb2->extended->getAll($sql);
```

Natomiast obecnie wystarczy wpisać:

```
$mdb2->getAll($sql);
```

getAsoc()

Kolejną użyteczną metodą z grupy `get*()`, która nie ma bezpośredniego odpowiednika w grupie `fetch*()` ani w grupie `query*()`, jest metoda `getAssoc()`. Zwraca ona wyniki w podobny sposób jak metoda `getAll()`, niemniej kluczami w tablicy wyników będą wartości z pierwszej kolumny tabeli. Dodatkowo zbiór wyników zawiera (w naszym przykładzie) tylko dwie kolumny, ponieważ jedną wykorzystaliśmy już jako indeks tablicy. Druga kolumna zostanie zwrócona w formie łańcucha (a nie w formie tablicy z jednym elementem). Oto kilka przykładów ilustrujących różnice pomiędzy metodami `getAll()` i `getAssoc()`:

```
$sql = 'SELECT id, name FROM people';
$mdb2->loadModule('Extended');
$data = $mdb2->getAll($sql);
```

Metoda `getAll()` zwróci uporządkowaną tablicę, w której każdy z elementów będzie tablicą asocjacyjną zawierającą wszystkie pola.

```
Array ( [0] => Array ( [id] => 1
                       [name] => Eddie
                     )
        [1] => Array ( [id] => 2
                       [name] => Mike
                     )
...
)
```

Jeśli wykonalibyśmy to samo zapytanie za pomocą metody `getAssoc()`, na przykład wpisując w kodzie `$data=$mdb2->getAssoc($sql)`, to otrzymalibyśmy następujący wynik:

```
Array ( [1] => Eddie
        [2] => Mike
        [3] => Stone
      )
```

Jeśli zapytanie zwraca więcej niż dwie kolumny, to każdy z wierszy będzie tablicą, a nie skalarem. Oto kod wykonujący takie zapytanie:

```
$sql = 'SELECT id, name, family FROM people';
$mdb2->loadModule('Extended');
$data = $mdb2->getAssoc($sql);
```

A oto wynik:

```
Array ( [1] => Array ( [name] => Eddie
                       [family] => Vedder
                     )
      ...
      )
```

Typy danych

Aby poradzić sobie z problemem wynikającym z tego, że różne systemy bazy danych obsługują różne typy danych dozwolone dla pól tabel, pakiet MDB2 dostarcza własnego, uniwersalnego zestawu typów danych. Programista może korzystać z typów danych oferowanych przez MDB2 i pozwolić, by sam pakiet MDB2 zadbał o przenośność typów danych między różnymi systemami RDBMS, po prostu mapując swoje typy na typy odpowiedniego systemu baz danych.

Oto lista typów danych oferowanych przez MDB2 i ich domyślne wartości:

```
$valid_types = array ( 'text' => '',
                       'boolean' => true,
                       'integer' => 0,
                       'decimal' => 0.0,
                       'float' => 0.0,
                       'timestamp' => '1970-01-01 00:00:00',
                       'time' => '00:00:00',
                       'date' => '1970-01-01',
                       'clob' => '',
                       'blob' => '',
                     )
```

Więcej informacji na temat typów danych MDB2 można znaleźć w pliku *datatypes.html*, znajdującym się w podkatalogu *docs* w katalogu, w którym zainstalowaliśmy PEAR. Dokument ten jest również dostępny w internecie, w witrynie repozytorium PEAR CVS:

```
http://cvs.php.net/viewcvs.cgi/pear/MDB2/docs/datatypes.html?view=co
```

Określanie typów danych

We wszystkich metodach służących do pobierania danych, którym się do tej pory przyglądaliśmy (z grup `query*()`, `fetch*()` i `get*()`), można było określać typ zbioru wyników, który chcemy otrzymać, i pakiet MDB2 automatycznie konwertował wartości na odpowiedni typ danych. Na przykład metodzie `query()` można było przesłać jako drugi parametr tablicę zawierającą oczekiwane typy danych dla pól:

```
$sql = 'SELECT * FROM people';
$types = array();
$result = $mdb2->query($sql, $types);
$row = $result->fetchRow();
var_dump($row);
```

W tym przypadku tablica typów `$types` była pusta, więc metoda zachowała się w domyślny sposób (nie wykonując żadnej konwersji typów danych) i wszystkie wyniki zostały zwrócone w formie łańcuchów. Przykład ten zwraca następujące dane:

```
array(2)
{
    ["id"] => string(1) "1"
    ["name"]=> string(5) "Eddie"
...
}
```

Możemy jednak zażyczyć sobie, aby pierwsze pole w każdym zwracanym rekordzie było typu `integer` (całkowitoliczbowego), a drugie typu `text` (tekstowego), definiując tablicę `$type` w następujący sposób:

```
$types = array('integer', 'text');
```

W tym przypadku otrzymamy następujący wynik:

```
array(2)
{
    ["id"]=> int(1)
    ["name"]=> string(5) "Eddie"
...
}
```

Podczas określania typów można również użyć tablicy asocjacyjnej, w której kluczami będą poszczególne pola tabeli. W takim przypadku można nawet pominąć niektóre pola, jeśli nie chcemy dla nich definiować typów. Oto przykłady poprawnych definicji takiej tablicy:

```
$types = array( 'id' => 'integer',
                'name' => 'text'
              );
$types = array('name'=>'text');
$types = array('integer');
```

Określanie typów danych podczas pobierania wyników

Jeśli nie chcemy określać typów danych już podczas przywoływania metody `query()`, możemy to zrobić później. Zanim rozpoczniemy pobieranie danych, możemy określić typy danych używając metody `setResultTypes()`.

```
// wykonujemy zapytanie
$sql = 'SELECT * FROM people';
$result = $mdb2->query($sql);

// pobieramy pierwszy wiersz bez konwertowania typów
$row = $result->fetchRow();
var_dump($row['id']);
// wynik będzie następujący: string(1) "1"

// określamy typy
$types = array('integer');
$result->setResultTypes($types);
```

```
// wszystkie następne pobrania będą konwertować
// pierwszą kolumnę na liczbę całkowitą
$row = $result->fetchRow();
var_dump($row['id']);
// wynik będzie następujący: int(2)
```

Określanie typów danych dla metod get*() i query*()

Wszystkie metody z grup `get*()` i `query*()`, które omawialiśmy wyżej w tym rozdziale, pozwalają na określanie w drugim przesyłanym im parametrze, jakie typy danych mają zwracać. Dokładnie tak samo jak w metodzie `query()`.

Możemy definiować parametr przesyłający typy danych nie tylko jako tablicę `$types = array('integer')`, ale również jako łańcuch `$types = 'integer'`. Jest to wygodne, kiedy pracujemy z metodami, które mają zwracać tylko jedną kolumnę danych, takimi jak `getOne()`, `queryOne()`, `getCol()` czy `queryCol()`, niemniej należy bardzo ostrożnie korzystać z tej techniki w przypadku metod typu `*All()` i `*Row()`, ponieważ wówczas łańcuch w parametrze podającym typ określi typ dla wszystkich pól w zbiorze danych.

Ujmowanie wartości i identyfikatorów w cudzysłowy

Różne systemy RDBMS zarządzające relacyjnymi bazami danych stosują różne konwencje ujmowania danych w cudzysłowy (na przykład tam gdzie jedne używają pojedynczych cudzysłowów ', inne stosują podwójne cudzysłowy "). Nie ma też żadnej powszechnie przyjętej konwencji ujmowania w cudzysłowy typów danych. Na przykład w systemie MySQL możemy (jeśli chcemy) ujmować wartości typu `integer` (całkowitoliczbowe) w cudzysłowy, natomiast w innych systemach może to być zabronione. Z tego powodu właśnie lepiej pozostawić obsługę cudzysłowów warstwie abstrakcji bazy danych, ponieważ pakiet MDB2 „wie", jakie konwencje stosują tutaj różne systemy baz danych.

Pakiet MDB2 oferuje specjalną metodę `quote()`, która umożliwia ujmowanie danych w cudzysłowy, i metodę `quoteIdentifier()` pozwalającą na ujmowanie w cudzysłowy nazw baz danych, tabel i pól. Wszelkie cudzysłowy wstawione przez pakiet MDB2 będą odpowiednie dla wykorzystywanego systemu RDBMS. Oto przykład:

```
$sql = 'UPDATE %s SET %s=%s WHERE id=%d';
$sql = sprintf( $sql,
                $mdb2->quoteIdentifier('people'),
                $mdb2->quoteIdentifier('name'),
                $mdb2->quote('Eddie'), // domniemany typ danych
                $mdb2->quote(1, 'integer') // wyraźnie określony typ danych
              );
```

Jeśli teraz w bazie MySQL wykonamy polecenie `echo $sql`, otrzymamy:

```
UPDATE `people` SET `name`='Eddie' WHERE id=1
```

Natomiast w bazie SQLite ten sam kod zwróci:

```
UPDATE "people" SET "name"='Eddie' WHERE id=1
```

Jak można było zauważyć, przyglądając się poprzednim przykładom, metoda `quote()` pozwala na przesłanie jej drugiego, opcjonalnego parametru, określającego typ danych (oczywiście typ MDB2), który ma zostać ujęty w cudzysłowy. Jeśli pominiemy drugi parametr, MD2 postara się zgadnąć typ danych.

Iteratory

Pakiet MDB2 korzysta ze standardowej biblioteki PHP (Standard PHP Library *http://php.net/spl*) i implementuje interfejs `Iterator`, który pozwala na znacznie prostsze i wygodniejsze przeglądanie wyników zapytania:

```
foreach ($result as $row)
{
   var_dump($row);
}
```

W każdej kolejnej iteracji (powtórzeniu) pętli zmienna `$row` zawierać będzie kolejny rekord (wiersz), przechowywany w formie tablicy. Prezentowany tu kod jest równoważny przywołaniu w pętli metody `fetchRow()`:

```
while ($row = $result->fetchRow())
{
   var_dump($row);
}
```

Aby skorzystać z zalet interfejsu iteratorów `Iterator`, konieczne będzie załączenie pliku *Iterator.php* z katalogu pakietu MDB2, przy użyciu metody `loadFile()`:

```
MDB2::loadFile('Iterator');
```

Po załadowaniu tego pliku będzie można przesyłać metodzie `query()` jako czwarty parametr nazwę klasy `Iterator`:

```
$query = 'SELECT * FROM people';
$result = $mdb2->query($query, null, true, 'MDB2_BufferedIterator');
```

Pakiet MDB2 oferuje dwie klasy iteratorów:

- `MDB2_Iterator` — Klasa ta implementuje klasę `Iterator` biblioteki SPL i najlepiej sprawdza się w pracy z niebuforowanymi zbiorami wyników.
- `MDB2_BufferedIterator` — Ta klasa jest rozszerzeniem klasy `MDB_Iterator` i implementuje interfejs `SeekableIterator`. Podczas pracy z buforowanymi zbiorami wyników (domyślnymi w MDB2) lepiej jest korzystać z klasy `MDB2_BufferedIterator`, ponieważ oferuje ona parę dodatkowych przydatnych w tej sytuacji metod, takich jak `count()` czy `rewind()`.

Wyszukiwanie błędów

Pakiet MDB2 umożliwia programiście przechowywanie listy wszystkich zapytań wykonywanych w danej instancji obiektu MDB2, znacznie ułatwiając wyszukiwanie błędów w naszych aplikacjach. Aby włączyć tę opcję wyszukiwania błędów (debugowania), trzeba przypisać opcji `debug` dodatnią liczbę całkowitą.

```
$mdb2->setOption('debug', 1);
```

Po włączeniu jej w ten sposób będziemy mogli w dowolnym momencie sięgnąć do zbieranych przez pakiet MDB2 danych debugowania:

```
$mdb2->getDebugOutput();
```

Można również włączyć opcję `log_line_break`, która pozwala określić, w jaki sposób będą oddzielane od siebie dane zapisywane w dzienniku debugowania. Domyślnie odseparowane są znakiem nowego wiersza `\n`.

Oto prosty przykład, w którym włączona została opcja `debug` i określony odpowiedni separator, następnie wykonano kilka zapytań, a na koniec pobrano nieuporządkowaną listę tych zapytań z zanotowanych w dzienniku danych debugowania.

```
$mdb2->setOption('debug', 1);
$mdb2->setOption('log_line_break', "\n\t");

$sql = 'SELECT * FROM people';
$result = $mdb2->query($sql);
$sql = 'SELECT * FROM people WHERE id = 1';
$result = $mdb2->query($sql);
$sql = 'SELECT name FROM people';
$result = $mdb2->query($sql);

$debug_array = explode("\n\t", trim($mdb2->getDebugOutput()));

echo '<ul><li>';
echo implode('</li><li>', $debug_array);
echo '</li></ul>';
```

Przykład ten zwróci następującą listę zapytań:

- **`query(1): SELECT * FROM people`**
- **`query(1): SELECT * FROM people WHERE id = 1`**
- **`query(1): SELECT name FROM people`**

Gdy już aplikacja wejdzie w fazę produkcyjną, warto ustawić poziom debugowania (rejestrowania informacji w dzienniku) jako 0, aby uniknąć niepotrzebnego zaśmiecania dziennika debugowania informacjami o wszystkich wykonywanych zapytaniach.