

» Idź do

- Spis treści
- Przykładowy rozdział

» 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

» Kontakt

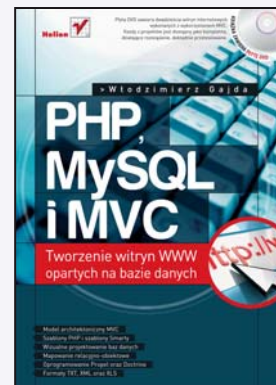
Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl
© Helion 1991-2010

PHP, MySQL i MVC. Tworzenie witryn WWW opartych na bazie danych

Autor: [Włodzimierz Gajda](#)

ISBN: 978-83-246-1258-1

Format: 158×235, stron: 528



Duża część popularnych serwisów internetowych dostępnych obecnie w sieci działa w oparciu o relacyjne bazy danych i język PHP. Tandem ten stał się już niemal standardem w dziedzinie tworzenia rozbudowanych, dynamicznych witryn i aplikacji WWW zarówno w przypadku wielkich przedsiębiorstw, jak i hobbystów pragnących dzielić się swoimi doświadczeniami za pośrednictwem internetu. PHP i MySQL doskonale nadają się do praktycznej realizacji wzorca architektonicznego MVC, ułatwiającego opracowywanie nawet najbardziej złożonych projektów. Dzięki odseparowaniu poszczególnych komponentów aplikacji i podzieleniu jej na mniejsze elementy funkcjonalne tworzona witryna jest znacznie wygodniejsza w zarządzaniu i modyfikacji niż serwisy oparte na innych wzorcach.

Wzorec architektoniczny MVC – choć opracowany z myślą o uproszczeniu życia programistom i twórcom witryn WWW – początkowo bardzo trudno zrozumieć, a postęp w dziedzinie tworzenia stron internetowych wymusza ciągle dostosowywanie się do panujących na rynku trendów i stałe odświeżanie wiedzy na ten temat. Pomocą posłuży tu odpowiednia książka – z pewnością może nią być „PHP, MySQL i MVC. Tworzenie witryn WWW opartych na bazie danych”. W przystępny sposób prezentuje ona podstawy zastosowania architektury MVC, pokazuje wykorzystanie dwóch najważniejszych rozwiązań do mapowania obiektowo-relacyjnego, podsuwa techniki tworzenia funkcjonalnych interfejsów użytkownika aplikacji WWW oraz proponuje stosowanie różnych formatów do przechowywania danych. Poszczególne zagadnienia ilustrowane są przykładami i praktycznymi projektami, dzięki czemu łatwiej zrozumieć opisywane techniki i wykorzystać je w swoich aplikacjach. Zdobyta wiedzę pomagają utrwalić liczne ćwiczenia do samodzielnego wykonania.

- Podstawy użycia wzorca MVC
- Moduły, akcje, widoki, szablony PHP i szablony Smarty
- Reguły translacji przyjaznych adresów URL
- Używanie baz danych w projektach WWW
- Wizualne projektowanie baz danych i mapowanie relacyjno-obiektowe
- Wykorzystanie oprogramowania Propel oraz Doctrine
- Metody zwiększania funkcjonalności interfejsów aplikacji internetowych
- Stronicowanie wyników i przewijanie rekordów
- Chmura tagów, korzystanie z formatów TXT, XML, XLS
- Odczyt plików i wypełnianie bazy danych
- Pliki skompresowane i osadzanie danych binarnych w plikach XML

Poznaj w praktyce nowoczesne metody tworzenia zaawansowanych aplikacji WWW!

Wszystkie projekty omówione w książce umieszczono na płycie dołączonej do książki. Każdy z projektów jest dostępny jako kompletne, działające rozwiązanie, dokładnie przetestowane.

Spis treści

Wstęp	13
Część I Akcje, widoki, translacje adresów URL i bazy danych, czyli podstawy użycia MVC	15
Rozdział 1. Hello world — pierwsza aplikacja korzystająca z MVC	17
Projekt 1.1. Hello world!	18
Krok pierwszy: utworzenie modułu main	19
Krok drugi: utworzenie akcji hello w module main	19
Krok trzeci: utworzenie układu witryny WWW	20
Krok czwarty: reguła translacji adresu pierwszy-projekt.html na wykonanie akcji main/hello	20
Krok piąty: skrypt index.php	21
Pliki, które należy utworzyć	22
Uruchomienie projektu	23
Przebieg wykonania	24
Rozdział 2. Dołączanie zewnętrznych zasobów .css, .jpg, .js	27
Projekt 2.1. Wlaził kotek na schody...	28
Analiza ścieżek zawartych w kodzie HTML	31
Rozdział 3. Błędy 404	35
Projekt 3.1. Żółta Turnia	36
Oglądanie strony błędu oraz nagłówek HTTP	38
Rozdział 4. Zmienne i widoki	41
Projekt 4.1. Data i godzina — szablon PHP	42
Projekt 4.2. Data i godzina — szablon Smarty	45
Projekt 4.3. Ojciec i syn — szablon PHP	47
Projekt 4.4. Ojciec i syn — szablony Smarty	49
Projekt 4.5. Stefan Żeromski: <i>Zmierzch</i> — szablon PHP	50
Projekt 4.6. Stefan Żeromski: <i>Zmierzch</i> — szablony Smarty	53
Projekt 4.7. Kolory CSS	54
Projekt 4.8. Kolory CSS — szablony Smarty	56
Rozdział 5. Pre- i postprzetwarzanie	59
Projekt 5.1. Fraszki	60
Projekt 5.2. Fraszki — szablony Smarty	68

Rozdział 6. Translacja adresów URL	71
Projekt 6.1. Kolędy	73
Analiza różnych rodzajów adresów URL	77
Dwukierunkowość konwersji adresów	78
Konwersje adresów w generowanych stronach WWW	79
Włączanie i wyłączanie translacji wyjściowych	80
Konwersja adresów URL przy użyciu funkcji pomocniczych	81
Implementacja funkcji pomocniczych w postaci wtyczek Smarty	82
Kilka zmiennych w adresach URL	83
Projekt 6.2. Ligi piłkarskie	84
Zmienne \$path_prefix oraz ####PATH_PREFIX###	92
Rozdział 7. Bazy danych	95
Wizualne projektowanie bazy danych	95
Oprogramowanie ORM	99
Konwersja pliku .mwb do formatu Propel XML	100
Konwersja pliku .mwb do formatu Doctrine YML	101
db-frame-tool	101
Propel — generowanie klas dostępu do bazy	102
ORM Propel — pierwsze kroki	104
Zestawienie wygenerowanych klas oraz najważniejszych metod	104
Podstawy użycia klas wygenerowanych przez Propel	105
Doctrine — generowanie klas dostępu do bazy	106
ORM Doctrine — pierwsze kroki	107
Zestawienie wygenerowanych klas oraz najważniejszych metod	107
Podstawy użycia klas wygenerowanych przez Doctrine	108
Projekt 7.1. Tatry (szablony PHP, Propel)	109
Krok pierwszy: projekt bazy danych	110
Krok drugi: generowanie klas dostępu do bazy danych	110
Krok trzeci: tworzenie pustej bazy danych	110
Krok czwarty: wypełnianie bazy danych na podstawie pliku tekstowego	110
Krok piąty: zrzut wypełnionej bazy danych	113
Krok szósty: aplikacja prezentująca zawartość bazy danych	114
Projekt 7.2. Tatry (szablony PHP, Doctrine)	116
Krok pierwszy: projekt bazy danych	116
Krok drugi: generowanie klas dostępu do bazy danych	116
Krok trzeci: tworzenie pustej bazy danych	117
Krok czwarty: wypełnianie bazy danych na podstawie pliku tekstowego	117
Krok piąty: zrzut wypełnionej bazy danych	118
Krok szósty: aplikacja prezentująca zawartość bazy danych	118
Projekt 7.3. Tatry (szablony Smarty, Propel)	119
Projekt 7.4. Tatry (szablony Smarty, Doctrine)	120
Rozdział 8. Czego powinieneś nauczyć się z części pierwszej?	123
Część II Operowanie klasami wygenerowanymi przez Propel oraz Doctrine	129
Rozdział 9. Wybieranie wszystkich rekordów z tabeli w zadanym porządku	131
Propel — sortowanie rekordów	132
Doctrine — sortowanie rekordów	134
Projekt 9.1. Słownik (Propel, PHP)	134
Krok pierwszy: projekt bazy danych	135
Krok drugi: generowanie klas dostępu do bazy danych	135

Krok trzeci: tworzenie pustej bazy danych	137
Krok czwarty: wypełnianie bazy danych na podstawie pliku tekstowego	137
Krok piąty: zrzut wypełnionej bazy danych	138
Krok szósty: aplikacja prezentująca zawartość bazy danych	138
Projekt 9.2. Słownik (Doctrine, PHP)	141
Krok pierwszy: projekt bazy danych	141
Krok drugi: generowanie klas dostępu do bazy danych	141
Krok trzeci: tworzenie pustej bazy danych	142
Krok czwarty: wypełnianie bazy danych na podstawie pliku tekstowego	142
Krok piąty: zrzut wypełnionej bazy danych	142
Krok szósty: aplikacja prezentująca zawartość bazy danych	143
Rozdział 10. Wybieranie pojedynczego rekordu	145
Propel	145
Doctrine	147
Projekt 10.1. <i>Treny</i> (Propel, PHP)	147
Identyfikacja trenu wewnątrz akcji tren/show	151
Projekt 10.2. <i>Treny</i> (Doctrine, PHP)	153
Rozdział 11. Relacje 1:n	157
Metody generowane przez Propel dla relacji 1:n	158
Doctrine i relacje 1:n	160
Projekt 11.1. Kontynenty, państwa, miasta (Propel, PHP)	162
Przygotowanie bazy danych	163
Aplikacja	167
Projekt 11.2. Kontynenty, państwa, miasta (Doctrine, PHP)	173
Rozszerzanie właściwości klas generowanych przez Doctrine	174
Wstawianie rekordów	176
Aplikacja	178
Rozdział 12. Relacje n:m	181
Metody generowane przez Propel dla relacji n:m	182
Doctrine i relacje n:m	183
Projekt 12.1. Filmy (Propel, PHP)	184
Przygotowanie bazy danych	184
Aplikacja	187
Projekt 12.2. Filmy (Doctrine, PHP)	189
Propel. Sortowanie rekordów stojących w relacji n:m	192
Projekt 12.3. Filmy (Propel, PHP, sortowanie)	192
Doctrine. Sortowanie rekordów stojących w relacji n:m	193
Projekt 12.4. Filmy (Doctrine, PHP, sortowanie)	194
Rozdział 13. Zagadnienia dodatkowe dotyczące warstw M oraz V	197
Czyszczenie zawartości bazy danych	197
Konwersja obiektu w napis	198
Konwersje toArray(), fromArray()	198
Warunkowe wstawianie nieistniejących obiektów	200
Wielokrotne wykorzystanie widoku	201
Projekt 13.1. Aparaty foto (Propel, PHP)	202
Warstwa M	202
Wypełnianie bazy danych	205
Aplikacja	206
Projekt 13.2. Aparaty foto (Doctrine, PHP)	208
Rozszerzanie funkcjonalności klas wygenerowanych przez Doctrine	209
Wstawianie rekordów do bazy danych	211
Aplikacja	212

Rozdział 14. Zapisywanie w bazie danych obrazów i plików binarnych	215
Zapisywanie w bazie danych zdjęć JPG	215
Prezentowanie zdjęć JPG zapisanych w bazie danych na stronie WWW	216
Zapisywanie w bazie danych dowolnych plików binarnych	218
Wysyłanie danych binarnych z bazy do przeglądarki	220
Projekt 14.1. NotH (Propel, PHP)	221
Skrypt wstaw.php	223
Aplikacja	224
Tytuły stron	225
Rozwijane menu pionowe	226
Wartości atrybutów href oraz src w tekstach zapisanych w bazie danych	227
Projekt 14.2. NotH (Doctrine, PHP)	229
Skrypt wstaw.php	230
Aplikacja	230
Rozdział 15. Akcje list i show, czyli publikowanie zawartości bazy danych w postaci witryny WWW	233
Projekt 15.1. Czcionki projektów CSS Zen Garden (Propel, PHP)	234
Aplikacja	235
Menu główne witryny oraz tytuły podstron	240
Rozdział 16. Czego powinieneś nauczyć się z części drugiej?	243
Część III Zwiększanie funkcjonalności interfejsu aplikacji internetowej	245
Rozdział 17. Kontekstowe hiperłącza do stron ze szczegółowymi informacjami ..	247
Projekt 17.1. Angaże (szablony PHP, Propel)	248
Unikatowość kolumny slug	248
Pliki tekstowe o bardziej złożonej strukturze	250
Rozszerzenia warstwy M	251
Filtry konwertujące generowany kod HTML	260
Aplikacja	261
Hiperłącza kontekstowe	263
Menu kontekstowe	264
Rozdział 18. Następny, poprzedni, czyli przewijanie zawartości witryny WWW	267
Projekt 18.1. PHP. Praktyczne projekty	269
Ograniczenia kluczy obcych	270
Wstępne opracowanie aplikacji	271
Implementacja hiperłączy następny/poprzedni	276
Hiperłącza link zawarte w nagłówku strony WWW	285
Rozdział 19. Wskaźnik położenia	287
Projekt 19.1. Kolekcja płyt DVD z fotografiami	288
Rozszerzanie klas dostępu do bazy danych	291
Wypełnianie bazy danych rekordami	295
Aplikacja	302
Translacje adresów stosujących cztery zmienne URL	304
Tabela zdjęć wykonana bez użycia tabel HTML	305
Wskaźniki następny/poprzedni do przewijania zdjęć i kategorii	306
Efekt rollover ze wskaźnikiem wybranej opcji	307
Fotografie podążające za wskaźnikiem myszki	307
Wskaźnik breadcrumbs	310

Rozdział 20. Sortowanie tabel	311
Projekt 20.1. Piłka nożna — sezon 2002/2003	312
Wielokrotne klucze obce z tej samej tabeli	314
Sortowanie złączeń dla wielokrotnych kluczy	314
Wypełnianie bazy danych	317
Zarys aplikacji	320
Implementacja sortowalnych tabeli HTML	320
Rozdział 21. Stronicowanie	331
Projekt 21.1. 33 ⅓	332
Automatyczne generowanie identyfikatorów slug dla rekordów o zdublowanych tytułach	333
Klasa Pager	335
Widok wskaźnika stronicowania	343
Prezentacja rekordów poddanych stronicowaniu	345
Kontekstowe stronicowanie rekordów	348
Rozdział 22. Alfabet	351
Projekt 22.1. Imiona	351
Wybieranie alfabetu liter	352
Komponent wyświetlający alfabet	353
Prezentacja liter rozpoczynających się od wybranej litery na stronie WWW	354
Umieszczanie kontrolki z listą liter w szablonie layout.html	355
Rozdział 23. Chmura tagów	357
Projekt 23.1. Katalog Open Clipart	358
Wypełnianie bazy danych	360
Waga słów kluczowych	363
Aplikacja	365
Rozdział 24. Spis treści	371
Projekt 24.1. Artykuły	372
Wypełnianie bazy danych rekordami	373
Funkcje odpowiedzialne za odczytywanie i usuwanie fragmentów kodu HTML	376
Funkcje odpowiedzialne za tworzenie spisu treści	378
Kolorowanie składni	380
Aplikacja	381
Rozdział 25. Czego powinieneś nauczyć się z części trzeciej?	383
Część IV Formaty danych	385
Rozdział 26. Podstawy przetwarzania dokumentów XML w PHP	387
Klasa SimpleXML	387
Tworzenie obiektu SimpleXMLElement	388
Dostęp do węzłów drzewa	389
Lista identycznych elementów	390
Dostęp do atrybutów	391
Przetwarzanie wszystkich elementów i ich atrybutów	391
Wielokrotne zagnieżdżenia	392
Język XPath	395
Przykładowe dokumenty XML dostępne w internecie	395
Kursy walut	395
Książki wydawnictwa Helion	397
Projekt 26.1. Turniej Czterech Skoczni	398
Aplikacja	400

Rozdział 27. Generowanie dokumentów XML w PHP	403
Statyczne pliki XML	403
Generowanie dokumentu XML w PHP	405
Echo — drukowanie kodu XML	406
Generowanie XML na podstawie tablicy	407
Generowanie XML na podstawie pliku tekstowego	408
Zapisywanie kodu XML do pliku	409
Konwersja pliku tekstowego do formatu XML	409
Dane w formacie XML opisujące witrynę WWW	410
Mapa witryny: sitemap.xml	410
Kanał RSS	411
Projekt 27.1. Kursy walut	412
Ustalanie adresów dokumentów XML z kursami walut	412
Projekt bazy danych	413
Wypełnianie bazy danych rekordami	413
Aplikacja	415
Kanał RSS	415
Mapa witryny	416
Wykresy kursów walut	418
Rozdział 28. XML_Serializer, XML_Unserializer	
— dwukierunkowe transformacje tablic w XML	423
XML_Serializer	423
Konwersja tablicy w kod XML	423
Tablica asocjacyjna	424
Opcje	425
Tablica opcji	426
Jednowymiarowa tablica indeksowana	426
Wielowymiarowe tablice indeksowane	427
Atrybuty	428
Wybiórcze stosowanie atrybutów	429
Przekształcenia	430
XML_Serializer — przykłady	431
Projekt 28.1. Konwersja pliku nobel.txt	431
Projekt 28.2. Konwersja pliku mecze.txt	432
Projekt 28.3. Konwersja pliku tcs.txt	434
Klasa XML_Unserializer	436
Podstawowe użycie	436
Odczyt pliku	437
Parsing atrybutów	437
Konwersja formatu XML	439
Projekt 28.4. Konwersja jeden-w-wiele	439
Projekt 28.5. Konwersja wiele-w-jeden	440
Projekt 28.6. Klasyfikacja zwierząt	441
Wypełnianie bazy danych rekordami	442
Moduły i akcje aplikacji	443
Akcja main/drzewo	444
Kanał RSS	444
Generowanie statycznego dokumentu sitemap.xml	447
Rozdział 29. Arkusze kalkulacyjne MS Excel XLS	449
Odczyt pliku XLS	449
Odczyt kilku arkuszy	451
Tworzenie pliku XLS	452

Wysyłanie arkusza do przeglądarki	453
Konwersja pliku tekstowego do formatu XLS	453
Konwersja pliku XLS do formatu tekstowego	454
Projekt 29.1. Generowanie danych autokomisu	455
Projekt 29.2. Autokomis	460
Wypełnianie bazy danych rekordami	460
Aplikacja	465
Rozdział 30. Konwersja plików z danymi	467
Format danych tekstowych	467
Format danych XML	469
Format danych XLS	470
Projekt 30.1. Konwersja formatu TXT do formatu XML	470
Projekt 30.2. Konwersja formatu TXT do formatu XLS	474
Projekt 30.3. Konwersja formatu XML do formatu TXT	476
Projekt 30.4. Konwersja formatu XML do formatu XLS	477
Projekt 30.5. Konwersja formatu XLS do formatu TXT	478
Projekt 30.6. Konwersja formatu XLS do formatu XML	480
Projekt 30.7. Zestawienia artykułów „Magazynu INTERNET”	480
Aplikacja	483
Akcja list prezentująca sortowalną i stronicowaną tabelkę HTML	484
Akcja show prezentująca sortowalną i stronicowaną tabelkę HTML	486
Rozdział 31. Skompresowane dokumenty XML zawierające dane binarne	489
Kodowanie base64	490
Dekodowanie base64	490
Kompresja danych	491
Dekompresja danych	492
Projekt 31.1. Format danych systemu do publikowania artykułów	493
Umieszczanie ilustracji, listingów, ramek i tabel w treści artykułu	495
Projekt 31.2. Konwersja artykułu z formatu tekstowego do spakowanego pliku XML	495
Projekt 31.3. System publikacji artykułów w postaci witryny WWW	498
Baza danych	499
Propel i dostęp tylko do wybranych kolumn tabeli	499
Wypełnianie bazy danych	501
Aplikacja	508
Rozdział 32. Czego powinieneś nauczyć się z części czwartej?	511
Skorowidz	513

Rozdział 21.

Stronicowanie

Częstym problemem, który pojawia się podczas tworzenia witryn internetowych, jest prezentacja dużej liczby rekordów. Powszechnie stosowanym rozwiązaniem jest stronicowanie, czyli podział liczby prezentowanych wyników na mniejsze grupy. Wskaźnik nawigacji zawierający numer strony oraz odsyłacze do stron sąsiednich ułatwia nawigowanie na witrynie stosującej stronicowanie.

Stronicowanie wyników opiszę na przykładzie internetowego katalogu płyt, wykonawców i piosenek.

Załóżmy, że w tabeli song zawierającej teksty piosenek znajduje się 1005 rekordów. Przyjmijmy, że chcemy wyświetlać 10 rekordów na stronie. Otrzymamy 101 stron, które będą prezentowały rekordy:

- ◆ strona 1: rekordy od 1 do 10;
- ◆ strona 2: rekordy od 11 do 20;
- ◆ ...
- ◆ strona 100: rekordy od 991 do 1000;
- ◆ strona 101: rekordy od 1001 do 1005.

Szerokość wskaźnika bieżącej strony będzie ustalała liczbę prezentowanych numerów stron. Jeśli szerokość wskaźnika ustalimy na 7, to na pierwszej stronie należy wyświetlić numery stron:

1 2 3 4 5 6 7

Jeśli szerokość wskaźnika wyniesie 3, to na stronie ostatniej wyświetlimy numery:

99 100 101

Parametrami wskaźnika będą więc:

- ◆ liczba wszystkich rekordów;
- ◆ liczba rekordów na stronie;
- ◆ szerokość wskaźnika.

Postać wskaźnika ma zależeć od numeru bieżącej strony. Wskaźnik wyświetlamy w taki sposób, by, jeśli to tylko możliwe, numer bieżącej strony znajdował się w środku. Jeśli szerokość wskaźnika wyniesie 5, to na stronie 13 zaprezentujemy numery:

11 12 13 14 15

a na stronie 39:

37 38 39 40 41

Na stronach początkowych, tj. pierwszej, drugiej i trzeciej, wyświetlimy identyczny wskaźnik o numerach:

1 2 3 4 5

Innymi słowy, jeśli numer strony jest mniejszy od połowy szerokości wskaźnika, to wyświetlamy wskaźnik zadanej szerokości rozpoczynający się od strony 1. Podobna sytuacja wystąpi dla stron, których numery będą bliskie numerowi ostatniej strony. Jeśli szerokość wskaźnika wyniesie 11, to na ostatnich pięciu stronach, tj. na stronach o numerach 97, 98, 99, 100, 101, wyświetlimy wskaźnik:

91 92 93 94 95 96 97 98 99 100 101

Jedynym przypadkiem, gdy wyświetlimy wskaźnik krótszy od zadanej szerokości, będzie sytuacja, w której liczba otrzymanych stron jest mniejsza od szerokości wskaźnika. Jeśli w bazie danych umieścimy 20 rekordów oraz ustalimy liczbę rekordów na stronie na 10, a szerokość wskaźnika na 5, to wskaźnik i tak będzie zawierał tylko dwa numery:

1 2

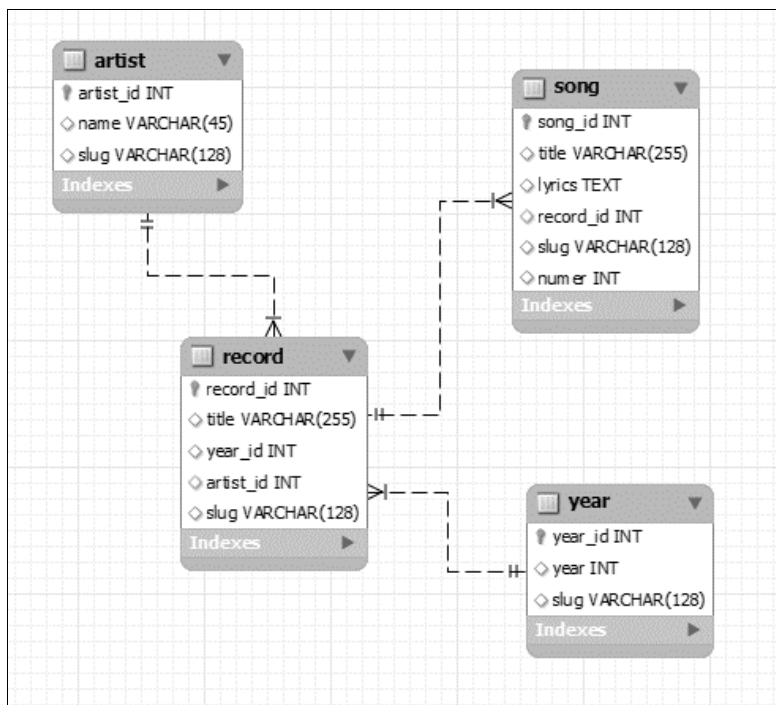
Projekt 21.1. 33 $\frac{1}{3}$

Wykonaj internetowy katalog płyt winylowych pt. 33 $\frac{1}{3}$. W interfejsie aplikacji uwzględnij fakt, że baza danych będzie zawierała setki rekordów. Wszelkie listy rekordów prezentowanych na stronie WWW poddaj stronicowaniu.

Pracę nad projektem rozpoczynamy od wykonania bazy danych przedstawionej na rysunku 21.1.

W omawianych do tej pory projektach mogliśmy przyjąć, że identyfikatory slug, tworzone na podstawie nazw, tytułów, imion czy nazwisk, są unikalne. W omawianym projekcie takiego założenia przyjąć nie możemy. Piosenka o tym samym tytule może pojawić się na wielu płytach. Na przykład utwór pt. *Good Times*, *Bad Times* występuje na pierwszej płycie zespołu Led Zeppelin oraz na krążku pt. *12 X 5* grupy The Rolling Stones. Powielenie tytułów wystąpi także w przypadku składanek *The Best of...* utworów wybranego zespołu. W celu rozwiązania tego problemu nadpiszemy metodę `setSlug()` ustalającą wartości kolumn `slug`.

Rysunek 21.1.
Baza danych
z projektu 21.1



Automatyczne generowanie identyfikatorów slug dla rekordów o zdublowanych tytułach

W jaki sposób rozwiążemy problem dublowania identyfikatorów slug? Kolejne zdublowane wartości będziemy numerowali. Pierwsza piosenka o tytule *Good Times, Bad Times*, którą wstawimy do bazy danych, otrzyma slug:

```
good_times_bad_times
```

Za drugim razem użyjemy wartości

```
good_times_bad_times2
```

Kolejne utwory, których tytuły po przekształceniu `string2slug()` dają napis `good_times_bad_times`, otrzymają wartości slug oznaczone kolejnymi liczbami:

```
good_times_bad_times3
good_times_bad_times4
good_times_bad_times5
...
itd.
```

Proces numeracji zdublowanych slugów zaimplementujemy w metodzie `setSlug()` klasy `Song`. Metoda ta jest przedstawiona na listingu 21.1.

Listing 21.1. *Metoda setSlug() klasy Song*

```

public function setSlug($slug)
{
    if (trim($slug) == '') {
        $slug = 'nieznany';
    }

    $next_slug = $slug;
    $c = new Criteria();
    $c->add(SongPeer::SLUG, $next_slug);
    $ile = SongPeer::doCount($c);

    $unikatowy = ($ile == 0);

    $min = 2;
    $max = 4;

    while (!$unikatowy) {

        $next_slug = $slug . $min;
        $min++;

        if ($min > $max + 1) {
            die("***** ERROR    Song::setSlug({$next_slug})");
        };

        $c->clear();
        $c->add(SongPeer::SLUG, $next_slug);
        $ile = SongPeer::doCount($c);
        $unikatowy = ($ile == 0);

    }

    parent::setSlug($next_slug);
}

```

Pracę rozpoczynamy od sprawdzenia, czy napis slug jest niepusty. Jeśli wartością parametru \$slug jest napis składający się z białych znaków, to jako wartość slug przyjmujemy napis nieznany:

```

if (trim($slug) == '') {
    $slug = 'nieznany';
}

```

Wartość zmiennej \$slug przypisujemy do zmiennej \$next_slug, po czym metodą doCount() zliczamy rekordy tabeli Song, które mają identyczną wartość slug. Liczba elementów o tej samej wartości kolumny slug zostaje zapamiętana w zmiennej \$ile:

```

$next_slug = $slug;
$c = new Criteria();
$c->add(SongPeer::SLUG, $next_slug);
$ile = SongPeer::doCount($c);

```

Zmienna \$poprawny przyjmuje wartość logiczną informującą nas o tym, czy wartość slug zawarta w zmiennej \$next_slug jest unikatowa. Jeśli liczba znalezionych rekordów wynosi zero, to zmienna \$unikatowy przyjmuje wartość true:

```
$unikatowy = ($ile == 0);
```

Przygotowanie iteracji kończymy, ustalając maksymalny oraz minimalny numer dodawany na końcu adresu slug:

```
$min = 2;  
$max = 10000;
```

Głównym fragmentem przetwarzania w metodzie setSlug() jest pętla while. Przetwarzanie powtarzamy, dopóki zmienna \$unikatowy przyjmuje wartość false:

```
while (!$unikatowy) {  
  
}
```

czyli dopóki adres zawarty w zmiennej \$next_slug nie jest unikatowy. W refrenie pętli najpierw tworzymy adres \$next_slug, dodając na końcu adresu \$slug kolejną liczbę:

```
$next_slug = $slug . $min;  
$min++;
```

Jeśli przekroczyliśmy wartość maksymalną, to działanie skryptu kończymy, drukując informację o błędzie:

```
if ($min > $max + 1) {  
    die("***** ERROR    Song::setSlug({$next_slug})");  
}
```

Jeśli liczba dodana na końcu zmiennej \$next_slug mieści się w ustalonym zakresie, to przechodzimy do sprawdzenia, czy otrzymany adres slug jest unikatowy:

```
$c->clear();  
$c->add(SongPeer::SLUG, $next_slug);  
$ile = SongPeer::doCount($c);  
$unikatowy = ($ile == 0);
```

Jeśli w którymkolwiek obrocie pętli while zmienna \$unikatowy przyjmie wartość true, oznaczać to będzie, że w bazie danych nie ma jeszcze rekordu o wartości slug takiej jak zmienna \$next_slug.

Po zakończeniu pętli wywołujemy oryginalną metodę setSlug(), przekazując do niej zmienną \$next_slug:

```
parent::setSlug($next_slug);
```

Klasa Pager

Implementacja stronicowania sprowadza się do przygotowania jednej klasy Pager. Dzięki wykorzystaniu klas generowanych przez Propel oraz kryteriów pojedyncza klasa Pager może służyć do stronicowania rekordów z dowolnych tabel. Treść klasy Pager jest przedstawiona na listingu 21.2.

Listing 21.2. *Klasa Pager*

```
class Pager
{
    private $klasa = '';
    private $klasaPeer = '';

    private $liczba_wszystkich_rekordow = 0;
    private $liczba_rekordow_na_stronie = 12;
    private $liczba_stron = 0;
    private $numer_strony = 0;
    private $numer_pierwszego_rekordu = 0;
    private $szerokosc = 11;

    private $base_url;
    private $rekordy;
    private $criteria = null;

    public function __construct(
        $klasa, $liczba_rekordow_na_stronie = 10,
        $criteria = null, $szerokosc = 11
    ) {
        if (is_null($criteria)) {
            $this->criteria = new Criteria();
        } else {
            $this->criteria = clone $criteria;
        }

        $this->klasa = $klasa;
        $this->klasaPeer = $klasa . 'Peer';
        $this->liczba_rekordow_na_stronie = $liczba_rekordow_na_stronie;
        $this->przeliczLiczbeRekordow();
        $this->przeliczLiczbeStron();
        $this->szerokosc = $szerokosc;
    }

    public function przeliczLiczbeRekordow()
    {
        $this->liczba_wszystkich_rekordow =
            call_user_func($this->klasaPeer . '::doCount', $this->criteria);
    }

    public function przeliczLiczbeStron()
    {
        $this->liczba_stron = (int)ceil(
            $this->liczba_wszystkich_rekordow / $this->liczba_rekordow_na_stronie
        );
    }

    public function isValidPage($page)
    {
        return str_irevpifr($page, 1, $this->liczba_stron);
    }

    public function setPage($page)
    {
        if ($this->isValidPage($page)) {
```

```

        $this->numer_strony = $page;
        $this->numer_pierwszego_rekordu = ($page - 1) * $this->
            ↳liczba_rekordow_na_stronie;
    }
}

public function getPages($szerokosc = false)
{
    if ($szerokosc) {
        $this->setWidth($szerokosc);
    }

    $polowa = (int)floor($this->szerokosc / 2);
    $minimum = $this->numer_strony;
    $minimum = $minimum - $polowa;
    $minimum = max($minimum, 1);
    $maksimum = $minimum + $this->szerokosc - 1;

    $za_duzo = $maksimum - $this->liczba_stron;
    if ($za_duzo > 0) {
        $minimum = $minimum - $za_duzo;
        $minimum = max($minimum, 1);
    }

    $maksimum = min($minimum + $this->szerokosc - 1, $this->liczba_stron);

    $pages = array();

    for ($i = $minimum; $i <= $maksimum; $i++) {
        $pages[] = $i;
    }
    return $pages;
}

public function assignRecords()
{
    $this->criteria->setLimit($this->liczba_rekordow_na_stronie);
    $this->criteria->setOffset($this->numer_pierwszego_rekordu);

    $tmp = call_user_func($this->klasaPeer . '::doSelect', $this->criteria);

    $i = $this->numer_pierwszego_rekordu + 1;

    $this->rekordy = array();
    foreach ($tmp as $obj) {
        $this->rekordy[$i] = $obj;
        $i++;
    }
}

public function getRecords()
{
    return $this->rekordy;
}

public function setBaseURL($url)
{

```

```
        return $this->base_url = $url;
    }

    public function getPageURL($page)
    {
        if ($page !== false) {
            return $this->base_url . $page;
        } else {
            return '';
        }
    }

    public function getPage()
    {
        return $this->numer_strony;
    }

    public function getFirstPage()
    {
        if ($this->liczba_stron > 0) {
            return 1;
        } else {
            return false;
        }
    }

    public function getLastPage()
    {
        if ($this->liczba_stron > 0) {
            return $this->liczba_stron;
        } else {
            return false;
        }
    }

    public function getPreviousPage()
    {
        if ($this->numer_strony > 1) {
            return $this->numer_strony - 1;
        } else {
            return false;
        }
    }

    public function getNextPage()
    {
        if ($this->numer_strony < $this->liczba_stron) {
            return $this->numer_strony + 1;
        } else {
            return false;
        }
    }

    public function getFirstPageURL()
    {
        return $this->getPageURL($this->getFirstPage());
    }
}
```

```
public function getPreviousPageURL()
{
    return $this->getPageURL($this->getPreviousPage());
}

public function getNextPageURL()
{
    return $this->getPageURL($this->getNextPage());
}

public function getLastPageURL()
{
    return $this->getPageURL($this->getLastPage());
}

public function isPager()
{
    return ($this->liczba_stron > 1);
}

public function isCurrentPage($strona)
{
    return ($strona == $pager->getPage());
}

public function getNbPages()
{
    return $this->liczba_stron;
}

public function leftDots($szerokosc = false)
{
    if ($szerokosc) {
        $this->setWidth($szerokosc);
    }

    $polowa = (int)floor($this->szerokosc / 2);
    $minimum = $this->numer_strony;
    $minimum = $minimum - $polowa;
    $minimum = max($minimum, 1);
    $maksimum = $minimum + $this->szerokosc - 1;

    $za_duzo = $maksimum - $this->liczba_stron;
    if ($za_duzo > 0) {
        $minimum = $minimum - $za_duzo;
        $minimum = max($minimum, 1);
    }

    return ($minimum > 1);
}

public function rightDots($szerokosc = false)
{
    ...
}
}
```

Parametrami konstruktora klasy `Pager` są nazwa klasy poddawanej stronicowaniu, liczba rekordów na stronie, dodatkowe kryteria oraz szerokość wskaźnika. Wywołanie:

```
$pager = new Pager('Song');
```

umożliwi stronicowanie wszystkich rekordów z tabeli `song`. Liczba rekordów na stronie wyniesie 10, a szerokość wskaźnika (mierzona liczbą numerów stron) będzie równa 11. Jeśli chcesz stronicować tylko piosenki rozpoczynające się na literę A, w taki sposób by na stronie pojawiała się 7 rekordów, a szerokość wskaźnika wynosiła trzy numery stron, to wywołaj konstruktor klasy `Pager` następująco:

```
$c = new Criteria();
$c->add(SongPeer::TYTUL, 'A%'. Criteria::LIKE);
$pager = new Pager('Song', 7, $c, 3);
```

W konstruktorze najpierw w razie potrzeby klonujemy kryteria, a następnie we właściwościach `$this->klasa`, `$this->klasaPeer` zapamiętujemy nazwę klasy oraz nazwę klasy `Peer` tabeli poddawanej stronicowaniu. Oprócz zapamiętania w odpowiednich właściwościach otrzymanych parametrów konstruktor odpowiada za przeliczenie rekordów i stron. Zadanie to realizują dwie metody: `przeliczLiczbeRekordow()` oraz `przeliczLiczbeStron()`.

Metoda `przeliczLiczbeRekordow()` musi wyznaczyć liczbę rekordów pasujących do zadanych kryteriów. W treści metody musimy wywołać metodę `doCount()` odpowiedniej klasy `Peer`. Przeliczenie liczby rekordów w tabeli `Song` z wykorzystaniem metody `doCount()` przyjmuje postać:

```
$c = New Criteria();
$x = SongPeer::doCount($c);
```

Nazwę klasy `Peer` ustaliliśmy we właściwości `$this->klasaPeer`, a kryteria — we właściwości `$this->criteria`. W celu wywołania metody o nazwie `x` dla klasy `y` z parametrem `z` należy użyć funkcji `call_user_func()`:

```
$wynik = call_user_func(y, x, z);
```

Treść metody `przeliczLiczbeRekordow()` sprowadza się więc do jednej instrukcji:

```
$this->liczba_wszystkich_rekordow =
    call_user_func($this->klasaPeer . '::doCount', $this->criteria);
```

Liczba stron jest zaokrąglonym w górę wynikiem dzielenia liczby rekordów przez liczbę rekordów na stronie:

```
$this->liczba_stron = (int)ceil(
    $this->liczba_wszystkich_rekordow / $this->liczba_rekordow_na_stronie
);
```

W ten sposób mamy ustaloną liczbę wszystkich rekordów oraz liczbę stron.

Numeracja stron będzie zawsze rozpoczynała się od 1 i kończyła na numerze ostatniej strony. Zatem walidacja numeru strony realizowana w metodzie `isValidPage()` będzie wykonana przy użyciu funkcji `str_ievpifr()`:

```
return str_ievpifr($page, 1, $this->liczba_stron);
```

Numer bieżącej strony ustalamy metodą `setPage()`. Metoda ta otrzymany parametr `$page` poddaje walidacji, po czym w razie powodzenia zapamiętuje numer strony oraz wyznacza numer pierwszego rekordu. Numer pierwszego rekordu wynika z liczby rekordów na stronie:

```
$this->numer_pierwszego_rekordu = ($page - 1) * $this->liczba_rekordow_na_stronie;
```

Kolejna metoda, `getPages()`, zwraca numery stron, które należy wyświetlić na bieżącej, tj. ustalonej metodą `setPage()`, stronie. W metodzie tej ustalamy dwie zmienne: `$minimum` oraz `$maksimum`. Zwracanym wynikiem jest tablica wartości całkowitych od minimum do maksimum.

Wartość minimum jest numerem pierwszej strony przesuniętym o połowę szerokości wskaźnika, jednak nigdy nie mniejszym od 1:

```
$minimum = $this->numer_strony;  
$minimum = $minimum - $polowa;  
$minimum = max($minimum, 1);
```

Wartość maksimum otrzymujemy przez dodanie szerokości wskaźnika do minimum:

```
$maksimum = $minimum + $this->szerokosc - 1;
```

Musimy sprawdzić, czy wartość maksimum nie wyskoczyła poza numer ostatniej strony. W zmiennej `$za_duzo` ustalamy, o ile wartość maksimum przekracza numer bieżącej strony:

```
$za_duzo = $maksimum - $this->liczba_stron;
```

Jeśli otrzymamy liczbę dodatnią, to musimy przesunąć wartość minimalną w lewo o otrzymaną nadwyżkę, ciągle gwarantując, że minimum nie jest mniejsze od 1:

```
if ($za_duzo > 0) {  
    $minimum = $minimum - $za_duzo;  
    $minimum = max($minimum, 1);  
}
```

Ostatnim etapem jest ustalenie maksimum dla nowej wartości minimum. Tym razem wartość maksimum ustalamy tak, by nie przekroczyć liczby stron:

```
$maksimum = min($minimum + $this->szerokosc - 1, $this->liczba_stron);
```

Wyznaczone wartości maksimum i minimum podlegają wymogom opisanym we wstępie:

- ♦ numeracja stron zawsze rozpoczyna się od 1 i dochodzi do numeru ostatniej strony;
- ♦ numery stron są podawane w taki sposób, by — jeśli to możliwe — numer bieżącej strony był w środku;
- ♦ jeśli liczba wszystkich stron jest większa od szerokości, to liczba podawanych stron jest zawsze równa szerokości wskaźnika;
- ♦ jeśli szerokość wskaźnika jest większa lub równa liczbie dostępnych stron, to podajemy numery wszystkich stron.

Wynikiem funkcji `getPages()` jest tablica zawierająca numery stron.

Kolejnymi ważnymi metodami są metody `assignRecords()` oraz `getRecords()`. Metody te zapewniają dostęp do rekordów, które należy wyświetlić na bieżącej stronie.

Metoda `getRecords()` najpierw modyfikuje kryteria wyboru rekordów, dołączając ograniczenia ilościowe:

```
$this->criteria->setLimit($this->liczba_rekordow_na_stronie);
$this->criteria->setOffset($this->numer_pierwszego_rekordu);
```

Powyższe kryteria gwarantują, że z bazy danych pobierzemy co najwyżej `liczba_rekordow_na_stronie` rekordów oraz że numer pierwszego pobieranego rekordu wyniesie `numer_pierwszego_rekordu`.

Rekordy pobieramy, wywołując metodę `doSelect()` klasy `Peer`, której nazwa jest zawarta we właściwości `$this->klasaPeer`:

```
$tmp = call_user_func($this->klasaPeer . '::doSelect', $this->criteria);
```

Otrzymane rekordy przepisujemy do tablicy `$this->rekordy`:

```
$i = $this->numer_pierwszego_rekordu + 1;

$this->rekordy = array();
foreach ($tmp as $obj) {
    $this->rekordy[$i] = $obj;
    $i++;
}
```

Zauważ, że indeksacja w tablicy `$this->rekordy` rozpoczyna się od wartości `numer_pierwszego_rekordu + 1`. Dzięki temu tabelka HTML prezentująca rekordy na stronie WWW będzie mogła zawierać numerację rekordów.

Zadaniem metody `getRecords()` jest tylko udostępnienie prywatnej tablicy `$this->rekordy`.

Metody `setBaseUrl()` oraz `getPageURL()` ułatwiają operowanie adresami URL do kolejnych stron. Metodą `setBaseUrl()` ustalamy adres bazowy kolejnych stron. Adresy stron powstają przez dopisanie na końcu adresu bazowego numeru bieżącej strony. Wywołanie metody `getPageURL(5)` z parametrem 5 zwróci adres URL piątej strony.

Metody `getPage()`, `getFirstPage()`, `getPreviousPage()`, `getNextPage()` oraz `getLastPage()` zwracają numer bieżącej, pierwszej, poprzedniej, następnej oraz ostatniej strony. Odpowiadające im metody `getFirstPageURL()`, `getPreviousPageURL()`, `getNextPageURL()` oraz `getLastPageURL()` udostępniają ich adresy URL.

Metody pomocnicze `isPager()` oraz `isCurrentPage()` ułatwiają operowanie wskaźnikiem stronicowania. Pierwsza z nich odpowiada, czy liczba stron jest większa od 1, a druga, czy podany numer strony jest identyczny jak numer strony bieżącej.

Metoda `getNbPages()` udostępnia liczbę stron.

Ostatnimi metodami pomocniczymi są metody `leftDots()` oraz `rightDots()`. Zwracają one informację logiczną mówiącą o tym, czy znajdujemy się przy lewej lub prawej krawędzi wskaźnika. Jeśli `leftDots()` zwraca wartość `true`, to oznacza, że numer minimalnej strony jest większy od 1 i w widoku wskaźnika stronicowania należy umieścić wykrótkowanie postaci:

```
...7 8 9
```

W podobny sposób funkcja `rightDots()` stwierdza konieczność wyświetlania wykrótkowania z prawej strony:

```
7 8 9...
```

Widok wskaźnika stronicowania

Dzięki sparametryzowaniu wskaźnika stronicowania nazwą klasy, kryteriami, szerokością, liczbą rekordów na stronie oraz bazowym adresem URL ten sam widok częściowy `_pager.html` będzie wykorzystany do stronicowania dowolnych rekordów. Widok częściowy `_pager.html` jest przedstawiony na listingu 21.3.

Listing 21.3. Widok częściowy `_pager.html`

```
<?php if (isset($pager) && $pager->isPager()): ?>
    <div class="pager">
        <?php if ($pager->getPreviousPage()): ?>
            <a href="<?php echo $pager->getFirstPageURL(); ?>">FIRST</a>
            <a href="<?php echo $pager->getPreviousPageURL(); ?>">PREV</a>
        <?php else: ?>
            FIRST
            PREV
        <?php endif: ?>

        <?php if ($pager->leftDots()): ?>
            ...
        <?php endif: ?>

        <?php foreach ($pager->getPages(8) as $strona): ?>
            <?php if (!$pager->isCurrentPage($strona)): ?>
                <a href="<?php echo $pager->getPageURL($strona); ?>">
                    <?php echo $strona; ?>
                </a>
            <?php else: ?>
                <strong><?php echo $strona; ?></strong>
            <?php endif: ?>
        <?php endforeach: ?>

        <?php if ($pager->rightDots()): ?>
            ...
        <?php endif: ?>

        <?php if ($pager->getNextPage()): ?>
            <a href="<?php echo $pager->getNextPageURL(); ?>">NEXT</a>
            <a href="<?php echo $pager->getLastPageURL(); ?>">LAST</a>
        <?php else: ?>
```

```

        NEXT
        LAST
    <?php endif; ?>
</div>
<?php endif; ?>

```

Widok rozpoczynamy od sprawdzenia, czy wskaźnik stronicowania jest konieczny. Wskaźnik wyświetlamy tylko wówczas, gdy liczba stron jest większa od 1:

```

<?php if (isset($pager) && $pager->isPager()): ?>
    ...
<?php endif; ?>

```

Cały wskaźnik jest zawarty w elemencie `div` o identyfikatorze `pager`. Umieszczamy w nim pięć grup elementów:

- ♦ wskaźniki do pierwszego i poprzedniego rekordu;
- ♦ lewy wielokropek;
- ♦ numery stron;
- ♦ prawy wielokropek;
- ♦ wskaźnik następnego i ostatniego rekordu.

Odsyłacze do pierwszego i poprzedniego rekordu wyświetlamy pod warunkiem, że poprzednia strona jest dostępna. Jeśli tak, to widok będzie zawierał dwa hiperłącza, a jeśli nie — napisy:

```

<?php if ($pager->getPreviousPage()): ?>
    <a href="<?php echo $pager->getFirstPageURL(); ?>">FIRST</a>
    <a href="<?php echo $pager->getPreviousPageURL(); ?>">PREV</a>
<?php else; ?>
    FIRST
    PREV
<?php endif; ?>

```

Adresy URL zawarte w hiperłączach są zwracane przez metody `getFirstPageURL()` oraz `getPreviousPageURL()`, co uniezależnia widok `_pager.html` od postaci adresów URL.

O wyświetlaniu lewego wielokropka decyduje metoda `leftDots()`:

```

<?php if ($pager->leftDots()): ?>
    ...
<?php endif; ?>

```

Najbardziej skomplikowanym fragmentem widoku `_pager.html` jest pętla `foreach` produkująca numery stron. Pętla ta przetwarza numery stron zwracane przez metodę `getPages()`. Parametrem tej metody jest szerokość wskaźnika. Jeśli podamy wartość 7, to otrzymamy wskaźnik o szerokości 7:

```

<?php foreach ($pager->getPages(7) as $strona): ?>
    <?php if (!$pager->isCurrentPage($strona)): ?>
        <strong><?php echo $strona; ?></strong>
    <?php else; ?>

```

```

        <a href="<?php echo $pager->getPageURL($strona); ?>">
            <?php echo $strona; ?>
        </a>
    <?php endif; ?>
<?php endforeach; ?>

```

W pętli `foreach` przetwarzamy numery stron zwrócone przez metodę `getPages()`. Sprawdzamy, czy numer kolejnej strony jest równy numerowi strony bieżącej. Jeśli tak, to drukujemy numer strony bieżącej ujęty w znaczniki `<strong></strong>`, a jeśli nie, drukujemy hiperłącze do strony o zadanym numerze.

Drukowanie prawego wielokropka, podobnie jak wielokropka z lewej strony, jest zabezpieczone instrukcją `if`:

```

<?php if ($pager->rightDots()): ?>
    ...
<?php endif; ?>

```

a wydruk hiperłączy do następnej i ostatniej strony realizujemy analogicznie jak wydruk hiperłączy *FIRST* i *PREV*.

Oczywiście w miejsce napisów *FIRST*, *PREV*, *NEXT*, *LAST* możemy użyć wskaźników graficznych. Każdy wskaźnik należy wykonać w dwóch wersjach: jako aktywny i jako nieaktywny. Jeśli obrazy dla opcji *FIRST* zapiszemy w plikach *first.png* oraz *first-brak.png* w folderze *aplikacja/www/img/*, wówczas wydruk hiperłącza do pierwszego rekordu przyjmie postać:

```

<a href="<?php echo $pager->getFirstPageURL(); ?>">
    
</a>

```

a etykietę *FIRST* prezentowaną, gdy przycisk *FIRST* jest nieaktywny, wykonamy jako:

```



```

W identyczny sposób wykonujemy graficzne wersje odsyłaczy *PREV*, *NEXT*, *LAST*.

Prezentacja rekordów poddanych stronicowaniu

Omawiana aplikacja zawiera tabele *artist*, *rekord*, *song* oraz *year*. Stronicowanie rekordów umieścimy najpierw w akcjach prezentujących zestawienie wszystkich rekordów, czyli w akcjach:

- ♦ *artist/list*,
- ♦ *rekord/list*,
- ♦ *song/list*,
- ♦ *year/list*.

Metoda akcji *artist/list* jest przedstawiona na listingu 21.4.

Listing 21.4. *Metoda akcji artist/list*

```

public function execute_list()
{
    if (isset($_GET['page'])) {
        if (str_iregi($_GET['page'])) {
            $strona = $_GET['page'];
        } else {
            $this->execute_404();
            return;
        }
    } else {
        $strona = '1';
    }

    $pager = new Pager('Artist', 10);

    if ($pager->isValidPage($strona)) {
        $pager->setPage($strona);
        $pager->assignRecords();
        $pager->setBaseURL('index.php?module=artist&action=list&page=');
        $this->set('pager', $pager);
    } else {
        $this->execute_404();
    }
}

```

Najpierw sprawdzamy, czy zmienna `$_GET['page']` została podana i jeśli tak, to czy ma poprawną wartość. Podany numer strony przypisujemy do zmiennej `$strona`. Jeśli zmienna `$_GET['page']` nie została podana, to zmiennej `$strona` przypisujemy wartość domyślną 1.

Wskaźnik stronicowania tworzymy jako obiekt klasy `Pager`, podając jako parametr nazwę klasy stronicowanych obiektów:

```
$pager = new Pager('Artist', 10);
```

Po utworzeniu wskaźnika sprawdzamy poprawność zmiennej `$strona`:

```

if ($pager->isValidPage($strona)) {
    ...
} else {
    $this->execute_404();
}

```

Jeśli zmienna ta jest poprawna, to we wskaźniku ustalamy numer bieżącej strony:

```
$pager->setPage($strona);
```

po czym pobieramy rekordy z bazy danych:

```
$pager->assignRecords();
```

Następnie ustalamy bazowy adres URL:

```
$pager->setBaseURL('index.php?module=artist&action=list&page=');
```

i przekazujemy wskaźnik stronicowania do widoku:

```
$this->set('pager', $pager);
```

Zwróć uwagę, że jedyną zmienną przekazywaną do widoku akcji `artist/list` jest obiekt `$pager`. Dostęp do rekordów, które mają być wyświetlone na bieżącej stronie WWW, zapewnia metoda `getRecords()` klasy `Pager`.

Widok akcji `artist/list` jest przedstawiony na listingu 21.5.

Listing 21.5. *Widok akcji `artist/list`*

```
<?php echo partial('../templates/_pager.html', array('pager' => $pager)); ?>
<table>
<tr>
  <th class="hash">#</th>
  <th>Artist</th>
</tr>
<?php foreach ($pager->getRecords() as $k => $a): ?>
<tr>
  <td><?php echo $k; ?>.</td>
  <td>
    <a href="index.php?module=artist&action=show&slug=<?php echo $a->getSlug(); ?>">
      <?php echo $a; ?>
    </a>
  </td>
</tr>
<?php endforeach; ?>
</table>
<?php echo partial('../templates/_pager.html', array('pager' => $pager)); ?>
```

Nad oraz pod tabelką HTML z rekordami umieszczamy wskaźnik prezentujący numery stron. Zadanie to wykonują instrukcje:

```
<?php echo partial('../templates/_pager.html', array('pager' => $pager)); ?>
```

Zawartość tabelki HTML powstaje w pętli przetwarzającej rekordy dostępne na bieżącej stronie:

```
<?php foreach ($pager->getRecords() as $k => $a): ?>
  ...
<?php endforeach; ?>
```

Pętla `foreach` jest sterowana zmiennymi `$k` oraz `$a`. Indeks `$k` będzie zawierał indeksy z tablicy `$this->kolumny` ustalone w metodzie `assignRecords()` klasy `Pager`. W ten sposób wydruk numerów rekordów (tj. wartości z kolumny `lp.`) przyjmie postać:

```
<td><?php echo $k; ?>.</td>
```

natomiast instrukcje:

```
<a href="index.php?module=artist&action=show&slug=<?php echo $a->getSlug(); ?>">
  <?php echo $a; ?>
</a>
```

wydrukują hiperłącze zawierające nazwę wykonawcy. Dzięki temu, że zmienna `$a` jest obiektem klasy `Artist`, mamy pełny dostęp do informacji o artyście.

Stronicowanie rekordów w akcjach `rekord/list`, `song/list` oraz `year/list` przebiega niemal identycznie jak w akcji `artist/list`.

Kontekstowe stronicowanie rekordów

Klasa `Pager` umożliwia stosowanie dowolnych kryteriów wyboru rekordów, dzięki czemu rekordy poddawane stronicowaniu możemy filtrować. W akcji `record/list` stronicowaniu poddajemy listę wszystkich płyt zawartych w tabeli `rekord`. Jeśli aplikację wzbogacimy o kontekstowe wyświetlanie danych, to na stronie akcji `show` dla tabeli `artist` staniemy przed zadaniem wyświetlenia listy płyt wybranego wykonawcy. Lista ta również może wymagać stronicowania. W jaki sposób wykonać stronicowanie płyt wybranego artysty? Wystarczy wykorzystać kryteria.

Metoda akcji `artist/show` jest przedstawiona na listingu 21.6.

Listing 21.6. *Metoda akcji `artist/show`, której zadaniem jest prezentacja stronicowanej listy płyt zadanego wykonawcy*

```
public function execute_show()
{
    if (
        isset($_GET['slug']) &&
        str_ireplace($_GET['slug']) &&
        ($artist = ArtistPeer::retrieveBySlug($_GET['slug']))
    ) {

        $this->set('artist', $artist);

        if (isset($_GET['page'])) {
            if (str_ireplace($_GET['page'])) {
                $strona = $_GET['page'];
            } else {
                $this->execute_404();
                return;
            }
        } else {
            $strona = '1';
        }

        $c = new Criteria();
        $c->add(RecordPeer::ARTIST_ID, $artist->getArtistId());

        $pager = new Pager('Record', 5, $c);

        if ($pager->isValidPage($strona)) {
            $pager->setPage($strona);
            $pager->assignRecords();
            $pager->setBaseUrl(
                'index.php?module=artist&action=show&slug=' .
                $artist->getSlug() . '&page='
            );
            $this->set('pager', $pager);
        } else {
            $this->execute_404();
        }
    }
}
```

```

    }
    } else {
        $this->execute_404();
    }
}

```

W akcji `artist/show` najpierw tworzymy obiekt `$artist` reprezentujący wybranego wykonawcę. Obiekt ten przekazujemy do widoku. Następnie identycznie jak w akcjach `list` poddajemy walidacji zmienną `$_GET['page']`. Po ustaleniu wartości zmiennej `$strona` przechodzimy do utworzenia wskaźnika stronicowania. W kryteriach dodajemy warunek:

```

$c = new Criteria();
$c->add(RecordPeer::ARTIST_ID, $artist->getArtistId());

```

który zagwarantuje, że rekordy pobierane z tabeli rekord będą płytami wybranego artysty. Utworzone kryteria przekazujemy do konstruktora wskaźnika:

```

$pager = new Pager('Record', 5, $c);

```

Cały pozostały kod akcji `artist/show` jest identyczny jak kod akcji `artist/list`.

Widok akcji `artist/show` wykonujemy, wykorzystując jako widok częściowy widok `modules/rekord/list.html`. Drukujemy nazwę artysty:

```

<h3><?php echo $artist; ?></h3>

```

a następnie listę jego płyt:

```

<?php echo partial('../modules/rekord/list.html', array('pager' => $pager)); ?>

```

W ten sposób w akcjach `show` nie musimy przygotowywać nowych widoków do prezentacji listy rekordów. Akcje `list` przygotowaliśmy w taki sposób, że możemy ich użyć do wyświetlenia rekordów spełniających dowolne kryteria.

Ćwiczenie 21.1

Wykonaj projekt 21.1. Użyj szablonów PHP oraz oprogramowania Propel. Pracę rozpocznij od pliku *cw-21-01-start.zip*. Zadanie wykonaj bez stronicowania.

Ćwiczenie 21.2

Wykonaj projekt 21.1. Wszystkie prezentowane listy rekordów poddaj stronicowaniu. Użyj szablonów PHP oraz oprogramowania Propel. Pracę rozpocznij od pliku *cw-21-02-start.zip*.