Technologia i rozwiązania

Magento
Przewodnik dla programistów PHP

Poznaj i rozbuduj możliwości Magento!

Helion

Allan MacGregor

PACKT PUBLISHING

Tytuł oryginału: Magento PHP Developer's Guide

Tłumaczenie: Daniel Kaczmarek

ISBN: 978-83-246-8940-8

First published in the English language under the title „Magento PHP Developer's Guide".

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: http://helion.pl (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/magphp
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

O autorze

Allan McGregor posiada certyfikat Magento Certified Developer Plus i od czterech lat pracuje z Magento. Uzyskał także certyfikat Linux System Administration wydany przez firmę IBM.

Swoją przygodę z Magento rozpoczął jako samodzielny programista, który szukał najlepszego narzędzia do tworzenia rozwiązań e-commerce'owych. Obecnie pracuje jako główny programista Magento w firmie Demac Media (*www.demacmedia.com*). Allan jest też pasjonatem programowania, stale poszukującym nowych, lepszych technologii i narzędzi programistycznych.

W Demac Media Allan współtworzył rozwiązania dla różnorodnych klientów. Dzięki temu zdobył doświadczenie oraz wiedzę, która pozwala mu stawiać czoła nawet najtrudniejszym wyzwaniom związanym z wykorzystaniem Magento.

W ramach jednego z projektów wewnętrznych prowadzonych w Demac Media Allan tworzył narzędzie *Triplecheck.io* (*http://triplecheck.io*) — pionierską usługę, która monitoruje i audytuje poprawność kodu źródłowego sklepu stworzonego w Magento. Wpisy McGregora można śledzić na Twitterze pod adresem: *http://www.twitter.com/allanmcgregor*.

Praca nad tą książką była dla mnie ogromnym wyzwaniem, które jednak w pełni się opłaciło. W trakcie pisania dowiedziałem się wielu nowych rzeczy na temat Magento, a także kilku o samym sobie — zarówno o człowieku, jak i o programiście.

W pierwszej kolejności chcę podziękować mojej wspaniałej żonie za jej bezwarunkowe wsparcie i zrozumienie, które okazuje mi, gdy pracuję nad coraz to nowymi projektami.

Dziękuję też Matthew Bertulliemu i Dimitriemu Colomvakosowi, współzałożycielom Demac Media, za wsparcie, które mi okazywali.

Michael Krietzer i Corey Slavnik, moi przyjaciele i współpracownicy, z wielką ochotą poświęcali swój wolny czas, aby zredagować tę książkę.

Specjalne podziękowania kieruję do całej rodziny Demac Media.

To, co udało mi się zdobyć, osiągnąłem dzięki Wam.

3

ROZDZIAŁ

ORM i kolekcje danych

Kolekcje i modele to chleb powszedni dla wszystkich programistów, którzy pracują z Magento. W tym rozdziale opisany zostanie system ORM obecny w Magento. Pokażę także, jak należy prawidłowo korzystać z kolekcji danych oraz z systemu EAV.

Magento, podobnie jak większość współczesnych systemów, implementuje system **mapowania obiektowo-relacyjnego** (ang. *object-relational mapping* — ORM).

> *Mapowanie obiektowo-relacyjne (ORM, O/RM lub mapowanie O/R) w technologiach informatycznych to technika programowania, w której dane występujące w różnych, niezgodnych formatach przekształcane są na języki programowania zorientowanego obiektowo. W ten sposób tworzy się „bazę danych wirtualnych obiektów", której można używać w konstrukcjach języka programowania.*

W tym rozdziale przedstawione zostaną następujące zagadnienia:

- modele Magento;
- struktura modelu danych Magento;
- EAV i modele EAV;
- wykorzystanie bezpośrednich zapytań języka SQL.

W rozdziale wykorzystane zostaną również liczne fragmenty przykładowego kodu źródłowego, na podstawie których łatwiej będzie zrozumieć sposób działania Magento.

Uruchomienie przykładowych kodów prezentowanych w tym rozdziale wymaga domyślnej instalacji Magento na maszynie VagrantBox lub instalacji Magento z danymi przykładowymi.

Dla celów tego rozdziału stworzyłem **interaktywną konsolę Magento** (ang. *Interactive Magento Console* — IMC), która jest skryptem powłoki zaimplementowanym specjalnie na potrzeby tej książki. Inspiracją dla niej jest przeznaczona dla Ruby **interaktywna konsola Ruby** (ang. *Interactive Ruby Console* — IRC). W celu uruchomienia IMC należy wykonać następujące czynności.

1. Zainstalować IMC. W tym celu trzeba pobrać pliki źródłowe ze strony: *https://github.com/amacgregor/mdg_imc* i rozpakować je w testowej instalacji Magento. IMC jest prostym skryptem powłoki Magento, który pozwoli nam testować kod w czasie rzeczywistym.
2. Po rozpakowaniu skryptu należy zalogować się w powłoce maszyny wirtualnej.
3. W kolejnym kroku trzeba przejść do głównego folderu Magento. Jeżeli korzysta się z domyślnej maszyny Vagrant, instalacja jest już w niej obecna. Folderem głównym jest */srv/www/ce1720/public_html/*, a aby do niego przejść, należy wpisać w wierszu poleceń następujące polecenie:

```
$ cd /srv/www/ce1720/public_html
```

4. Na koniec można uruchomić IMC następującym poleceniem:

```
$ php shell/imc.php
```

5. Jeżeli instalacja przebiegła poprawnie, nowy wiersz w wierszu poleceń powinien zaczynać się symbolem `magento >`.

Struktura modelu Magento

Jak powiedziano w poprzednim rozdziale, modele danych Magento służą do manipulowania danymi i ich odczytywania. Warstwa modeli podzielona jest na dwa podstawowe typy: modele proste i modele EAV.

- **Modele proste.** Tego typu implementacje modeli są zwykłymi odwzorowaniami jednego obiektu na jedną tabelę, co oznacza, że atrybuty obiektu odpowiadają każdemu polu oraz strukturze tabeli.
- **Modele encja – atrybut – wartość (EAV).** W modelach tego rodzaju encje opisuje się atrybutami o zmiennej liczbie.

> Należy podkreślić, że nie wszystkie modele Magento używają systemu ORM lub rozszerzają jego możliwości. Obserwatory są doskonałym przykładem prostych klas modeli, które to modele nie są odwzorowane na konkretną tabelę lub encję bazy danych.

Dodatkowo każdy typ modelu jest kształtowany przez następujące warstwy.

- **Klasa modelu.** W niej implementuje się logikę biznesową. Modele służą do manipulowania danymi, lecz nie mają bezpośredniego dostępu do tych danych.

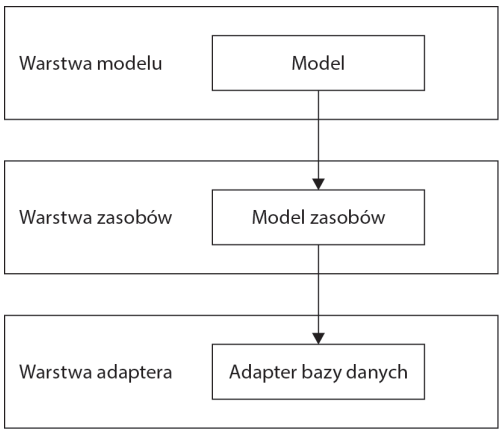
- **Klasa modelu zasobów.** Modele zasobów komunikują się z bazą danych w imieniu modeli. Modele zasobów wykonują wszelkie operacje typu CRUD.
- **Klasa modelu kolekcji.** Każdy model danych zawiera klasę kolekcji. Kolekcje są obiektami, które przechowują jedną lub więcej instancji modelu Magento.

> CRUD oznacza cztery podstawowe operacje na danych w bazie danych: *Create* (tworzenie), *Read* (odczyt), *Update* (zmiana), *Delete* (usuwanie).

Modele Magento nie zawierają żadnej logiki komunikacji z bazą danych — wręcz są one niezależne od bazy danych. Odpowiedni kod implementuje się w warstwie modelu zasobów.

Dzięki opisanej konstrukcji Magento może obsługiwać różne rodzaje baz danych i platform. Wprawdzie na chwilę obecną oficjalnie obsługiwany jest jedynie serwer MySQL, bez trudu można jednak napisać nową klasę zasobu przeznaczoną dla nowej bazy danych, która to klasa nie będzie w żaden sposób wpływać na logikę modeli.

Schemat struktury modeli Magento przedstawiono na rysunku 3.1.



Rysunek 3.1. Schemat struktury modeli Magento

Wykonajmy zatem pewien eksperyment, który będzie polegał na stworzeniu instancji obiektu produktu i ustawieniu jego wybranych atrybutów. Należy w tym celu wykonać następujące czynności.

1. Uruchomić interaktywną konsolę Magento w głównym folderze rozwojowej instalacji narzędzia:

```
php shell/imc.php
```

2. Pierwszy krok polega na stworzeniu nowej instancji obiektu produktu, do czego służy następujące polecenie:

```
magento> $product = Mage::getModel('catalog/product');
```

3. Poniższym poleceniem potwierdzimy, że jest to pusta instancja klasy produktu:

```
magento> echo get_class($product);
```

4. Jeżeli wszystko pójdzie dobrze, w konsoli powinien pojawić się następujący komunikat:

```
magento> Magento_Catalog_Model_Product
```

5. Aby dowiedzieć się więcej na temat metod klasy, można wykonać polecenie o poniższej treści:

```
magento> print_r(get_class_methods($product));
```

W efekcie zwrócona zostanie tablica, w której widnieć będą wszystkie metody udostępniane przez klasę. Spróbujmy zatem wykonać poniższy fragment kodu źródłowego, aby zmodyfikować cenę i nazwę produktu:

```
$product = Mage::getModel('catalog/product')->load(2);
$name    = $product->getName() . '-TEST';
$price   = $product->getPrice();
$product->setPrice($price + 15);
$product->setName($name);
$product->save();
```

W pierwszym wierszu przykładowego kodu źródłowego tworzona jest instancja wskazanego obiektu, po czym odczytywana jest wartość atrybutu obiektu, w którym zapisana jest nazwa produktu. Następnie ustawiana jest cena i nazwa, po czym obiekt zostaje zapisany.

Analiza implementacji klasy produktu Magento `Mage_Catalog_Model_Product` szybko wykaże, że o ile funkcje `getName()` i `getPrice()` są w niej zaimplementowane, o tyle już definicji funkcji `setPrice()` i `setName()` w niej nie ma.

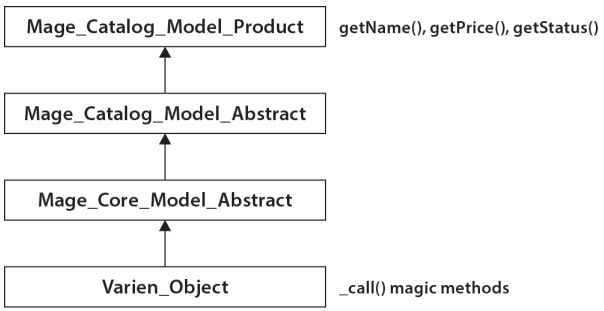
Powstaje zatem kluczowe pytanie: w jakiż to magiczny sposób Magento definiuje określone metody ustawiania i odczytywania danych w obiekcie produktu? Wprawdzie `getPrice()` i `getName()` są jawnie zaimplementowane, jednak nigdzie nie ma definicji metod ustawiających i odczytujących inne atrybuty produktu, takie jak kolor albo nazwa producenta.

Metody magiczne

Cóż, rzeczywiście zdarza się, że działanie systemu ORM Magento ociera się o magię. Mówiąc bardziej precyzyjnie: w systemie ORM wykorzystuje się jeden z najciekawszych mechanizmów dostępnych w PHP, który umożliwia definiowanie metod ustawiających i odczytujących dane — mechanizm ten opiera się na magicznej metodzie `__call()`. Dzięki niej metod Magento można używać do ustawiania, usuwania, sprawdzania i odczytywania danych.

Gdy podjęta zostanie próba wywołania metody, która nie jest zaimplementowana w klasie, PHP zacznie szukać w klasach rodziców deklaracji tej metody. Jeżeli odpowiednia funkcja nie zostanie znaleziona w żadnej z klas rodziców, podjęta zostanie ostatnia próba, która polega na wywołaniu metody `__call()`. Jeżeli funkcja zostanie znaleziona, Magento (a właściwie PHP) wywoła magiczną metodę i przekaże do niej nazwę pierwotnie wywoływanej metody oraz jej argumenty.

Model `Product` nie ma zdefiniowanej metody `__call()`, lecz dziedziczy ją z klasy `Varien_Object`, która jest klasą podstawową dla wszystkich modeli Magento. Drzewo dziedziczenia dla klasy `Mage_Catalog_Model_Product` znajduje się na schemacie przedstawionym na rysunku 3.2.



Rysunek 3.2. Drzewo dziedziczenia dla klasy Mage_Catalog_Model_Product

> Każdy model Magento dziedziczy po klasie `Varien_Object`.

Przyjrzyjmy się bliżej klasie `Varien_Object`. W tym celu należy wykonać następujące czynności.

1. Otworzyć plik *folder_główny_magento/lib/Varien/Object.php*.
2. Klasa `Varien_Object` ma zdefiniowaną metodę `__call()`, a także implementuje dwie przestarzałe metody: `__set()` i `__get()`. Te dwie ostatnie metody są zastąpione metodą `__call()` i dlatego już się ich nie używa.

```
public function __call($method, $args)
{
   switch (substr($method, 0, 3)) {
       case 'get' :
           //Varien_Profiler::start('GETTER: '.get_class($this).'::'.$method);
           $key = $this->_underscore(substr($method,3));
           $data = $this->getData($key, isset($args[0]) ? $args[0] : null);
           //Varien_Profiler::stop('GETTER: '.get_class($this).'::'.$method);
           return $data;
       case 'set' :
           //Varien_Profiler::start('SETTER: '.get_class($this).'::'.$method);
           $key = $this->_underscore(substr($method,3));
           $result = $this->setData($key, isset($args[0]) ? $args[0] : null);
           //Varien_Profiler::stop('SETTER: '.get_class($this).'::'.$method);
           return $result;
       case 'uns' :
           //Varien_Profiler::start('UNS: '.get_class($this).'::'.$method);
           $key = $this->_underscore(substr($method,3));
           $result = $this->unsetData($key);
           //Varien_Profiler::stop('UNS: '.get_class($this).'::'.$method);
           return $result;
       case 'has' :
```

```
            //Varien_Profiler::start('HAS: '.get_class($this).'::'.$method);
            $key = $this->_underscore(substr($method,3));
            //Varien_Profiler::stop('HAS: '.get_class($this).'::'.$method);
            return isset($this->_data[$key]);
        }
        throw new Varien_Exception("Invalid method" .
    get_class($this)."::".$method."(".print_r($args,1).")");
    }
```

W metodzie `__call()` znajduje się instrukcja `switch`, która obsługuje nie tylko funkcje ustawiania (`set`) i odczytywania (`get`) danych, ale również funkcje `unset` i `has`.

Po uruchomieniu debuggera i prześledzeniu wywołań metody `__call()` w przykładowym fragmencie kodu okaże się, że przyjmuje ona dwa argumenty: nazwę metody (na przykład `setName()`) oraz argumenty pochodzące z wywołania oryginalnego.

Co ciekawe, Magento próbuje zidentyfikować typ metody na podstawie pierwszych trzech liter nazwy metody wywoływanej. Operacja ta zachodzi w momencie, gdy instrukcja `switch` wykonuje funkcję `substr()`:

```
substr($method, 0, 3)
```

Pierwszą czynnością wykonywaną w każdym przypadku analizowanym przez instrukcję `switch` jest wykonanie funkcji `_underscore()`, która przyjmuje parametr w postaci reszty znaków nazwy metody oprócz trzech pierwszych liter. W naszym przykładzie argumentem dla `_underscore()` będzie `Name`.

Funkcja `_underscore()` zwraca klucz danych. Klucz ten jest wykorzystywany w każdym przypadku analizowanym przez instrukcję, aby wykonać odpowiednie operacje na danych. Istnieją cztery podstawowe operacje na danych i każda z nich jest wywoływana w odpowiadającym jej przypadku instrukcji `switch`:

- `setData($parameters)`,
- `getData($parameters)`,
- `unsetData($parameters)`,
- `isset($parameters)`.

Każda z wymienionych funkcji wykonuje odpowiednie dla niej operacje na tablicy danych klasy `Varien_Object`. W większości przypadków wywoływana jest magiczna metoda `set/get`, która wykonuje odpowiednie czynności na atrybutach obiektu. Istnieje tylko kilka wyjątków od tej reguły — na przykład gdy wymagana jest dodatkowa logika biznesowa, metody ustawiania i odczytywania danych są definiowane jawnie. W naszym przykładzie takimi metodami są `getName()` i `getPrice()`:

```
public function getPrice()
{
    if ($this->_calculatePrice || !$this->getData('price')) {
        return $this->getPriceModel()->getPrice($this);
```

```
        } else {
            return $this->getData('price');
        }
    }
```

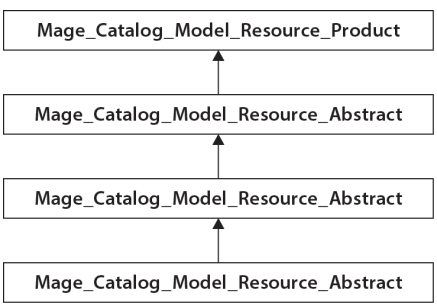
Nie będziemy się na razie wgłębiać w szczegóły działania funkcji `getPrice()`. Na jej podstawie widać jednak wyraźnie, że dla niektórych części modelu konieczne może być zaimplementowanie dodatkowej logiki:

```
public function getName()
{
    return $this->_getData('name');
}
```

Natomiast metoda `getName()` nie została jawnie zaimplementowana po to, by realizować dodatkową logikę biznesową, ale po to, by zoptymalizować działanie kluczowego elementu Magento. Funkcja `getName()` klasy `Mage_Catalog_Model_Product` może być teoretycznie wykonywana setki razy przy każdym ładowaniu strony i jest jedną z najczęściej używanych w całym Magento. W końcu czym byłaby platforma e-commerce'owa, gdyby nie skupiała się na produktach?

Zarówno w interfejsie użytkownika, jak i w modułach wewnętrznych funkcja `getName()` zostanie prędzej czy później wywołana. Na przykład jeżeli ładujemy stronę kategorii z 24 produktami, oznacza to konieczność wykonania 24 niezależnych wywołań funkcji `getName()` i w każdym z tych wywołań poszukiwana będzie metoda `getName()` na każdej klasie rodzica; następnie podjęta zostanie próba wykonania magicznej metody `__call()`. Ostatecznie cały proces może zająć długie milisekundy.

Modele zasobów zawierają kompletną logikę komunikacji z bazą danych i tworzą instancje wymaganych adapterów odczytywania danych i zapisywania ich do odpowiadających im źródeł danych. Wróćmy do przykładu z produktami i spójrzmy na model zasobów produktów z rysunku 3.3, zlokalizowany w klasie `Mage_Catalog_Model_Resource_Product`.



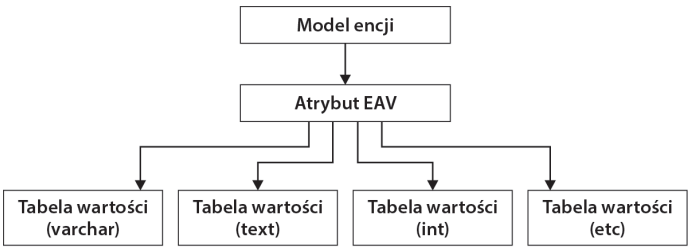
Rysunek 3.3. Model zasobów produktów

Modele zasobów występują w dwóch odmianach: Entity oraz MySQL4. Drugi z nich jest standardową implementacją relacji jedna tabela – jeden model, natomiast pierwszy jest zdecydowanie bardziej skomplikowany.

Model EAV

EAV jest skrótowcem słów *entity* (encja), *attribute* (atrybut) i *value* (wartość) i oznacza koncepcję, z której zrozumieniem początkujący programiści Magento mają zwykle największe trudności. Koncepcja EAV jest w Magento dość powszechnie wykorzystywana, jednak w innych współczesnych systemach informatycznych spotyka się ją stosunkowo rzadko. Poza tym implementacja modelu w Magento sama w sobie jest dość złożona.

Schemat modelu EAV przedstawiono na rysunku 3.4.



Rysunek 3.4. Schemat modelu EAV

Czym jest model EAV?

Aby zrozumieć, czym w ogóle jest model EAV oraz jaką funkcję pełni w Magento, trzeba najpierw opisać jego części składowe.

- **Encja.** Encja reprezentuje pojedyncze dane w obiektach Magento — produktach, klientach, kategoriach i zamówieniach. Każda encja jest przechowywana w bazie danych i ma unikatowy identyfikator.
- **Atrybut.** Atrybut reprezentuje właściwości obiektów. Poszczególne atrybuty nie są umieszczane w oddzielnych kolumnach tabeli produktów — wszystkie atrybuty są przechowywane w odrębnych zbiorach tabel.
- **Wartość.** Jak wskazuje nazwa, jest to zwykła wartość skojarzona z określonym atrybutem.

To właśnie ten wzorzec projektowy stoi za niespotykaną elastycznością i niemal nieograniczonymi możliwościami Magento, ponieważ dzięki niemu można dodawać i usuwać nowe właściwości bez konieczności wprowadzania jakichkolwiek zmian w kodzie źródłowym czy szablonach.

Podczas gdy model w ujęciu Magento można postrzegać jako mechanizm rozrostu bazy danych w pionie (nowe atrybuty dodawane są w postaci nowych wierszy), model tradycyjny powiększa bazę danych w poziomie (nowe atrybuty oznaczają nowe kolumny), ponieważ wiąże się z koniecznością każdorazowej zmiany w schemacie bazy danych, gdy zachodzi potrzeba dodania nowego atrybutu.

Oprócz tego, że model EAV umożliwia dokonywanie coraz to nowych zmian w bazie danych, to również działa bardziej wydajnie, ponieważ przetwarzane są tylko atrybuty niepuste. Nie trzeba więc rezerwować dodatkowego miejsca w bazie danych na atrybuty `null`.

> Więcej szczegółowych informacji na temat struktury bazy danych Magento można znaleźć na stronie: *www.magereverse.com*.

Dodawanie nowego atrybutu produktu jest bardzo proste i sprowadza się do określenia w Magento jego typu — może to być kolor, rozmiar, marka i tym podobne. Równie prosta jest czynność odwrotna, gdy trzeba pozbyć się nieużywanych atrybutów w modelach produktów albo klientów.

> Więcej informacji na temat zarządzania atrybutami można znaleźć na stronie: *http://www.magentocommerce.com/knowledge-base/entry/how-do-attributes-work-in-magento*.

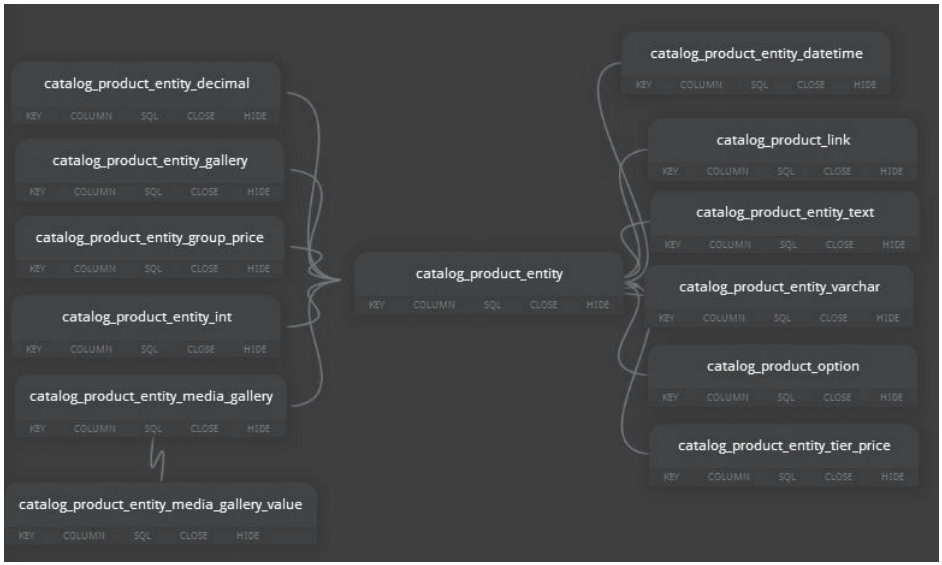
Magento w wersji Community Edition obecnie obsługuje osiem różnych typów obiektów EAV. Są to:

- klient,
- adres klienta,
- produkty,
- kategorie produktów,
- zamówienia,
- faktury,
- noty kredytowe,
- wysyłki.

> W Magento Enterprise Edition obsługiwany jest jeszcze jeden typ — obiekt RMA symbolizujący zlecenie odbioru stosowane w przypadku zwrotu towarów. Jest on częścią systemu **Return Merchandise Authorization** (**RMA**).

Elastyczność i szerokie możliwości mają niestety swoją cenę — implementacja modelu EAV sprawia, że dane na temat encji ulegają rozproszeniu w wielu tabelach. Na przykład dane na temat samego modelu produktu są przechowywane w około 40 różnych tabelach.

Diagram widoczny na rysunku 3.5 prezentuje zaledwie kilka tabel, w których przechowywane są dane na temat produktów przetwarzanych w Magento.



Rysunek 3.5. Schemat kilku wybranych tabel, w których przechowywane są dane na temat produktów

Kolejną wadą stosowania modelu EAV jest to, że odczytywanie dużych kolekcji obiektów EAV znacząco wpływa na wydajność systemu, a jednocześnie wymaga tworzenia bardzo skomplikowanych zapytań do bazy danych. Dane są w tym modelu bardziej pofragmentowane (znajdują się w wielu tabelach), zatem odczytanie pojedynczego rekordu wymaga wykonania co najmniej kilku złączeń.

Kontynuując nasz przykład oparty na produktach przechowywanych w Magento, stworzymy teraz ręcznie zapytanie, które będzie zwracać rekord pojedynczego produktu.

> Prezentowane w dalszej części punktu zapytania można wykonywać i zmieniać w narzędziu PHPMyAdmin lub MySQL Workbench. PHPMyAdmin można pobrać ze strony: *http://www.phpmyadmin.net/*, zaś MySQL Workbench jest dostępne na witrynie: *http://www.mysql.com/products/workbench/*.

Pierwszą tabelą, z jakiej będziemy musieli skorzystać, jest `catalog_product_entity`. Można ją traktować jako główną tabelę produktów w modelu EAV, ponieważ znajdują się w niej najważniejsze atrybuty encji produktów. Zawartość tabeli `catalog_product_entity` przedstawiono na rysunku 3.6.

Zawartość tabeli `catalog_product_entity` zostanie zwrócona po wykonaniu następującego zapytania języka SQL:

```
SELECT * FROM 'catalog_product_entity';
```

entity_id	entity_type_id	attribute_set_id	type_id	sku	category_ids	created_at	updated_at	has_options
16	10	38	simple	n2610	8	2007-08-23 13:03:05	2008-08-08 14:50:04	0
17	10	38	simple	bb8100	8	2007-08-23 15:40:26	2008-08-08 14:50:23	0
18	10	38	simple	sw810i	8	2007-08-23 15:47:44	2008-08-08 14:50:56	0
19	10	38	simple	8525PDA	8	2007-08-23 15:55:29	2008-08-08 14:51:11	0
20	10	38	simple	MM-A900M	8	2007-08-23 18:06:42	2008-08-08 14:51:27	0
25	10	39	simple	MA464LL/A	15,28	2007-08-24 14:28:50	2008-07-28 21:27:34	1
26	10	39	simple	LX.FR206.001	15,28	2007-08-24 14:35:16	2008-08-05 07:14:52	1
27	10	39	simple	VGN-TXN27N/B		2007-08-24 14:41:56	2008-08-06 05:12:24	0
28	10	39	simple	M285-E	15,28	2007-08-24 14:47:57	2008-08-08 14:51:44	0
29	10	40	simple	cn_3		2007-08-24 18:53:19	2008-08-08 00:51:34	0
30	10	40	simple	asc_8	16	2007-08-24 19:00:49	2008-07-31 17:23:28	0
31	10	40	simple	steve_4	17	2007-08-24 19:05:50	2008-08-08 01:02:40	0
32	10	40	simple	nine_3	17	2007-08-24 19:12:54	2008-07-31 18:06:25	0
33	10	40	simple	ecco_3	17	2007-08-24 19:21:42	2008-08-08 01:04:49	0
34	10	40	simple	ken_8	16	2007-08-24 19:27:04	2008-08-08 00:58:35	0
35	10	41	simple	coal_sm	4	2007-08-24 19:49:30	2008-07-31 17:36:42	0
36	10	41	simple	ink_sm	4	2007-08-24 19:53:06	2008-08-08 00:49:48	0
37	10	41	simple	oc_sm	4	2007-08-24 19:59:39	2008-08-08 00:47:47	0
38	10	41	simple	zol_r_sm	4	2007-08-24 20:02:08	2008-08-08 00:39:57	0
39	10	41	simple	4fasd5f5	19	2007-08-24 20:07:02	2008-06-24 23:45:44	0
41	10	42	simple	384822	23	2007-08-27 10:43:59	2008-07-29 22:28:43	0
42	10	42	simple	bar1234	23	2007-08-27 10:50:01	2008-06-24 23:59:46	0
44	10	44	simple	Rebel XT	26	2007-08-28 13:06:05	2008-06-24 22:50:46	0
45	10	44	simple	QC-2185	26	2007-08-28 13:18:56	2008-08-08 14:52:48	0
46	10	44	simple	750	12,26	2007-08-28 13:23:34	2008-08-08 14:57:57	0
47	10	44	simple	A630	26	2007-08-28 13:27:14	2008-08-08 14:58:14	0
48	10	44	simple	C530	26	2007-08-28 13:32:20	2008-08-08 14:58:41	0
49	10	40	simple	ana_9	17	2007-08-28 15:09:50	2008-08-08 01:07:50	0
51	10	42	simple	1111	22	2007-08-28 16:25:46	2008-08-08 14:59:04	0

Rysunek 3.6. Fragment zawartości tabeli catalog_product_entity

W tabeli znajdują się opisane niżej pola.

- `entity_id` — unikatowy identyfikator produktu, używany wewnętrznie przez Magento.
- `entity_type_id` — Magento używa w systemie EAV kilku różnych typów, między innymi modeli, produktów, klientów i zamówień. Dzięki temu, że każdy z tych typów ma unikatowy identyfikator, Magento może odczytać ich atrybuty i wartości z odpowiednich tabel.
- `attribute_set_id` — atrybuty produktów można lokalnie grupować w zbiory atrybutów. Zbiory atrybutów zapewniają jeszcze dalej idącą elastyczność struktury produktów, ponieważ dzięki nim produkty mogą mieć tylko niektóre spośród wszystkich dostępnych atrybutów.
- `type_id` — w Magento występuje kilka różnych typów produktów: proste, konfigurowalne, łączone, dostępne do pobrania i grupowane. Każdy typ produktu ma unikatowe ustawienia i funkcje.
- `sku` — **jednostka magazynowa** (ang. *Stock Keeping Unit* — SKU) to liczba lub kod, który identyfikuje unikatowy produkt lub artykuł dostępny w sklepie do sprzedaży. Wartość SKU jest definiowana przez użytkownika.
- `has_options` — wskazuje, czy produkt ma dodatkowe opcje.
- `required_options` — wskazuje, czy wymagane są jakieś dodatkowe opcje.
- `created_at` — data utworzenia wiersza.
- `updated_at` — data ostatniej modyfikacji wiersza.

Znamy już strukturę tabeli, która przechowuje encje produktów, a także wiemy, że każdy rekord tej tabeli reprezentuje pojedynczy produkt w sklepie Magento. Nie mamy natomiast jeszcze wystarczających informacji na temat samego produktu, oprócz kodu jednostki magazynowej SKU oraz typu produktu.

Gdzie zatem znajdują się pozostałe atrybuty produktów? I skąd Magento wie, który atrybut dotyczy produktu, a który klienta?

Brakujące informacje można uzyskać z tabeli `eav_attribute` — w tym celu należy wykonać następujące zapytanie SQL:

```
SELECT * FROM 'eav_attribute';
```

Wynik zapytania będzie zawierał nie tylko atrybuty produktów, ale również atrybuty charakterystyczne dla modelu klienta, modelu zamówienia i im podobnych. Na szczęście znamy już klucz, na podstawie którego można wyizolować atrybuty, jakie nas interesują. Należy w tym celu wykonać zapytanie w następującej postaci:

```
SELECT * FROM 'eav_attribute'
WHERE entity_type_id = 4;
```

Zapytanie o takiej treści nakazuje bazie danych zwrócenie tylko tych atrybutów, dla których wartość w polu `entity_type_id` odpowiada analogicznemu identyfikatorowi `entity_type_id` produktu, czyli ma wartość `4`. Zanim przejdziemy dalej, warto zapoznać się z najważniejszymi polami tabeli `eav_attribute`.

- `attribute_id` — unikatowy identyfikator każdego atrybutu; stanowi jednocześnie klucz główny tabeli.
- `entity_type_id` — to pole kojarzy każdy atrybut z odpowiednim typem modelu EAV.
- `attribute_code` — nazwa lub klucz atrybutu; na podstawie tej wartości magiczne metody generują metody do odczytywania i ustawiania wartości.
- `backend_model` — model wewnętrzny, który zarządza ładowaniem danych z bazy danych i zapisywaniem ich do niej.
- `backend_type` — wskazuje typ wartości zapisywanej w magazynie danych (bazie danych).
- `backend_table` — wartość w tym polu wskazuje, czy atrybut powinien być przechowywany w tabeli specjalnej zamiast w domyślnych tabelach systemu EAV.
- `frontend_model` — model interfejsu użytkownika, odpowiada za generowanie elementu atrybutu na potrzeby przeglądarki internetowej.
- `frontend_input` — analogicznie do modelu interfejsu użytkownika wartość w tym polu wskazuje typ pola wejściowego, jakie powinno zostać wyświetlone przez przeglądarkę.
- `frontend_label` — w tym polu znajduje się etykieta (nazwa) atrybutu, która zostanie wyświetlona w przeglądarce.
- `source_model` — na podstawie modeli źródłowych atrybuty są wypełniane dozwolonymi wartościami. Magento zawiera kilka predefiniowanych modeli źródłowych, między innymi dla krajów, wartości typu „tak” lub „nie” i im podobnych.

Odczytywanie danych

Na tym etapie wiemy już, jak pozyskuje się encje produktów oraz ich atrybuty, które dotyczą całej encji. Czas więc odczytać rzeczywiste dane. Aby nie komplikować zbytnio przykładu (i zapytania), skupimy się na odczytaniu atrybutu, który zawiera nazwę produktu.

Skąd wiadomo, w której tabeli przechowywane są wartości atrybutów? Cóż, na szczęście w Magento konsekwentnie używa się jasno określonej konwencji nazewniczej, zgodnie z którą nadaje się odpowiednie nazwy tabelom. Rzut oka na strukturę bazy danych wykaże, że w bazie występuje kilka tabel, których nazwa zaczyna się od przedrostka `catalog_product_entity`:

- `catalog_product_entity`,
- `catalog_product_entity_datetime`,
- `catalog_product_entity_decimal`,
- `catalog_product_entity_int`,
- `catalog_product_entity_text`,
- `catalog_product_entity_varchar`,
- `catalog_product_entity_gallery`,
- `catalog_product_entity_media_gallery`,
- `catalog_product_entity_tier_price`.

No dobrze, ale skąd mamy wiedzieć, z której tabeli należy uzyskać wartość atrybutu wskazującego nazwę produktu? Uważny czytelnik na pewno zna już odpowiedź — wystarczy sobie przypomnieć, że w tabeli `eav_attribute` znajduje się kolumna o nazwie `backend_type`.

W systemie EAV Magento każdy atrybut jest przechowywany w oddzielnej tabeli, zgodnie z typem wartości `backend_type` tego atrybutu. Aby upewnić się co do typu wartości nazwy produktu, wystarczy wykonać zapytanie SQL o następującej postaci:

```
SELECT * FROM 'eav_attribute'
WHERE 'entity_type_id' =4 AND 'attribute_code' = 'name';
```

W wyniku wykonania zapytania okaże się, że typem wartości jest `varchar` oraz że wartości dla tego atrybutu są przechowywane w tabeli `catalog_product_entity_varchar`. Spójrzmy na zawartość tabeli widoczną na rysunku 3.7.

Tabela `catalog_product_entity_varchar` zawiera sześć następujących kolumn.

- `value_id` — unikatowy identyfikator wartości, który jest jednocześnie kluczem głównym tabeli.
- `entity_type_id` — identyfikator typu encji dla tej wartości.
- `attribute_id` — klucz obcy, którego wartość odnosi się do zawartości tabeli `eav_entity`.

value_id	entity_type_id	attribute_id	store_id	entity_id	value
228	10	96	1	16	Nokia 2610 Phone
230	10	102	1	16	20
232	10	103	1	16	Nokia 2610
233	10	105	1	16	Offering advanced media and calling features witho...
246	10	96	1	17	BlackBerry 8100 Pearl
248	10	102	1	17	21
250	10	103	1	17	BlackBerry 8100 Pearl
251	10	105	1	17	BlackBerry 8100 Pearl sports a large 240 x 260 scr...
264	10	96	1	18	Sony Ericsson W810i
266	10	102	1	18	2
268	10	103	1	18	Sony Ericsson W810i
269	10	105	1	18	The W810i follows a long tradition of beautifully ...
298	10	96	1	20	Samsung MM-A900M Ace
300	10	102	1	20	3
302	10	103	1	20	Samsung MM-A900M Ace Phone
303	10	105	1	20	New services supported by both the MM-A900m includ...
382	10	96	1	25	Apple MacBook Pro MA464LL/A 15.4" Notebook PC
386	10	103	1	25	Apple MacBook Pro MA464LL/A 15.4" Notebook PC (2.0...
387	10	105	1	25	Apple MacBook Pro MA464LL/A 15.4" Notebook PC (2.0...
400	10	96	1	26	Acer Ferrari 3200 Notebook Computer PC
404	10	103	1	26	Acer Ferrari 3200 Notebook Computer PC
405	10	105	1	26	Acer Ferrari 3200 Notebook Computer PC
418	10	96	1	27	Sony VAIO VGN-TXN27N/B 11.1" Notebook PC
422	10	103	1	27	Sony VAIO VGN-TXN27N/B 11.1" Notebook PC (Intel Co...
423	10	105	1	27	Sony VAIO VGN-TXN27N/B 11.1" Notebook PC (Intel Co...
436	10	96	1	28	Toshiba M285-E 14"
440	10	103	1	28	Toshiba Satellite A135-S4527 155.4" Notebook PC (I...
441	10	105	1	28	Toshiba M285-E 14" Convertible Notebook PC (Intel ...
453	10	96	1	29	CN Clogs Beach/Garden Clog
456	10	103	1	29	CN Clogs Beach/Garden Clog

Rysunek 3.7. Zawartość tabeli catalog_product_entity_varchar

- `store_id` — klucz obcy, który kojarzy wartość atrybutu z widokiem sklepu.
- `entity_id` — klucz obcy do odpowiedniej tabeli encji. W naszym przykładzie taką tabelą jest `catalog_product_entity`.
- `value` — rzeczywista wartość atrybutu, którą chcemy uzyskać.

> Atrybut można skonfigurować w taki sposób, aby jego wartość była wartością globalną, czyli dostępną we wszystkich widokach sklepów, lub też by w każdym widoku sklepu atrybut miał inną wartość.

Znamy już wszystkie tabele, w których znajdują się interesujące nas informacje na temat produktów. Możemy więc napisać docelowe zapytanie:

```
SELECT p.entity_id AS product_id, var.value AS product_name, p.sku AS product_sku
FROM catalog_product_entity p, eav_attribute eav, catalog_product_entity_varchar var
WHERE p.entity_type_id = eav.entity_type_id
   AND var.entity_id = p.entity_id
   AND eav.attribute_code = 'name'
   AND eav.attribute_id = var.attribute_id
```

Wynik wykonania zapytania znajduje się na rysunku 3.8.

product_id	product_name	product_sku
16	Nokia 2610 Phone	n2610
16	Nokia 2610 Phone	n2610
16	Nokia 2610 Phone	n2610
16	Nokia 2610 Phone	n2610
17	BlackBerry 8100 Pearl	bb8100
17	BlackBerry 8100 Pearl	bb8100
17	BlackBerry 8100 Pearl	bb8100
17	BlackBerry 8100 Pearl	bb8100
18	Sony Ericsson W810i	sw810i
18	Sony Ericsson W810i	sw810i
18	Sony Ericsson W810i	sw810i
18	Sony Ericsson W810i	sw810i
19	AT&T 8525 PDA	8525PDA
19	AT&T 8525 PDA	8525PDA
19	AT&T 8525 PDA	8525PDA
19	AT&T 8525 PDA	8525PDA
20	Samsung MM-A900M Ace	MM-A900M
20	Samsung MM-A900M Ace	MM-A900M
20	Samsung MM-A900M Ace	MM-A900M
20	Samsung MM-A900M Ace	MM-A900M
25	Apple MacBook Pro MA464LL/A 15.4" Notebook PC	MA464LL/A
25	Apple MacBook Pro MA464LL/A 15.4" Notebook PC	MA464LL/A
25	Apple MacBook Pro MA464LL/A 15.4" Notebook PC	MA464LL/A
25	Apple MacBook Pro MA464LL/A 15.4" Notebook PC	MA464LL/A
26	Acer Ferrari 3200 Notebook Computer PC	LX.FR206.001
26	Acer Ferrari 3200 Notebook Computer PC	LX.FR206.001
26	Acer Ferrari 3200 Notebook Computer PC	LX.FR206.001
26	Acer Ferrari 3200 Notebook Computer PC	LX.FR206.001
27	Sony VAIO VGN-TXN27N/B 11.1" Notebook PC	VGN-TXN27N/B
27	Sony VAIO VGN-TXN27N/B 11.1" Notebook PC	VGN-TXN27N/B

Rysunek 3.8. Wynik wykonania zapytania o dane na temat produktów

W wyniku wykonania zapytania zwrócone zostały trzy kolumny: `product_id`, `product_name` oraz `product_sku`. Cofnijmy się zatem o krok, aby zobaczyć, jak uzyskać jedynie nazwy produktów na podstawie SKU. Otóż odpowiednie zapytanie języka SQL musiałoby się składać z pięciu wierszy i zwracałoby wyłącznie jedną z dwóch danych na temat produktu: wartość pola numerycznego (na przykład cenę) z jednej tabeli wartości EAV albo wartość tekstową (na przykład nazwę produktu) z innej tabeli wartości EAV.

Gdyby nie ORM zaimplementowany w Magento, utrzymywanie danych w systemie byłoby w zasadzie niemożliwe. Na szczęście dzięki ORM prawdopodobnie nigdy nie trzeba będzie pisać standardowego kodu SQL, aby odczytywać potrzebne informacje.

Spójrzmy zatem, jak ten sam zestaw danych na temat produktów można uzyskać dzięki ORM Magento.

1. W pierwszym kroku trzeba stworzyć instancję kolekcji produktów:

```
$collection = Mage::getModel('catalog/product')->getCollection();
```

2. Następnie konieczne będzie jawne nakazanie Magento, że wybrany ma zostać atrybut, który wskazuje nazwę produktu:

```
$collection->addAttributeToSelect('name');
```

3. Teraz trzeba posortować kolekcję względem nazw produktów:

```
$collection->setOrder('name', 'asc');
```

4. Końcowy krok polega na załadowaniu przez Magento całej kolekcji:

```
$collection->load();
```

5. Wynikiem wykonania opisanych czynności jest kolekcja wszystkich produktów przechowywanych w sklepie, uporządkowana względem nazwy. Pełną treść zapytania języka SQL można uzyskać po wykonaniu następującej instrukcji:

```
echo $collection->getSelect()->__toString();
```

Ostatecznie w zaledwie trzech wierszach kodu źródłowego nakazaliśmy Magento, aby system odczytał wszystkie produkty przechowywane w sklepie, wyizolował z nich nazwy produktów i na koniec uporządkował je alfabetycznie.

> Ostatni wiersz przykładowego kodu, `$collection->getSelect()->__toString()`, pozwala programiście podejrzeć rzeczywiste zapytanie języka SQL przetwarzane przez Magento na podstawie wykonanego kodu źródłowego.

Rzeczywiste zapytanie języka SQL, które wykona Magento na podstawie przykładowego kodu, ma następującą treść:

```
SELECT 'e'.*. IF( at_name.value_id >0, at_name.value, at_name_default.value ) AS 'name'
FROM 'catalog_product_entity' AS 'e'
LEFT JOIN 'catalog_product_entity_varchar' AS 'at_name_default' ON ('at_name_default'.
↪'entity_id' = 'e'.'entity_id')
AND ('at_name_default'.'attribute_id' = '65')
AND 'at_name_default'.'store_id' =0
LEFT JOIN 'catalog_product_entity_varchar' AS 'at_name' ON ( 'at_name'.'entity_id' =
↪'e'.'entity_id' )
AND ('at_name'.'attribute_id' = '65')
AND ('at_name'.'store_id' =1)
ORDER BY 'name' ASC
```

Widać więc wyraźnie, że ORM i modele EAV są doskonałymi narzędziami, które nie tylko dają programistom wiele możliwości i warunkują elastyczność, ale również pozwalają tworzyć rozwiązania zwięzłe i czytelne.

Korzystanie z kolekcji Magento

Gdy przyjrzymy się jeszcze raz kodowi źródłowemu z poprzedniego przykładu, warto zwrócić uwagę, że oprócz stworzenia instancji modelu produktu wywołana również została metoda `getCollection()`. Metoda `getCollection()` należy do klasy `Mage_Core_Model_Abstract`, co oznacza, że może ją wywoływać każdy model w Magento.

Wszystkie kolekcje są dziedziczone po klasie `Varien_Data_Collection`.

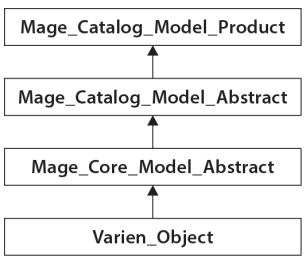
Kolekcja Magento jest w zasadzie modelem, który zawiera inne modele. Dlatego zamiast przechowywać listę produktów w tablicy, można użyć do tego celu kolekcji produktów. Struktura danych obecna w kolekcji sprzyja grupowaniu modeli, a ponadto kolekcje udostępniają specjalne metody, za pomocą których można przetwarzać encje przechowywane w kolekcji.

Oto najbardziej przydatne metody udostępniane przez kolekcje.

- `addAttributeToSelect` — dodaje atrybut do encji w kolekcji. W szczególności w wywołaniu metody można użyć symbolu wieloznacznego *, aby dodać do encji wszystkie dostępne atrybuty.
- `addFieldToFilter` — dodaje pole filtrowania do kolekcji. Wywołuje się ją na zwykłych modelach, które nie są modelami EAV.
- `addAttributeToFilter` — służy do filtrowania kolekcji encji EAV.
- `addAttributeToSort` — służy do dodawania atrybutu do definicji porządku sortowania.
- `addStoreFilter` — pozwala na filtrowanie względem sklepu; uwzględnia dostępność produktu w sklepie.
- `addWebsiteFilter` — dodaje do kolekcji filtr strony WWW.
- `addCategoryFilter` — wskazuje filtr kategorii dla kolekcji produktów.
- `addUrlRewrite` — służy do dodawania przepisanych adresów URL dla produktu.
- `setOrder` — ustawia porządek sortowania kolekcji.

Są to tylko niektóre spośród wszystkich dostępnych metod przetwarzania kolekcji. Każda kolekcja implementuje unikatowe metody, których charakter zależy od rodzaju encji przechowywanych w kolekcji. Na przykład kolekcja klientów `Mage_Customer_Model_Resource_Customer_Collection` ma zaimplementowaną unikatową metodę `groupByEmail()`, która — zgodnie z nazwą — grupuje encje w kolekcji względem adresu poczty elektronicznej.

Podobnie jak w poprzednich przykładach, nadal będziemy opierać się na modelach produktów. Tym razem skupimy się na kolekcji produktów. Dla przypomnienia na rysunku 3.9 przedstawiono ponownie model dziedziczenia dla klasy `Mage_Catalog_Model_Product`.



Rysunek 3.9. Drzewo dziedziczenia dla klasy Mage_Catalog_Model_Product

W celu lepszego zilustrowania sposobu, w jaki można używać kolekcji, weźmiemy pod uwagę następujące standardowe scenariusze działań na produktach.

1. Uzyskanie kolekcji produktów, które należą do określonej kategorii.
2. Uzyskanie nowych produktów, które pojawiły się w dniu x lub później.
3. Uzyskanie produktów, które najlepiej się sprzedają.
4. Filtrowanie kolekcji produktów względem widoczności produktów.
5. Filtrowanie produktów, którym nie przypisano obrazka.
6. Dodanie wielu kryteriów porządkowania.

Uzyskanie kolekcji produktów, które należą do określonej kategorii

Pierwszym zadaniem, z jakim próbuje się uporać większość programistów rozpoczynających swoją przygodę z Magento, jest załadowanie kolekcji produktów, które należą do określonej kategorii. Najczęściej spotkać można rozwiązania oparte na wykorzystaniu metod `addCategoryFilter()` lub `addAttributeToFilter()`. Jednak w większości przypadków używanych w praktyce zdecydowanie łatwiej jest zastosować rozwiązanie znacznie prostsze, ale również nie do końca intuicyjne w kontekście informacji przedstawionych wcześniej w tej książce.

Najprostszy sposób realizacji zadania nie polega wcale na uzyskaniu kolekcji produktów i jej późniejszemu filtrowaniu względem kategorii, lecz na stworzeniu najpierw instancji interesującej nas kategorii, a następnie pobraniu z niej kolekcji produktów. Aby przekonać się o skuteczności takiego podejścia, w IMC należy wykonać następujący fragment kodu:

```
$category = Mage::getModel('catalog/category')->load(5);
$productCollection = $category->getProductCollection();
```

W klasie `Mage_Catalog_Model_Category` zaimplementowana jest metoda `getProductCollection()`. Warto przeanalizować kod źródłowy tej metody:

```
public function getProductCollection()
{
    $collection = Mage::getResourceModel('catalog/product_collection')
        ->setStoreId($this->getStoreId())
        ->addCategoryFilter($this);
    return $collection;
}
```

Jak widać, funkcja tworzy jedynie instancję modelu zasobów dla kolekcji produktów, to znaczy jako sklep aktywny ustawia sklep o podanym identyfikatorze, a następnie przekazuje do metody `addCategoryFilter()` bieżącą kategorię.

Opisane rozwiązanie jest bezpośrednim następstwem decyzji, które miały na celu optymalizację wydajności Magento i ułatwienie życia programistom korzystającym z tego narzędzia. Dzięki tym decyzjom bowiem kategoria będzie w znakomitej większości przypadków dostępna zawsze — w taki lub inny sposób.

Uzyskanie nowych produktów, które pojawiły się w dniu x lub później

Skoro wiadomo już, jak uzyskać kolekcję produktów z danej kategorii, warto sprawdzić, czy możliwe jest zastosowanie filtrów na wynikowym zbiorze produktów, aby ostatecznie otrzymać tylko te, które pasują do założonych warunków. W tym konkretnym przykładzie zadanie będzie polegać na uzyskaniu wszystkich produktów, które zostały dodane nie wcześniej niż w grudniu 2012 roku. Analogicznie do poprzedniego przykładu kolekcję produktów można przefiltrować na podstawie daty ich stworzenia — w tym celu w IMC należy wykonać następujący kod:

```
// kolekcja produktów z poprzedniego przykładu
$productCollection->addFieldToFilter('created_at', array('from' => '2012-12-01));
```

Proste, prawda? Można by nawet dodać kolejny warunek i uzyskać produkty, które zostały dodane w okresie między dwiema podanymi datami. Powiedzmy, że konieczne jest odczytanie rekordów produktów dodanych w grudniu:

```
$productCollection->addFieldToFilter('created_at', array('from' => '2012-12-01));
$productCollection->addFieldToFilter('created_at', array('to' => '2012-12-30));
```

Metoda `addFieldToFilter` Magento obsługuje warunki opisane w tabeli 3.1.

Tabela 3.1. Warunki obsługiwane przez metodę addFieldToFilter

Kod atrybutu	Warunek SQL
eq	=
neq	!=
like	LIKE
nlike	NOT LIKE
in	IN ()
nin	NOT IN ()
is	IS
notnull	NOT NULL
null	NULL
moreq	>=
gt	>
lt	<
gteq	>=
lteq	<=

Można też stosować inne rodzaje filtrów. Na przykład wykonanie poniższego kodu w IMC po nałożeniu filtra na datę utworzenia produktu spowoduje, że zwrócone zostaną tylko produkty widoczne:

```
$productCollection->addAttributeToFilter('visibility', 4);
```

`visibility` jest specjalnym atrybutem, za pomocą którego wskazuje się, gdzie produkty mają być widoczne. Atrybut ten może mieć następujące wartości:

- `1` — produkty nie są widoczne pojedynczo.
- `2` — produkty są widoczne w katalogu.
- `3` — produkty są widoczne w wynikach wyszukiwania.
- `4` — produkty są widoczne w katalogach i w wynikach wyszukiwania.

Uzyskanie produktów, które najlepiej się sprzedają

Aby uzyskać listę produktów, które najlepiej sprzedają się w danej kategorii, należy wykonać połączenie z tabelą `sales_order`. Mechanizm odczytywania rekordów bestsellerowych produktów przyda się później do stworzenia specjalnej kategorii produktów albo umieszczenia takich danych w raportach. W IMC trzeba wykonać następujący kod:

```
$category = Mage::getModel('catalog/category')->load(5);
$productCollection = $category->getProductCollection();
$productCollection->getSelect()->join(array('o'=> 'sales_flat_order_item'),
↪'main_table.entity_id = o.product_id', array('o.row_total','o.product_id'))
↪->group(array('sku'));
```

Warto przeanalizować zwłaszcza operacje wykonywane w trzecim wierszu kodu. Metoda `getSelect()` jest dziedziczona bezpośrednio z klasy `Varien_Data_Collection_Db` i zwraca zmienną, w której znajduje się instrukcja `Select`. Zmienna ta zawiera również kolekcje, które udostępniają metody odpowiedzialne za definiowanie złączeń oraz za wykonywanie grupowania bez konieczności pisania kodu języka SQL.

Nie jest to jedyny możliwy sposób dodawania złączenia do kolekcji. Tak naprawdę istnieje również inne rozwiązanie, o wiele prostsze. Polega ono na wykorzystaniu funkcji `joinField()`. Nowa wersja kodu, w której wykorzystana zostanie ta funkcja, będzie mieć następującą postać:

```
$category = Mage::getModel('catalog/category')->load(5);
$productCollection = $category->getProductCollection();
$productCollection->joinField('o', 'sales_flat_order_item', array('o.row_total',
↪'o.product_id'), 'main_table.entity_id = o.product_id')->group(array('sku'));
```

Filtrowanie kolekcji produktów względem widoczności produktów

To zadanie można wykonać bardzo łatwo za pomocą metody `addAttributeToFilter()`. Produkty w Magento są wyposażone w systemowy atrybut `visibility`, który określa ich widoczność. Atrybut `visibility` może mieć jedną z czterech wartości liczbowych z przedziału od 1 do 4. W naszym przykładzie interesuje nas wyłącznie pokazywanie produktów, dla których atrybut widoczności ma wartość 4, co oznacza, że produkty te są widoczne zarówno w wynikach wyszukiwania, jak i w katalogu. W IMC należy wykonać następujący kod źródłowy:

```
$category = Mage::getModel('catalog/category')->load(5);
$productCollection = $category->getProductCollection();
$productCollection->addAttributeToFilter('visibility', 4);
```

Gdy widoczność produktu zostanie zmieniona, będzie można porównać wynik wykonania kodu na różnych kolekcjach.

Filtrowanie produktów, którym nie przypisano obrazka

Filtrowanie produktów bez przypisanego obrazka przydatne jest między innymi wówczas, gdy importuje się dane z zewnętrznego systemu, który czasami zawodzi. Podobnie jak we wszystkich wcześniejszych przykładach, również dla obrazka skojarzonego z produktem istnieje odpowiedni atrybut:

```
$category = Mage::getModel('catalog/category')->load(5);
$productCollection = $category->getProductCollection();
$productCollection->addAttributeToFilter('small_image',array('notnull'=>'','neq'=>
↪'no_selection'));
```

Dzięki zdefiniowaniu dodatkowego filtra produkty zwrócone w kolekcji wynikowej będą mieć skojarzony z nimi mały obrazek. Domyślnie w Magento występują trzy typy obrazków dla produktów: miniatury, małe obrazki `small_image` oraz obrazki właściwe. Każdy z tych trzech typów obrazków jest wykorzystywany w różnych częściach aplikacji. Zatem regułę wyszukiwania produktów można by jeszcze bardziej zawęzić:

```
$productCollection->addAttributeToFilter('small_image',
↪array('notnull'=>'','neq'=>'no_selection'))->addAttributeToFilter('thumbnail,
↪array('notnull'=>'','neq'=>'no_selection'))->addAttributeToFilter('image',
↪array('notnull'=>'','neq'=>'no_selection'));
```

Tak skonstruowany kod spowoduje, że zwrócona zostanie kolekcja produktów, dla których wskazano wszystkie trzy typy obrazków. Można poeksperymentować na własną rękę i filtrować produkty względem różnych typów obrazków.

Dodanie wielu kryteriów porządkowania

W ostatnim zadaniu uporządkujemy zawartość kolekcji najpierw względem stanu magazynowego, a następnie względem ceny — od najwyższej do najniższej. Informację o stanie magazynowym zwróci metoda `addStockStatusToSelect()`, która jest dostępna tylko w modelu zasobów reprezentującym właśnie stan magazynowy. Metoda `addStockStatusToSelect()` sama wygeneruje odpowiednie zapytanie języka SQL, które zwróci kolekcję:

```
$category = Mage::getModel('catalog/category')->load(5);
$productCollection = $category->getProductCollection();
$select = $productCollection->getSelect();
Mage::getResourceModel('cataloginventory/stock_status')->addStockStatusToSelect
↪($select, Mage::app()->getWebsite());
$select->order('salable desc');
$select->order('price asc');
```

Zapytanie to spowoduje, że Magento uporządkuje produkty względem ich dostępności do sprzedaży (będącej wartością logiczną — produkt może być dostępny w sprzedaży lub nie) oraz względem ceny. Jako efekt końcowy zwrócona zostanie kolekcja produktów uporządkowanych w ten sposób, że na początku występować będą produkty dostępne w sprzedaży ułożone względem ceny od najwyższej do najniższej, a w dalszej kolejności widnieć będą produkty niedostępne w sprzedaży, także uporządkowane od najdroższego do najtańszego.

Warto poeksperymentować z różnymi kombinacjami kryteriów sortowania, aby zobaczyć, jak Magento organizuje i porządkuje kolekcje produktów.

Wykonywanie bezpośrednich zapytań języka SQL

Wiemy już, w jaki sposób modele danych Magento oraz system ORM ułatwiają odczytywanie i zapisywanie danych oraz manipulowanie nimi. Zanim zajmiemy się głównymi tematami tego punktu, czyli adapterami baz danych w Magento oraz wykonywaniem bezpośrednich zapytań języka SQL, najpierw koniecznie trzeba powiedzieć, dlaczego zasadniczo powinno się unikać stosowania tych technik.

Magento jest systemem niezwykle złożonym, przynajmniej częściowo sterowanym zdarzeniami, o czym była już mowa w poprzednim rozdziale. Samo zapisanie produktu wyzwala różnego rodzaju zdarzenia, z których każde wykonuje inne zadanie. Zdarzenia nie zajdą jednak wówczas, gdy zmiana danych na temat produktu zostanie wprowadzona bezpośrednio przez zapytanie SQL. Dlatego programiści muszą pracować z Magento z niezwykłą ostrożnością i zawsze się upewniać, czy istnieje wystarczający powód, by nie korzystać z ORM.

Istnieją rzecz jasna okoliczności, w których możliwość bezpośredniego operowania na bazie danych jest bardzo przydatna i okazuje się łatwiejszym sposobem wykonania niektórych zadań niż wykorzystywanie modeli Magento. Na przykład aby zmienić globalnie określony atrybut produktu albo zmodyfikować status produktów w kolekcji, można by załadować kolekcję produktów i w pętli przejść przez każdy z nich, wprowadzając w nich wymagane zmiany i zapisując je jedną po drugim. W przypadku niewielkiej kolekcji podejście takie jeszcze by się sprawdziło, jednak im większy będzie rozmiar zbioru danych, tym niższa stanie się wydajność pętli przetwarzającej kolekcję. W przypadku większych kolekcji wykonanie pętli może trwać nawet długie sekundy.

Bezpośrednie zapytanie języka SQL zostanie wykonane zdecydowanie szybciej, zwykle w ciągu mniej niż jednej sekundy, zależnie od rozmiaru zbioru przetwarzanych danych oraz charakteru wykonywanego zapytania.

Magento ma zaimplementowane mechanizmy, które odpowiadają za nawiązanie połączenia z bazą danych. Odpowiednie metody znajdują się w klasie `Mage_Core_Model_Resource` i umożliwiają nawiązanie połączenia jednego z dwóch typów: `core_read` lub `core_write`.

Na początek stworzymy model zasobu oraz dwa połączenia — jedno do odczytu i drugie do zapisu:

```
$resource = Mage::getModel('core/resource');
$read = $resource->getConnection('core_read');
$write = $resource->getConnection('core_write');
```

Nawet gdy trzeba wykonać bezpośrednie zapytanie języka SQL, dzięki Magento nie trzeba się martwić o zestawienie połączenia z bazą danych — wystarczy tylko stworzyć instancję modelu zasobu i wybrać odpowiedni rodzaj połączenia.

Odczyt

Działanie połączenia do odczytu można sprawdzić przez wykonanie następującego kodu źródłowego:

```
$resource = Mage::getModel('core/resource');
$read = $resource->getConnection('core_read');
$query = 'SELECT * FROM catalog_product_entity';
$results = $read->fetchAll($query);
```

Tak skonstruowane zapytanie jest prawidłowe i powinno zwrócić wszystkie produkty z tabeli `catalog_product_entity`. A co się stanie, gdy ten sam kod spróbujemy uruchomić w instancji Magento, w której nazwy tabel są poprzedzone prefiksem? Albo gdy wraz z kolejną aktualizacją nazwy tabel Magento ulegną zmianie? Kod źródłowy w takiej postaci nie jest ani przenośny, ani łatwy w utrzymaniu. Na szczęście model zasobu udostępnia przydatną metodę o nazwie `getTableName()`.

Metoda `getTableName()` przyjmuje parametr, którym jest nazwa wytwórcza, a następnie na podstawie konfiguracji zdefiniowanej w pliku *config.xml* nie tylko znajduje właściwą tabelę, ale również od razu sprawdza, czy tabela ta istnieje w bazie danych. W celu wykorzystania metody `getTableName()` przykładowy kod należy zmienić do następującej postaci:

```
$resource = Mage::getModel('core/resource');
$read = $resource->getConnection('core_read');
$query = 'SELECT * FROM ' . $resource->getTableName('catalog/product');
$results = $read->fetchAll($query);
```

W przykładowym kodzie wykonywana jest również metoda `fetchAll()`, która zwraca wszystkie wiersze wynikowe zapytania umieszczone w tablicy. Nie jest to jednak jedyna używana opcja — dostępne są również metody `fetchCol()` i `fetchOne()`. Poniżej opisano działanie trzech wymienionych metod.

- `fetchAll` — funkcja zwraca wszystkie wiersze uzyskane w wyniku wykonania oryginalnego zapytania.
- `fetchOne` — funkcja zwraca jedynie wartości z pierwszego wiersza stanowiącego wynik wykonania zapytania.
- `fetchCol` — funkcja zwraca wszystkie wiersze uzyskane w wyniku wykonania zapytania, lecz tylko pierwszą kolumnę każdego wiersza. Funkcja przydaje się na przykład wówczas, gdy wystarczy odczytać pierwszą kolumnę z unikatowymi identyfikatorami, takimi jak identyfikatory produktów albo jednostki magazynowe SKU.

Zapisywanie

Jak wspomniano już wcześniej, zapisywanie modelu — produktu, kategorii, klienta i tak dalej — może trwać w Magento stosunkowo długo, ze względu na znaczną liczbę obserwatorów i zdarzeń wywoływanych w tle.

Jeśli jednak zadanie sprowadza się do zapisania zmienionych prostych, statycznych wartości, dokonanie takiej modyfikacji w obszernych kolekcjach z wykorzystaniem ORM Magento może być czynnością czasochłonną. Załóżmy na przykład, że wszystkie produkty należy oznaczyć jako niedostępne w magazynie. Zamiast wykonywania tej operacji z wykorzystaniem modeli udostępnianych przez Magento albo tworzenia własnego skryptu, który będzie iterować przez kolejne pozycje kolekcji wszystkich produktów, wystarczy wykonać następujący przykładowy kod źródłowy:

```
$resource = Mage::getModel('core/resource');
$read = $resource->getConnection('core_write');
$tablename = $resource->getTableName('cataloginventory/stock_status');
$query = "UPDATE {$tablename} SET 'is_in_stock` = 1";
$write->query($query);
```

Podsumowanie

W tym rozdziale opisane zostały następujące zagadnienia:

- modele Magento, ich dziedziczenie i przeznaczenie;
- sposób, w jaki Magento używa modeli zasobów i kolekcji;
- model EAV oraz jego znaczenie w Magento;
- sposób działania EAV oraz struktura danych w bazie;
- model ORM Magento i jego implementacja;
- sposób korzystania z bezpośrednich zapytań języka SQL oraz adapterów zasobów Magento.

Dotychczasowe rozdziały miały charakter raczej teoretyczny niż praktyczny, a ich celem było uświadomienie czytelnikowi stopnia złożoności Magento oraz przedstawienie narzędzi i informacji niezbędnych w trakcie lektury kolejnych rozdziałów. W dalszej części książki skupimy się na bardziej praktycznych zagadnieniach i stopniowo zaczniemy tworzyć własne rozszerzenia, z wykorzystaniem dotychczas poznanych mechanizmów.

W następnym rozdziale ubrudzimy sobie nieco dłonie i stworzymy swoje pierwsze rozszerzenie Magento.

Skorowidz

G

H

I

J

K

L

M

N

O

P

R

S

Ś

T

U

V

W

X

Y

Z

Ż

Magento
Przewodnik dla programistów PHP

Magento to platforma pozwalająca na prowadzenie handlu w internecie. Jej pierwsza wersja ukazała się w 2008 roku i od tego czasu platforma jest ciągle rozwijana. Magento powstało w oparciu o Zend Framework, a fakt ten cieszy wielu programistów PHP. Jeżeli chcesz poznać możliwości tej platformy, wdrożyć ją, dostosować do własnych potrzeb lub napisać nowy moduł, to masz w rękach doskonałą książkę.

Ten przewodnik pozwoli Ci zgłębić tajniki Magento. W pierwszej kolejności poznasz architekturę platformy, niezbędne narzędzia oraz techniki — to pomoże Ci sprawnie poruszać się w środowisku Magento. Po tym wstępie przejdziesz do bardziej zaawansowanych zagadnień. Poznasz model EAV oraz nauczysz się rozszerzać interfejs użytkownika. Ponadto przekonasz się, że stworzenie nowego modułu w panelu administracyjnym wcale nie musi być trudne. W tej książce znajdziesz również dokładny opis API platformy oraz dowiesz się, jak testować stworzony kod. Na sam koniec zobaczysz, w jaki sposób przygotować Twój produkt do wdrażania i dystrybucji. Książka ta jest obowiązkową lekturą dla wszystkich programistów PHP pracujących w środowisku Magento.

Dzięki tej książce:

- poznasz API Magento
- zaznajomisz się z modelem EAV
- przetestujesz stworzony kod
- przygotujesz Twój moduł do dystrybucji i wdrożenia

Wykorzystaj potencjał platformy Magento!

ISBN 978-83-246-8940-8

9 788324 689408

Cena: 39,90 zł