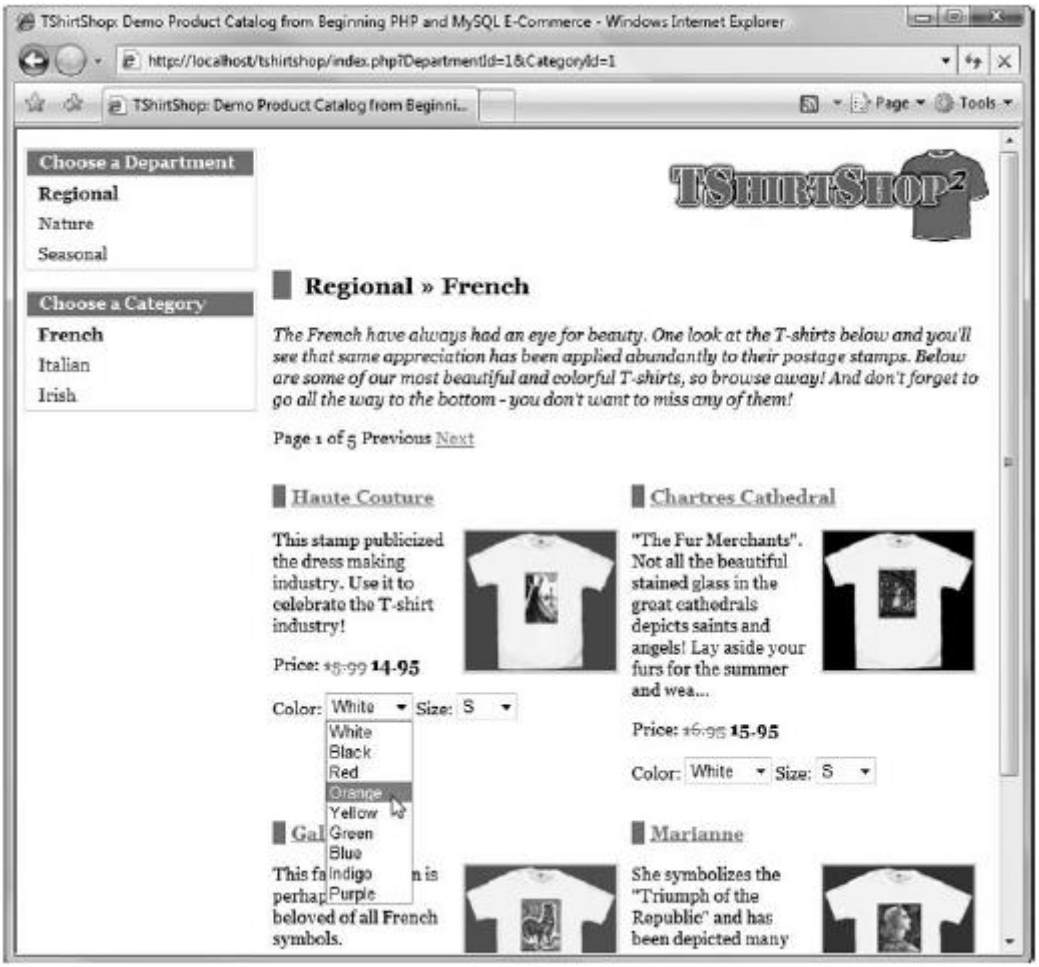
Cechy produktu

Wiele sklepów internetowych umożliwia kupującym dostosowanie kupowanych produktów. Na przykład podczas sprzedaży t-shirtów często pozwala się klientowi wybrać rozmiar i kolor koszulki – oszczędzając mu ryzyka związanego z modą, jeśli chodzi o jeden rozmiar w jednym kolorze dla wszystkich. W tej części zaimplementujemy funkcję atrybutów produktu w TShirtShop.



Zrobimy to, zaczynając od warstwy danych, gdzie utworzysz tabele danych i procedurę składowaną, następnie napiszesz kod warstwy biznesowej, który wywołuje tę procedurę, a na koniec użyjemy tej funkcji do zaktualizowania szablonu Smarty dla interfejsu użytkownika. Na końcu tej części nasz katalog pozwoli klientom wybrać rozmiar i kolor koszulki, jak pokazano na rysunku 6-1. Ponieważ dane atrybutów są przechowywane w bazie danych, możesz dodawać własne atrybuty i wartości atrybutów.

Implementacja warstwy danych

Składniki warstwy danych, które obsługują funkcję atrybutów produktu, obejmują trzy tabele danych (atrybut, atrybut_wartość i atrybut_produktu) oraz procedurę składowaną o nazwie catalog_get_product_attributes.

Poniżej znajdują się trzy tabele danych:

• atrybut przechowuje nazwy atrybutów, takie jak Rozmiar i Kolor.

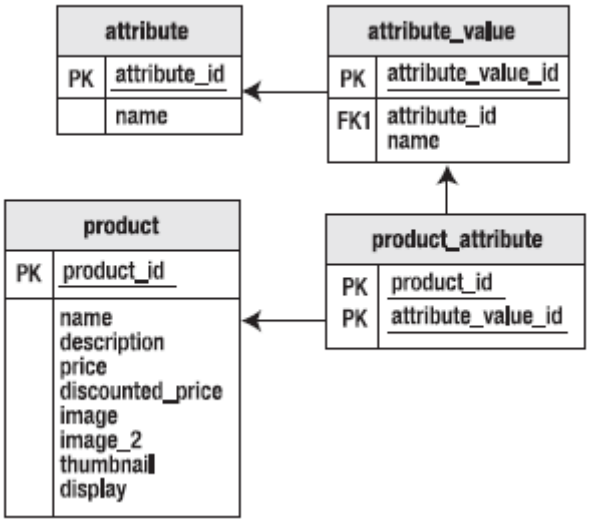
• atrybut_wartość zawiera możliwe wartości atrybutów dla każdej grupy atrybutów. Istnieje relacja jeden-do-wielu między atrybutem a wartością_atrybutu. Z każdym atrybutem, weźmy na przykład

kolor, może być powiązanych kilka wartości — czerwony, pomarańczowy, żółty i tak dalej. Potrzebujemy tej tabeli, aby pomóc nam połączyć identyfikator atrybutu (jak 1 dla koloru) z jego identyfikatorami_wartości_atrybutu dla jego możliwych wartości (czerwony, pomarańczowy, żółty i tak dalej). Będzie więc zawierał trzy kolumny: atrybut_id, który znajduje się w tabeli atrybutów i mówi nam, o jakim rodzaju atrybutu mówimy (np. kolor lub rozmiar); atrybut_wartość_id, liczba całkowita, którą przypisujemy do jednoznacznej identyfikacji elementów w samej tabeli atrybut_wartość; i wartość, która będzie zawierać opis tekstowy, taki jak Pomarańczowy, Czerwony, S (małe), M (średnie) i tak dalej.

• product_attribute to tabela skojarzona implementująca relację wiele-do-wielu między tabelą produkt i atrybut_wartość, z parami (id_produktu, id_wartości_atrybutu) (tabela atrybut_wartość pozwoli nam szybko i łatwo wypełnić tę tabelę).

Uwaga Wdrażany system nie pozwala na zmianę ceny produktu przez atrybut. Na przykład biała koszulka zawsze będzie miała tę samą cenę, co czarna. Jeśli chcesz mieć różne ceny produktów, musisz stworzyć różne produkty. Jeśli potrzebujesz obsługiwać atrybuty, które mają wpływ na cenę produktu, jedną z opcji byłoby rozszerzenie tabeli product_attribute przez uwzględnienie danych o zmianie ceny produktu oraz wprowadzenie wielu innych zmian i dodatków do kodu warstwy biznesowej i warstwy prezentacji. Radzimy nie dokonywać w tym momencie żadnych zmian, ponieważ rozprzestrzeniłyby się one również na kolejne Części, utrudniając zadanie podążania za księgą.

W Części 5 dowiedziałeś się, że diagramy bazy danych mogą pomóc w zrozumieniu relacji między tabelami danych. Wizualną reprezentację tych tabel i relacji między tabelami atrybutów można zobaczyć na rysunku.



Jeśli jesteś nowy w bazach danych, powinieneś poświęcić chwilę na przestudiowanie tego diagramu, aby poniższy kod był łatwiejszy do negocjowania i zrozumienia. Zacznijmy od aktualizacji bazy danych tshirtshop, a następnie zaimplementujmy nową procedurę składowaną catalog_get_product_attributes w poniższym ćwiczeniu.

Ćwiczenie: Tworzenie funkcjonalności warstwy danych dla atrybutów produktu

1. Otwórz phpMyAdmin i wykonaj następujący kod, który utworzy i wypełni tabelę atrybutów:

-- Create attribute table (stores attributes such as Size and Color)

```
CREATE TABLE `attribute` (
`attribute_id` INT NOT NULL AUTO_INCREMENT,
`name` VARCHAR(100) NOT NULL, -- E.g. Color, Size
PRIMARY KEY (`attribute_id`)
) ENGINE=MyISAM;
-- Populate attribute table
INSERT INTO `attribute` (`attribute_id`, `name`) VALUES
(1, 'Size'), (2, 'Color');
```

2. Kontynuuj, tworząc i wypełniając tabelę atrybut_wartość, używając następującego kodu SQL:

```
-- Create attribute_value table (stores values such as Yellow or XXL)
CREATE TABLE `attribute_value` (
`attribute_value_id` INT NOT NULL AUTO_INCREMENT,
`attribute_id` INT NOT NULL, -- The ID of the attribute
`value` VARCHAR(100) NOT NULL, -- E.g. Yellow
PRIMARY KEY (`attribute_value_id`),
KEY `idx_attribute_value_attribute_id` (`attribute_id`)
) ENGINE=MyISAM;
-- Populate attribute_value table
INSERT INTO `attribute_value` (`attribute_value_id`, `attribute_id`, `value`)
VALUES
(1, 1, 'S'), (2, 1, 'M'), (3, 1, 'L'), (4, 1, 'XL'), (5, 1, 'XXL'),
(6, 2, 'White'), (7, 2, 'Black'), (8, 2, 'Red'), (9, 2, 'Orange'),
(10, 2, 'Yellow'), (11, 2, 'Green'), (12, 2, 'Blue'),
(13, 2, 'Indigo'), (14, 2, 'Purple');
```

3. Utwórz i wypełnij tabelę product_attribute, używając kodu z poniższej listy:

```
-- Create product_attribute table (associates attribute values to products)
CREATE TABLE `product_attribute` (
`product_id` INT NOT NULL,
`attribute_value_id` INT NOT NULL,
PRIMARY KEY (`product_id`, `attribute_value_id`)
) ENGINE=MyISAM;
```

```
-- Populate product_attribute table

INSERT INTO `product_attribute` (`product_id`, `attribute_value_id`)

SELECT `p`.`product_id`, `av`.`attribute_value_id`

FROM `product` `p`, `attribute_value` `av`;
```

4. Wykonaj następujący kod SQL, który utworzy procedurę składowaną catalog_get_product_attributes. Ta procedura składowana skojarzy atrybuty przypisane do produktu. Nie zapomnij ustawić ogranicznika na $$ przed wykonaniem kodu.

```
-- Create catalog_get_product_attributes stored procedure

CREATE PROCEDURE catalog_get_product_attributes(IN inProductId INT)

BEGIN

SELECT a.name AS attribute_name,

av.attribute_value_id, av.value AS attribute_value

FROM attribute_value av

INNER JOIN attribute a

ON av.attribute_id = a.attribute_id

WHERE av.attribute_value_id IN

(SELECT attribute_value_id

FROM product_attribute

WHERE product_id = inProductId)

ORDER BY a.name;

END$$
```

Jak to działa: logika danych dla atrybutów produktu

Tutaj omówimy kod użyty do wypełnienia atrybutu product_attribute tabeli i procedury przechowywanej catalog_get_product_attributes. Zanim przejdziesz dalej, upewnij się, że rozumiesz przeznaczenie każdego pola w tabelach atrybut i atrybut_wartość oraz kod używany do wypełniania tych tabel. Dokładne zrozumienie struktury tabeli może trochę potrwać. Nie będziemy tutaj powtarzać teorii, ale zachęcamy do zapoznania się z rozdziałami 4 i 5, gdzie szczegółowo omówiliśmy główne koncepcje projektowania relacyjnych baz danych. Podczas wypełniania atrybutu product_attribute celem jest powiązanie wszystkich istniejących wartości atrybutów (poprzez pole atrybut_wartość_id) z każdym z naszych produktów (poprzez pole identyfikator_produktu). Na naszej stronie nasze produkty to koszulki, a każdy z nich chcemy sprzedawać we wszystkich możliwych rozmiarach i kolorach. Kod wypełniający tabelę product_attribute używa polecenia INSERT INTO w celu wstawienia liczby rekordów utworzonych przez zapytanie SELECT:

```
INSERT INTO `product_attribute` (`product_id`, `attribute_value_id`)
```

W naszym przypadku zapytanie SELECT generujące dane do wstawienia do atrybutu product_attribute jest sprzężeniem krzyżowym. Ten typ JOIN tworzy iloczyn kartezjański między dwoma zestawami

danych. Wynikiem jest lista zawierająca wszystkie możliwe kombinacje między rekordami pierwszego zestawu danych a rekordami drugiego zestawu danych. Na przykład iloczyn kartezjański między {1, 2, 3} i {a, b, c}, który jest matematycznie zapisany jako {1, 2, 3} × {a, b, c}, jest następującym zbiorem danych : { {1, a}, {1, b}, {1, c}, {2, a}, {2, b}, {2, c}, {3, a}, {3, b}, {3, c} }. W naszym przypadku, jeśli tworzymy produkt kartezjański między identyfikatorami istniejących produktów a identyfikatorami istniejących wartości atrybutów, otrzymujemy listę utworzoną z elementów (product_id, attribute_value_id), czyli dokładnie taką listę, którą chcemy dodać do tabela atrybut_produktu. Podkreślono składnię do zaimplementowania tej operacji łączenia krzyżowego z MySQL:

INSERT INTO `product_attribute` (`product_id`, `attribute_value_id`)

SELECT `p`.`product_id`, `av`.`attribute_value_id`

FROM `product` `p`, `attribute_value` `av`;

Jest to z pewnością dobry sposób na tworzenie wielu rekordów w naszej bazie danych przy minimalnym wysiłku. Przykładowe dane naszej bazy danych zawierają 14 możliwych wartości atrybutów i 101 produktów. Operacja łączenia krzyżowego, która kojarzy wszystkie wartości atrybutów ze wszystkimi produktami, generuje 1414 (czyli 14 pomnożonych przez 101) rekordów w tabeli product_attribute. Zauważ, że użycie SELECT do utworzenia połączenia krzyżowego nie jest standardowym SQL; jeśli chcesz zaimplementować taką operację z innymi serwerami baz danych, być może będziesz musiał użyć innej składni. „Oficjalna” składnia implementacji łączeń krzyżowych określa użycie składni CROSS JOIN. Istnieje kilka alternatywnych metod użycia sprzężenia krzyżowego do wypełnienia tabeli product_attribute niezbędnymi danymi. Jedną z metod, którą naszym zdaniem warto tutaj omówić, jest użycie UNION, ponieważ jest ona dość często stosowana w takich sytuacjach. UNION sumuje wyniki wielu instrukcji SELECT w jeden zestaw wyników. Na przykład, jeśli używasz UNION dla dwóch zapytań, które zwracają po pięć rekordów, otrzymasz zestaw wyników składający się z dziesięciu rekordów. Oczywiście, aby UNION działał, wszystkie zaangażowane zapytania muszą zwracać tę samą liczbę kolumn zgodnych typów danych. Nie będziemy wchodzić w więcej szczegółów na temat UNION, ale jeśli chcesz zobaczyć implementację opartą na UNION sprzężenie krzyżowe, oto jest. Wymaga to większej liczby naciśnięć klawiszy, ale zapytanie zapewnia większą elastyczność; na przykład, używając UNION, możesz dodać do swoich produktów tylko niektóre wartości atrybutów.

-- Populate product_attribute table

INSERT INTO `product_attribute` (`product_id`, `attribute_value_id`)

SELECT product_id, 1 AS attribute_value_id FROM product

UNION SELECT product_id, 2 AS attribute_value_id FROM product

UNION SELECT product_id, 3 AS attribute_value_id FROM product

UNION SELECT product_id, 4 AS attribute_value_id FROM product

UNION SELECT product_id, 5 AS attribute_value_id FROM product

UNION SELECT product_id, 6 AS attribute_value_id FROM product

UNION SELECT product_id, 7 AS attribute_value_id FROM product

UNION SELECT product_id, 8 AS attribute_value_id FROM product

UNION SELECT product_id, 9 AS attribute_value_id FROM product

UNION SELECT product_id, 10 AS attribute_value_id FROM product

UNION SELECT product_id, 11 AS attribute_value_id FROM product

UNION SELECT product_id, 12 AS attribute_value_id FROM product

UNION SELECT product_id, 13 AS attribute_value_id FROM product

UNION SELECT product_id, 14 AS attribute_value_id FROM product;

Na koniec przyjrzyjmy się procedurze składowanej catalog_get_product_attributes. Ta procedura składowana odbiera jako parametr identyfikator produktu i zwraca listę atrybutów tego produktu. To poręczne małe urządzenie, do którego wywoła się warstwa biznesowa, aby uzyskać listę atrybutów naszych produktów, dzięki czemu nasi klienci mogą wybrać rozmiar i kolor dla każdej koszulki:

-- Create catalog_get_product_attributes stored procedure

CREATE PROCEDURE catalog_get_product_attributes(IN inProductId INT)

BEGIN

The SQL code in this procedure returns a list with the attribute_name, attribute_value_id, and attribute_

value for all the attributes of the mentioned product:

SELECT a.name AS attribute_name,

av.attribute_value_id, av.value AS attribute_value

FROM attribute_value av

INNER JOIN attribute a

ON av.attribute_id = a.attribute_id

WHERE av.attribute_value_id IN

(SELECT attribute_value_id

FROM product_attribute

WHERE product_id = inProductId)

ORDER BY a.name;

Aby sprawdzić, czy procedura działa tak, jak powinna, możesz wykonać kod procedury składowanej za pomocą phpMyAdmin, zastępując nazwy parametrów ich wartościami. Na przykład możesz wziąć poprzedni kod SQL, zastąpić inProductId identyfikatorem produktu, a następnie wykonać kod w phpMyAdmin. Alternatywnie możesz wywołać samą procedurę składowaną, ale ta operacja nie jest obsługiwana przez phpMyAdmin w momencie pisania tego tekstu. Aby wykonać procedurę składowaną, możesz użyć konsoli MySQL. Aby uruchomić konsolę w systemie Windows, musisz wykonać następujące polecenie w C:\xampp\mysql\bin (zakładając, że zainstalowałeś XAMPP).

mysql -u root

Jeśli twój serwer MySQL jest zabezpieczony hasłem (przy domyślnej instalacji XAMPP tak nie jest), będziesz musiał użyć również parametru -p, w którym to przypadku zostaniesz poproszony o podanie hasła:

mysql -u root -p

Po połączeniu się z serwerem MySQL musisz połączyć się z bazą danych tshirtshop:

USE tshirtshop;

Po wybraniu tshirtshop możesz wykonać procedurę catalog_get_product_attributes, podając 1 jako parametr, na przykład:

CALL directory_get_product_attributes (1);

Podczas wykonywania takiej procedury składowanej otrzymujesz wszystkie atrybuty i wartości atrybutów dla produktu o identyfikatorze 1 w Twojej bazie danych:

attribute_name	attribute_value_id	attribute_value
Color	6	White
Color	7	Black
Color	8	Red
Color	9	Orange
Color	10	Yellow
Color	11	Green
Color	12	Blue
Color	13	Indigo
Color	14	Purple
Size	1	S
Size	2	M
Size	3	L
Size	4	XL
Size	5	XXL

Wdrażanie poziomu biznesowego

Bit warstwy biznesowej funkcji atrybutów produktu jest bardzo prosty — wystarczy napisać kod, który wywołuje procedurę składowaną catalog_get_product_attributes. Dodaj następujący kod do klasy Catalog w business/catalog.php:

```
// Retrieves product attributes

public static function GetProductAttributes($productId)

{

// Build SQL query

$sql = 'CALL catalog_get_product_attributes(:product_id)';

// Build the parameters array

$params = array (':product_id' => $productId);
```

```
// Execute the query and return the results
return DatabaseHandler::GetAll($sql, $params);
}
```

Wdrażanie poziomu prezentacji

Utworzenie warstwy prezentacji oznacza dodanie kontrolek, które pozwalają wybrać wartość z listy wartości atrybutów produktu. Jak zobaczysz, atrybuty nie są zakodowane w szablonach. Zamiast tego atrybuty (takie jak Rozmiar i Kolor) i ich wartości (takie jak L, XL, Biały, Zielony itd.) są odczytywane z bazy danych Już teraz możesz zobaczyć, jak dokonywanie zmian w atrybutach dla wszystkich Twoich produktów jest o wiele łatwiej w ten sposób. Po prostu zmieniasz dane w tabelach atrybutów i voilà - zmiany są automatycznie odzwierciedlane na Twojej stronie! Zaimplementujmy kod prezentacji, a później omówimy go. Aby ułatwić zrozumienie, pamiętaj, aby skorelować kod, który wpisujesz teraz, z wynikami, jakich oczekujesz od procedur przechowywanych w katalogu_get_product_attributes. Nieco wcześniej w tej części widziałeś przykładowe wyniki tej procedury.

Ćwiczenie: Implementacja prezentacji atrybutów produktu

1. Dodaj następujący podświetlony kod do presentation\templates\products_list.tpl:

```
<p class="section">
Price:
{if $obj->mProducts[k].discounted_price != 0}
<span class="old-price">{$obj->mProducts[k].price}</span>
<span class="price">{$obj->mProducts[k].discounted_price}</span>
{else}
<span class="price">{$obj->mProducts[k].price}</span>
{/if}
</p>
{* Generate the list of attribute values *}
<p class="attributes">
{* Parse the list of attributes and attribute values *}
{section name=l loop=$obj->mProducts[k].attributes}
{* Generate a new select tag? *}
{if $smarty.section.l.first ||
$obj->mProducts[k].attributes[l].attribute_name !==
$obj->mProducts[k].attributes[l.index_prev].attribute_name}
{$obj->mProducts[k].attributes[l].attribute_name}:
<select name="attr_{$obj->mProducts[k].attributes[l].attribute_name}">
```

```
{/if}
{* Generate a new option tag *}
<option value="{$obj->mProducts[k].attributes[l].attribute_value}">
{$obj->mProducts[k].attributes[l].attribute_value}
</option>
{* Close the select tag? *}
{if $smarty.section.l.last ||
$obj->mProducts[k].attributes[l].attribute_name !==
$obj->mProducts[k].attributes[l.index_next].attribute_name}
</select>
{/if}
{/section}
</p>
</td>
{if $smarty.section.k.index % 2 != 0 && !$smarty.section.k.first ||
$smarty.section.k.last}
</tr>
{/if}
{/section}
</tbody>
</table>
{/if}
```

2. Dodaj podświetlony fragment kodu do metody init() w Presentation/products_list.php. Spowoduje to wyświetlenie listy atrybutów dla każdego produktu na liście.

```
// Build links for product details pages
for ($i = 0; $i < count($this->mProducts); $i++)
{
$this->mProducts[$i]['link_to_product'] =
Link::ToProduct($this->mProducts[$i]['product_id']);
if ($this->mProducts[$i]['thumbnail'])
$this->mProducts[$i]['thumbnail'] =
```

```
Link::Build('product_images/' . $this->mProducts[$i]['thumbnail']);
$this->mProducts[$i]['attributes'] =
Catalog::GetProductAttributes($this->mProducts[$i]['product_id']);
}
}
}
?>
```

3. Otwórz plik Presentation/templates/product.tpl i dodaj podświetlony kod, który generuje listę wartości atrybutów dla strony szczegółów produktu:

```
<p class="section">
Price:
{if $obj->mProduct.discounted_price != 0}
<span class="old-price">{$obj->mProduct.price}</span>
<span class="price">{$obj->mProduct.discounted_price}</span>
{else}
<span class="price">{$obj->mProduct.price}</span>
```



```
{/if}
</p>
{* Generate the list of attribute values *}
<p class="attributes">
{* Parse the list of attributes and attribute values *}
{section name=k loop=$obj->mProduct.attributes}
{* Generate a new select tag? *}
{if $smarty.section.k.first ||
$obj->mProduct.attributes[k].attribute_name !==
$obj->mProduct.attributes[k.index_prev].attribute_name}
{$obj->mProduct.attributes[k].attribute_name}:
<select name="attr_{$obj->mProduct.attributes[k].attribute_name}">
{/if}
```

```
{* Generate a new option tag *}
<option value="{$obj->mProduct.attributes[k].attribute_value}">
{$obj->mProduct.attributes[k].attribute_value}
</option>
{* Close the select tag? *}
{if $smarty.section.k.last ||
$obj->mProduct.attributes[k].attribute_name !==
$obj->mProduct.attributes[k.index_next].attribute_name}
</select>
{/if}
{/section}
</p>
{if $obj->mLinkToContinueShopping}
<a href="{$obj->mLinkToContinueShopping}">Continue Shopping</a>
{/if}
```

4. Dodaj podświetlony kod do metody init() klasy Product z pliku Presentation/product.php:

```
if ($this->mProduct['image_2'])
$this->mProduct['image_2'] =
Link::Build('product_images/' . $this->mProduct['image_2']);
$this->mProduct['attributes'] =
Catalog::GetProductAttributes($this->mProduct['product_id']);
$this->mLocations = Catalog::GetProductLocations($this->_mProductId);
```

5. Dodaj następujący styl do styles/tshirtshop.css:

```
.attributes {
clear: both;
display: block;
padding-top: 5px;
}
```

6. Załaduj http://localhost/tshirtshop/ i podziwiaj nowe atrybuty produktu w akcji. Rysunek 6-3 przedstawia atrybuty produktu na stronie szczegółów produktu. Rysunek przedstawia atrybuty produktu na liście produktów.

Jak to działa: prezentacja atrybutów produktu

Interfejs użytkownika dla atrybutów produktu został utworzony w dwóch krokach. Wywołałeś metodę Catalog::GetProductAttributes() warstwy biznesowej, aby uzyskać listę atrybutów dla określonego produktu. Następnie zaktualizowałeś szablon products_list.tpl, aby wyświetlić listę atrybutów dla każdego produktu. W tej chwili te atrybuty nie mają większego znaczenia dla klienta, ponieważ produktów nie można jeszcze zamówić - ale o tym szczegółu zadbamy w kolejnych rozdziałach. Wyzwanie ćwiczenia polega na zrozumieniu, w jaki sposób atrybuty produktu są wyświetlane na stronie szczegółów produktu oraz na listach produktów. Przydatną strategią zrozumienia, jak działa kod Smarty, jest analiza danych wejściowych otrzymanych przez Smarty i kodu, który generuje. Przyjrzyjmy się raz jeszcze dane atrybutów zwracane przez procedurę catalog_get_product_attributes dla produktu o identyfikatorze 1:

attribute_name	attribute_value_id	attribute_value
Color	6	White
Color	7	Black
Color	8	Red
Color	9	Orange
Color	10	Yellow
Color	11	Green
Color	12	Blue
Color	13	Indigo
Color	14	Purple
Size	1	S
Size	2	M
Size	3	L
Size	4	XL
Size	5	XXL

Teraz spójrzmy na kod źródłowy HTML wygenerowany dla strony szczegółów tego produktu. Możesz załadować tę stronę pod adresem http://localhost/tshirtshop/index.php?ProductId=1 i możesz zobaczyć jej źródło HTML, klikając prawym przyciskiem myszy pusty obszar strony i wybierając opcję Wyświetl źródło (w Internet Explorerze) lub Wyświetl źródło strony (w Firefoksie). Oto, co znajdziesz. Usunęliśmy kilka opcji, aby lista była łatwiejsza do odczytania, i dodaliśmy komentarze Smarty, które pomogą Ci skorelować szablon źródłowy z danymi wyjściowymi, chociaż komentarze nie pojawiają się w rzeczywistych danych wyjściowych:

{* Generate the list of attribute values *}

<p class="attributes">

{* Parse the list of attributes and attribute values *}

{* Generate a new select tag? *}

Color:

<select name="attr_Color">

{* Generate a new option tag *}

<option value="White">

White

</option>

{* Close the select tag? *}

{* Generate a new select tag? *}

{* Generate a new option tag *}

<option value="Black">

Black

```
</option>
{* Close the select tag? *}
{* Generate a new select tag? *}
{* Generate a new option tag *}
<option value="Red">
Red
</option>
{* Close the select tag? *}
{* Generate a new select tag? *}
...
...
{* Close the select tag? *}
</select>
{* Generate a new select tag? *}
Size:
<select name="attr_Size">
{* Generate a new option tag *}
<option value="S">
S
</option>
{* Close the select tag? *}
{* Generate a new select tag? *}
{* Generate a new option tag *}
<option value="M">
M
</option>
...
...
{* Close the select tag? *}
</select>
</p>
```

Jak więc obliczana jest ta wydajność? Kluczem jest logika Smarty zawarta w szablonach product.tpl i products_list.tpl. Ten kod analizuje listę atrybutów ładowanych przez warstwę biznesową. Te dane są w rzeczywistości danymi zwróconymi przez procedurę składowaną catalog_get_product_attributes:

{section name=k loop=$obj->mProduct.attributes}

Dla każdego elementu tej listy podejmuje się kilka kroków, aby zdecydować, jaki rodzaj danych wyjściowych ma zostać wygenerowany. Weźmy na przykład poniższy fragment kodu, który jest instrukcją warunkową {if}, która zwraca prawdę dla pierwszego elementu oraz dla elementów, których nazwa atrybutu jest inna niż nazwa atrybutu poprzedniego elementu:

{* Generate a new select tag? *}

{if $smarty.section.k.first ||

$obj->mProduct.attributes[k].attribute_name !==

$obj->mProduct.attributes[k.index_prev].attribute_name}

Krótko mówiąc, ten warunek zwraca prawdę za każdym razem, gdy trzeba wyświetlić nowy atrybut (taki jak rozmiar i kolor). Kiedy tak się stanie, generujemy nowy element <select>, rozpoczynając nowy element selekcji:

{$obj->mProduct.attributes[k].attribute_name}:

<select name="attr_{$obj->mProduct.attributes[k].attribute_name}">

{/if}

Reszta kodu przebiega w tych samych liniach. Poświęć kilka minut, aby upewnić się, że wszystko rozumiesz, a następnie pogratuluj sobie ukończenia kolejnej ważnej funkcji witryny internetowej TShirtShop!

Podsumowanie

W tej małej części dodaliśmy atrybuty do naszych produktów! Od teraz nasi klienci będą mogli wybrać kolory i rozmiary swoich koszulek. Ponadto bardzo łatwo jest dodawać własne grupy atrybutów i wartości atrybutów według własnego uznania. Nasz sklep z T-shirtami naprawdę się rozwija! Ale nie będziemy dobrze konkurować w Internecie, jeśli wyszukiwarki będą miały trudności ze zrozumieniem i rankingiem naszej witryny. Co gorsza, jeśli nie przygotujemy starannie naszej witryny pod kątem wyszukiwarek, możemy w rzeczywistości stracić pozycję w rankingu, zamiast ją zyskać! Ponieważ modernizacja witryny pod kątem optymalizacji pod kątem wyszukiwarek może być bardzo trudna, ważne jest, abyśmy zajęli się tym na wczesnym etapie projektowania i tworzenia. W niektórych przypadkach próba modernizacji witryny jest tak trudna, że programiści postanawiają po prostu zacząć od nowa! Ale wiemy, że musimy planować od początku, więc przejdźmy teraz do ulepszania naszego katalogu pod kątem optymalizacji pod kątem wyszukiwarek.