Administracja katalogiem: produkty i atrybuty

Twoi administratorzy mogą teraz edytować działy i kategorie Twojej witryny e-commerce. W tym rozdziale dodajemy brakujące funkcje związane z zarządzaniem produktami i ich atrybutami. Dokładniej, w tej części zaimplementujemy funkcje, które pozwolą administratorowi witryny wykonać następujące czynności:

• Zarządzaj atrybutami produktów, w szczególności wyświetlaj wartości atrybutów, dodawaj i usuwaj atrybuty oraz przypisuj atrybuty do produktów.

• Zobacz listę produktów w określonej kategorii.

• Edytuj szczegóły produktu, takie jak nazwa produktu, opis, cena lub informacja, czy jest on objęty promocją.

• Przypisz istniejący produkt do dodatkowej kategorii (produkt może należeć do wielu kategorii) lub przenieś go do innej kategorii.

• Usuń produkt z kategorii.

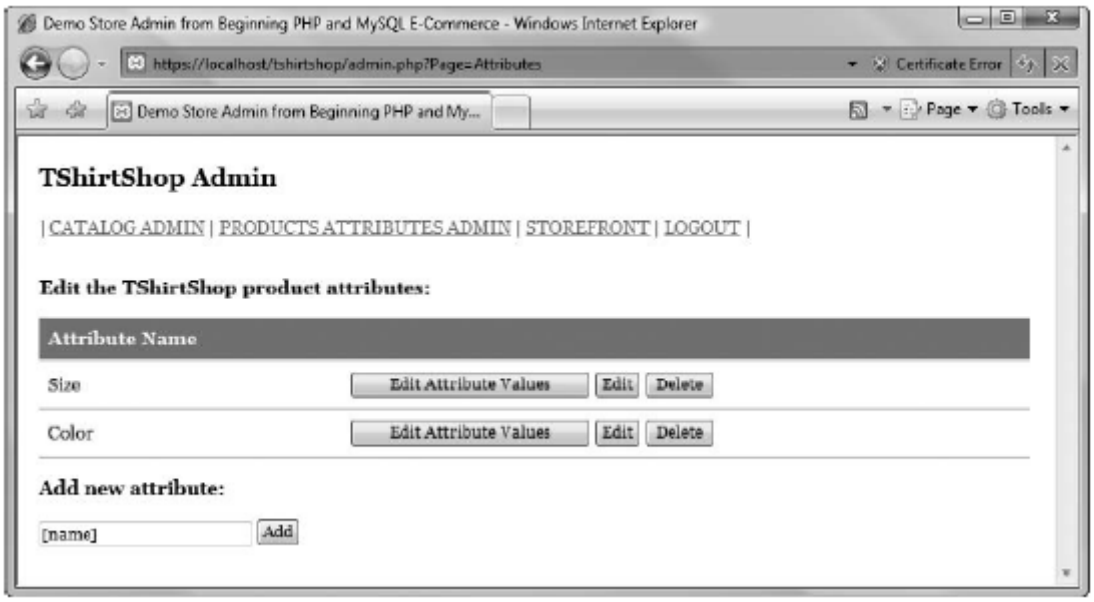
• Usuń produkt z katalogu.

• Zezwalaj administratorom na dostęp do stron administracyjnych działów, kategorii lub produktów bezpośrednio z katalogu.

Jest sporo do przejścia, ale niewiele więcej niż to, co musiałeś znieść w poprzedniej części rozdziale. Zachęcamy do przyjrzenia się liczbom, aby zobaczyć wygląd nowych funkcji. Zaczynajmy!

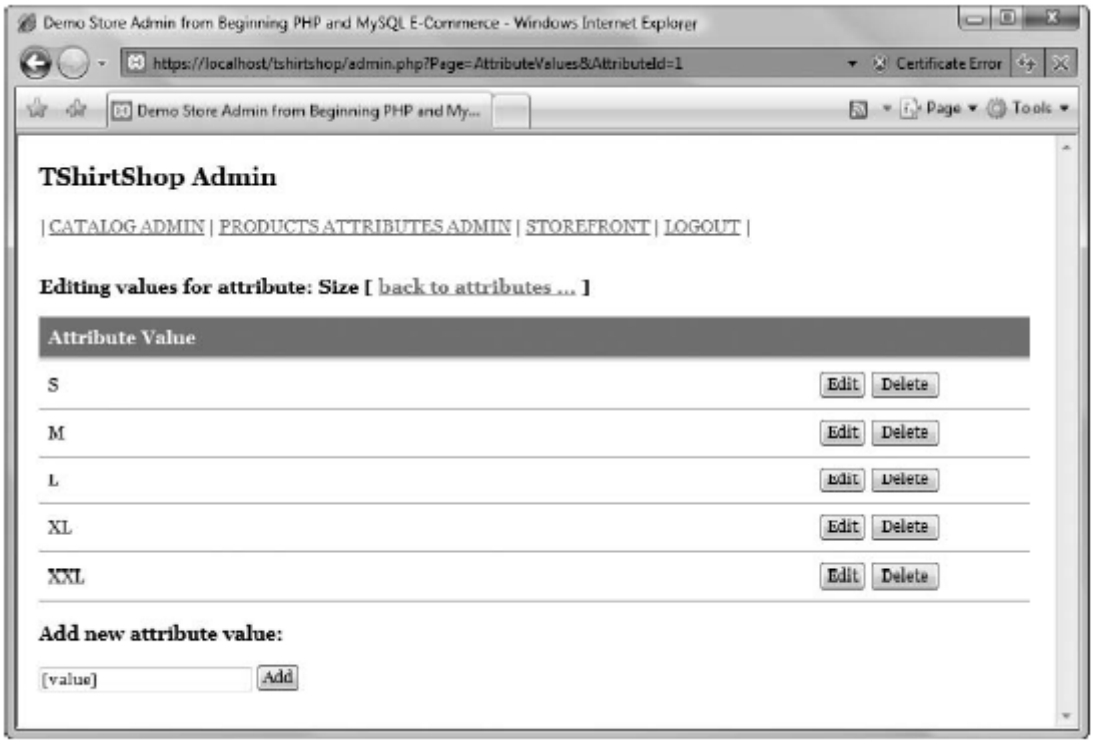
Administrowanie atrybutami produktów

Pierwszą funkcją, którą stworzymy w tym rozdziale, jest strona zarządzania atrybutami produktów. Ta strona jest dostępna za pośrednictwem nowego łącza w menu administracyjnym, które brzmi „ADMIN ATRYBUTÓW PRODUKTU”. Kliknięcie tego linku przeniesie Cię do strony pokazanej na rysunku



Jak wiecie z Części 6, gdzie dodaliśmy obsługę atrybutów produktów, każdy atrybut może mieć kilka możliwych wartości atrybutów. Na przykład w przypadku atrybutu Kolor możliwe wartości mogą

obejmować zielony, różowy, czarny itd. Gdy produkt obsługuje atrybut, musimy również określić możliwe wartości atrybutów, które klient musi wybrać podczas dokonywania zakupu. Kliknięcie łącza Edytuj wartości atrybutów dla atrybutu przeniesie Cię do strony, na której możesz zarządzać możliwymi wartościami tego atrybutu, jak pokazano na rysunku



Zaimplementujmy te funkcje w ćwiczeniu, a szczegóły omówimy później.

Ćwiczenie: Implementacja admin_products

1. Utwórz nowy plik szablonu o nazwie admin_attributes.tpl w pliku folderu prezentacji/templates i dodaj do niego następujący kod:

```
{* admin_attributes.tpl *}

{load_presentation_object filename="admin_attributes" assign="obj"}

<form method="post"

action="{$obj->mLinkToAttributesAdmin}">

<h3>Edit the TShirtShop product attributes:</h3>

{if $obj->mErrorMessage}<p class="error">{$obj->mErrorMessage}</p>{/if}

{if $obj->mAttributesCount eq 0}

<p class="no-items-found">

There are no products attributes in your database!

</p>

{else}
```

```
<table class="tss-table">
<tr>
<th>Attribute Name</th>
<th width="240"> </th>
</tr>
{section name=i loop=$obj->mAttributes}
{if $obj->mEditItem == $obj->mAttributes[i].attribute_id}
<tr>
<td>
<input type="text" name="name"
value="{$obj->mAttributes[i].name}" size="30" />
</td>
<td>
<input type="submit"
name="submit_edit_attr_val_{$obj->mAttributes[i].attribute_id}"
value="Edit Attribute Values" />
<input type="submit"
name="submit_update_attr_{$obj->mAttributes[i].attribute_id}"
value="Update" />
<input type="submit" name="cancel" value="Cancel" />
<input type="submit"
name="submit_delete_attr_{$obj->mAttributes[i].attribute_id}"
value="Delete" />
</td>
</tr>
{else}
<tr>
<td>{$obj->mAttributes[i].name}</td>
<td>
<input type="submit"
name="submit_edit_val_{$obj->mAttributes[i].attribute_id}"
```

```
value="Edit Attribute Values" />
<input type="submit"
name="submit_edit_attr_{$obj->mAttributes[i].attribute_id}"
value="Edit" />
<input type="submit"
name="submit_delete_attr_{$obj->mAttributes[i].attribute_id}"
value="Delete" />
</td>
</tr>
{/if}
{/section}
</table>
{/if}
<h3>Add new attribute:</h3>
<p>
<input type="text" name="attribute_name" value="[name]" size="30" />
<input type="submit" name="submit_add_attr_0" value="Add" />
</p>
</form>
```

2. Utwórz nowy plik obiektu prezentacji o nazwie admin_attributes.php w folderze prezentacji i dodaj do niego następujący kod:

```
<?php
// Class that supports attributes admin functionality
class AdminAttributes
{
// Public variables available in smarty template
public $mAttributesCount;
public $mAttributes;
public $mErrorMessage;
public $mEditItem;
public $mLinkToAttributesAdmin;
```

```
// Private members
private $_mAction;
private $_mActionedAttributeId;
// Class constructor
public function __construct()
{
// Parse the list with posted variables
foreach ($_POST as $key => $value)
// If a submit button was clicked ...
if (substr($key, 0, 6) == 'submit')
{
/* Get the position of the last '_' underscore from submit
button name e.g strtpos('submit_edit_attr_1', '_') is 17 */
$last_underscore = strrpos($key, '_');
/* Get the scope of submit button
(e.g 'edit_dep' from 'submit_edit_attr_1') */
$this->_mAction = substr($key, strlen('submit_'),
$last_underscore - strlen('submit_'));
/* Get the attribute id targeted by submit button
(the number at the end of submit button name)
e.g '1' from 'submit_edit_attr_1' */
$this->_mActionedAttributeId = substr($key, $last_underscore + 1);
break;
}
$this->mLinkToAttributesAdmin = Link::ToAttributesAdmin();
}
public function init()
{
// If adding a new attribute ...
if ($this->_mAction == 'add_attr')
{
```

```
$attribute_name = $_POST['attribute_name'];
if ($attribute_name == null)
$this->mErrorMessage = 'Attribute name required';
if ($this->mErrorMessage == null)
{
Catalog::AddAttribute($attribute_name);
header('Location: ' . $this->mLinkToAttributesAdmin);
}
}
// If editing an existing attribute ...
if ($this->_mAction == 'edit_attr')
$this->mEditItem = $this->_mActionedAttributeId;
// If updating an attribute ...
if ($this->_mAction == 'update_attr')
{
$attribute_name = $_POST['name'];
if ($attribute_name == null)
$this->mErrorMessage = 'Attribute name required';
if ($this->mErrorMessage == null)
{
Catalog::UpdateAttribute($this->_mActionedAttributeId,
$attribute_name);
header('Location: ' . $this->mLinkToAttributesAdmin);
}
}
// If deleting an attribute ...
if ($this->_mAction == 'delete_attr')
{
$status = Catalog::DeleteAttribute($this->_mActionedAttributeId);
if ($status < 0)
$this->mErrorMessage =
```

```
'Attribute has one or more values and cannot be deleted';
else
header('Location: ' . $this->mLinkToAttributesAdmin);
}
// If editing an attribute value ...
if ($this->_mAction == 'edit_val')
{
header('Location: ' .
htmlspecialchars_decode(
Link::ToAttributeValuesAdmin(
$this->_mActionedAttributeId)));
exit();
}
// Load the list of attributes
$this->mAttributes = Catalog::GetAttributes();
$this->mAttributesCount = count($this->mAttributes);
}
}
?>
```

3. Utwórz nowy plik szablonu o nazwie admin_attribute_values.tpl w pliku folderu prezentacji/templates i dodaj do niego następujący kod:

```
{* admin_attribute_values.tpl *}
{load_presentation_object filename="admin_attribute_values" assign="obj"}
<form method="post"
action="{$obj->mLinkToAttributeValuesAdmin}">
<h3>
Editing values for attribute: {$obj->mAttributeName} [
<a href="{$obj->mLinkToAttributesAdmin}">back to attributes ...</a> ]
</h3>
{if $obj->mErrorMessage}<p class="error">{$obj->mErrorMessage}</p>{/if}
{if $obj->mAttributeValuesCount eq 0}
```

```
<p class="no-items-found">There are no values for this attribute!</p>
{else}
<table class="tss-table">
<tr>
<th>Attribute Value</th>
<th width="170"> </th>
</tr>
{section name=i loop=$obj->mAttributeValues}
{if $obj->mEditItem == $obj->mAttributeValues[i].attribute_value_id}
<tr>
<td>
<input type="text" name="value"
value="{$obj->mAttributeValues[i].value}" size="30" />
</td>
<td>
<input type="submit"
name="submit_update_val_{$obj->mAttributeValues[i].attribute_value_id}"
value="Update" />
<input type="submit" name="cancel" value="Cancel" />
<input type="submit"
name="submit_delete_val_{$obj->mAttributeValues[i].attribute_value_id}"
value="Delete" />
</td>
</tr>
{else}
<tr>
<td>{$obj->mAttributeValues[i].value}</td>
<td>
<input type="submit"
name="submit_edit_val_{$obj->mAttributeValues[i].attribute_value_id}"
value="Edit" />
```

```
<input type="submit"
name="submit_delete_val_{$obj->mAttributeValues[i].attribute_value_id}"
value="Delete" />
</td>
</tr>
{/if}
{/section}
</table>
{/if}
<h3>Add new attribute value:</h3>
<input type="text" name="attribute_value" value="[value]" size="30" />
<input type="submit" name="submit_add_val_0" value="Add" />
</form>
```

4. Utwórz nowy plik obiektu prezentacji o nazwie admin_attribute_values.php w folderze prezentacji i dodaj do niego:

```
<?php
// Class that deals with attribute values admin
class AdminAttributeValues
{
// Public variables available in smarty template
public $mAttributeValuesCount;
public $mAttributeValues;
public $mErrorMessage;
public $mEditItem;
public $mAttributeId;
public $mAttributeName;
public $mLinkToAttributeAdmin;
public $mLinkToAttributeValuesAdmin;
// Private members
private $_mAction;
private $_mActionedAttributeValueId;
```

```
// Class constructor
public function __construct()
{
if (isset ($_GET['AttributeId']))
$this->mAttributeId = (int)$_GET['AttributeId'];
else
trigger_error('AttributeId not set');
$attribute_details = Catalog::GetAttributeDetails($this->mAttributeId);
$this->mAttributeName = $attribute_details['name'];
foreach ($_POST as $key => $value)
// If a submit button was clicked ...
if (substr($key, 0, 6) == 'submit')
{
/* Get the position of the last '_' underscore from submit
button name e.g strtpos('submit_edit_val_1', '_') is 16 */
$last_underscore = strrpos($key, '_');
/* Get the scope of submit button
(e.g 'edit_cat' from 'submit_edit_val_1') */
$this->_mAction = substr($key, strlen('submit_'),
$last_underscore - strlen('submit_'));
/* Get the attribute value id targeted by submit button
(the number at the end of submit button name)
e.g '1' from 'submit_edit_val_1' */
$this->_mActionedAttributeValueId =
(int)substr($key, $last_underscore + 1);
break;
}
$this->mLinkToAttributesAdmin = Link::ToAttributesAdmin();
$this->mLinkToAttributeValuesAdmin =
Link::ToAttributeValuesAdmin($this->mAttributeId);
}
```

```
public function init()
{
// If adding a new attribute value ...
if ($this->_mAction == 'add_val')
{
$attribute_value = $_POST['attribute_value'];
if ($attribute_value == null)
$this->mErrorMessage = 'Attribute value is empty';
if ($this->mErrorMessage == null)
{
Catalog::AddAttributeValue($this->mAttributeId, $attribute_value);
header('Location: ' .
htmlspecialchars_decode(
$this->mLinkToAttributeValuesAdmin));
}
}
// If editing an existing attribute value ...
if ($this->_mAction == 'edit_val')
{
$this->mEditItem = $this->_mActionedAttributeValueId;
}
// If updating an attribute value ...
if ($this->_mAction == 'update_val')
{
$attribute_value = $_POST['value'];
if ($attribute_value == null)
$this->mErrorMessage = 'Attribute value is empty';
if ($this->mErrorMessage == null)
{
Catalog::UpdateAttributeValue(
$this->_mActionedAttributeValueId, $attribute_value);
```

```
header('Location: ' .
htmlspecialchars_decode(
$this->mLinkToAttributeValuesAdmin));
}
}
// If deleting an attribute value ...
if ($this->_mAction == 'delete_val')
{
$status =
Catalog::DeleteAttributeValue($this->_mActionedAttributeValueId);
if ($status < 0)
$this->mErrorMessage = 'Cannot delete this attribute value. ' .
'One or more products are using it!';
else
header('Location: ' .
htmlspecialchars_decode(
$this->mLinkToAttributeValuesAdmin));
}
// Load the list of attribute values
$this->mAttributeValues =
Catalog::GetAttributeValues($this->mAttributeId);
$this->mAttributeValuesCount = count($this->mAttributeValues);
}
}
?>
```

5. Otwórz plik Presentation/templates/admin_menu.tpl i dodaj następujący podświetlony kod, aby dodać link do strony administratora atrybutów:

```
<p> |
<a href="{$obj->mLinkToStoreAdmin}">CATALOG ADMIN</a> |
<a href="{$obj->mLinkToAttributesAdmin}">PRODUCTS ATTRIBUTES ADMIN</a> |
<a href="{$obj->mLinkToStoreFront}">STOREFRONT</a> |
```

```
<a href="{$obj->mLinkToLogout}">LOGOUT</a> |
</p>
```

6. Otwórz plik Presentation/admin_menu.php i dodaj następujący podświetlony kod:

```
<?php
class AdminMenu
{
public $mLinkToStoreAdmin;
public $mLinkToAttributesAdmin;
public $mLinkToStoreFront;
public $mLinkToLogout;
public function __construct()
{
$this->mLinkToStoreAdmin = Link::ToAdmin();
$this->mLinkToAttributesAdmin = Link::ToAttributesAdmin();
$this->mLinkToStoreFront = Link::ToIndex();
$this->mLinkToLogout = Link::ToLogout();
}
}
?>
```

7. Teraz otwórz plik Presentation/link.php i dodaj następujące metody na końcu klasy Link:

```
// Create link to the attributes administration page
public static function ToAttributesAdmin()
{
return self::ToAdmin('Page=Attributes');
}
// Create link to the attribute values administration page
public static function ToAttributeValuesAdmin($attributeId)
{
$link = 'Page=AttributeValues&AttributeId=' . $attributeId;
return self::ToAdmin($link);
}
```

8. Otwórz business/catalog.php, aby dodać następujące metody warstwy biznesowej do klasy Catalog. Te metody są potrzebne do zarządzania atrybutami:

```
// Retrieves all attributes
public static function GetAttributes()
{
// Build the SQL query
$sql = 'CALL catalog_get_attributes()';
// Execute the query and return the results
return DatabaseHandler::GetAll($sql);
}
// Add an attribute
public static function AddAttribute($attributeName)
{
// Build the SQL query
$sql = 'CALL catalog_add_attribute(:attribute_name)';
// Build the parameters array
$params = array (':attribute_name' => $attributeName);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
// Updates attribute name
public static function UpdateAttribute($attributeId, $attributeName)
{
// Build the SQL query
$sql = 'CALL catalog_update_attribute(:attribute_id, :attribute_name)';
// Build the parameters array
$params = array (':attribute_id' => $attributeId,
':attribute_name' => $attributeName);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
```

```
// Deletes an attribute
public static function DeleteAttribute($attributeId)
{
// Build the SQL query
$sql = 'CALL catalog_delete_attribute(:attribute_id)';
// Build the parameters array
$params = array (':attribute_id' => $attributeId);
// Execute the query and return the results
return DatabaseHandler::GetOne($sql, $params);
}
// Retrieves details for the specified attribute
public static function GetAttributeDetails($attributeId)
{
// Build SQL query
$sql = 'CALL catalog_get_attribute_details(:attribute_id)';
// Build the parameters array
$params = array (':attribute_id' => $attributeId);
// Execute the query and return the results
return DatabaseHandler::GetRow($sql, $params);
}
// Gets atribute values
public static function GetAttributeValues($attributeId)
{
// Build the SQL query
$sql = 'CALL catalog_get_attribute_values(:attribute_id)';
// Build the parameters array
$params = array (':attribute_id' => $attributeId);
// Execute the query and return the results
return DatabaseHandler::GetAll($sql, $params);
}
// Adds an attribute value
```

```
public static function AddAttributeValue($attributeId, $attributeValue)
{
// Build the SQL query
$sql = 'CALL catalog_add_attribute_value(:attribute_id, :value)';
// Build the parameters array
$params = array (':attribute_id' => $attributeId,
':value' => $attributeValue);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
// Updates an attribute value
public static function UpdateAttributeValue(
$attributeValueId, $attributeValue)
{
// Build the SQL query
$sql = 'CALL catalog_update_attribute_value(
:attribute_value_id, :value)';
// Build the parameters array
$params = array (':attribute_value_id' => $attributeValueId,
':value' => $attributeValue);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
// Deletes an attribute value
public static function DeleteAttributeValue($attributeValueId)
{
// Build the SQL query
$sql = 'CALL catalog_delete_attribute_value(:attribute_value_id)';
// Build the parameters array
$params = array (':attribute_value_id' => $attributeValueId);
// Execute the query and return the results
```

return DatabaseHandler::GetOne($sql, $params);

}

9. Zmodyfikuj metodę init() klasy StoreAdmin znajdującą się w pliku Presentation/store_admin.php, aby załadować nowo dodane szablony składowe:

// Choose what admin page to load ...

if ($admin_page == 'Departments')

$this->mContentsCell = 'admin_departments.tpl';

elseif ($admin_page == 'Categories')

$this->mContentsCell = 'admin_categories.tpl';

elseif ($admin_page == 'Attributes')

$this->mContentsCell = 'admin_attributes.tpl';

elseif ($admin_page == 'AttributeValues')

$this->mContentsCell = 'admin_attribute_values.tpl';

10. Użyj phpMyAdmin, aby wykonać następujący kod, który tworzy procedury składowane warstwy danych w bazie danych tshirtshop. Nie zapomnij ustawić separatora $$. (Pamiętaj, że jeśli chcesz zobaczyć listę wszystkich procedur składowanych w bazie danych tshirtshop, możesz użyć polecenia SHOW PROCEDURE STATUS).

-- Create catalog_get_attributes stored procedure

CREATE PROCEDURE catalog_get_attributes()

BEGIN

SELECT attribute_id, name FROM attribute ORDER BY attribute_id;

END$$

-- Create catalog_add_attribute stored procedure

CREATE PROCEDURE catalog_add_attribute(IN inName VARCHAR(100))

BEGIN

INSERT INTO attribute (name) VALUES (inName);

END$$

-- Create catalog_update_attribute stored procedure

CREATE PROCEDURE catalog_update_attribute(

IN inAttributeId INT, IN inName VARCHAR(100))

BEGIN

UPDATE attribute SET name = inName WHERE attribute_id = inAttributeId;

```
END$$
-- Create catalog_delete_attribute stored procedure
CREATE PROCEDURE catalog_delete_attribute(IN inAttributeId INT)
BEGIN
DECLARE attributeRowsCount INT;
SELECT count(*)
FROM attribute_value
WHERE attribute_id = inAttributeId
INTO attributeRowsCount;
IF attributeRowsCount = 0 THEN
DELETE FROM attribute WHERE attribute_id = inAttributeId;
SELECT 1;
ELSE
SELECT -1;
END IF;
END$$
-- Create catalog_get_attribute_details stored procedure
CREATE PROCEDURE catalog_get_attribute_details(IN inAttributeId INT)
BEGIN
SELECT attribute_id, name
FROM attribute
WHERE attribute_id = inAttributeId;
END$$
-- Create catalog_get_attribute_values stored procedure
CREATE PROCEDURE catalog_get_attribute_values(IN inAttributeId INT)
BEGIN
SELECT attribute_value_id, value
FROM attribute_value
WHERE attribute_id = inAttributeId
ORDER BY attribute_id;
END$$
```

```
-- Create catalog_add_attribute_value stored procedure
CREATE PROCEDURE catalog_add_attribute_value(
IN inAttributeId INT, IN inValue VARCHAR(100))
BEGIN
INSERT INTO attribute_value (attribute_id, value)
VALUES (inAttributeId, inValue);
END$$
-- Create catalog_update_attribute_value stored procedure
CREATE PROCEDURE catalog_update_attribute_value(
IN inAttributeValueId INT, IN inValue VARCHAR(100))
BEGIN
UPDATE attribute_value
SET value = inValue
WHERE attribute_value_id = inAttributeValueId;
END$$
-- Create catalog_delete_attribute_value stored procedure
CREATE PROCEDURE catalog_delete_attribute_value(IN inAttributeValueId INT)
BEGIN
DECLARE productAttributeRowsCount INT;
SELECT count(*)
FROM product p
INNER JOIN product_attribute pa
ON p.product_id = pa.product_id
WHERE pa.attribute_value_id = inAttributeValueId
INTO productAttributeRowsCount;
IF productAttributeRowsCount = 0 THEN
DELETE FROM attribute_value WHERE attribute_value_id = inAttributeValueId;
SELECT 1;
ELSE
SELECT -1;
END IF;
```

END$$

11. Załaduj stronę administracyjną TShirtShop i upewnij się, że funkcje administracyjne atrybutów (pokazane na powyższych rysunkach) działają poprawnie.

Jak to działa: administracja atrybutami

Trzeba przyznać, że pisanie kodu do administrowania katalogiem może stać się trochę nudne, ponieważ nie jest to szczególnie trudne, ale wiąże się z napisaniem dużej ilości kodu. Niemniej jednak funkcjonalny panel kontrolny jest niezbędny dla dobrego samopoczucia takiej strony internetowej (ponieważ czytasz tę książkę, zawsze możesz użyć jej kodu do pobrania, jeśli chcesz). Oto ważne zadania, przez które przeszedłeś w tym ćwiczeniu:

• Utworzenie skomponowanego szablonu admin_attributes, który składa się z admin_attributes.tpl i admin_attributes.php. To jest szablon, który implementuje stronę pokazaną na rysunku 1.

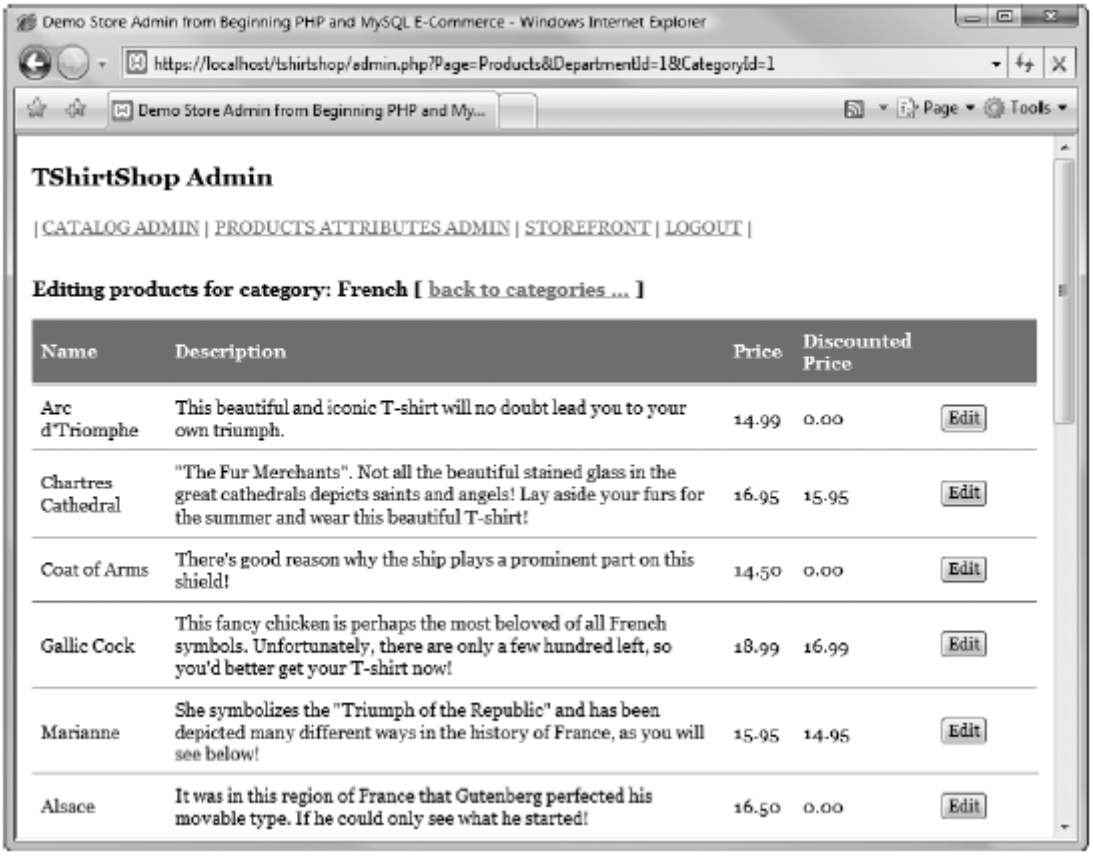
• Utworzenie skomponowanego szablonu admin_attribute_values, który składa się z admin_attribute_values.tpl i admin_attribute_values.php. Jest to szablon, który implementuje funkcję zarządzania wartościami atrybutów przedstawioną na rysunku 2.

• Kodowanie niezbędnych czynności porządkowych, takich jak dodawanie linków do nowej funkcji w szablonie admin_menu (zawierającego linki do wszystkich części administracyjnych witryny) oraz nowe funkcje budowania linków w klasie Link.

• Napisanie kodu warstwy biznesowej i procedur składowanych w bazie danych, które obsługują dodawanie, usuwanie i edytowanie atrybutów oraz wartości atrybutów.

Administrowanie produktami

Jeśli istnieje coś takiego jak nienudna funkcja administracyjna, zarządzanie produktami musi nią być. Produkty mają nie tylko zdjęcia (a wszyscy wiemy, że zdjęcia są mniej nudne niż tekst), ale także dodatkowe szczegóły, którymi można zarządzać, takie jak kategorie, których są częścią i czy są objęte promocją. Ponieważ zarządzanie produktami jest nieco bardziej złożone niż zarządzanie działami lub kategoriami, będziemy wspierać tę funkcję na dwóch stronach administracyjnych. Pierwsza nazywa się Administrowanie Produktami i jest to strona, która pokazuje listę produktów należących do kategorii. Ta strona pojawia się po kliknięciu przycisku Edytuj produkty na liście kategorii i możesz ją podziwiać na rysunku 3. Po uruchomieniu tej strony utworzymy stronę administratora szczegółów produktu, do której można uzyskać dostęp za pomocą przycisków Edytuj można zobaczyć na rysunku 3. Ale o tym później - na razie skupmy się na stworzeniu strony zarządzania produktami.



Ćwiczenie: Administrowanie produktami

1. Utwórz nowy plik szablonu o nazwie admin_products.tpl w pliku folderu prezentacji/templates i dodaj do niego następujący kod:

```
{* admin_products.tpl *}

{load_presentation_object filename="admin_products" assign="obj"}

<form method="post"

action="{$obj->mLinkToCategoryProductsAdmin}">

<h3>

Editing products for category: {$obj->mCategoryName} [

<a href="{$obj->mLinkToDepartmentCategoriesAdmin}">

back to categories ...</a> ]

</h3>

{if $obj->mErrorMessage}<p class="error">{$obj->mErrorMessage}</p>{/if}

{if $obj->mProductsCount eq 0}

<p class="no-items-found">There are no products in this category!</p>

{else}
```

```
<table class="tss-table">
<tr>
<th>Name</th>
<th>Description</th>
<th>Price</th>
<th>Discounted Price</th>
<th width="80"> </th>
</tr>
{section name=i loop=$obj->mProducts}
<tr>
<td>{$obj->mProducts[i].name}</td>
<td>{$obj->mProducts[i].description}</td>
<td>{$obj->mProducts[i].price}</td>
<td>{$obj->mProducts[i].discounted_price}</td>
<td>
<input type="submit"
name="submit_edit_prod_{$obj->mProducts[i].product_id}"
value="Edit" />
</td>
</tr>
{/section}
</table>
{/if}
<h3>Add new product:</h3>
<input type="text" name="product_name" value="[name]" size="30" />
<input type="text" name="product_description" value="[description]"
size="60" />
<input type="text" name="product_price" value="[price]" size="10" />
<input type="submit" name="submit_add_prod_0" value="Add" />
</form>
```

2. Utwórz nowy plik obiektu prezentacji o nazwie admin_products.php w folderze prezentacji i dodaj do niego:

```
<?php
// Class that deals with products administration from a specific category
class AdminProducts
{
// Public variables available in smarty template
public $mProductsCount;
public $mProducts;
public $mErrorMessage;
public $mDepartmentId;
public $mCategoryId;
public $mCategoryName;
public $mLinkToDepartmentCategoriesAdmin;
public $mLinkToCategoryProductsAdmin;
// Private attributes
private $_mAction;
private $_mActionedProductId;
// Class constructor
public function __construct()
{
if (isset ($_GET['DepartmentId']))
$this->mDepartmentId = (int)$_GET['DepartmentId'];
else
trigger_error('DepartmentId not set');
if (isset ($_GET['CategoryId']))
$this->mCategoryId = (int)$_GET['CategoryId'];
else
trigger_error('CategoryId not set');
$category_details = Catalog::GetCategoryDetails($this->mCategoryId);
$this->mCategoryName = $category_details['name'];
```

```
foreach ($_POST as $key => $value)
// If a submit button was clicked ...
if (substr($key, 0, 6) == 'submit')
{
/* Get the position of the last '_' underscore from submit button name
e.g strtpos('submit_edit_prod_1', '_') is 17 */
$last_underscore = strrpos($key, '_');
/* Get the scope of submit button
(e.g 'edit_dep' from 'submit_edit_prod_1') */
$this->_mAction = substr($key, strlen('submit_'),
$last_underscore - strlen('submit_'));
/* Get the product id targeted by submit button
(the number at the end of submit button name)
e.g '1' from 'submit_edit_prod_1' */
$this->_mActionedProductId = (int)substr($key, $last_underscore + 1);
break;
}
$this->mLinkToDepartmentCategoriesAdmin =
Link::ToDepartmentCategoriesAdmin($this->mDepartmentId);
$this->mLinkToCategoryProductsAdmin =
Link::ToCategoryProductsAdmin($this->mDepartmentId, $this->mCategoryId);
}
public function init()
{
// If adding a new product ...
if ($this->_mAction == 'add_prod')
{
$product_name = $_POST['product_name'];
$product_description = $_POST['product_description'];
$product_price = $_POST['product_price'];
if ($product_name == null)
```

```
$this->mErrorMessage = 'Product name is empty';
if ($product_description == null)
$this->mErrorMessage = 'Product description is empty';
if ($product_price == null || !is_numeric($product_price))
$this->mErrorMessage = 'Product price must be a number!';
if ($this->mErrorMessage == null)
{
Catalog::AddProductToCategory($this->mCategoryId, $product_name,
$product_description, $product_price);
header('Location: ' .
htmlspecialchars_decode(
$this->mLinkToCategoryProductsAdmin));
}
}
// If we want to see a product details
if ($this->_mAction == 'edit_prod')
{
header('Location: ' .
htmlspecialchars_decode(
Link::ToProductAdmin($this->mDepartmentId,
$this->mCategoryId,
$this->_mActionedProductId)));
exit();
}
$this->mProducts = Catalog::GetCategoryProducts($this->mCategoryId);
$this->mProductsCount = count($this->mProducts);
}
}
?>
```

3. Otwórz plik Presentation/link.php i dodaj następującą metodę na końcu klasy Link:

```
// Create link to a products administration page
```

```
public static function ToCategoryProductsAdmin($departmentId, $categoryId)
{
$link = 'Page=Products&DepartmentId=' . $departmentId .
'&CategoryId=' . $categoryId;
return self::ToAdmin($link);
}
```

4. Zmodyfikuj metodę init() klasy StoreAdmin w store_admin.php, aby załadować skomponentowany szablon admin_products:

```
// Choose what admin page to load ...
if ($admin_page == 'Departments')
$this->mContentsCell = 'admin_departments.tpl';
elseif ($admin_page == 'Categories')
$this->mContentsCell = 'admin_categories.tpl';
elseif ($admin_page == 'Attributes')
$this->mContentsCell = 'admin_attributes.tpl';
elseif ($admin_page == 'AttributeValues')
$this->mContentsCell = 'admin_attribute_values.tpl';
elseif ($admin_page == 'Products')
$this->mContentsCell = 'admin_products.tpl';
```

5. Dodaj następujący kod warstwy biznesowej do klasy Catalog w business/catalog.php:

```
// Gets products in a category
public static function GetCategoryProducts($categoryId)
{
// Build the SQL query
$sql = 'CALL catalog_get_category_products(:category_id)';
// Build the parameters array
$params = array (':category_id' => $categoryId);
// Execute the query and return the results
return DatabaseHandler::GetAll($sql, $params);
}
// Creates a product and assigns it to a category
```

```
public static function AddProductToCategory($categoryId, $productName,
$productDescription, $productPrice)
{
// Build the SQL query
$sql = 'CALL catalog_add_product_to_category(:category_id, :product_name,
:product_description, :product_price)';
// Build the parameters array
$params = array (':category_id' => $categoryId,
':product_name' => $productName,
':product_description' => $productDescription,
':product_price' => $productPrice);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
```

6. Użyj phpMyAdmin, aby wykonać i utworzyć następujące procedury składowane. Nie zapomnij ustawić $$ jako ogranicznika przed wykonaniem kodu.

```
-- Create catalog_get_category_products stored procedure
CREATE PROCEDURE catalog_get_category_products(IN inCategoryId INT)
BEGIN
SELECT p.product_id, p.name, p.description, p.price,
p.discounted_price
FROM product p
INNER JOIN product_category pc
ON p.product_id = pc.product_id
WHERE pc.category_id = inCategoryId
ORDER BY p.product_id;
END$$
-- Create catalog_add_product_to_category stored procedure
CREATE PROCEDURE catalog_add_product_to_category(IN inCategoryId INT,
IN inName VARCHAR(100), IN inDescription VARCHAR(1000),
IN inPrice DECIMAL(10, 2))
```

```
BEGIN

DECLARE productLastInsertId INT;

INSERT INTO product (name, description, price)

VALUES (inName, inDescription, inPrice);

SELECT LAST_INSERT_ID() INTO productLastInsertId;

INSERT INTO product_category (product_id, category_id)

VALUES (productLastInsertId, inCategoryId);

END$$
```

7. Możesz teraz załadować swoją stronę administracyjną, przejść do kategorii i kliknąć przycisk Edytuj produkty. Powinieneś otrzymać stronę podobną do tej pokazanej na rysunku 3.

Jak to działa: administracja produktem

Więc teraz możesz zobaczyć produkty należące do kategorii. Nie możesz jeszcze edytować szczegółów produktu, ale posiadanie ich na liście jest ważnym krokiem do osiągnięcia tego celu. Możesz również dodać nowy produkt do wybranej kategorii, korzystając z pól tekstowych znajdujących się pod listą produktów. Zaimplementowałeś te funkcje w typowy sposób:

• Zaimplementowałeś podstawową funkcjonalność za pomocą składowanego szablonu Smarty o nazwie admin_products. Jest on utworzony z pliku szablonu admin_products.tpl i skojarzonego z nim obiektu prezentacji w admin_products.php.

• Zintegrowałeś nową funkcję z witryną administracyjną, łącząc ją z store_admin.

• Napisano pomocnicze metody warstwy biznesowej i procedury składowane warstwy danych, które umożliwiają dodawanie nowego produktu do bazy danych i odczytywanie istniejących produktów.

Administrowanie szczegółami produktu

Lista produktów, którą zbudowałeś wcześniej, jest wspaniała, ale brakuje jej kilku ważnych funkcji. Ostateczny szablon składowy, który wdrażasz, admin_product_details, pojawia się, gdy użytkownik kliknie przycisk Edytuj dla produktu na liście produktów. Szablon umożliwia:

• Zobacz zdjęcie produktu

• Usuń produkt z kategorii

• Całkowicie usuń produkt z bazy danych

• Przypisz aktualny produkt do dodatkowej kategorii

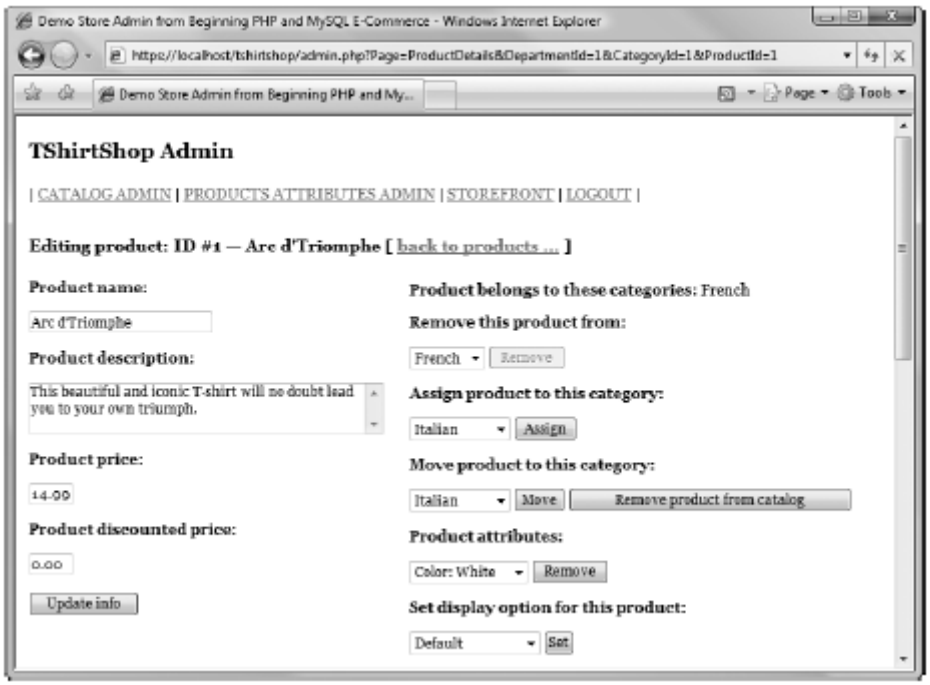
• Przenieś obecny produkt do innej kategorii

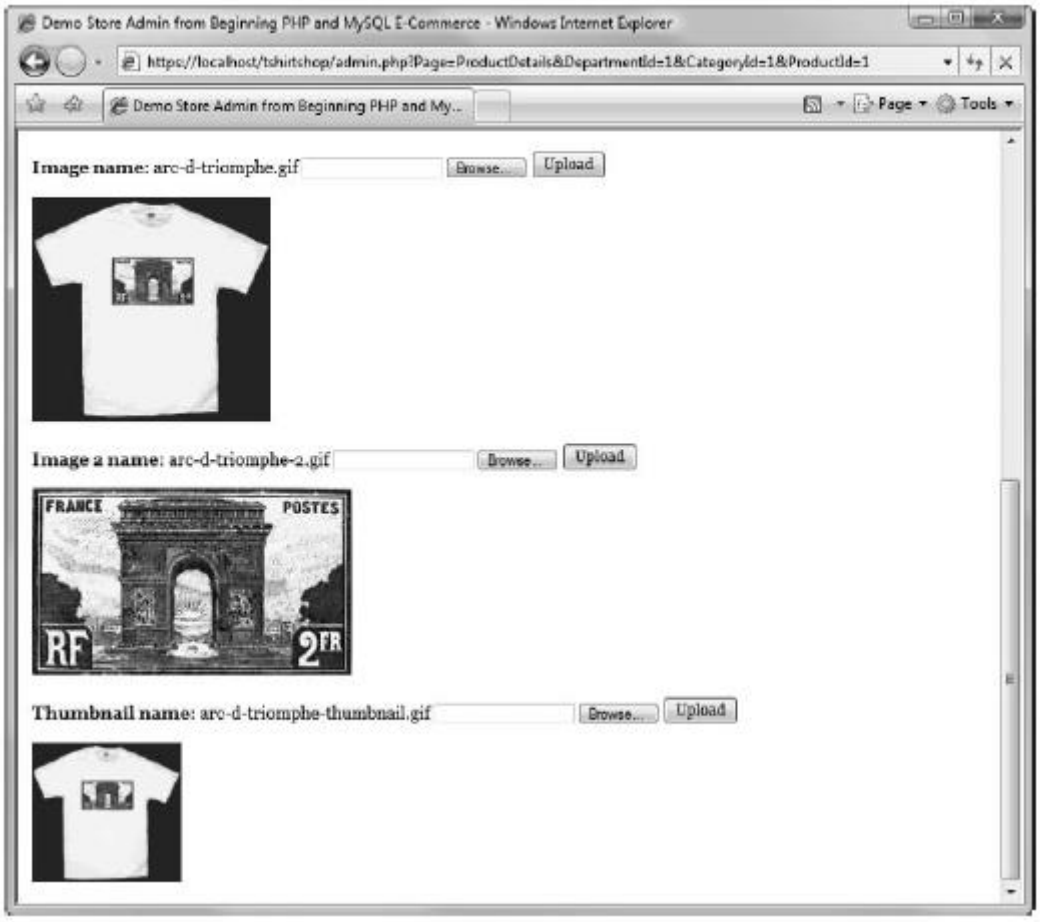
• Zmień cenę produktu i obniżoną cenę

• Zmień nazwę lub opis produktu

Jeśli chodzi o usuwanie produktu, sprawy nie są takie proste. Możesz anulować przypisanie produktu do kategorii, usuwając rekord z tabeli product_category lub skutecznie usunąć produkt z tabeli produktów. Ponieważ produkty są dostępne w katalogu poprzez wybranie kategorii, musisz upewnić

się, że nie ma produktów osieroconych (produktów, które nie należą do żadnej kategorii). Pojawią się dwa przyciski usuwania: przycisk „Usuń z kategorii”, który umożliwia usunięcie produktu z jednej kategorii oraz przycisk „Usuń z katalogu”, który całkowicie usuwa produkt z katalogu poprzez usunięcie jego wpisów w produkcie oraz tabele product_category. Jeśli produkt należy do wielu kategorii, aktywny będzie tylko przycisk „Usuń z kategorii”. Jeśli produkt należy do jednej kategorii, dostępny będzie tylko przycisk „Usuń z katalogu”. W ten sposób oferujemy administratorom niezbędne funkcje i unikamy tworzenia produktów osieroconych, które nie należą do żadnej kategorii. Rzuć okiem na stronę administrowania szczegółami produktu na rysunkach 4 i , a następnie zaimplementuj ją, wykonując (kolejne ekscytujące) ćwiczenie krok po kroku.





Szczegóły produktu: Implementacja poziomu prezentacji

Należy pamiętać, że ponieważ do napisania jest sporo kodu, implementujemy (i wyjaśniamy) warstwę prezentacji, warstwę biznesową i warstwę danych dla funkcji administrowania szczegółami produktu w osobnych ćwiczeniach krok po kroku. Podobnie jak w przypadku wszystkich funkcji administracyjnych, wdrożenie rozpoczynamy od warstwy prezentacji.

Ćwiczenie: Implementacja warstwy prezentacji szczegółów produktu

1. Otwórz tshirtshop.css z folderu stylów i dodaj następujące definicje stylów:

```
.borderless-table {
width: 100%;
}
.borderless-table td {
border: none;
padding-left: 0;
}
.bold-text {
font-size: 93%;
```

```
font-weight: bold;
}
```

2. Stwórz prezentację/templates/admin_product_details.tpl i dodaj w niej następujący kod:

```
{* admin_product_details.tpl *}
{load_presentation_object filename="admin_product_details" assign="obj"}
<form enctype="multipart/form-data" method="post"
action="{$obj->mLinkToProductDetailsAdmin}">
<h3>
Editing product: ID #{$obj->mProduct.product_id} &mdash;
{$obj->mProduct.name} [
<a href="{$obj->mLinkToCategoryProductsAdmin}">
back to products ...</a> ]
</h3>
{if $obj->mErrorMessage}<p class="error">{$obj->mErrorMessage}</p>{/if}
<table class="borderless-table">
<tbody>
<tr>
<td valign="top">
<p class="bold-text">
Product name:
</p>
<p>
<input type="text" name="name"
value="{$obj->mProduct.name}" size="30" />
</p>
<p class="bold-text">
Product description:
</p>
<p>
{strip}
<textarea name="description" rows="3" cols="60">
```

{$obj->mProduct.description}

</textarea>

{/strip}

Product price:

<input type="text" name="price"

value="{$obj->mProduct.price}" size="5" />

Product discounted price:

<input type="text" name="discounted_price"

value="{$obj->mProduct.discounted_price}" size="5" />

<input type="submit" name="UpdateProductInfo"

value="Update info" />

</td>

<td valign="top">

Product belongs to these categories:

{$obj->mProductCategoriesString}

Remove this product from:

```
<p>
{html_options name="TargetCategoryIdRemove"
options=$obj->mRemoveFromCategories}
<input type="submit" name="RemoveFromCategory" value="Remove"
{if $obj->mRemoveFromCategoryButtonDisabled}
disabled="disabled" {/if}/>
</p>
<p class="bold-text">
Assign product to this category:
</p>
<p>
{html_options name="TargetCategoryIdAssign"
options=$obj->mAssignOrMoveTo}
<input type="submit" name="Assign" value="Assign" />
</p>
<p class="bold-text">
Move product to this category:
</p>
<p>
{html_options name="TargetCategoryIdMove"
options=$obj->mAssignOrMoveTo}
<input type="submit" name="Move" value="Move" />
<input type="submit" name="RemoveFromCatalog"
value="Remove product from catalog"
{if !$obj->mRemoveFromCategoryButtonDisabled}
disabled="disabled" {/if}/>
</p>
{if $obj->mProductAttributes}
<p class="bold-text">
Product attributes:
</p>
```

```
<p>
{html_options name="TargetAttributeValueIdRemove"
options=$obj->mProductAttributes}
<input type="submit" name="RemoveAttributeValue"
value="Remove" />
</p>
{/if}
{if $obj->mCatalogAttributes}
<p class="bold-text">
Assign attribute to product:
</p>
<p>
{html_options name="TargetAttributeValueIdAssign"
options=$obj->mCatalogAttributes}
<input type="submit" name="AssignAttributeValue"
value="Assign" />
</p>
{/if}
<p class="bold-text">
Set display option for this product:
</p>
<p>
{html_options name="ProductDisplay"
options=$obj->mProductDisplayOptions
selected=$obj->mProduct.display}
<input type="submit" name="SetProductDisplayOption" value="Set" />
</p>
</td>
</tr>
</tbody>
</table>
```

```
<p>
<font class="bold-text">Image name:</font> {$obj->mProduct.image}
<input name="ImageUpload" type="file" value="Upload" />
<input type="submit" name="Upload" value="Upload" />
</p>
{if $obj->mProduct.image}
<p>
<img src="product_images/{$obj->mProduct.image}"
border="0" alt="{$obj->mProduct.name} image" />
</p>
{/if}
<p>
<font class="bold-text">Image 2 name:</font> {$obj->mProduct.image_2}
<input name="Image2Upload" type="file" value="Upload" />
<input type="submit" name="Upload" value="Upload" />
</p>
{if $obj->mProduct.image_2}
<p>
<img src="product_images/{$obj->mProduct.image_2}"
border="0" alt="{$obj->mProduct.name} image 2" />
</p>
{/if}
<p>
<font class="bold-text">Thumbnail name:</font> {$obj->mProduct.thumbnail}
<input name="ThumbnailUpload" type="file" value="Upload" />
<input type="submit" name="Upload" value="Upload" />
</p>
{if $obj->mProduct.thumbnail}
<p>
<img src="product_images/{$obj->mProduct.thumbnail}"
border="0" alt="{$obj->mProduct.name} thumbnail" />
```

</p>

{/if}

</form>

3. Otwórz business/catalog.php, aby dodać członka $mProductDisplayOptions do klasy Catalog potrzebnej dla admin_products, jak zaznaczono:

<?php

// Business tier class for reading product catalog information

class Catalog

{

// Defines product display options

public static $mProductDisplayOptions = array ('Default', // 0

'On Catalog', // 1

'On Department', // 2

'On Both'); // 3

// Retrieves all departments

public static function GetDepartments()

{

4. Utwórz obiekt prezentacji Presentation/admin_product_details.php i dodaj w nim:

<?php

// Class that deals with product administration

class AdminProductDetails

{

// Public attributes

public $mProduct;

public $mErrorMessage;

public $mProductCategoriesString;

public $mProductDisplayOptions;

public $mProductAttributes;

public $mCatalogAttributes;

public $mAssignOrMoveTo;

public $mRemoveFromCategories;

```
public $mRemoveFromCategoryButtonDisabled = false;
public $mLinkToCategoryProductsAdmin;
public $mLinkToProductDetailsAdmin;
// Private attributes
private $_mProductId;
private $_mCategoryId;
private $_mDepartmentId;
// Class constructor
public function __construct()
{
// Need to have DepartmentId in the query string
if (!isset ($_GET['DepartmentId']))
trigger_error('DepartmentId not set');
else
$this->_mDepartmentId = (int)$_GET['DepartmentId'];
// Need to have CategoryId in the query string
if (!isset ($_GET['CategoryId']))
trigger_error('CategoryId not set');
else
$this->_mCategoryId = (int)$_GET['CategoryId'];
// Need to have ProductId in the query string
if (!isset ($_GET['ProductId']))
trigger_error('ProductId not set');
else
$this->_mProductId = (int)$_GET['ProductId'];
$this->mProductDisplayOptions = Catalog::$mProductDisplayOptions;
$this->mLinkToCategoryProductsAdmin =
Link::ToCategoryProductsAdmin($this->_mDepartmentId, $this->_mCategoryId);
$this->mLinkToProductDetailsAdmin =
Link::ToProductAdmin($this->_mDepartmentId,
$this->_mCategoryId,
```

```
$this->_mProductId);
}
public function init()
{
// If uploading a product picture ...
if (isset ($_POST['Upload']))
{
/* Check whether we have write permission on the
product_images folder */
if (!is_writeable(SITE_ROOT . '/product_images/'))
{
echo "Can't write to the product_images folder";
exit();
}
// If the error code is 0, the file was uploaded ok
if ($_FILES['ImageUpload']['error'] == 0)
{
/* Use the move_uploaded_file PHP function to move the file
from its temporary location to the product_images folder */
move_uploaded_file($_FILES['ImageUpload']['tmp_name'],
SITE_ROOT . '/product_images/' .
$_FILES['ImageUpload']['name']);
// Update the product's information in the database
Catalog::SetImage($this->_mProductId,
$_FILES['ImageUpload']['name']);
}
// If the error code is 0, the file was uploaded ok
if ($_FILES['Image2Upload']['error'] == 0)
{
/* Use the move_uploaded_file PHP function to move the file
from its temporary location to the product_images folder */
```

```
move_uploaded_file($_FILES['Image2Upload']['tmp_name'],
SITE_ROOT . '/product_images/' .
$_FILES['Image2Upload']['name']);
// Update the product's information in the database
Catalog::SetImage2($this->_mProductId,
$_FILES['Image2Upload']['name']);
}
// If the error code is 0, the file was uploaded ok
if ($_FILES['ThumbnailUpload']['error'] == 0)
{
// Move the uploaded file to the product_images folder
move_uploaded_file($_FILES['ThumbnailUpload']['tmp_name'],
SITE_ROOT . '/product_images/' .
$_FILES['ThumbnailUpload']['name']);
// Update the product's information in the database
Catalog::SetThumbnail($this->_mProductId,
$_FILES['ThumbnailUpload']['name']);
}
}
// If updating product info ...
if (isset ($_POST['UpdateProductInfo']))
{
$product_name = $_POST['name'];
$product_description = $_POST['description'];
$product_price = $_POST['price'];
$product_discounted_price = $_POST['discounted_price'];
if ($product_name == null)
$this->mErrorMessage = 'Product name is empty';
if ($product_description == null)
$this->mErrorMessage = 'Product description is empty';
if ($product_price == null || !is_numeric($product_price))
```

```
$this->mErrorMessage = 'Product price must be a number!';
if ($product_discounted_price == null ||
!is_numeric($product_discounted_price))
$this->mErrorMessage = 'Product discounted price must be a number!';
if ($this->mErrorMessage == null)
Catalog::UpdateProduct($this->_mProductId, $product_name,
$product_description, $product_price, $product_discounted_price);
}
// If removing the product from a category ...
if (isset ($_POST['RemoveFromCategory']))
{
$target_category_id = $_POST['TargetCategoryIdRemove'];
$still_exists = Catalog::RemoveProductFromCategory(
$this->_mProductId, $target_category_id);
if ($still_exists == 0)
{
header('Location: ' .
htmlspecialchars_decode(
$this->mLinkToCategoryProductsAdmin));
exit();
}
}
// If setting product display option ...
if (isset ($_POST['SetProductDisplayOption']))
{
$product_display = $_POST['ProductDisplay'];
Catalog::SetProductDisplayOption($this->_mProductId, $product_display);
}
// If removing the product from catalog ...
if (isset ($_POST['RemoveFromCatalog']))
{
```

```
Catalog::DeleteProduct($this->_mProductId);
header('Location: ' .
htmlspecialchars_decode(
$this->mLinkToCategoryProductsAdmin));
exit();
}
// If assigning the product to another category ...
if (isset ($_POST['Assign']))
{
$target_category_id = $_POST['TargetCategoryIdAssign'];
Catalog::AssignProductToCategory($this->_mProductId,
$target_category_id);
}
// If moving the product to another category ...
if (isset ($_POST['Move']))
{
$target_category_id = $_POST['TargetCategoryIdMove'];
Catalog::MoveProductToCategory($this->_mProductId,
$this->_mCategoryId, $target_category_id);
header('Location: ' .
htmlspecialchars_decode(
Link::ToProductAdmin($this->_mDepartmentId,
$target_category_id,
$this->_mProductId)));
exit();
}
// If assigning an attribute value to the product ...
if (isset ($_POST['AssignAttributeValue']))
{
$target_attribute_value_id = $_POST['TargetAttributeValueIdAssign'];
Catalog::AssignAttributeValueToProduct($this->_mProductId,
```

```
$target_attribute_value_id);
}
// If removing an attribute value from the product ...
if (isset ($_POST['RemoveAttributeValue']))
{
$target_attribute_value_id = $_POST['TargetAttributeValueIdRemove'];
Catalog::RemoveProductAttributeValue($this->_mProductId,
$target_attribute_value_id);
}
// If moving the product to another category ...
if (isset ($_POST['Move']))
{
$target_category_id = $_POST['TargetCategoryIdMove'];
Catalog::MoveProductToCategory($this->_mProductId,
$this->_mCategoryId, $target_category_id);
header('Location: ' .
htmlspecialchars_decode(
Link::ToProductAdmin($this->_mDepartmentId,
$target_category_id,
$this->_mProductId)));
exit();
}
// Get product info
$this->mProduct = Catalog::GetProductInfo($this->_mProductId);
$product_categories = Catalog::GetCategoriesForProduct($this->_mProductId);
$product_attributes =
Catalog::GetProductAttributes($this->_mProductId);
for ($i = 0; $i < count($product_attributes); $i++)
$this->mProductAttributes[$product_attributes[$i]['attribute_value_id']] =
$product_attributes[$i]['attribute_name'] . ': ' .
$product_attributes[$i]['attribute_value'];
```

```
$catalog_attributes =
Catalog::GetAttributesNotAssignedToProduct($this->_mProductId);
for ($i = 0; $i < count($catalog_attributes); $i++)
$this->mCatalogAttributes[$catalog_attributes[$i]['attribute_value_id']] =
$catalog_attributes[$i]['attribute_name'] . ': ' .
$catalog_attributes[$i]['attribute_value'];
if (count($product_categories) == 1)
$this->mRemoveFromCategoryButtonDisabled = true;
// Show the categories the product belongs to
for ($i = 0; $i < count($product_categories); $i++)
$temp1[$product_categories[$i]['category_id']] =
$product_categories[$i]['name'];
$this->mRemoveFromCategories = $temp1;
$this->mProductCategoriesString = implode(', ', $temp1);
$all_categories = Catalog::GetCategories();
for ($i = 0; $i < count($all_categories); $i++)
$temp2[$all_categories[$i]['category_id']] =
$all_categories[$i]['name'];
$this->mAssignOrMoveTo = array_diff($temp2, $temp1);
}
}
?>
```

5. Otwórz plik Presentation/link.php i dodaj następującą metodę na końcu klasy Link:

```
// Create link to product details administration page
public static function ToProductAdmin($departmentId, $categoryId, $productId)
{
$link = 'Page=ProductDetails&DepartmentId=' . $departmentId .
'&CategoryId=' . $categoryId . '&ProductId=' . $productId;
return self::ToAdmin($link);
}
```

6. Zmodyfikuj metodę init() w klasie StoreAdmin w store_admin.php, aby załadować skomponowany szablon admin_product_details:

```
// Choose what admin page to load ...

if ($admin_page == 'Departments')

$this->mContentsCell = 'admin_departments.tpl';

elseif ($admin_page == 'Categories')

$this->mContentsCell = 'admin_categories.tpl';

elseif ($admin_page == 'Attributes')

$this->mContentsCell = 'admin_attributes.tpl';

elseif ($admin_page == 'AttributeValues')

$this->mContentsCell = 'admin_attribute_values.tpl';

elseif ($admin_page == 'Products')

$this->mContentsCell = 'admin_products.tpl';

elseif ($admin_page == 'ProductDetails')

$this->mContentsCell = 'admin_product_details.tpl';
```

Jak to działa: tworzenie admin_product

Mimo że nie możesz jeszcze uruchomić strony, warto przyjrzeć się nowym elementom zawartym w nowym szablonie. Szablon admin_product_details.tpl zawiera pojedynczy formularz z atrybutem enctype="multipart/form-data". Ten atrybut jest potrzebny do przesyłania zdjęć produktów i działa w połączeniu z kodem HTML, który umożliwia przesyłanie plików:

```
...

<input name="ImageUpload" type="file" value="Upload" />

<input type="submit" name="Upload" value="Upload" />

...
```

Reakcja na kliknięcie jednego z trzech przycisków uploadu odpowiadających obrazom produktów jest zaimplementowana w metodzie init() z klasy AdminProductDetails (w Presentation/admin_product_details.php):

```
// If uploading a product picture ...

if (isset ($_POST['Upload']))

{

/* Check whether we have write permission on the

product_images folder */

if (!is_writeable(SITE_ROOT . '/product_images/'))
```

```
{
echo "Can't write to the product_images folder";
exit();
}
// If the error code is 0, the file was uploaded ok
if ($_FILES['ImageUpload']['error'] == 0)
{
/* Use the move_uploaded_file PHP function to move the file
from its temporary location to the product_images folder */
move_uploaded_file($_FILES['ImageUpload']['tmp_name'],
SITE_ROOT . '/product_images/' .
$_FILES['ImageUpload']['name']);
// Update the product's information in the database
Catalog::SetImage($this->_mProductId,
$_FILES['ImageUpload']['name']);
}
// If the error code is 0, the file was uploaded ok
if ($_FILES['Image2Upload']['error'] == 0)
{
/* Use the move_uploaded_file PHP function to move the file
from its temporary location to the product_images folder */
move_uploaded_file($_FILES['Image2Upload']['tmp_name'],
SITE_ROOT . '/product_images/' .
$_FILES['Image2Upload']['name']);
// Update the product's information in the database
Catalog::SetImage2($this->_mProductId,
$_FILES['Image2Upload']['name']);
}
// If the error code is 0, the file was uploaded ok
if ($_FILES['ThumbnailUpload']['error'] == 0)
{
```

```
// Move the uploaded file to the product_images folder
move_uploaded_file($_FILES['ThumbnailUpload']['tmp_name'],
SITE_ROOT . '/product_images/' .
$_FILES['ThumbnailUpload']['name']);
// Update the product's information in the database
Catalog::SetThumbnail($this->_mProductId,
$_FILES['ThumbnailUpload']['name']);
}
}
```

Zmienna superglobalna $_FILES to dwuwymiarowa tablica, która przechowuje informacje o przesłanym pliku (lub plikach). Jeśli zmienna $_FILES['ImageUpload']['error'] jest ustawiona na 0, główny obraz produktu został pomyślnie przesłany i musi zostać obsłużony. Zmienna $_FILES['ImageUpload']['tmp_name'] przechowuje tymczasową nazwę przesłanego pliku na serwerze, a zmienna $_FILES['ImageUpload']['name'] przechowuje określoną nazwę pliku po przesłaniu na serwer.

Funkcja PHP move_uploaded_file() służy do przeniesienia pliku z lokalizacji tymczasowej do folderu product_images:

```
/* Use the move_uploaded_file PHP function to move the file
from its temporary location to the product_images folder */
move_uploaded_file($_FILES['ImageUpload']['tmp_name'],
SITE_ROOT . '/product_images/' .
$_FILES['ImageUpload']['name']);
```

Po wgraniu zdjęcia produktu nazwa pliku musi zostać zapisana w bazie danych (w przeciwnym razie wgranie pliku nie będzie miało żadnego skutku):

```
// Update the product's information in the database
Catalog::SetImage($this->_mProductId,
$_FILES['ImageUpload']['name']);
```

Jak widać, obsługa przesyłania plików za pomocą PHP jest całkiem prosta.

Szczegóły produktu: wdrażanie poziomu biznesowego

Aby zaimplementować warstwę biznesową, musisz dodać następujące metody do klasy Catalog:

• UpdateProduct aktualizuje szczegóły produktu: nazwę, opis, cenę i obniżoną cenę.

• DeleteProduct całkowicie usuwa produkt z katalogu.

• RemoveProductFromCategory jest wywoływane po kliknięciu przycisku „Usuń z kategorii” w celu usunięcia przypisania produktu do kategorii.

• GetCategories zwraca wszystkie kategorie z naszego katalogu.

• GetProductInfo zwraca szczegóły produktu.

• GetCategoriesForProduct służy do pobrania listy kategorii powiązanych z określonym produktem.

• SetProductDisplayOption ustawia ustawienia wyświetlania produktu.

• AssignProductToCategory przypisuje produkt do kategorii.

• MoveProductToCategory przenosi produkt z jednej kategorii do drugiej.

• GetAttributesNotAssignedToProduct zwraca wszystkie wartości atrybutów z tabeli atrybutów, które nie zostały przypisane do produktu.

• AssignAttributeValueToProduct przypisuje wartość atrybutu do produktu.

• RemoveProductAttributeValue usuwa powiązanie między produktem a wartością atrybutu z tabeli product_attribute.

• SetImage1 zmienia nazwę pliku obrazu w bazie danych dla określonego produktu.

• SetImage2 zmienia nazwę drugiego pliku obrazu w bazie danych dla określonego produktu.

• SetThumbnail zmienia nazwę pliku obrazu miniatury dla określonego produktu.

Ćwiczenie: Implementacja metod poziomu biznesowego

Ponieważ funkcjonalność jest lepiej wyrażana przez funkcje warstwy danych, które wywołują metody, omówimy je bardziej szczegółowo podczas implementacji warstwy danych. Na razie wystarczy dodać następujący kod do klasy Catalog w business/catalog.php:

```
// Updates a product

public static function UpdateProduct($productId, $productName,

$productDescription, $productPrice,

$productDiscountedPrice)

{

// Build the SQL query

$sql = 'CALL catalog_update_product(:product_id, :product_name,

:product_description, :product_price,

:product_discounted_price)';

// Build the parameters array

$params = array (':product_id' => $productId,

':product_name' => $productName,

':product_description' => $productDescription,

':product_price' => $productPrice,
```

```
':product_discounted_price' => $productDiscountedPrice);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
// Removes a product from the product catalog
public static function DeleteProduct($productId)
{
// Build SQL query
$sql = 'CALL catalog_delete_product(:product_id)';
// Build the parameters array
$params = array (':product_id' => $productId);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
// Unassigns a product from a category
public static function RemoveProductFromCategory($productId, $categoryId)
{
// Build SQL query
$sql = 'CALL catalog_remove_product_from_category(
:product_id, :category_id)';
// Build the parameters array
$params = array (':product_id' => $productId,
':category_id' => $categoryId);
// Execute the query and return the results
return DatabaseHandler::GetOne($sql, $params);
}
// Retrieves the list of categories a product belongs to
public static function GetCategories()
{
// Build SQL query
$sql = 'CALL catalog_get_categories()';
```

```
// Execute the query and return the results
return DatabaseHandler::GetAll($sql);
}
// Retrieves product info
public static function GetProductInfo($productId)
{
// Build SQL query
$sql = 'CALL catalog_get_product_info(:product_id)';
// Build the parameters array
$params = array (':product_id' => $productId);
// Execute the query and return the results
return DatabaseHandler::GetRow($sql, $params);
}
// Retrieves the list of categories a product belongs to
public static function GetCategoriesForProduct($productId)
{
// Build SQL query
$sql = 'CALL catalog_get_categories_for_product(:product_id)';
// Build the parameters array
$params = array (':product_id' => $productId);
// Execute the query and return the results
return DatabaseHandler::GetAll($sql, $params);
}
// Assigns a product to a category
public static function SetProductDisplayOption($productId, $display)
{
// Build SQL query
$sql = 'CALL catalog_set_product_display_option(
:product_id, :display)';
// Build the parameters array
$params = array (':product_id' => $productId,
```

```
':display' => $display);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
// Assigns a product to a category
public static function AssignProductToCategory($productId, $categoryId)
{
// Build SQL query
$sql = 'CALL catalog_assign_product_to_category(
:product_id, :category_id)';
// Build the parameters array
$params = array (':product_id' => $productId,
':category_id' => $categoryId);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
// Moves a product from one category to another
public static function MoveProductToCategory($productId, $sourceCategoryId,
$targetCategoryId)
{
// Build SQL query
$sql = 'CALL catalog_move_product_to_category(:product_id,
:source_category_id, :target_category_id)';
// Build the parameters array
$params = array (':product_id' => $productId,
':source_category_id' => $sourceCategoryId,
':target_category_id' => $targetCategoryId);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
// Gets the catalog attributes that are not assigned to the specified product
```

```
public static function GetAttributesNotAssignedToProduct($productId)
{
// Build the SQL query
$sql = 'CALL catalog_get_attributes_not_assigned_to_product(:product_id)';
// Build the parameters array
$params = array (':product_id' => $productId);
// Execute the query and return the results
return DatabaseHandler::GetAll($sql, $params);
}
// Assign an attribute value to the specified product
public static function AssignAttributeValueToProduct($productId,
$attributeValueId)
{
// Build SQL query
$sql = 'CALL catalog_assign_attribute_value_to_product(
:product_id, :attribute_value_id)';
// Build the parameters array
$params = array (':product_id' => $productId,
':attribute_value_id' => $attributeValueId);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
// Removes a product attribute value
public static function RemoveProductAttributeValue($productId,
$attributeValueId)
{
// Build SQL query
$sql = 'CALL catalog_remove_product_attribute_value(
:product_id, :attribute_value_id)';
// Build the parameters array
$params = array (':product_id' => $productId,
```

```
':attribute_value_id' => $attributeValueId);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
// Changes the name of the product image file in the database
public static function SetImage($productId, $imageName)
{
// Build SQL query
$sql = 'CALL catalog_set_image(:product_id, :image_name)';
// Build the parameters array
$params = array (':product_id' => $productId, ':image_name' => $imageName);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
// Changes the name of the second product image file in the database
public static function SetImage2($productId, $imageName)
{
// Build SQL query
$sql = 'CALL catalog_set_image_2(:product_id, :image_name)';
// Build the parameters array
$params = array (':product_id' => $productId, ':image_name' => $imageName);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
// Changes the name of the product thumbnail file in the database
public static function SetThumbnail($productId, $thumbnailName)
{
// Build SQL query
$sql = 'CALL catalog_set_thumbnail(:product_id, :thumbnail_name)';
// Build the parameters array
$params = array (':product_id' => $productId,
```

':thumbnail_name' => $thumbnailName);

// Execute the query

DatabaseHandler::Execute($sql, $params);

}

Szczegóły produktu: Implementacja warstwy danych

W warstwie danych dodajesz procedury składowane, które odpowiadają metodom warstwy biznesowej w właśnie wyświetlonej klasie Catalog.

Ćwiczenie: Dodawanie procedur składowanych

1. Użyj phpMyAdmin, aby wykonać i utworzyć procedury składowane opisane w poniższych krokach i nie zapomnij ustawić $$ jako ogranicznika przed wykonaniem kodu.

2. Wykonaj następujący kod, który utworzy procedurę składowaną catalog_update_product w bazie danych sklepu tshirtshop. Procedura składowana catalog_update_product aktualizuje szczegóły produktu przy użyciu danych otrzymanych za pośrednictwem parametrów wejściowych inProductId, inName, inDescription, inPrice i inDiscountedPrice.

-- Create catalog_update_product stored procedure

CREATE PROCEDURE catalog_update_product(IN inProductId INT,

IN inName VARCHAR(100), IN inDescription VARCHAR(1000),

IN inPrice DECIMAL(10, 2), IN inDiscountedPrice DECIMAL(10, 2))

BEGIN

UPDATE product

SET name = inName, description = inDescription, price = inPrice,

discounted_price = inDiscountedPrice

WHERE product_id = inProductId;

END$$

3. Wykonaj następujący kod, który utworzy procedurę składowaną catalog_delete_product w bazie danych tshirtshop. Procedura składowana catalog_delete_product całkowicie usuwa produkt z katalogu, usuwając jego wpisy w tabelach product_attribute, product_category i product.

-- Create catalog_delete_product stored procedure

CREATE PROCEDURE catalog_delete_product(IN inProductId INT)

BEGIN

DELETE FROM product_attribute WHERE product_id = inProductId;

DELETE FROM product_category WHERE product_id = inProductId;

DELETE FROM product WHERE product_id = inProductId;

END$$

4. Wykonaj następujący kod, który utworzy procedurę składowaną catalog_remove_product_from_category w bazie danych sklepu tshirtshop. Procedura składowana catalog_remove_product_from_category weryfikuje, w ilu kategoriach istnieje produkt. Jeśli produkt istnieje w więcej niż jednej kategorii, po prostu usuwa produkt z określonej kategorii (identyfikator otrzymany jako parametr). Jeśli produkt jest powiązany z jedną kategorią, jest całkowicie usuwany z bazy danych.

```
-- Create catalog_remove_product_from_category stored procedure

CREATE PROCEDURE catalog_remove_product_from_category(

IN inProductId INT, IN inCategoryId INT)

BEGIN

DECLARE productCategoryRowsCount INT;

SELECT count(*)

FROM product_category

WHERE product_id = inProductId

INTO productCategoryRowsCount;

IF productCategoryRowsCount = 1 THEN

CALL catalog_delete_product(inProductId);

SELECT 0;

ELSE

DELETE FROM product_category

WHERE category_id = inCategoryId AND product_id = inProductId;

SELECT 1;

END IF;

END$$
```

5. Wykonaj następujący kod, który utworzy procedurę składowaną catalog_get_categories w Twojej bazie danych tshirtshop; Catalog_get_categories po prostu zwraca wszystkie kategorie z Twojego katalogu.

```
-- Create catalog_get_categories stored procedure

CREATE PROCEDURE catalog_get_categories()

BEGIN

SELECT category_id, name, description

FROM category

ORDER BY category_id;

END$$
```

6. Wykonaj następujący kod, który utworzy procedurę składowaną catalog_get_product_info w Twojej bazie danych tshirtshop. Procedura przechowywana catalog_get_product_info pobiera nazwę produktu, opis, cenę, obniżoną cenę, obraz, drugi obraz, miniaturę i opcję wyświetlania dla produktu identyfikowanego przez identyfikator produktu (inProductId).

```
-- Create catalog_get_product_info stored procedure

CREATE PROCEDURE catalog_get_product_info(IN inProductId INT)

BEGIN

SELECT product_id, name, description, price, discounted_price,

image, image_2, thumbnail, display

FROM product

WHERE product_id = inProductId;

END$$
```

7. Wykonaj ten kod, który utworzy procedurę przechowywaną catalog_get_categories_for_product w bazie danych tshirtshop. Procedura składowana catalog_get_categories_for_product zwraca listę kategorii, które należą do określonego produktu. Zwracane są tylko ich identyfikatory i imiona i nazwiska, bo to jedyne informacje, które nas interesują.

```
-- Create catalog_get_categories_for_product stored procedure

CREATE PROCEDURE catalog_get_categories_for_product(IN inProductId INT)

BEGIN

SELECT c.category_id, c.department_id, c.name

FROM category c

JOIN product_category pc

ON c.category_id = pc.category_id

WHERE pc.product_id = inProductId

ORDER BY category_id;

END$$
```

8. Wykonaj ten kod, który utworzy procedurę składowaną catalog_set_product_display_option w bazie danych tshirtshop:

```
-- Create catalog_set_product_display_option stored procedure

CREATE PROCEDURE catalog_set_product_display_option(

IN inProductId INT, IN inDisplay SMALLINT)

BEGIN

UPDATE product SET display = inDisplay WHERE product_id = inProductId;

END$$
```

9. Wykonaj następujący kod, który utworzy procedurę składowaną catalog_assign_product_to_category w bazie danych tshirtshop. Procedura składowana catalog_assign_product_to_category kojarzy produkt z kategorią przez dodanie pary wartości (product_id, category_id) do tabeli product_ category.

```
-- Create catalog_assign_product_to_category stored procedure

CREATE PROCEDURE catalog_assign_product_to_category(

IN inProductId INT, IN inCategoryId INT)

BEGIN

INSERT INTO product_category (product_id, category_id)

VALUES (inProductId, inCategoryId);

END$$
```

10. Wykonaj następujący kod, który utworzy procedurę składowaną catalog_assign_product_to_category w Twojej bazie danych tshirtshop. Procedura składowana catalog_move_product_to_category usuwa produkt z kategorii i umieszcza go w innej.

```
-- Create catalog_move_product_to_category stored procedure

CREATE PROCEDURE catalog_move_product_to_category(IN inProductId INT,

IN inSourceCategoryId INT, IN inTargetCategoryId INT)

BEGIN

UPDATE product_category

SET category_id = inTargetCategoryId

WHERE product_id = inProductId

AND category_id = inSourceCategoryId;

END$$
```

11. Wykonaj następujący kod, który utworzy procedurę składowaną catalog_get_attributes_not_assigned_to_product w bazie danych tshirtshop. Ta procedura zwraca wszystkie wartości atrybutów, które nie były jeszcze powiązane z produktem w tabeli product_attribute.

```
-- Create catalog_get_attributes_not_assigned_to_product stored procedure

CREATE PROCEDURE catalog_get_attributes_not_assigned_to_product(

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 NOT IN

(SELECT attribute_value_id

FROM product_attribute

WHERE product_id = inProductId)

ORDER BY attribute_name, av.attribute_value_id;

END$$
```

12. Wykonaj następujący kod, który utworzy procedurę przechowywaną catalog_assign_attribute_value_to_product w Twojej bazie danych tshirtshop. Ta procedura przypisuje wartość atrybutu do produktu poprzez dodanie nowego rekordu do tabeli product_attribute.

```
-- Create catalog_assign_attribute_value_to_product stored procedure

CREATE PROCEDURE catalog_assign_attribute_value_to_product(

IN inProductId INT, IN inAttributeValueId INT)

BEGIN

INSERT INTO product_attribute (product_id, attribute_value_id)

VALUES (inProductId, inAttributeValueId);

END$$
```

13. Wykonaj następujący kod, który utworzy procedurę składowaną catalog_remove_product_attribute_value w Twojej bazie danych tshirtshop. Ta procedura cofa przypisanie wartości atrybutu do produktu poprzez usunięcie niezbędnego rekordu z tabeli product_attribute.

```
-- Create catalog_remove_product_attribute_value stored procedure

CREATE PROCEDURE catalog_remove_product_attribute_value(

IN inProductId INT, IN inAttributeValueId INT)

BEGIN

DELETE FROM product_attribute

WHERE product_id = inProductId AND

attribute_value_id = inAttributeValueId;

END$$
```

14. Wykonaj ten kod, który utworzy procedury składowane catalog_set_image, catalog_set_image_2 i catalog_set_thumbnail w bazie danych sklepu tshirtshop. Potrzebujemy tych funkcji do zmiany głównego, dodatkowego i/lub miniaturowego obrazu produktu podczas przesyłania nowego zdjęcia.

```
-- Create catalog_set_image stored procedure
CREATE PROCEDURE catalog_set_image(
IN inProductId INT, IN inImage VARCHAR(150))
BEGIN
UPDATE product SET image = inImage WHERE product_id = inProductId;
END$$
-- Create catalog_set_image_2 stored procedure
CREATE PROCEDURE catalog_set_image_2(
IN inProductId INT, IN inImage VARCHAR(150))
BEGIN
UPDATE product SET image_2 = inImage WHERE product_id = inProductId;
END$$
-- Create catalog_set_thumbnail stored procedure
CREATE PROCEDURE catalog_set_thumbnail(
IN inProductId INT, IN inThumbnail VARCHAR(150))
BEGIN
UPDATE product
SET thumbnail = inThumbnail
WHERE product_id = inProductId;
END$$
```

15. Załaduj stronę szczegółów produktu i upewnij się, że wszystko działa tak, jak powinno. Masz wiele funkcji do przetestowania! Rysunki 4 i 5 pokazują stronę administratora szczegółów produktu w akcji.

Jak to działa: administrowanie szczegółami produktu

Żadna funkcja administracyjna nie jest fajna do zaimplementowania, ale to, co do tej pory osiągnąłeś, jest imponujące. Administratorzy Twojego sklepu mogą teraz edytować nazwę, opis i cenę produktu; usuwać produkty z bazy danych lub tylko z kategorii; przypisać produkt do jednej lub więcej kategorii; oraz zarządzać atrybutami i zdjęciami produktów. W tej chwili oferujesz wszystkie ważne funkcje, które są niezbędne do administrowania stroną internetową, taką jak TShirtShop.

Dodamy jednak jeszcze jedną funkcję, która znacznie ułatwi życie administratorom naszych sklepów: linki administracyjne w sklepie.

Tworzenie linków administracyjnych w sklepie

W tej chwili strona administracyjna spełnia wszystkie ważne wymagania: administratorzy mogą zarządzać działami katalogu, kategoriami, produktami i ich atrybutami. W ostatniej części tego rozdziału wdrażamy dodatkową funkcję, która nieco ułatwia zadanie administratora. Ta funkcja składa

się z przycisków Edytuj, takich jak Edytuj szczegóły działu i Edytuj szczegóły produktu, które można zobaczyć na rysunku 6. Przyciski te pojawiają się tylko wtedy, gdy odwiedzający jest uwierzytelniony jako administrator i przekierowuje go bezpośrednio na stronę administracyjną elementu.

Ćwiczenie: Implementacja linków administracyjnych w sklepie

1. Otwórz plik tshirtshop.css z folderu stylów i dodaj następujące definicje stylów:

```
div.yui-b div form.edit-form
{
margin: 0;
padding: 0;
}
.edit-form
{
margin: 0;
padding: 0 0 10px 0;
}
```

2. Otwórz prezentację/szablony/department.tpl i edytuj ją w ten sposób

```
{* department.tpl *}

{load_presentation_object filename="department" assign="obj"}

<h1>{$obj->mName}</h1>

<p class="description">{$obj->mDescription}</p>

{if $obj->mShowEditButton}

<form action="{$obj->mEditActionTarget}" method="post" class="edit-form">

<input type="submit" name="submit_{$obj->mEditAction}"

value="{$obj->mEditButtonCaption}" />

</form>

{/if}

{include file="products_list.tpl"}
```

3. Dodaj następujących wyróżnionych członków do klasy Department w Presentation/department.php:

```
// Deals with retrieving department details

class Department

{

// Public variables for the smarty template

public $mName;

public $mDescription;

public $mEditActionTarget;

public $mEditAction;

public $mEditButtonCaption;

public $mShowEditButton;
```

4. Również w Presentation/department.php dodaj następujący fragment kodu na końcu konstruktora klasy Department. Dzięki temu przycisk Edytuj jest widoczny, jeśli użytkownik jest administratorem.

```
/* If CategoryId is in the query string we save it

(casting it to integer to protect against invalid values) */

if (isset ($_GET['CategoryId']))

$this->_mCategoryId = (int)$_GET['CategoryId'];

// Show Edit button if the user is administrator

if (!(isset ($_SESSION['admin_logged'])) ||

$_SESSION['admin_logged'] != true)
```

```
$this->mShowEditButton = false;
else
$this->mShowEditButton = true;
}
```

5. Nadal w Presentation/department.php, dodaj następujący podświetlony fragment kodu na końcu metody init():

```
// If visiting a category ...
if (isset ($this->_mCategoryId))
{
$category_details =
Catalog::GetCategoryDetails($this->_mCategoryId);
$this->mName = $this->mName . ' &raquo; ' .
$category_details['name'];
$this->mDescription = $category_details['description'];
$this->mEditActionTarget =
Link::ToDepartmentCategoriesAdmin($this->_mDepartmentId);
$this->mEditAction = 'edit_cat_' . $this->_mCategoryId;
$this->mEditButtonCaption = 'Edit Category Details';
}
else
{
$this->mEditActionTarget = Link::ToDepartmentsAdmin();
$this->mEditAction = 'edit_dept_' . $this->_mDepartmentId;
$this->mEditButtonCaption = 'Edit Department Details';
}
}
}
?>
```

6. Dodaj następujący fragment kodu do prezentacji/templates/product.tpl:

```
{* Add the submit button and close the form *}
<p>
```

<input type="submit" name="submit" value="Add to Cart" />

</form>

{* Show edit button for administrators *}

{if $obj->mShowEditButton}

<form action="{$obj->mEditActionTarget}" target="_self"

method="post" class="edit-form">

<input type="submit" name="submit_edit" value="Edit Product Details" />

</form>

{/if}

{if $obj->mLinkToContinueShopping}

mLinkToContinueShopping}">Continue Shopping

{/if}

7. Dodaj te elementy do klasy Product w Presentation/product.php:

// Handles product details

class Product

{

// Public variables to be used in Smarty template

public $mProduct;

public $mProductLocations;

public $mLinkToContinueShopping;

public $mLocations;

public $mEditActionTarget;

public $mShowEditButton;

8. Dodaj ten kod na końcu konstruktora klasy Product, w Presentation/product.php. Ten kod zapewnia, że przycisk Edytuj jest wyświetlany tylko wtedy, gdy użytkownik jest administratorem:

else

trigger_error('ProductId not set');

// Show Edit button for administrators

```
if (!(isset ($_SESSION['admin_logged'])) ||
$_SESSION['admin_logged'] != true)
$this->mShowEditButton = false;
else
$this->mShowEditButton = true;
}
```

9. W tym samym pliku dodaj podświetlony fragment kodu na końcu metody init() klasy Product. Spowoduje to utworzenie łącza do strony edycji produktu.

```
// Build links for product departments and categories pages
for ($i = 0; $i < count($this->mLocations); $i++)
{
$this->mLocations[$i]['link_to_department'] =
Link::ToDepartment($this->mLocations[$i]['department_id']);
$this->mLocations[$i]['link_to_category'] =
Link::ToCategory($this->mLocations[$i]['department_id'],
$this->mLocations[$i]['category_id']);
}
// Prepare the Edit button
$this->mEditActionTarget =
Link::Build(str_replace(VIRTUAL_LOCATION, '', getenv('REQUEST_URI')));
if (isset ($_SESSION['admin_logged']) &&
$_SESSION['admin_logged'] == true &&
isset ($_POST['submit_edit']))
{
$product_locations = $this->mLocations;
if (count($product_locations) > 0)
{
$department_id = $product_locations[0]['department_id'];
$category_id = $product_locations[0]['category_id'];
header('Location: ' .
htmlspecialchars_decode(
```

```
Link::ToProductAdmin($department_id,
$category_id,
$this->_mProductId)));
}
}
}
}
?>
```

10. Dodaj następujący fragment kodu do Presentation/templates/products_list.tpl. To dodaje Edytuj do list produktów.

```
{* Add the submit button and close the form *}
<p>
<input type="submit" name="submit" value="Add to Cart" />
</p>
</form>
{* Show Edit button for administrators *}
{if $obj->mShowEditButton}
<form action="{$obj->mEditActionTarget}" target="_self"
method="post" class="edit-form">
<input type="hidden" name="product_id"
value="{$obj->mProducts[k].product_id}" />
<input type="submit" name="submit" value="Edit Product Details" />
</form>
{/if}
</td>
{if $smarty.section.k.index % 2 != 0 && !$smarty.section.k.first ||
$smarty.section.k.last}
</tr>
{/if}
{/section}
```

11. Otwórz prezentację/products_list.php i dodaj następujące elementy do klasy ProductsList:

```
public $mAllWords = 'off';
public $mSearchString;
public $mEditActionTarget;
public $mShowEditButton;
// Private members
private $_mDepartmentId;
private $_mCategoryId;
```

12. Dodaj następujący fragment kodu na końcu metody __construct() klasy ProductsList:

```
if ($this->mPage < 1)
trigger_error('Incorrect Page value');
// Save page request for continue shopping functionality
$_SESSION['link_to_continue_shopping'] = $_SERVER['QUERY_STRING'];
// Show Edit button for administrators
if (!(isset ($_SESSION['admin_logged'])) ||
$_SESSION['admin_logged'] != true)
$this->mShowEditButton = false;
else
$this->mShowEditButton = true;
}
```

13. Również w Presentation/products_list.php dodaj następujący fragment kodu na początku metody init() w ProductsList:

```
public function init()
{
// Prepare the Edit button
$this->mEditActionTarget =
Link::Build(str_replace(VIRTUAL_LOCATION, '', getenv('REQUEST_URI')));
if (isset ($_SESSION['admin_logged']) &&
$_SESSION['admin_logged'] == true &&
isset ($_POST['product_id']))
{
if (isset ($this->_mDepartmentId) && isset ($this->_mCategoryId))
```

```
header('Location: ' .
htmlspecialchars_decode(
Link::ToProductAdmin($this->_mDepartmentId,
$this->_mCategoryId,
(int)$_POST['product_id'])));
else
{
$product_locations =
Catalog::GetProductLocations((int)$_POST['product_id']);
if (count($product_locations) > 0)
{
$department_id = $product_locations[0]['department_id'];
$category_id = $product_locations[0]['category_id'];
header('Location: ' .
htmlspecialchars_decode(
Link::ToProductAdmin($department_id,
$category_id,
(int)$_POST['product_id'])));
}
}
}
/* If searching the catalog, get the list of products by calling
the Search business tier method */
if (isset ($this->mSearchString))
{
// Get search results
$search_results = Catalog::Search($this->mSearchString,
$this->mAllWords,
$this->mPage,
$this->mrTotalPages);
```

14. Otwórz Presentation/store_front.php i dodaj następujący fragment kodu na początku metody init() klasy StoreFront:

```
public function init()

{

$_SESSION['link_to_store_front'] =

Link::Build(str_replace(VIRTUAL_LOCATION, '', getenv('REQUEST_URI')));

// Create "Continue Shopping" link for the PayPal shopping cart

if (!isset ($_GET['AddProduct']))

{
```

15. Otwórz Presentation/admin_menu.php i zmodyfikuj konstruktor klasy AdminMenu w następujący sposób:

```
public function __construct()

{

$this->mLinkToStoreAdmin = Link::ToAdmin();

$this->mLinkToAttributesAdmin = Link::ToAttributesAdmin();

if (isset ($_SESSION['link_to_store_front']))

$this->mLinkToStoreFront = $_SESSION['link_to_store_front'];

else

$this->mLinkToStoreFront = Link::ToIndex();

$this->mLinkToLogout = Link::ToLogout();

}
```

16. Nawiguj po TShirtShop będąc zalogowanym jako administrator i zauważ pojawiające się przyciski Edytuj, jak pokazano na rysunkach 6 i 7.

Jak to działa: linki administracyjne w sklepie

W tym ćwiczeniu utworzyliśmy przyciski Edytuj w całym katalogu produktów, które są wyświetlane tylko wtedy, gdy bieżący użytkownik jest zalogowany jako administrator. W ten sposób administrator, który zauważy, powiedzmy, opis produktu, który wymaga aktualizacji, może kliknąć przycisk Edytuj dla tego produktu bezpośrednio z katalogu, zamiast przeglądać strony administracyjne w celu znalezienia tego produktu. Wdrożenie tej funkcji nie było prawdopodobnie najbardziej ekscytującym ćwiczeniem z kodowania, jakie kiedykolwiek wykonałeś, ale jest to naprawdę przydatna funkcja, którą z pewnością docenią Twoi klienci!

Podsumowanie

Wdrożyłeś funkcje administracyjne dla swoich produktów, w tym funkcje dodawania lub edytowania atrybutów produktów, przypisywania produktów do kategorii, przesyłania zdjęć produktów i tak dalej. Zaktualizowałeś również TShirtShop, aby zawierał przyciski edycji na stronie katalogu, dzięki czemu administratorzy mogą znacznie łatwiej uzyskać dostęp do stron administracyjnych w celu aktualizacji informacji o katalogu. Teraz, gdy sucha część tworzenia strony administracyjnych naszej strony została zakończona, jesteśmy w końcu gotowi do przejścia do naprawdę ekscytujących funkcji. W Części 12 zaimplementujesz własny koszyk na zakupy w TShirtShop, zastępując koszyk PayPal, z którego korzystałeś do tej pory.