Tworzenie własnego koszyka

Witamy w drugim etapie rozwoju! Na tym etapie zaczniesz ulepszać i dodawać nowe funkcje do już istniejącej, w pełni funkcjonalnej witryny e-commerce. Co dokładnie możesz poprawić? Cóż, odpowiedź na to pytanie nie jest trudna do znalezienia, jeśli rzucisz okiem na popularne witryny e-commerce w sieci. Personalizują one wrażenia użytkownika, zapewniają rekomendacje produktów, zapamiętują preferencje klientów i oferują wiele innych funkcji, które sprawiają, że witryna jest łatwa do zapamiętania i trudna do opuszczenia bez uprzedniego zakupu czegoś. Na pierwszym etapie rozwoju w dużym stopniu polegałeś na zewnętrznym procesorze płatności (PayPal), który dostarczył zintegrowany koszyk na zakupy, więc nie rejestrowałeś żadnego koszyka ani informacji o zamówieniach w bazie danych. W tej chwili Twoja witryna nie jest w stanie wyświetlić listy „najbardziej poszukiwanych" produktów ani żadnych innych informacji o produktach, które zostały sprzedane za pośrednictwem witryny, ponieważ na tym etapie nie śledzisz sprzedawanych produktów. Zapisywanie informacji o zamówieniach w bazie danych jest teraz jednym z naszych priorytetów, ponieważ większość funkcji, które chcesz następnie wdrożyć, opiera się na posiadaniu tych informacji. Na końcu tego rozdziału będziesz miał sprawny koszyk na zakupy, ale odwiedzający nie będzie mógł jeszcze zamawiać zawartych w nim produktów. Dodasz tę funkcję w części 14, kiedy wdrożysz niestandardowy system kasowy, który integruje się z Twoim nowym koszykiem na zakupy. W szczególności w tym rozdziale dowiesz się, jak:

• Przeanalizuj elementy koszyka.

• Utwórz strukturę bazy danych, która przechowuje rekordy koszyka zakupów.

• Implementuj składniki warstwy danych, warstwy biznesowej i warstwy prezentacji koszyka zakupów.

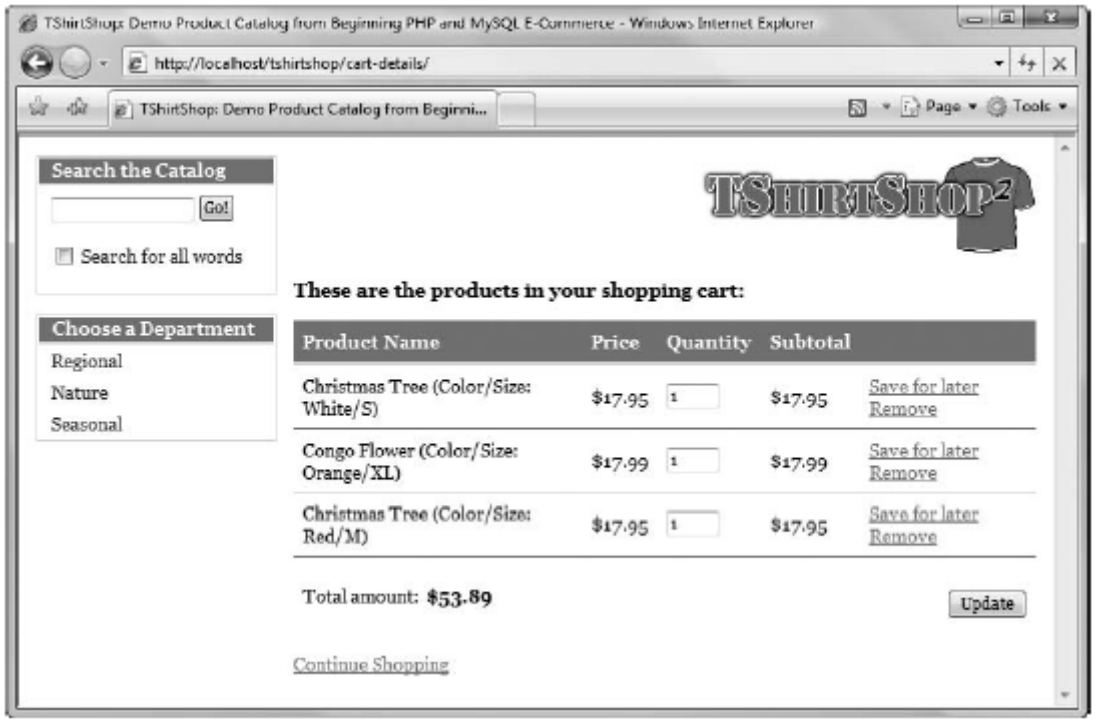
• Zaktualizuj przyciski Dodaj do koszyka PayPal utworzone w Części 9, aby działały z nowym koszykiem.

• Utwórz okno podsumowania koszyka, aby przypomnieć użytkownikom o produktach w ich koszykach i całkowitych kwotach.

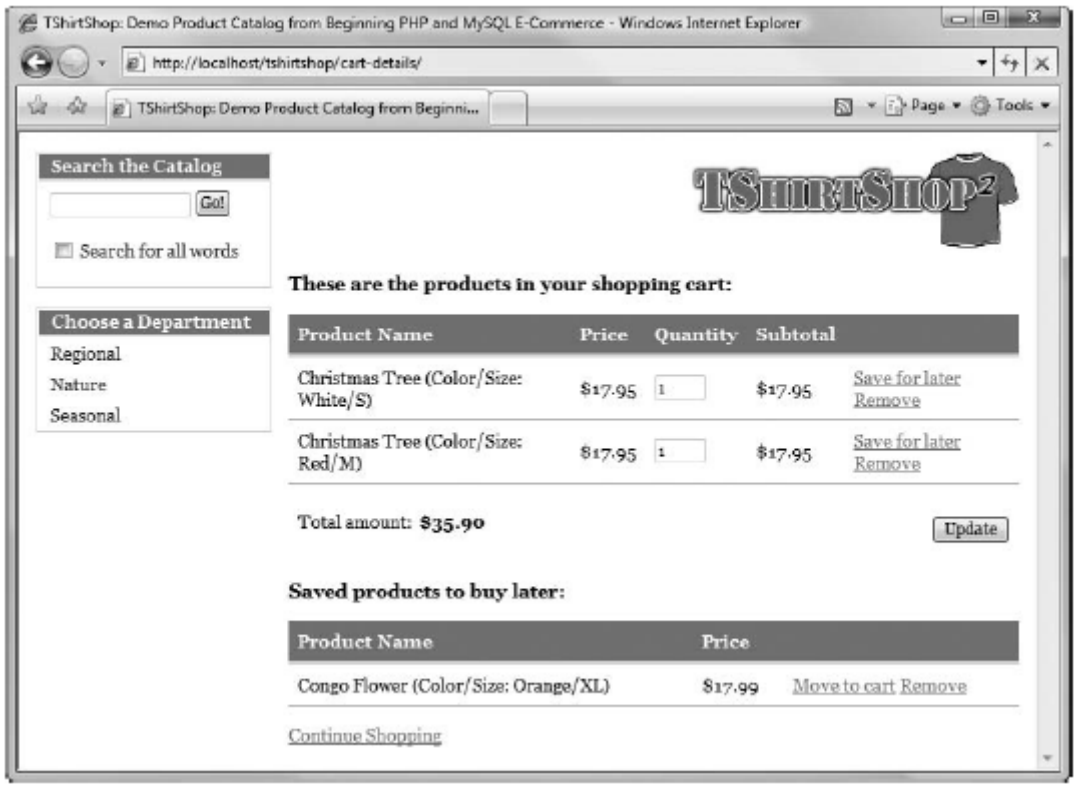
• Zaimplementuj stronę administrowania koszykiem na zakupy, która umożliwia administratorom witryny usuwanie koszyków, które nie były aktualizowane przez określoną liczbę dni.

Projektowanie koszyka na zakupy

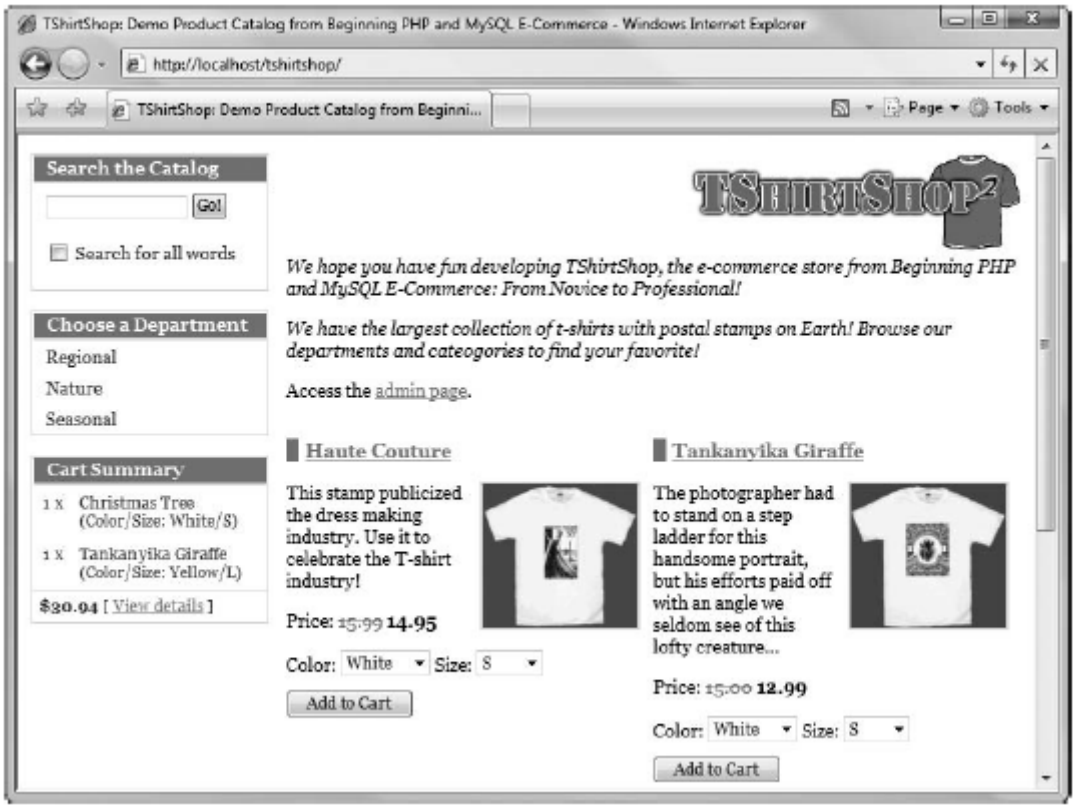
Tu zaimplementujemy niestandardowy koszyk, który przechowuje dane w lokalnej bazie tshirtshop. Zapewni to znacznie większą elastyczność niż koszyk PayPal, nad którym nie masz kontroli i którego nie można łatwo zapisać w bazie danych w celu dalszego przetwarzania i analizy. W przypadku niestandardowego koszyka na zakupy, gdy odwiedzający kliknie przycisk Dodaj do koszyka dla produktu, produkt jest nadal dodawany do koszyka odwiedzającego, ale ten koszyk i informacje o produkcie będą przechowywane bezpośrednio w bazie danych tshirtshop, a nie w niedostępnej bazie danych PayPal . Gdy odwiedzający kliknie przycisk Wyświetl koszyk, pojawi się strona podobna do tej pokazanej na rysunku



Nasz koszyk będzie miał funkcję „Zapisz na później”, dzięki której odwiedzający może zamówić tylko podzbiór produktów w koszyku i zapisać pozostałe produkty do zakupu w późniejszym czasie. Gdy produkt zostanie zapisany na później, jest przenoszony na osobną listę koszyka i nie jest uwzględniany w zamówieniu, gdy odwiedzający się wymeldowuje.



Na wszystkich innych stronach z wyjątkiem strony koszyka, odwiedzający będzie mógł zobaczyć podsumowanie koszyka zakupów w lewej części ekranu, jak pokazano na rysunku



Zanim zaczniemy pisać kod do koszyka, przyjrzyjmy się bliżej temu, co zamierzamy zrobić. Po pierwsze, pamiętaj, że na tym etapie witryny nie będziesz mieć żadnych funkcji personalizacji użytkownika. W tym momencie nie ma znaczenia, kto kupuje Twoje produkty; chcesz tylko wiedzieć, jakie produkty zostały sprzedane i kiedy. Kiedy dodasz funkcje dostosowywania użytkownika w późniejszych częściach, twoje zadanie będzie dość proste: kiedy odwiedzający się uwierzytelni, tymczasowy (anonimowy) koszyk odwiedzającego zostanie powiązany z kontem odwiedzającego. Ponieważ pracujesz z tymczasowymi koszykami zakupowymi, nawet po wdrożeniu systemu kont klienta, odwiedzający nie musi podawać dodatkowych informacji (logowania) wcześniej niż to konieczne. Używamy plików cookie do śledzenia koszyków zakupowych. Kiedy odwiedzający kliknie przycisk Dodaj do koszyka, serwer najpierw sprawdza, czy plik cookie koszyka na zakupy już istnieje na komputerze odwiedzającego. Jeśli tak, określony produkt zostanie dodany do istniejącego koszyka. W przeciwnym razie serwer generuje unikalny identyfikator koszyka, zapisuje go w ciasteczku klienta, a następnie dodaje produkt do nowo wygenerowanego koszyka.

Przechowywanie informacji o koszyku

Będziesz przechowywać wszystkie informacje z koszyków w jednej tabeli o nazwie shopping_cart. Wykonaj następne ćwiczenie, aby utworzyć tabelę shopping_cart.

Ćwiczenie: Tworzenie tabeli koszyka_koszyków

1. Załaduj phpMyAdmin, wybierz bazę danych tshirtshop i otwórz nową stronę zapytań SQL.

2. Wykonaj następujący kod, który utworzy tabelę shopping_cart w Twojej bazie danych tshirtshop:

```
-- Create shopping_cart table
```

```
CREATE TABLE `shopping_cart` (
`item_id` INT NOT NULL AUTO_INCREMENT,
`cart_id` CHAR(32) NOT NULL,
`product_id` INT NOT NULL,
`attributes` VARCHAR(1000) NOT NULL,
`quantity` INT NOT NULL,
`buy_now` BOOL NOT NULL DEFAULT true,
`added_on` DATETIME NOT NULL,
PRIMARY KEY (`item_id`),
KEY `idx_shopping_cart_cart_id` (`cart_id`)
);
```

Jak to działa: tabela z koszykiem

Przyjrzyjmy się każdemu polu w shopping_cart:

• item_id to klucz podstawowy tabeli. Jego wartość jednoznacznie identyfikuje rekord koszyka.

• cart_id to wartość CHAR(32), która jednoznacznie identyfikuje koszyk na zakupy odwiedzającego (w przeciwieństwie do tylko jednego rekordu koszyka).

• product_id odwołuje się do identyfikatora istniejącego produktu.

• atrybuty to zmienne pole znaków, które przechowuje atrybuty wybrane przez odwiedzającego podczas dodawania produktu do koszyka. Możliwą wartością tego pola może być np. "Kolor/Rozmiar: Biały/XL".

• ilość przechowuje ilość produktu w koszyku.

• kup_teraz to pole logiczne z domyślną wartością true, które obsługuje funkcję „Zapisz na później”. Gdy klient wymelduje się, do zamówienia dodawane są tylko te produkty, dla których ta wartość jest ustawiona na true, natomiast produkty „Zapisz na później” pozostają w koszyku.

• add_on to pole daty wypełnione datą dodania produktu do koszyka. Wartość ta służy do obliczania wieku koszyka, co daje nam możliwość usunięcia starych koszyków z bazy danych.

Wartość pierwszego pola, item_id, jest obliczana za każdym razem, gdy w tabeli tworzony jest nowy rekord. Żadnych niespodzianek — już wcześniej pracowałeś z kolumnami AUTO_INCREMENT. Wartość cart_id reprezentuje identyfikator koszyka na zakupy. Ta wartość jest obliczana przez warstwę biznesową za każdym razem, gdy tworzony jest nowy koszyk. Pole cart_id zawiera wartość, która jednoznacznie identyfikuje koszyk na zakupy odwiedzającego. Ważne jest, aby zrozumieć, że użytkownik może mieć kilka produktów w koszyku, że każdy produkt może mieć różne konfiguracje atrybutów oraz że klient może kupić ten sam produkt kilka razy, każdy z innymi atrybutami. Na przykład, odwiedzający może mieć w koszyku koszulkę Black/L Torch i Pink/M Torch. Z tego powodu jedyną kombinacją pól gwarantującą unikalność pozycji koszyka są (cart_id, product_id, attributes), które mogą służyć jako klucz podstawowy tabeli. Ale poczekaj! MySQL ma ograniczenia dotyczące rozmiaru kolumn tworzących klucz podstawowy. Próba utworzenia klucza podstawowego, jak

określono wcześniej, powoduje następujący błąd: „#1071 — Określony klucz był za długi; maksymalna długość klucza to 999 bajtów.” Błąd zasadniczo mówi, że skumulowany rozmiar kolumn tworzących klucz podstawowy nie może przekroczyć 999 bajtów, aw naszym przypadku trzy pola przekraczają 3000 bajtów (znak zakodowany w UT8 zajmuje trzy bajty). Oznacza to, że jeśli chcesz utworzyć klucz podstawowy z trzech pól, musisz ograniczyć rozmiar atrybutów do 269 znaków. To ograniczenie może, ale nie musi być dla Ciebie do zaakceptowania, w zależności od tego, co sprzedajesz, a nawet jeśli dzisiaj jest w porządku, jutro możesz potrzebować większych atrybutów! Nasze obejście polega na utworzeniu wartości sztucznego klucza podstawowego — item_id — a następnie użyciu tego pola jako klucza podstawowego. To rozwiązanie polegające na dodaniu nowego pola łamie zasady trzeciej postaci normalnej lub po prostu nie jest idealnym projektem bazy danych; jednak przy pewnym starannym zaplanowaniu pozwala nam na wdrożenie funkcjonalnego koszyka na zakupy.

Implementacja warstwy danych

Teraz utworzymy procedury składowane, które obsługują niezbędne operacje koszyka na zakupy. Stworzymy te procedury w bazie danych tshirtshop:

• shopping_cart_add_product dodaje produkt do koszyka.

• shopping_cart_update modyfikuje ilości produktów w koszyku i późniejszą ich wycenę.

• shopping_cart_remove_product usuwa produkt z koszyka odwiedzającego.

• shopping_cart_get_products pobiera listę produktów w określonym koszyku i jest wywoływana, gdy chcesz pokazać użytkownikowi koszyk.

• shopping_cart_get_saved_products pobiera listę produktów zapisanych do kupienia później i jest wywoływana, gdy użytkownik zażąda strony ze szczegółami koszyka.

• shopping_cart_get_total_amount zwraca łączne koszty produktów w określonym koszyku produktów.

• shopping_cart_save_product_for_later zapisuje produkt w koszyku do późniejszego zakupu.

• shopping_cart_move_product_to_cart przenosi produkt z listy „Zapisz na później” z powrotem do „głównego” koszyka.

Teraz stwórzmy każdą metodę pojedynczo w poniższym ćwiczeniu.

Ćwiczenie: Implementacja procedur składowanych

1. Użyj phpMyAdmin, aby utworzyć procedury składowane opisane w poniższych krokach. Nie zapomnij ustawić separatora $$ przed wykonaniem kodu każdego kroku.

2. Utwórz procedurę składowaną shopping_cart_add_product w bazie danych sklepu tshirtshop, wykonując ten kod:

```
-- Create shopping_cart_add_product stored procedure

CREATE PROCEDURE shopping_cart_add_product(IN inCartId CHAR(32),

IN inProductId INT, IN inAttributes VARCHAR(1000))

BEGIN

DECLARE productQuantity INT;
```

```
-- Obtain current shopping cart quantity for the product
SELECT quantity
FROM shopping_cart
WHERE cart_id = inCartId
AND product_id = inProductId
AND attributes = inAttributes
INTO productQuantity;
-- Create new shopping cart record, or increase quantity of existing record
IF productQuantity IS NULL THEN
INSERT INTO shopping_cart(cart_id, product_id, attributes,
quantity, added_on)
VALUES (inCartId, inProductId, inAttributes, 1, NOW());
ELSE
UPDATE shopping_cart
SET quantity = quantity + 1, buy_now = true
WHERE cart_id = inCartId
AND product_id = inProductId
AND attributes = inAttributes;
END IF;
END$$
```

Procedura przechowywana shopping_cart_add_product jest wywoływana, gdy użytkownik kliknie przycisk Dodaj do koszyka dla jednego z produktów. Jeżeli wybrany produkt znajduje się już w koszyku, jego ilość zwiększa się o jeden; w przeciwnym razie do koszyka zostanie dodana jedna nowa jednostka (utworzony zostanie nowy rekord shopping_cart). Procedura otrzymuje trzy parametry: inCartId, inProductId i inAttributes. Najpierw określa, czy produkt już istnieje w koszyku, szukając identyfikatora koszyka, identyfikatora produktu i atrybutów. Jeśli ta kombinacja istnieje w tabeli shopping_cart, oznacza to, że odwiedzający ma już produkt w swoim koszyku, więc aktualizujesz istniejącą ilość, dodając jedną jednostkę. W przeciwnym razie shopping_cart_add_product tworzy nowy rekord dla produktu w shopping_cart z domyślną liczbą 1. Funkcja NOW() MySQL służy do pobierania bieżącej daty w celu wypełnienia pola add_on.

3. Wykonaj następujący kod, który utworzy procedurę składowaną shopping_cart_update w Twojej bazie danych tshirtshop:

```
-- Create shopping_cart_update_product stored procedure
CREATE PROCEDURE shopping_cart_update(IN inItemId INT, IN inQuantity INT)
BEGIN
```

```
IF inQuantity > 0 THEN

UPDATE shopping_cart

SET quantity = inQuantity, added_on = NOW()

WHERE item_id = inItemId;

ELSE

CALL shopping_cart_remove_product(inItemId);

END IF;

END$$
```

Procedura składowana shopping_cart_update aktualizuje ilość jednego towaru. Ta procedura składowana jest wywoływana, gdy użytkownik kliknie przycisk Aktualizuj na stronie koszyka i jest wywoływana raz dla każdego elementu, który wymaga aktualizacji. Procedura otrzymuje dwa parametry: inItemId oraz inQuantity. Jeśli inQuantity wynosi zero lub mniej, shopping_cart_update jest wystarczająco sprytny, aby usunąć wspomnianą pozycję z koszyka. W przeciwnym razie aktualizuje ilość produktu w koszyku, a także aktualizuje add_on, aby dokładnie odzwierciedlić czas ostatniej modyfikacji rekordu. Aktualizacja add_on jest przydatna dla strony zarządzania katalogiem, gdzie to pole służy do obliczania wieku koszyka i usuwania starych koszyków. W tym celu uważamy, że towar, którego ilość została zmodyfikowana, jest „nowym towarem".

4. Wykonaj następujący kod, który utworzy procedurę przechowywaną shopping_cart_remove_product w Twojej bazie danych tshirtshop:

```
-- Create shopping_cart_remove_product stored procedure

CREATE PROCEDURE shopping_cart_remove_product(IN inItemId INT)

BEGIN

DELETE FROM shopping_cart WHERE item_id = inItemId;

END$$
```

Procedura przechowywana shopping_cart_remove_product usuwa pozycję z koszyka, gdy odwiedzający kliknie przycisk Usuń dla jednego z produktów w koszyku.

5. Wykonaj ten kod, który utworzy procedurę składowaną shopping_cart_get_products w Twojej bazie danych tshirtshop:

```
-- Create shopping_cart_get_products stored procedure

CREATE PROCEDURE shopping_cart_get_products(IN inCartId CHAR(32))

BEGIN

SELECT sc.item_id, p.name, sc.attributes,

COALESCE(NULLIF(p.discounted_price, 0), p.price) AS price,

sc.quantity,

COALESCE(NULLIF(p.discounted_price, 0),
```

p.price) * sc.quantity AS subtotal

FROM shopping_cart sc

INNER JOIN product p

ON sc.product_id = p.product_id

WHERE sc.cart_id = inCartId AND sc.buy_now;

END$$

Procedura składowana shopping_cart_get_products zwraca pozycje w koszyku wymienionym w parametrze inCartId. Ponieważ tabela shopping_cart przechowuje identyfikator produktu dla każdego przechowywanego produktu, musisz połączyć tabele shopping_cart i product, aby uzyskać potrzebne informacje. Pamiętaj, że niektóre artykuły mogą mieć obniżone ceny. Gdy towar ma obniżoną cenę (co ma miejsce, gdy jego wartość zdyskontowana_cena jest różna od 0), wówczas do obliczeń należy użyć jego zdyskontowanej ceny. W przeciwnym razie należy użyć jego ceny katalogowej. Poniższe wyrażenie zwraca Discounted_price, jeśli jest różne od 0; w przeciwnym razie zwraca cenę.

COALESCE(NULLIF(p.cena_zniżkowa; 0), p.cena)

Uwaga Po raz pierwszy pracujesz z wyrażeniami warunkowymi COALESCE i NULLIF, więc teraz wyjaśnimy, co one robią. COALESCE może otrzymać dowolną liczbę parametrów i zwraca pierwszy, który nie jest NULL. NULLIF otrzymuje dwa parametry i zwraca NULL, jeśli są one równe; w przeciwnym razie zwraca pierwszy z parametrów. W naszym przypadku używamy NULLIF, aby sprawdzić, czy p.discounted_price wynosi 0; jeśli ten warunek jest prawdziwy, NULLIF zwraca fałsz, a funkcja COALESCE zwróci cenę p.price. Jeśli p.discounted_price jest różne od 0, całe wyrażenie zwraca p.discounted_price.

6. Wykonaj następujący kod, który utworzy procedurę składowaną shopping_cart_get_saved_products w Twojej bazie danych tshirtshop:

-- Create shopping_cart_get_saved_products stored procedure

CREATE PROCEDURE shopping_cart_get_saved_products(IN inCartId CHAR(32))

BEGIN

SELECT sc.item_id, p.name, sc.attributes,

COALESCE(NULLIF(p.discounted_price, 0), p.price) AS price

FROM shopping_cart sc

INNER JOIN product p

ON sc.product_id = p.product_id

WHERE sc.cart_id = inCartId AND NOT sc.buy_now;

END$$

Procedura składowana shopping_cart_get_saved_products zwraca pozycje zapisane na później w koszyku określonym przez parametr inCartId.

7. Wykonaj następujący kod, który utworzy procedurę przechowywaną shopping_cart_get_total_amount w Twojej bazie danych tshirtshop:

```
-- Create shopping_cart_get_total_amount stored procedure

CREATE PROCEDURE shopping_cart_get_total_amount(IN inCartId CHAR(32))

BEGIN

SELECT SUM(COALESCE(NULLIF(p.discounted_price, 0), p.price)

* sc.quantity) AS total_amount

FROM shopping_cart sc

INNER JOIN product p

ON sc.product_id = p.product_id

WHERE sc.cart_id = inCartId AND sc.buy_now;

END$$
```

Procedura składowana shopping_cart_get_total_amount zwraca całkowitą wartość pozycji w koszyku przed należnymi podatkami i kosztami wysyłki. Jest to wywoływane podczas wyświetlania całkowitej kwoty za koszyk. Jeśli koszyk jest pusty, total_amount będzie wynosić 0.

8. Wykonaj następujący kod, który utworzy procedurę składowaną shopping_cart_save_product_for_later w Twojej bazie danych tshirtshop:

```
-- Create shopping_cart_save_product_for_later stored procedure

CREATE PROCEDURE shopping_cart_save_product_for_later(IN inItemId INT)

BEGIN

UPDATE shopping_cart

SET buy_now = false, quantity = 1

WHERE item_id = inItemId;

END$$
```

Procedura przechowywana shopping_cart_save_product_for_later zapisuje element koszyka na liście „Zapisz na później”, aby odwiedzający mógł go kupić później (element nie jest wysyłany do kasy podczas składania zamówienia). Jest to realizowane poprzez ustawienie wartości pola buy_now na false.

9. Wykonaj ten kod, który utworzy procedurę składowaną shopping_cart_move_product_to_cart w bazie danych sklepu tshirtshop:

```
-- Create shopping_cart_move_product_to_cart stored procedure

CREATE PROCEDURE shopping_cart_move_product_to_cart(IN inItemId INT)

BEGIN

UPDATE shopping_cart
```

```
SET buy_now = true, added_on = NOW()

WHERE item_id = inItemId;

END$$
```

Procedura przechowywana shopping_cart_move_product_to_cart ustawia stan buy_now dla pozycji koszyka na wartość true, dzięki czemu odwiedzający może kupić produkt podczas składania zamówienia.

Wdrażanie poziomu biznesowego

Aby zaimplementować warstwę biznesową dla koszyka na zakupy, utworzymy plik o nazwie shopping_cart.php, który zawiera klasę ShoppingCart. Ta klasa ma następujące metody:

• SetCartId() generuje nowy identyfikator koszyka i zapisuje go na

przeglądarki internetowej jako plik cookie oraz w sesji.

• GetCartId() zwraca identyfikator koszyka na zakupy.

• AddProduct() dodaje nowy produkt do koszyka odwiedzającego.

• Update() modyfikuje ilość produktu w koszyku odwiedzającego lub usuwa produkt z koszyka, jeśli ilość jest zerowa lub ujemna.

• RemoveProduct() usuwa produkt z koszyka.

• GetCartProducts() pobiera wszystkie produkty w koszyku.

• GetTotalAmount() zwraca całkowitą ilość produktów w koszyku.

• SaveProductForLater() zapisuje produkt w koszyku na później.

• MoveProductToCart() przenosi produkt z listy „Zapisz na później” z powrotem do „głównego” koszyka.

Napiszmy kod.

Ćwiczenie: Implementacja logiki biznesowej koszyka zakupów

1. Najpierw dodaj następujące dwie linie na końcu pliku include/config.php. Te stałe służą do rozróżniania między bieżącymi pozycjami koszyka na zakupy a pozycjami, które zostały zapisane na później.

```
// Shopping cart item types

define('GET_CART_PRODUCTS', 1);

define('GET_CART_SAVED_PRODUCTS', 2);
```

2. Utwórz nowy plik o nazwie shopping_cart.php w folderze biznesowym. Dodaj następujący kod do pliku, a potem go skomentujemy:

```
<?php

// Business tier class for the shopping cart

class ShoppingCart
```

```
{
// Stores the visitor's Cart ID
private static $_mCartId;
// Private constructor to prevent direct creation of object
private function __construct()
{
}
/* This will be called by GetCartId to ensure we have the
visitor's cart ID in the visitor's session in case
$_mCartID has no value set */
public static function SetCartId()
{
// If the cart ID hasn't already been set ...
if (self::$_mCartId == '')
{
// If the visitor's cart ID is in the session, get it from there
if (isset ($_SESSION['cart_id']))
{
self::$_mCartId = $_SESSION['cart_id'];
}
// If not, check whether the cart ID was saved as a cookie
elseif (isset ($_COOKIE['cart_id']))
{
// Save the cart ID from the cookie
self::$_mCartId = $_COOKIE['cart_id'];
$_SESSION['cart_id'] = self::$_mCartId;
// Regenerate cookie to be valid for 7 days (604800 seconds)
setcookie('cart_id', self::$_mCartId, time() + 604800);
}
else
{
```

```
/* Generate cart id and save it to the $_mCartId class member,
the session and a cookie (on subsequent requests $_mCartId
will be populated from the session) */
self::$_mCartId = md5(uniqid(rand(), true));
// Store cart id in session
$_SESSION['cart_id'] = self::$_mCartId;
// Cookie will be valid for 7 days (604800 seconds)
setcookie('cart_id', self::$_mCartId, time() + 604800);
}
}
}
// Returns the current visitor's card id
public static function GetCartId()
{
// Ensure we have a cart id for the current visitor
if (!isset (self::$_mCartId))
self::SetCartId();
return self::$_mCartId;
}
// Adds product to the shopping cart
public static function AddProduct($productId, $attributes)
{
// Build SQL query
$sql = 'CALL shopping_cart_add_product(
:cart_id, :product_id, :attributes)';
// Build the parameters array
$params = array (':cart_id' => self::GetCartId(),
':product_id' => $productId,
':attributes' => $attributes);
// Execute the query
DatabaseHandler::Execute($sql, $params);
```

```
}
// Updates the shopping cart with new product quantities
public static function Update($itemId, $quantity)
{
// Build SQL query
$sql = 'CALL shopping_cart_update(:item_id, :quantity)';
// Build the parameters array
$params = array (':item_id' => $itemId,
':quantity' => $quantity);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
// Removes product from shopping cart
public static function RemoveProduct($itemId)
{
// Build SQL query
$sql = 'CALL shopping_cart_remove_product(:item_id)';
// Build the parameters array
$params = array (':item_id' => $itemId);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
// Gets shopping cart products
public static function GetCartProducts($cartProductsType)
{
$sql = '';
// If retrieving "active" shopping cart products ...
if ($cartProductsType == GET_CART_PRODUCTS)
{
// Build SQL query
$sql = 'CALL shopping_cart_get_products(:cart_id)';
```

```
}
// If retrieving products saved for later ...
elseif ($cartProductsType == GET_CART_SAVED_PRODUCTS)
{
// Build SQL query
$sql = 'CALL shopping_cart_get_saved_products(:cart_id)';
}
else
trigger_error($cartProductsType. ' value unknown', E_USER_ERROR);
// Build the parameters array
$params = array (':cart_id' => self::GetCartId());
// Execute the query and return the results
return DatabaseHandler::GetAll($sql, $params);
}
/* Gets total amount of shopping cart products before tax and/or
shipping charges (not including the ones that are being
saved for later) */
public static function GetTotalAmount()
{
// Build SQL query
$sql = 'CALL shopping_cart_get_total_amount(:cart_id)';
// Build the parameters array
$params = array (':cart_id' => self::GetCartId());
// Execute the query and return the results
return DatabaseHandler::GetOne($sql, $params);
}
// Save product to the Save for Later list
public static function SaveProductForLater($itemId)
{
// Build SQL query
$sql = 'CALL shopping_cart_save_product_for_later(:item_id)';
```

```
// Build the parameters array
$params = array (':item_id' => $itemId);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
// Get product from the Save for Later list back to the cart
public static function MoveProductToCart($itemId)
{
// Build SQL query
$sql = 'CALL shopping_cart_move_product_to_cart(:item_id)';
// Build the parameters array
$params = array (':item_id' => $itemId);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
}
?>
```

3. Dołącz odniesienie do shopping_cart.php w index.php:

```
// Load Business Tier
require_once BUSINESS_DIR . 'catalog.php';
require_once BUSINESS_DIR . 'shopping_cart.php';
```

Jak to działa: poziom biznesowy koszyka

Gdy odwiedzający dodaje produkt lub żąda jakiejkolwiek operacji koszyka na zakupy, generujemy identyfikator koszyka na zakupy dla odwiedzającego, jeśli go nie ma. Zajmujesz się tym w metodzie SetCartId(), która generuje identyfikator koszyka do elementu $_mCartId klasy ShoppingCart. Identyfikator koszyka zakupów jest również zapisywany w sesji odwiedzającego oraz w trwałym pliku cookie. SetCartId() rozpoczyna się od sprawdzenia, czy element $_mCartId został już ustawiony, w takim przypadku nie musimy czytać go z zewnętrznych źródeł:

```
public static function SetCartId()
{
// If the cart ID hasn't already been set ...
if (self::$_mCartId == '')
{
```

Jeśli nie mamy identyfikatora w zmiennej member, kolejnym miejscem do obejrzenia jest sesja odwiedzającego:

```
// If the visitor's cart ID is in the session, get it from there

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

{

self::$_mCartId = $_SESSION['cart_id'];

}
```

Jeśli identyfikatora również nie udało się znaleźć w sesji, sprawdzamy, czy został on zapisany jako plik cookie. Jeśli tak, zapisujemy wartość zarówno do sesji, jak i do elementu $_mCartId i ponownie generujemy plik cookie, aby zresetować jego datę wygaśnięcia:

```
// If not, check whether the cart ID was saved as a cookie

elseif (isset ($_COOKIE['cart_id']))

{

// Save the cart ID from the cookie

self::$_mCartId = $_COOKIE['cart_id'];

$_SESSION['cart_id'] = self::$_mCartId;

// Regenerate cookie to be valid for 7 days (604800 seconds)

setcookie('cart_id', self::$_mCartId, time() + 604800);

}
```

Wreszcie, jeśli nie można nigdzie znaleźć identyfikatora koszyka, nowy jest generowany i zapisywany w sesji, w członku $_mCartId oraz w trwałym pliku cookie:

```
else

{

/* Generate cart id and save it to the $_mCartId class member, the session and a cookie (on subsequent requests $_mCartId will be populated from the session) */

self::$_mCartId = md5(uniqid(rand(), true));

// Store cart id in session

$_SESSION['cart_id'] = self::$_mCartId;

// Cookie will be valid for 7 days (604800 seconds)

setcookie('cart_id', self::$_mCartId, time() + 604800);

}

}

}
```

Do wygenerowania identyfikatora koszyka używane są trzy funkcje: md5(), uniqid() i rand(). Wywołanie md5(uniqid(rand(), true)) generuje unikalną, trudną do przewidzenia, 32-bajtową wartość, która reprezentuje identyfikator koszyka.

Uwaga: jeśli chcesz poznać szczegóły dotyczące generowania identyfikatora koszyka, oto one. Funkcja md5() używa algorytmu Message-Digest 5 (MD5) do obliczenia wartości skrótu wartości, którą otrzymuje jako parametr. Wartość skrótu ma długość 32 znaków. Funkcja uniqid() zwraca unikalny identyfikator na podstawie bieżącego czasu w mikrosekundach; jego pierwszym parametrem jest prefiks, który zostanie dołączony do wygenerowanej wartości, w tym przypadku funkcja rand(), która zwraca pseudolosową wartość z zakresu od 0 do RAND_MAX, która jest zależna od platformy. Jeśli drugi parametr uniqid() jest prawdziwy, uniqid() dodaje dodatkową kombinowaną liniową entropię kongruencyjną (połączone LCG) na końcu wartości zwracanej, co powinno sprawić, że wyniki są jeszcze „bardziej wyjątkowe".

W skrócie, uniquid(rand(), true) generuje „bardzo unikalną" wartość, która jest przekazywana przez md5(), aby zapewnić, że stanie się losową sekwencją znaków o długości 32 znaków. Metoda SetCartId jest używana tylko przez metodę GetCartId(), która zwraca identyfikator koszyka. GetCartId() najpierw sprawdza, czy ustawiono $_mCartId, a jeśli nie, wywołuje SetCartId() przed zwróceniem wartości $_mCartId:

```
// Returns the current visitor's cart id

public static function GetCartId()

{

// Ensure we have a cart id for the current visitor

if (!isset (self::$_mCartId))

self::SetCartId();

return self::$_mCartId;

}
```

Ta wartość jest zapisywana w polu cart_id w tabeli shopping_cart, która jest polem CHAR(32) specjalnie po to, aby pasowała do wartości zwracanej przez funkcję md5(). Przyjrzyjmy się też metodzie GetCartProducts(). Ta metoda zwraca produkty w koszyku. Otrzymuje $cartProductsType jako parametr, który określa, czy szukasz produktów z bieżącego koszyka, czy z produktów zapisanych na później. Jeśli $cartProductsType jest równe stałej GET_CART_PRODUCTS, GetCartProducts() zwróci produkty z koszyka. Jeśli $cartProductsType jest równe stałej GET_CART_SAVED_PRODUCTS, GetCartProducts() zwróci produkty „Zapisz na później". Jeśli $cartProductsType nie jest ani GET_CART_ PRODUCTS, ani GET_CART_SAVED_PRODUCTS, metoda zgłosi błąd. Wszystkie inne napisane przez Ciebie metody warstwy biznesowej w zasadzie wywołują powiązane z nimi funkcje warstwy danych w celu wykonywania różnych zadań koszyka na zakupy.

Wdrażanie poziomu prezentacji

Teraz zbudujmy interfejs użytkownika koszyka. Po zaktualizowaniu witryny sklepowej będziesz mieć przyciski Dodaj do koszyka dla każdego produktu i łącze Wyświetl koszyk w polu podsumowania koszyka w lewej części strony. Jeśli koszyk odwiedzającego jest pusty, link nie jest już wyświetlany, jak widać na rysunku:

Jeśli dodałeś integrację PayPal, jak przedstawiono w części 9, masz już te przyciski w swojej witrynie i tutaj zaktualizujesz ich funkcje. Po kliknięciu Wyświetl koszyk szablon złożony ze szczegółów koszyka jest ładowany w store_front.tpl. Możesz zobaczyć ten komponent skomponentowany w działaniu na rysunku 1. Mechanizm ładowania szablonu skomponentowego szczegółów koszyka jest taki sam, jak ten, którego już użyłeś w store_front.php do załadowania innych komponentów. Po kliknięciu przycisku Dodaj do koszyka, plik index.php jest ponownie ładowany z dodatkowym parametrem (CartAction) w ciągu zapytania:

http://localhost/tshirtshop/index.php?CartAction=1&ItemId=10

Po kliknięciu Wyświetl koszyk parametr CartAction dodany do ciągu zapytania nie przyjmuje żadnej wartości. Koszyk ma pięć akcji koszyka, które są opisane za pomocą następujących nie wyjaśniających się stałych w pliku konfiguracyjnym (include/config.php):

ADD_PRODUCT, REMOVE_PRODUCT, UPDATE_PRODUCTS_QUANTITIES, SAVE_PRODUCT_FOR_LATER, and MOVE_PRODUCT_TO_CART.

Zanim przejdziemy dalej, podsumujmy główne kroki, które podejmiesz, aby zaimplementować cały interfejs użytkownika koszyka:

1. Zmodyfikuj przyciski Dodaj do koszyka, aby użyć niestandardowego koszyka na zakupy.

2. Dodaj pole podsumowania koszyka zakupów do store_front.tpl zamiast przycisku Wyświetl koszyk.

3. Zmodyfikuj metodę CheckRequest() z klasy Link z pliku Presentation/link.php, aby rozpoznawała parametr ciągu zapytania CartAction.

4. Zaimplementuj szablon z komponentami cart_details.

Aktualizacja przycisków Dodaj do koszyka

Należy zmienić kod products_list.tpl, aby każdy wyświetlany produkt zawierał przycisk Dodaj do koszyka z linkiem jak te pokazane wcześniej (link do index.php z dodatkowym parametrem CartAction w ciągu zapytania).

Ćwiczenie: Dodawanie produktów do nowego koszyka

1. Dodaj następujący kod na końcu pliku include/config.php:

```
// Cart actions

define('ADD_PRODUCT', 1);

define('REMOVE_PRODUCT', 2);

define('UPDATE_PRODUCTS_QUANTITIES', 3);

define('SAVE_PRODUCT_FOR_LATER', 4);

define('MOVE_PRODUCT_TO_CART', 5);
```

2. Jeśli wdrożyłeś koszyk PayPal, musisz usunąć kod z prezentacji/store_front.php, który przekierowuje do łącza PayPal po kliknięciu przycisku Dodaj do koszyka. Otwórz Presentation/store_front.php i usuń następujący kod z metody init():

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

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

{

/* Store the current request needed for the paypal

continue shopping functionality */

$_SESSION['paypal_continue_shopping'] =

Link::Build(str_replace(VIRTUAL_LOCATION, '',

$_SERVER['REQUEST_URI']));

...

...

// Redirect to the PayPal cart page

header('HTTP/1.1 302 Found');

header('Location: ' . $paypal_url);

// clear the output buffer and stop execution

flush();

ob_flush();

ob_end_clean();

exit();
```

```
}
```

3. Otwórz Presentation/link.php i zmodyfikuj kod metody CheckRequest() klasy Link, tak jak zaznaczono:

```
// Redirects to proper URL if not already there
public static function CheckRequest()
{
$proper_url = '';
if (isset ($_GET['Search']) || isset($_GET['SearchResults']) ||
isset ($_GET['CartAction']))
{
return ;
}
```

4. Również w Presentation/link.php usuń metodę ToAddProduct() i dodaj metodę o nazwie ToCart(), która tworzy łącza Dodaj produkt i Wyświetl koszyk:

```
// Create a shopping cart link
public static function ToCart($action = 0, $target = null)
{
$link = '';
switch ($action)
{
case ADD_PRODUCT:
$link = 'index.php?CartAction=' . ADD_PRODUCT . '&ItemId=' . $target;
break;
default:
$link = 'cart-details/';
}
return self::Build($link);
}
```

5. Otwórz Presentation/products_list.php i znajdź następujący kod z metody init() klasy ProductList:

```
// Create the Add to Cart link
$this->mProducts[$i]['link_to_add_product'] =
Link::ToAddProduct($this->mProducts[$i]['product_id']);
```

Replace it with the following code that builds the Add to Cart links for our shopping cart:

```
// Create the Add to Cart link
$this->mProducts[$i]['link_to_add_product'] =
Link::ToCart(ADD_PRODUCT, $this->mProducts[$i]['product_id']);
```

6. Otwórz Presentation/product.php i znajdź następujący kod z metody init() klasy Product:

```
// Create the Add to Cart link
$this->mProduct['link_to_add_product'] =
Link::ToAddProduct($this->_mProductId);
```

Zastąp go następującym kodem, który tworzy łącza Dodaj do koszyka dla naszego koszyka::

```
// Create the Add to Cart link
$this->mProduct['link_to_add_product'] =
Link::ToCart(ADD_PRODUCT, $this->_mProductId);
```

Jak to działa: dodawanie linków do produktów

Utworzyłeś przyciski Dodaj do koszyka, które prowadzą do index.php z dodatkowym parametrem CartAction do oryginalnego ciągu zapytania. Po wprowadzeniu tej zmiany uruchom stronę, aby upewnić się, że masz przycisk na swoim miejscu, chociaż nie możesz tak naprawdę przetestować, jak to działa, dopóki nie zakończysz warstwy prezentacji. Jeśli przejdziesz teraz do swojego ulubionego działu lub kategorii i klikniesz przycisk Dodaj do koszyka jednego z produktów, zostaniesz przeniesiony do adresu URL, takiego jak ten:

```
index.php?CartAction=1&ItemId=10
```

Parametr CartAction dołączony na początku ciągu zapytania określa akcję koszyka na zakupy żądaną przez odwiedzającego. Podczas dodawania nowego produktu akcja jest kodowana z 1. W tym momencie nowy link prowadzi do strony głównej, ponieważ Twoja witryna nie wie jeszcze, jak zinterpretować parametr ciągu zapytania CartAction.

Wyświetlanie podsumowania koszyka

Zamiast przycisków Wyświetl koszyk PayPal chcemy mieć element podsumowania koszyka z łączem „Wyświetl szczegóły”, jak pokazano na rysunku 3. Zaimplementuj szablon z elementami cart_summary, wykonując kroki następnego ćwiczenia.

Ćwiczenie: Wyświetlanie podsumowania koszyka

1. Zacznijmy od usunięcia przycisku Wyświetl koszyk. Zlokalizuj i usuń następujący kod w prezentacji/templates/store_front.tpl:

```
<div class="view-cart">
<form target="_self" method="post"
action="{$smarty.const.PAYPAL_URL}">
<input type="hidden" name="cmd" value="_cart" />
```

```
<input type="hidden" name="business"
value="{$smarty.const.PAYPAL_EMAIL}" />
<input type="hidden" name="display" value="1" />
<input type="hidden" name="shopping_url"
value="{$obj->mPayPalContinueShoppingLink}" />
<input type="hidden" name="return"
value="{$smarty.const.PAYPAL_RETURN_URL}" />
<input type="hidden" name="cancel_return"
value="{$smarty.const.PAYPAL_CANCEL_RETURN_URL}" />
<input type="submit" name="view_cart" value="View Cart" />
</form>
</div>
```

2. W tym samym pliku dodaj odnośnik do komponentu podsumowania koszyka:

```
{include file="search_box.tpl"}
{include file="departments_list.tpl"}
{include file=$obj->mCategoriesCell}
{include file=$obj->mCartSummaryCell}
```

3. Otwórz Presentation/store_front.php i zaktualizuj go zgodnie z poniższym fragmentem kodu. W ten sposób klasa StoreFront rozpozna parametr ciągu zapytania CartAction.

```
<?php
class StoreFront
{
public $mSiteUrl;
// Define the template file for the page contents
public $mContentsCell = 'first_page_contents.tpl';
// Define the template file for the categories cell
public $mCategoriesCell = 'blank.tpl';
// Define the template file for the cart summary cell
public $mCartSummaryCell = 'blank.tpl';
// Page title
public $mPageTitle;
```

```
...
public function init()
{
...
// Load product details page if visiting a product
if (isset ($_GET['ProductId']))
$this->mContentsCell = 'product.tpl';
// Load search result page if we're searching the catalog
elseif (isset ($_GET['SearchResults']))
$this->mContentsCell = 'search_results.tpl';
// Load shopping cart or cart summary template
if (isset ($_GET['CartAction']))
$this->mContentsCell = 'cart_details.tpl';
else
$this->mCartSummaryCell = 'cart_summary.tpl';
// Load the page title
$this->mPageTitle = $this->_GetPageTitle();
}
```

4. Utwórz nowy plik obiektu prezentacji o nazwie Presentation/cart_summary.php i dodaj do niego następujący kod:

```
<?php
// Class that deals with managing the shopping cart summary
class CartSummary
{
// Public variables to be used in Smarty template
public $mTotalAmount;
public $mItems;
public $mLinkToCartDetails;
public $mEmptyCart;
// Class constructor
public function __construct()
```

```
{
/* Calculate the total amount for the shopping cart
before applicable taxes and/or shipping charges */
$this->mTotalAmount = ShoppingCart::GetTotalAmount();
// Get shopping cart products
$this->mItems = ShoppingCart::GetCartProducts(GET_CART_PRODUCTS);
if (empty($this->mItems))
$this->mEmptyCart = true;
else
$this->mEmptyCart = false;
$this->mLinkToCartDetails = Link::ToCart();
}
}
?>
```

5. Utwórz nowy plik w folderze prezentacji/szablonów o nazwie cart_summary.tpl i zapisz do niego następujący kod:

```
{* cart_summary.tpl *}
{load_presentation_object filename="cart_summary" assign="obj"}
{* Start cart summary *}
<div class="box">
<p class="box-title">Cart Summary</p>
{if $obj->mEmptyCart}
<p class="empty-cart">Your shopping cart is empty!</p>
{else}
<table class="cart-summary">
<tbody>
{section name=i loop=$obj->mItems}
<tr>
<td width="30" valign="top" align="right">
{$obj->mItems[i].quantity} x
</td>
```

```
<td>
{$obj->mItems[i].name} ({$obj->mItems[i].attributes})
</td>
</tr>
{/section}
<tr>
<td colspan="2" class="cart-summary-subtotal">
<span class="price">${$obj->mTotalAmount}</span>
<span>
[ <a href="{$obj->mLinkToCartDetails}">View details</a> ]
</span>
</td>
</tr>
</tbody>
</table>
{/if}
</div>
{* End cart summary *}
```

6. Otwórz .htaccess i dodaj następujące wiersze:

```
# Rewrite cart details pages
RewriteRule ^cart-details/?$ index.php?CartAction [L]
```

7. Dodaj następujące style do pliku tshirtshop.css z folderu stylów:

```
.empty-cart {
margin: 0px;
padding: 5px 10px;
}
table.cart-summary {
font-size: 85%;
margin: 0px;
padding: 0px 5px;
}
```

```
table.cart-summary td {

border: none;

}

table.cart-summary td.cart-summary-subtotal {

border-top: 1px solid #C6E1EC;

padding: 5px 5px;

}
```

Jak to działa: wyświetlanie podsumowania koszyka

Jeśli przeładujesz TShirtShop, zobaczysz teraz pole podsumowania koszyka po lewej stronie. W tym momencie nadal nie możesz dodawać nowych produktów do koszyka, ponieważ musisz utworzyć stronę szczegółów koszyka. Będziesz mógł w pełni przetestować komponent podsumowania koszyka po zaimplementowaniu strony szczegółów koszyka w następnym ćwiczeniu.

Wyświetlanie szczegółów koszyka

W tej chwili kliknięcie przycisków Dodaj do koszyka i Wyświetl koszyk powoduje wygenerowanie błędu, ponieważ nie napisałeś jeszcze złożonego szablonu cart_details, który wyświetla szczegóły koszyka odwiedzającego. Aby utworzyć nowy szablon z komponentami, najpierw utwórz nowy szablon o nazwie cart_details.tpl w folderze Presentation/Templates. Następnie tworzysz plik obiektu prezentacji cart_details.php, który będzie przechowywać Twoją klasę CartDetails za szablonem cart_details.tpl.

Ćwiczenie: Wyświetlanie szczegółów koszyka

1. Zmodyfikuj prezentację/store_front.php, aby zarządzać funkcją Kontynuuj zakupy na stronie szczegółów koszyka. Dodaj następujący kod w metodzie init() z klasy StoreFront:

```
public function init()

{

$_SESSION['link_to_store_front'] =

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

// Build the "continue shopping" link

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

$_SESSION['link_to_last_page_loaded'] = $_SESSION['link_to_store_front'];

// Load department details if visiting a department

if (isset ($_GET['DepartmentId']))
```

2. Utwórz nowy plik obiektu prezentacji o nazwie Presentation/cart_details.php i dodaj do niego następujący kod:

```
<?php

// Class that deals with managing the shopping cart
```

```
class CartDetails
{
// Public variables available in smarty template
public $mCartProducts;
public $mSavedCartProducts;
public $mTotalAmount;
public $mIsCartNowEmpty = 0; // Is the shopping cart empty?
public $mIsCartLaterEmpty = 0; // Is the 'saved for later' list empty?
public $mLinkToContinueShopping;
public $mUpdateCartTarget;
// Private attributes
private $_mItemId;
private $_mCartAction;
// Class constructor
public function __construct()
{
if (isset ($_GET['CartAction']))
$this->_mCartAction = $_GET['CartAction'];
else
trigger_error('CartAction not set', E_USER_ERROR);
// These cart operations require a valid product id
if ($this->_mCartAction == ADD_PRODUCT ||
$this->_mCartAction == REMOVE_PRODUCT ||
$this->_mCartAction == SAVE_PRODUCT_FOR_LATER ||
$this->_mCartAction == MOVE_PRODUCT_TO_CART)
{
if (isset ($_GET['ItemId']))
$this->_mItemId = $_GET['ItemId'];
else
trigger_error('ItemId must be set for this type of request',
E_USER_ERROR);
```

```
}
$this->mUpdateCartTarget = Link::ToCart(UPDATE_PRODUCTS_QUANTITIES);
// Setting the "Continue shopping" link target
if (isset ($_SESSION['link_to_last_page_loaded']))
$this->mLinkToContinueShopping = $_SESSION['link_to_last_page_loaded'];
}
public function init()
{
switch ($this->_mCartAction)
{
case ADD_PRODUCT:
$selected_attributes = array ();
$selected_attribute_values = array ();
// Get selected product attributes if any
foreach ($_POST as $key => $value)
{
// If there are fields starting with "attr_" in the POST array
if (substr($key, 0, 5) == 'attr_')
{
// Get the selected attribute name and value
$selected_attributes[] = substr($key, strlen('attr_'));
$selected_attribute_values[] = $_POST[$key];
}
}
$attributes = '';
if (count($selected_attributes) > 0)
$attributes = implode('/', $selected_attributes) . ': ' .
implode('/', $selected_attribute_values);
ShoppingCart::AddProduct($this->_mItemId, $attributes);
header('Location: ' . $this->mLinkToContinueShopping);
break;
```

```
case REMOVE_PRODUCT:
ShoppingCart::RemoveProduct($this->_mItemId);
header('Location: ' . Link::ToCart());
break;
case UPDATE_PRODUCTS_QUANTITIES:
for($i = 0; $i < count($_POST['itemId']); $i++)
ShoppingCart::Update($_POST['itemId'][$i], $_POST['quantity'][$i]);
header('Location: ' . Link::ToCart());
break;
case SAVE_PRODUCT_FOR_LATER:
ShoppingCart::SaveProductForLater($this->_mItemId);
header('Location: ' . Link::ToCart());
break;
case MOVE_PRODUCT_TO_CART:
ShoppingCart::MoveProductToCart($this->_mItemId);
header('Location: ' . Link::ToCart());
break;
default:
// Do nothing
break;
}
/* Calculate the total amount for the shopping cart
before applicable taxes and/or shipping */
$this->mTotalAmount = ShoppingCart::GetTotalAmount();
// Get shopping cart products
$this->mCartProducts =
ShoppingCart::GetCartProducts(GET_CART_PRODUCTS);
// Gets the Saved for Later products
$this->mSavedCartProducts =
ShoppingCart::GetCartProducts(GET_CART_SAVED_PRODUCTS);
// Check whether we have an empty shopping cart
```

```
if (count($this->mCartProducts) == 0)
$this->mIsCartNowEmpty = 1;
// Check whether we have an empty Saved for Later list
if (count($this->mSavedCartProducts) == 0)
$this->mIsCartLaterEmpty = 1;
// Build the links for cart actions
for ($i = 0; $i < count($this->mCartProducts); $i++)
{
$this->mCartProducts[$i]['save'] =
Link::ToCart(SAVE_PRODUCT_FOR_LATER,
$this->mCartProducts[$i]['item_id']);
$this->mCartProducts[$i]['remove'] =
Link::ToCart(REMOVE_PRODUCT,
$this->mCartProducts[$i]['item_id']);
}
for ($i = 0; $i < count($this->mSavedCartProducts); $i++)
{
$this->mSavedCartProducts[$i]['move'] =
Link::ToCart(MOVE_PRODUCT_TO_CART,
$this->mSavedCartProducts[$i]['item_id']);
$this->mSavedCartProducts[$i]['remove'] =
Link::ToCart(REMOVE_PRODUCT,
$this->mSavedCartProducts[$i]['item_id']);
}
}
}
?>
```

3. Utwórz nowy plik o nazwie cart_details.tpl w folderze Presentation/Templates i dodaj do niego następujący kod:

```
{* cart_details.tpl *}
{load_presentation_object filename="cart_details" assign="obj"}
```

```
{if $obj->mIsCartNowEmpty eq 1}
<h3>Your shopping cart is empty!</h3>
{else}
<h3>These are the products in your shopping cart:</h3>
<form class="cart-form" method="post" action="{$obj->mUpdateCartTarget}">
<table class="tss-table">
<tr>
<th>Product Name</th>
<th>Price</th>
<th>Quantity</th>
<th>Subtotal</th>
<th> </th>
</tr>
{section name=i loop=$obj->mCartProducts}
<tr>
<td>
<input name="itemId[]" type="hidden"
value="{$obj->mCartProducts[i].item_id}" />
{$obj->mCartProducts[i].name}
({$obj->mCartProducts[i].attributes})
</td>
<td>${$obj->mCartProducts[i].price}</td>
<td>
<input type="text" name="quantity[]" size="5"
value="{$obj->mCartProducts[i].quantity}" />
</td>
<td>${$obj->mCartProducts[i].subtotal}</td>
<td>
<a href="{$obj->mCartProducts[i].save}">Save for later</a>
<a href="{$obj->mCartProducts[i].remove}">Remove</a>
</td>
```

```
</tr>
{/section}
</table>
<table class="cart-subtotal">
<tr>
<td>
<p>
Total amount: 
<font class="price">${$obj->mTotalAmount}</font>
</p>
</td>
<td align="right">
<input type="submit" name="update" value="Update" />
</td>
</tr>
</table>
</form>
{/if}
{if ($obj->mIsCartLaterEmpty eq 0)}
<h3>Saved products to buy later:</h3>
<table class="tss-table">
<tr>
<th>Product Name</th>
<th>Price</th>
<th> </th>
</tr>
{section name=j loop=$obj->mSavedCartProducts}
<tr>
<td>
{$obj->mSavedCartProducts[j].name}
({$obj->mSavedCartProducts[j].attributes})
```

```
</td>
<td>
${$obj->mSavedCartProducts[j].price}
</td>
<td>
<a href="{$obj->mSavedCartProducts[j].move}">Move to cart</a>
<a href="{$obj->mSavedCartProducts[j].remove}">Remove</a>
</td>
</tr>
{/section}
</table>
{/if}
{if $obj->mLinkToContinueShopping}
<p><a href="{$obj->mLinkToContinueShopping}">Continue Shopping </a></p>
{/if}
```

4. Otwórz Presentation/link.php i zmodyfikuj metodę ToCart() z klasy Link, jak zaznaczono tutaj:

```
// Create a shopping cart link
public static function ToCart($action = 0, $target = null)
{
$link = '';
switch ($action)
{
case ADD_PRODUCT:
$link = 'index.php?CartAction=' . ADD_PRODUCT . '&ItemId=' . $target;
break;
case REMOVE_PRODUCT:
$link = 'index.php?CartAction=' .
REMOVE_PRODUCT . '&ItemId=' . $target;
break;
case UPDATE_PRODUCTS_QUANTITIES:
$link = 'index.php?CartAction=' . UPDATE_PRODUCTS_QUANTITIES;
```

```
break;
case SAVE_PRODUCT_FOR_LATER:
$link = 'index.php?CartAction=' .
SAVE_PRODUCT_FOR_LATER . '&ItemId=' . $target;
break;
case MOVE_PRODUCT_TO_CART:
$link = 'index.php?CartAction=' .
MOVE_PRODUCT_TO_CART . '&ItemId=' . $target;
break;
default:
$link = 'cart-details/';
}
return self::Build($link);
}
```

5. Dodaj następujące style do pliku tshirtshop.css z folderu stylów:

```
.cart-subtotal td {
border: none;
margin: 10px 0px;
width: 100%;
}
.yui-b div form.cart-form {
margin: 0;
padding: 0;
}
```

Właśnie skończyłeś część kodu odwiedzającego dla tego rozdziału, więc teraz czas wypróbować i upewnić się, że wszystko działa zgodnie z oczekiwaniami. Przetestuj go, dodając produkty do koszyka, zmieniając ilość i usuwając przedmioty.

Jak to działa: wózek na zakupy

Akcje, które koszyk może wykonać, są zdefiniowane przez stałe zdefiniowane w pliku include/config.php:

ADD_PRODUCT, REMOVE_PRODUCT, UPDATE_PRODUCTS_QUANTITIES, SAVE_PRODUCT_FOR_LATER i MOVE_PRODUCT_TO_CART. Zauważ, że nie zdefiniowaliśmy żadnej zmiennej do przeglądania koszyka, więc jeśli CartAction nie przyjmie żadnej wartości lub jego wartość nie jest równa jednej ze

zmiennych akcji, po prostu wyświetli zawartość koszyka. Każda akcja koszyka, z wyjątkiem przeglądania i aktualizowania koszyka, opiera się na parametrze ciągu zapytania ItemId (jeśli nie jest ustawiony, zgłaszany jest błąd). Jeśli spełnione są odpowiednie warunki, wywoływana jest metoda warstwy biznesowej, która odpowiada działaniu użytkownika.

Administrowanie koszykiem

Teraz, gdy skończyłeś pisać koszyk, musisz wziąć pod uwagę jeszcze dwie kwestie, oba związane z administracją:

• Jak usunąć z katalogu produkt, który istnieje w koszykach.

• Jak liczyć lub usuwać stare elementy koszyka, budując prostą stronę zarządzania koszykiem. Jest to ważne, ponieważ bez tej funkcji stół shopping_cart wciąż rośnie, wypełniony starymi tymczasowymi (i bezużytecznymi) koszykami.

Usuwanie produktów znajdujących się w koszyku

Strony administracyjne katalogu umożliwiają całkowite usunięcie produktów z katalogu. Przed usunięciem produktu należy również usunąć jego wygląd w koszykach odwiedzających. Zaktualizuj funkcję catalog_delete_product z bazy danych tshirtshop, wykonując następujące kroki:

1. Użyj phpMyAdmin, aby wykonać kod opisany w poniższych krokach. Nie zapomnij też ustawić $$ jako ogranicznika przed wykonaniem kodu na każdym kroku.

2. Wykonaj następujący kod, który usunie starą procedurę przechowywaną catalog_delete_product z bazy danych tshirtshop:

-- Usuń starą procedurę przechowywaną catalog_delete_product

PROCEDURA UPUSZCZANIA directory_delete_product$$

3. Wykonaj ten kod, który utworzy nową procedurę przechowywaną catalog_delete_product w Twojej bazie danych tshirtshop:

-- 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 shopping_cart WHERE product_id = inProductId;

DELETE FROM product WHERE product_id = inProductId;

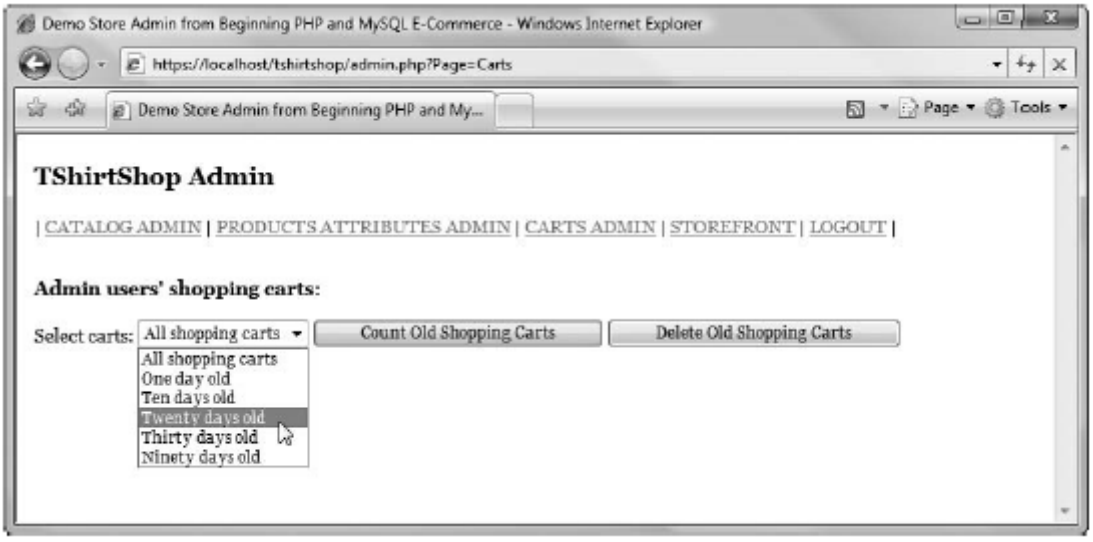
END$$

Tworzenie strony administratora koszyka

Drugi problem z koszykiem polega na tym, że w tej chwili nie istnieje mechanizm usuwania starych rekordów z tabeli shopping_cart. Na bardzo aktywnej stronie internetowej tabela shopping_cart może urosnąć do bardzo dużych rozmiarów. W obecnej wersji kodu identyfikatory koszyka są przechowywane w przeglądarce klienta przez siedem dni. W rezultacie możesz założyć, że wszystkie

koszyki, które nie były aktualizowane w ciągu ostatnich dziesięciu dni, są nieprawidłowe i można je bezpiecznie usunąć.

W poniższym ćwiczeniu szybko zaimplementujesz prostą stronę zarządzania koszykiem na zakupy, na której administrator może zobaczyć, ile istnieje starych wpisów koszyka i może je usunąć, jeśli to konieczne. Rysunek przedstawia tę stronę.



Najciekawszym aspektem, który musisz zrozumieć, jest logika SQL, która usuwa wszystkie koszyki, które nie zostały zaktualizowane przez określony czas. To nie jest takie proste, jak się wydaje – na pierwszy rzut oka możesz pomyśleć, że wystarczy usunąć wszystkie rekordy w koszyku, których data dodania jest starsza niż określona data. Jednak ta strategia nie działa w przypadku koszyków, które są modyfikowane w czasie (powiedzmy, że użytkownik co tydzień dodawał elementy do koszyka w ciągu ostatnich trzech miesięcy). Jeśli ostatnia zmiana w koszyku jest niedawna, żaden z jej elementów nie powinien zostać usunięty, nawet jeśli niektóre są bardzo stare. Innymi słowy, powinieneś albo usunąć wszystkie elementy z koszyka, albo żaden z nich. Wiek koszyka jest określany przez wiek ostatnio zmodyfikowanego lub dodanego produktu. Biorąc to pod uwagę, zaimplementuj nową funkcjonalność, wykonując to ćwiczenie.

Ćwiczenie: Tworzenie strony administratora koszyka

1. Użyj phpMyAdmin, aby wykonać i utworzyć procedury składowane opisane w następnym kroku. Nie zapomnij też ustawić $$ jako separatora przed wykonaniem kodu.

2. Dodaj następujące procedury składowane w warstwie danych do bazy danych tshirtshop:

```
-- Create shopping_cart_count_old_carts stored procedure

CREATE PROCEDURE shopping_cart_count_old_carts(IN inDays INT)

BEGIN

SELECT COUNT(cart_id) AS old_shopping_carts_count

FROM (SELECT cart_id

FROM shopping_cart

GROUP BY cart_id
```

```
HAVING DATE_SUB(NOW(), INTERVAL inDays DAY) >= MAX(added_on))
AS old_carts;
END$$
-- Create shopping_cart_delete_old_carts stored procedure
CREATE PROCEDURE shopping_cart_delete_old_carts(IN inDays INT)
BEGIN
DELETE FROM shopping_cart
WHERE cart_id IN
(SELECT cart_id
FROM (SELECT cart_id
FROM shopping_cart
GROUP BY cart_id
HAVING DATE_SUB(NOW(), INTERVAL inDays DAY) >=
MAX(added_on))
AS sc);
END$$
```

3. Dodaj następującą metodę poziomu biznesowego do business/shopping_cart.php:

```
// Count old shopping carts
public static function CountOldShoppingCarts($days)
{
// Build SQL query
$sql = 'CALL shopping_cart_count_old_carts(:days)';
// Build the parameters array
$params = array (':days' => $days);
// Execute the query and return the results
return DatabaseHandler::GetOne($sql, $params);
}
// Deletes old shopping carts
public static function DeleteOldShoppingCarts($days)
{
// Build SQL query
```

```
$sql = 'CALL shopping_cart_delete_old_carts(:days)';

// Build the parameters array

$params = array (':days' => $days);

// Execute the query

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

}
```

4. Dołącz odniesienie do shopping_cart.php w admin.php:

```
// Load Business Tier

require_once BUSINESS_DIR . 'catalog.php';

require_once BUSINESS_DIR . 'shopping_cart.php';
```

5. Utwórz nowy plik obiektu prezentacji o nazwie Presentation/admin_carts.php i dodaj do niego następujący kod:

```
<?php

// Class that supports cart admin functionality

class AdminCarts

{

// Public variables available in smarty template

public $mMessage;

public $mDaysOptions = array (0 => 'All shopping carts',

1 => 'One day old',

10 => 'Ten days old',

20 => 'Twenty days old',

30 => 'Thirty days old',

90 => 'Ninety days old');

public $mSelectedDaysNumber = 0;

public $mLinkToCartsAdmin;

// Private members

public $_mAction = '';

// Class constructor

public function __construct()

{
```

```
foreach ($_POST as $key => $value)
// If a submit button was clicked ...
if (substr($key, 0, 6) == 'submit')
{
// Get the scope of submit button
$this->_mAction = substr($key, strlen('submit_'), strlen($key));
// Get selected days number
if (isset ($_POST['days']))
$this->mSelectedDaysNumber = (int) $_POST['days'];
else
trigger_error('days value not set');
}
$this->mLinkToCartsAdmin = Link::ToCartsAdmin();
}
public function init()
{
// If counting shopping carts ...
if ($this->_mAction == 'count')
{
$count_old_carts =
ShoppingCart::CountOldShoppingCarts($this->mSelectedDaysNumber);
if ($count_old_carts == 0)
$count_old_carts = 'no';
$this->mMessage = 'There are ' . $count_old_carts .
' old shopping carts (selected option: ' .
$this->mDaysOptions[$this->mSelectedDaysNumber] .
').';
}
// If deleting shopping carts ...
if ($this->_mAction == 'delete')
{
```

```
$this->mDeletedCarts =
ShoppingCart::DeleteOldShoppingCarts($this->mSelectedDaysNumber);
$this->mMessage = 'The old shopping carts were removed from the
database (selected option: ' .
$this->mDaysOptions[$this->mSelectedDaysNumber] .').';
}
}
}
?>
```

6. Utwórz nowy plik w folderze Presentation/templates o nazwie admin_carts.tpl i wpisz następujący kod:

```
{* admin_carts.tpl *}
{load_presentation_object filename="admin_carts" assign="obj"}
<form action="{$obj->mLinkToCartsAdmin}" method="post">
<h3>Admin users&amp;#039; shopping carts:</h3>
{if $obj->mMessage}<p>{$obj->mMessage}</p>{/if}
<p>
Select carts:
{html_options name="days" options=$obj->mDaysOptions
selected=$obj->mSelectedDaysNumber}
<input type="submit" name="submit_count" value="Count Old Shopping Carts" />
<input type="submit" name="submit_delete"
value="Delete Old Shopping Carts" />
</p>
</form>
```

7. Otwórz Presentation/link.php i dodaj następujący kod na końcu klasy Link:

```
// Create link to shopping carts administration page
public static function ToCartsAdmin()
{
return self::ToAdmin('Page=Carts');
}
```

8. Zmodyfikuj prezentację/templates/admin_menu.tpl, dodając następujący podświetlony kod linku do strony administratora koszyków:

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

9. Otwórz Presentation/admin_menu.php i dodaj następujący kod, który utworzy nowy link do menu:

```
<?php
class AdminMenu
{
public $mLinkToStoreAdmin;
public $mLinkToAttributesAdmin;
public $mLinkToCartsAdmin;
public $mLinkToStoreFront;
public $mLinkToLogout;
public function __construct()
{
$this->mLinkToStoreAdmin = Link::ToAdmin();
$this->mLinkToAttributesAdmin = Link::ToAttributesAdmin();
$this->mLinkToCartsAdmin = Link::ToCartsAdmin();
if (isset ($_SESSION['link_to_store_front']))
$this->mLinkToStoreFront = $_SESSION['link_to_store_front'];
```

10. Dodaj podświetlony kod na końcu metody init() klasy StoreAdmin w Presentation/store_admin.php, która ładuje Presentation/templates/admin_carts.tpl:

```
elseif ($admin_page == 'Products')
$this->mContentsCell = 'admin_products.tpl';
elseif ($admin_page == 'ProductDetails')
$this->mContentsCell = 'admin_product_details.tpl';
elseif ($admin_page == 'Carts')
```

```
$this->mContentsCell = 'admin_carts.tpl';
}
}
}
?>
```

Jak to działa: strona administratora koszyka

Ciężka praca strony administratora koszyka na zakupy jest wykonywana przez dwie przechowywane procedury dodane do bazy danych tshirtshop: shopping_cart_count_old_carts i shopping_cart_delete_old_carts. Oboje otrzymują jako parametr liczbę dni, które określają, kiedy koszyk jest stary, i używają tej samej logiki do obliczania starych elementów koszyka, które należy usunąć. Wiek koszyka jest określany jako wiek ostatnio dodanej lub zmienionej pozycji i jest obliczany przy użyciu klauzuli GROUP BY SQL. Warunek, który decyduje o tym, czy koszyk należy uznać za stary:

```
WHERE cart_id IN
(SELECT cart_id
FROM (SELECT cart_id
FROM shopping_cart
GROUP BY cart_id
HAVING DATE_SUB(NOW(), INTERVAL inDays DAY) >=
MAX(added_on))
AS sc);
```

Dwa zagnieżdżone podzapytania są niezbędne do przezwyciężenia ograniczenia MySQL, które nie pozwala na wybór z aktualizowanej tabeli.

Podsumowanie

Dowiedziałeś się, jak przechowywać informacje o koszyku w bazie danych, a także nauczyłeś się kilku innych rzeczy. Prawdopodobnie najciekawszy był sposób, w jaki możesz przechowywać identyfikator koszyka zakupów jako plik cookie na kliencie, ponieważ do tej pory nie zrobiłeś niczego podobnego w tej książce. Po przejściu przez proces tworzenia koszyka, począwszy od bazy danych, a skończywszy na warstwie prezentacji, poruszyliśmy również nowe wyzwania administracyjne. W rozdziale 13 użyjesz technologii o nazwie AJAX, aby uaktualnić funkcjonalność katalogu produktów i koszyka. W rozdziale 14 rozszerzysz koszyk zakupowy, tworząc system kasowy, i zaczniesz przechowywać szczegóły koszyków zakupowych swoich klientów w bazie danych. Umożliwi to zaimplementowanie wymyślnych funkcji, takich jak dynamiczne rekomendacje produktów.