Poznaj zasady wirtualnego handlu i zarabiaj prawdziwe pieniądze

Jak **stworzyć doskonałą witrynę** sklepu internetowego?
Jak **zapewnić maksymalne bezpieczeństwo** Twojej strony?
Jak **prowadzić sprzedaż** i zarządzać stanem magazynowym?

# E-commerce

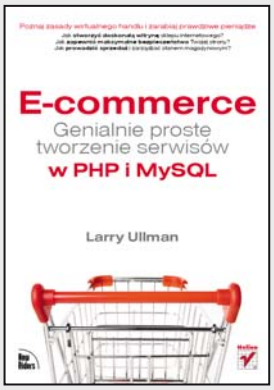
## Genialnie proste tworzenie serwisów w PHP i MySQL

Larry Ullman

New Riders

Helion

# E-commerce. Genialnie proste tworzenie serwisów w PHP i MySQL




Autor: Larry Ullman
Tłumaczenie: Aleksander Lamża
ISBN: 978-83-246-3213-8
Tytuł oryginału: Effortless E-Commerce with PHP and MySQL
Format: 168×237, stron: 400

**Poznaj zasady wirtualnego handlu i zarabiaj prawdziwe pieniądze**

- Jak stworzyć doskonałą witrynę sklepu internetowego?
- Jak zapewnić maksymalne bezpieczeństwo Twojej strony?
- Jak przeprowadzać sprzedaż i zarządzać stanem magazynowym?

Handel elektroniczny to znakomity sposób prowadzenia działalności zarobkowej. Twoje przedsięwzięcie może odnieść prawdziwy sukces – pod warunkiem, że się do niego dobrze przygotujesz. Oto jedna z nielicznych książek na rynku dostarczających szczegółowych informacji na temat tworzenia serwisów e-commerce z wykorzystaniem PHP i MySQL. Bez względu na to, czy tworzysz dynamiczne strony internetowe od lat, czy dopiero od kilku tygodni, z pewnością znajdziesz tu mnóstwo bezcennych informacji.

Dzięki książce E-commerce. Genialnie proste tworzenie serwisów w PHP i MySQL dowiesz się, jak zaprojektować bazę danych, generować katalog produktów, zarządzać koszykiem zakupów, obsługiwać zamówienia i płatności oraz sprawić, by Twój sklep internetowy nie wymagał od Ciebie pracy ponad siły, a jednak był dochodowy. Podręcznik zawiera również omówienie tak istotnych zagadnień, jak zapewnienie bezpieczeństwa witryny oraz zadbanie o interfejs przyjazny użytkownikom, a także informacje dotyczące modułowego programowania, gotowego do dalszej rozbudowy. Przytoczone tu praktyczne przykłady pozwolą Ci spojrzeć na systemy e-commerce z możliwie jak najszerszej perspektywy.

- Wybór technologii internetowych
- Struktura i projekt witryny
- Zarządzanie zawartością witryny
- Tworzenie kont użytkowników
- Łączenie różnych systemów płatniczych
- Sprzedaż wirtualnych produktów
- Tworzenie bezpiecznego środowiska serwera i baz danych
- Tworzenie paneli administracyjnych
- Zasady składania zamówień

**Magia tworzenia profesjonalnych serwisów e-commerce**



Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl

# SPIS TREŚCI

# 9 TWORZENIE KOSZYKA NA ZAKUPY

W poprzednim rozdziale został opracowany katalog produktów zawierający przyciski służące do dodawania wybranych produktów do koszyka na zakupy. Teraz przyszedł czas na zaimplementowanie koszyka. Dobrze zaprojektowany koszyk pozwala klientom wyświetlić wszystkie wybrane przez nich produkty, w łatwy sposób usunąć wskazane pozycje lub zmienić ilość, a także przejść do etapu realizacji zamówienia. Projektowana witryna ma posiadać również funkcję „przechowalni", do której klient może dodać produkty, ale nie musi ich od razu zamawiać.

Główna uwaga w tym rozdziale skupi się na zaimplementowaniu koszyka i przechowalni. Musisz jednak zacząć od zdefiniowania nowych procedur składowanych. Na końcu rozdziału są omówione zagadnienia związane z różnymi możliwościami uwzględnienia kosztów wysyłki oraz inne dodatkowe funkcje, które można zaimplementować.

## DEFINIOWANIE PROCEDUR SKŁADOWANYCH

Ponieważ w projekcie witryny został zastosowany wzorzec MVC, pracę nad koszykiem rozpoczniemy od zaprojektowania **modelu**, czyli bazy danych. W tym rozdziale zostanie opisanych osiem nowych procedur składowanych: cztery dla koszyka na zakupy i cztery dla przechowalni. Tylko dwie z nich są skomplikowane porównywalnie do tych opisanych w rozdziale 8., „Tworzenie katalogu", jednak tym razem w procedurach znajdzie się więcej tzw. logiki. Ponieważ tabele `carts` i `wish_lists` mają taką samą strukturę i będą wykorzystywane w ten sam sposób, opiszę tylko procedury dla koszyka. Aby utworzyć procedury dla przechowalni, wystarczy zmienić nazwę tabeli `cart` na `wish_list`.

# Dodawanie produktów

### Wskazówka

Sposób tworzenia procedur składowanych w bazie danych jest opisany w rozdziale 8.

Procedura składowana `add_to_cart()` będzie wywoływana po żądaniu zgłoszonym przez klienta wkładającego produkt do koszyka. Procedurze trzeba przekazać cztery argumenty: unikalny identyfikator użytkownika, typ produktu (*kawa* lub *inne*) i jego identyfikator, a także ilość. Dodatkowo, jeśli w koszyku nie ma jeszcze danego produktu, procedura powinna go **dodać**, natomiast jeśli już jest, powinna tylko **zwiększyć jego ilość**. Poniżej znajduje się kod procedury i jego objaśnienie:

```
DELIMITER $$
CREATE PROCEDURE add_to_cart (uid CHAR(32), type VARCHAR(6), pid MEDIUMINT,
↪qty TINYINT)
BEGIN
   DECLARE cid INT;
   SELECT id INTO cid FROM carts WHERE user_session_id=uid AND product_type=type
   ↪AND product_id=pid;
   IF cid > 0 THEN
      UPDATE carts SET quantity=quantity+qty, date_modified=NOW( ) WHERE id=cid;
   ELSE
      INSERT INTO carts (user_session_id, product_type, product_id, quantity)
      ↪VALUES (uid, type, pid, qty);
   END IF;
END$$
DELIMITER ;
```

### Wskazówka

Pełny kod SQL i pozostałe pliki możesz pobrać ze strony *www.DMCInsights.com/ecom/*.

Najpierw, tak jak to już robiliśmy, zmieniamy symbol zakończenia definicji na dwa znaki dolara (`$$`). Następnie jest definiowana **sygnatura** procedury, czyli jej nazwa i lista argumentów. Pierwszy argument (`uid`) to identyfikator sesji użytkownika, który będzie się składał z dokładnie 32 znaków. Kolejny (`type`) to typ produktu, który może mieć długość do sześciu znaków (w naszym przypadku nazwy obu typów mają po cztery znaki: *kawa* i *inne*). Dwa ostatnie argumenty to identyfikator produktu (`pid`) oraz ilość (`qty`).

### Wskazówka

Symbol zakończenia definicji procedury musi być zmieniony na coś innego niż znak średnika.

Aby w procedurze sprawdzić, czy dany produkt znajduje się już w koszyku, konieczne jest wprowadzenie dodatkowej, wewnętrznej zmiennej. Można ją zadeklarować za pomocą instrukcji `DECLARE`, po której

następuje nazwa zmiennej i typ danych MySQL. W kodzie jest tworzona zmienna `cid` (skrót od *cart ID*, czyli identyfikator koszyka).

### Uwaga

Zmienne muszą być deklarowane bezpośrednio pod instrukcją `BEGIN`.

Kwerenda `SELECT` poszukuje w tabeli `carts` określonego produktu. Jeśli go znajdzie, zapisuje jego identyfikator (`id`) do zmiennej `cid`, za co odpowiada konstrukcja `SELECT...INTO`.

Po kwerendzie `SELECT` znajduje się instrukcja warunkowa `IF-ELSE`, która — w zależności od wartości zmiennej `cid` — wykonuje zapytanie `UPDATE` lub `INSERT`. Wartość `cid` większa od zera oznacza, że produkt już jest w koszyku, więc trzeba jedynie zaktualizować ilość. W przeciwnym przypadku do tabeli jest wstawiany nowy rekord. Kolumna `date_modified` jest uaktualniana tylko w przypadku wykonania zapytania `UPDATE`.

# Usuwanie produktów

### Wskazówka

Argumenty przekazywane do procedur składowanych mogą być wykorzystywane w kwerendach, tak samo jak w przypadku predefiniowanych zapytań, co oznacza, że łańcuchy tekstowe nie mogą być umieszczane w cudzysłowach.

Procedura składowana usuwająca produkt z koszyka jest najprostszą z wszystkich czterech opisanych w tym rozdziale. Wymaga przekazania trzech z czterech argumentów zdefiniowanych w procedurze `add_to_cart()` (z oczywistych względów podczas usuwania produktu nie jest wymagane podanie jego ilości). Procedura wykonuje tylko jedną kwerendę `DELETE`:

```
DELIMITER $$
CREATE PROCEDURE remove_from_cart (uid CHAR(32), type VARCHAR(6), pid MEDIUMINT)
BEGIN
   DELETE FROM carts WHERE user_session_id=uid AND product_type=type AND
   ↪product_id=pid;
END$$
DELIMITER ;
```

# Aktualizowanie koszyka

### Wskazówka

Są dwie możliwości usunięcia produktu z koszyka: kliknięcie linku usuwającego lub wpisanie zera w polu ilości.

Procedura składowana aktualizująca koszyk będzie wywoływana po kliknięciu przez użytkownika przycisku *Aktualizuj koszyk* znajdującego się na stronie koszyka (rysunek 9.1). Procedura przyjmuje te same cztery parametry co `add_to_cart()`, ale nie musi sprawdzać, czy produkt już istnieje w tabeli (ponieważ produkt musi znajdować się w bazie, by był wyświetlony w koszyku). Jeśli jednak użytkownik w polu ilości wpisze zero, procedura powinna usunąć taki produkt. Zamiast umieszczać w tej procedurze kod usuwający produkt, wywołamy procedurę `remove_from_cart()` realizującą to zadanie.


**Rysunek 9.1**

```
DELIMITER $$
CREATE PROCEDURE update_cart (uid CHAR(32), type VARCHAR(6), pid MEDIUMINT,
↪qty TINYINT)
BEGIN
   IF qty > 0 THEN
      UPDATE carts SET quantity=qty, date_modified=NOW( ) WHERE user_session_id=uid
      ↪AND product_type=type AND product_id=pid;
   ELSEIF qty = 0 THEN
      CALL remove_from_cart (uid, type, pid);
   END IF;
END$$
DELIMITER ;
```

### Wskazówka

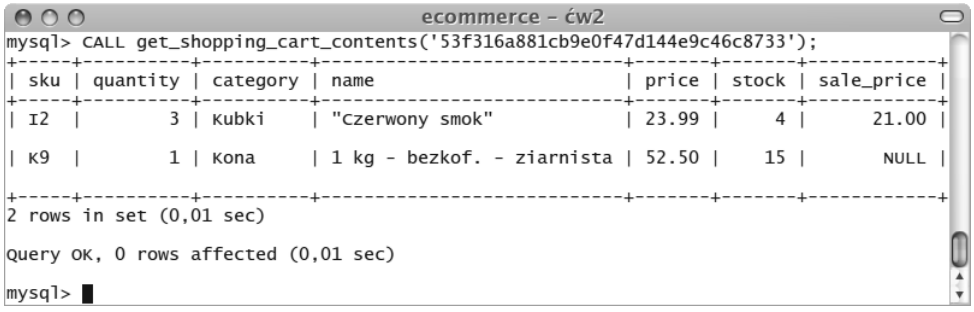
Za każdym razem, gdy w tabelach `carts` lub `wish_lists` zmieni się ilość jakiegoś produktu (ale nie zostanie on usunięty), zaktualizowana zostanie również data modyfikacji (`date_modified`).

## Pobieranie zawartości koszyka

Czwarta i ostatnia procedura składowana (dla koszyka) wykonuje kwerendę `SELECT` pobierającą całą zawartość koszyka. Stopnień skomplikowania kwerendy zależy od tego, jak dużo informacji chcesz uzyskać, jednak jej podstawowa budowa przypomina kwerendy związane z promocjami, opisane w rozdziale 8. Zapytanie łączy dwie kwerendy `SELECT` za pomocą instrukcji `UNION`: pierwsza to złączenie `JOIN` czterech tabel, a druga — aż pięciu. W poniższym kodzie kwerenda jest przedstawiona w czytelny sposób z podziałem na wiersze. Procedura wymaga podania jedynie jednego argumentu, którym jest identyfikator sesji użytkownika.

```
DELIMITER $$
CREATE PROCEDURE get_shopping_cart_contents (uid CHAR(32))
BEGIN
   SELECT CONCAT("I", ncp.id) AS sku, c.quantity, ncc.category,
ncp.name, ncp.price, ncp.stock, sales.price AS sale_price
FROM carts AS c
INNER JOIN non_coffee_products AS ncp ON c.product_id=ncp.id
INNER JOIN non_coffee_categories AS ncc ON ncc.id=ncp.non_coffee_category_id
LEFT OUTER JOIN sales ON
(sales.product_id=ncp.id AND sales.product_type='inne' AND
((NOW( ) BETWEEN sales.start_date AND sales.end_date) OR (NOW( ) > sales.start_date
↪AND sales.end_date IS NULL)) )
WHERE c.product_type="inne" AND c.user_session_id=uid
   UNION
   SELECT CONCAT("K", sc.id), c.quantity, gc.category,
CONCAT_WS(" - ", s.size, sc.caf_decaf, sc.ground_whole), sc.price, sc.stock,
↪sales.price
FROM carts AS c
INNER JOIN specific_coffees AS sc ON c.product_id=sc.id
INNER JOIN sizes AS s ON s.id=sc.size_id
INNER JOIN general_coffees AS gc ON gc.id=sc.general_coffee_id
LEFT OUTER JOIN sales ON
(sales.product_id=sc.id AND sales.product_type='kawa' AND
((NOW( ) BETWEEN sales.start_date AND sales.end_date) OR (NOW( ) > sales.start_date
↪AND sales.end_date IS NULL)) )
WHERE c.product_type="kawa" AND c.user_session_id=uid;
END$$
DELIMITER ;
```

Na rysunku 9.2 widać efekt wywołania procedury. Zwróć uwagę, że kwerenda zwraca domyślną cenę (w kolumnie *price*), stan magazynowy (*stock*) i cenę promocyjną (*sale_price*), jeżeli obowiązuje dla danego produktu. Te informacje będą wykorzystane później.




**Rysunek 9.2**

# DEFINIOWANIE FUNKCJI POMOCNICZYCH

Zanim przejdziemy do tworzenia koszyka na zakupy, przygotujemy dwie funkcje pomocnicze, które będą potrzebne w skrypcie koszyka. Pierwsza funkcja przyjmuje dwie ceny — normalną oraz promocyjną — i zwraca tę, która powinna być uwzględniona w koszyku:

```
function get_just_price($regular, $sales) {
   if ((0 < $sales) && ($sales < $regular)) {
      return number_format($sales, 2);
   } else {
      return number_format($regular, 2);
   }
}
```

Podobny kod znajduje się w funkcji `get_price()` zdefiniowanej w rozdziale 8., ale tam jest zwracana wartość liczbowa ceny (bez żadnego kontekstu).

Druga funkcja przetwarza numer SKU na dwie wartości, np. `K12` zostanie zamienione na `kawa` i `12`. Ta operacja będzie często potrzebna, ponieważ w bazie danych każdy produkt jest opisany typem i identyfikatorem. Funkcja ta przyjmuje jeden argument — numer SKU — i zwraca dwuelementową tablicę:

```
function parse_sku($sku) {

   // Wydziel pierwszy znak:
   $type_abbr = substr($sku, 0, 1);

   // Wydziel pozostałe znaki:
   $pid = substr($sku, 1);

   // Sprawdź typ:
   if ($type_abbr == 'K') {
      $sp_type = 'kawa';
   } elseif ($type_abbr == 'I') {
      $sp_type = 'inne';
```

```
    } else {
       $sp_type = NULL;
    }

    // Sprawdź identyfikator produktu:
    $pid = (filter_var($pid, FILTER_VALIDATE_INT, array('min_range' => 1))) ?
    ➥$pid : NULL;

    // Zwróć wartości:
    return array($sp_type, $pid);
```

} *// Koniec funkcji parse_sku().*

Numer SKU jest dzielony na części za pomocą dwóch funkcji `substr()`. Następnie na podstawie pierwszego znaku numeru SKU (który może przyjąć dwie wartości: `K` dla kawy i `I` dla innych produktów) jest ustalany typ (zmienna `$sp_type`). Jeżeli nie jest spełniony żaden warunek, zmiennej `$sp_type` jest przypisywana wartość `NULL`.

Druga część numeru SKU musi być liczbą całkowitą co najmniej równą 1. Do sprawdzenia jej poprawności służy funkcja `filter_var()`. Jeżeli liczba jest prawidłowa, jest przypisywana zmiennej `$pid`, a w przeciwnym przypadku przyjmuje wartość `NULL`.

Na koniec obie wartości są zwracane jako tablica, więc w kodzie wywołującym tę funkcję trzeba zastosować instrukcję `list()`:

```
list($type, $id) = parse_sku($sku);
```

Funkcja `list()` przypisuje zmiennym `$type` i `$id` wartości z tablicy zwróconej przez wyrażenie z prawej strony przypisania, czyli funkcję `parse_sku()`.

Najbardziej odpowiednim miejscem dla tych dwóch funkcji jest skrypt *product_functions.inc.php* znajdujący się w katalogu *includes*. Przed przejściem do kolejnego etapu koniecznie umieść tam te funkcje.

# BUDOWANIE KOSZYKA NA ZAKUPY

Po zdefiniowaniu procedur składowanych i funkcji pomocniczych musisz się zająć utworzeniem trzech plików:

- *cart.php*, który wykonuje wszystkie niezbędne operacje (jest kontrolerem),
- *cart.html*, który jest plikiem widoku dla koszyka,
- *emptycart.html*, który jest plikiem widoku dla pustego koszyka.

Utworzysz teraz te pliki, rozpoczynając od skryptu PHP.

## Tworzenie skryptu PHP

Skrypty PHP opracowane w rozdziale 8., dzięki zastosowaniu procedur składowanych i dołączanych plików HTML, były bardzo krótkie i stosunkowo „czyste”. Cały plik *cart.php* składa się tylko z 50 wierszy kodu,

włączając w to komentarze i puste wiersze! Większość kodu odpowiada za uruchomienie odpowiednich procedur składowanych w zależności od sposobu wywołania skryptu.

1. W edytorze lub IDE utwórz nowy skrypt PHP i zapisz go w głównym katalogu serwera WWW pod nazwą `cart.php`.
2. Dołącz plik konfiguracyjny:

```
<?php
require ('./includes/config.inc.php');
```

3. Sprawdź, czy jest już ustawiony identyfikator sesji użytkownika, a jeżeli nie — utwórz go.

```
if (isset($_COOKIE['SESSION'] )) {
   $uid = $_COOKIE['SESSION'];
} else {
   $uid = md5(uniqid('biped',true));
}
```

Ta witryna korzysta tylko z jednego ciasteczka do obsługi zarówno koszyka, jak i przechowalni. Ciasteczko ma nazwę *SESSION*. Chociaż nie są tu stosowane prawdziwe sesje PHP, nazwa wskazuje przeznaczenie ciasteczka.

Jeśli ciasteczko istnieje, jego wartość jest przypisywana zmiennej `$uid`. W przeciwnym przypadku musi zostać utworzony nowy identyfikator sesji. Służy do tego kombinacja funkcji `uniqid()` i `md5()`.

### Wskazówka

Funkcja `md5()` zwraca łańcuch tekstowy o długości 32 znaków.

4. Wyślij ciasteczko:

```
setcookie('SESSION', $uid, time()+(60*60*24*30));
```

Gdy użytkownik powraca na stronę albo pojawia się na niej pierwszy raz, ciasteczko musi zostać przesłane. Dla nowych użytkowników jest to oczywiste, natomiast w przypadku powracających klientów chodzi o zaktualizowanie ciastka, tak by było dłużej ważne. Przy przedstawionych ustawieniach termin ważności ciasteczka mija po 30 dniach od daty utworzenia.

### Uwaga

Ciasteczka muszą zostać przesłane do przeglądarki internetowej jako pierwsze.

5. Dołącz plik nagłówka:

```
$page_title = 'Kawy świata — koszyk na zakupy';
include ('./includes/header.html');
```

6. Dołącz skrypty odpowiedzialne za połączenie z bazą danych oraz funkcje pomocnicze:

```
require (MYSQL);
include ('./includes/product_functions.inc.php');
```

7. Jeśli w adresie URL znajduje się wartość SKU, podziel ją na części:

```
if (isset($_GET['sku'] )) {
   list($sp_type, $pid) = parse_sku($_GET['sku'] );
}
```

Dzięki wywołaniu funkcji `parse_sku()` numer SKU, który może być przekazany w adresie URL, zostanie podzielony na dwie części: typ produktu oraz jego identyfikator.

8. Sprawdź, czy produkt ma być dodany do koszyka:

```
if (isset ($sp_type, $pid, $_GET['action'] ) && ($_GET['action'] =='add') ) {
   $r = mysqli_query($dbc, "CALL add_to_cart('$uid', '$sp_type', $pid, 1)");
```

### Wskazówka

Za pomocą funkcji `isset()` można za jednym razem sprawdzić istnienie wielu zmiennych.

Cała logika tego skryptu opiera się na długiej instrukcji warunkowej `IF-ELSEIF`, sprawdzającej różne możliwości wywołania skryptu. Pierwszą możliwością jest kliknięcie przycisku *Do koszyka*, co powoduje utworzenie adresu URL o postaci np. *http://**nazwahosta**/cart.php?sku=C8&action=add*. W tym przypadku numer SKU zostanie podzielony na wartości `$sp_type` i `$pid`, a zmienna `$_GET['action']` będzie równa `add`.

Jeśli wszystkie warunki są spełnione, jest wywoływana procedura składowana `add_to_cart()` i przekazywany jest jej identyfikator użytkownika (`$uid`), typ (`$sp_type`) i identyfikator produktu (`$pid`) oraz ilość wynosząca `1`.

### Wskazówka

Jeśli chcesz umożliwić klientom dodawanie do koszyka większych ilości produktu za jednym razem, musiałbyś pobrać tę wartość z adresu URL, sprawdzić ją i przekazać procedurze `add_to_cart()`.

9. Sprawdź, czy produkt ma zostać usunięty z koszyka:

```
} elseif (isset ($sp_type, $pid, $_GET['action'] ) && ($_GET['action']
↪=='remove') ) {
   $r = mysqli_query($dbc, "CALL remove_from_cart('$uid', '$sp_type', $pid)");
```

Warunek sprawdzany za pomocą funkcji `isset()` jest podobny do poprzedniego (dla dodawania produktów). Jeśli wartość `$_GET['action']` jest równa `remove`, wywoływana jest procedura składowana `remove_from_cart()`. Do takiej sytuacji dojdzie po kliknięciu przez użytkownika linku *Usuń z koszyka* (patrz rysunek 9.1).

10. Sprawdź, czy produkt ma zostać przeniesiony do koszyka:

```
} elseif (isset ($sp_type, $pid, $_GET['action'], $_GET['qty'] ) &&
↪($_GET['action'] == 'move') ) {
   $qty = (filter_var($_GET['qty'], FILTER_VALIDATE_INT, array('min_range' => 1)))
   ↪? $_GET['qty'] : 1;
```

```
$r = mysqli_query($dbc, "CALL add_to_cart('$uid', '$sp_type', $pid, $qty)");
$r = mysqli_query($dbc, "CALL remove_from_wish_list('$uid', '$sp_type', $pid)");
```

Klient ma możliwość przenoszenia produktów między koszykiem i przechowalnią. Jeśli coś znajduje się w przechowalni i jest przenoszone do koszyka, wykonywana jest podobna operacja do dodawania do koszyka. Między tymi działaniami są jednak dwie istotne różnice. Po pierwsze, brana jest pod uwagę również liczba produktów, czyli jeśli w przechowalni były np. trzy kubki, do koszyka zostaną przeniesione wszystkie trzy. Po drugie, po przeniesieniu produktu do koszyka, musi on zostać usunięty z przechowalni.

Aby to wszystko zadziałało, w instrukcji warunkowej jest sprawdzane istnienie wartości `$_GET['qty']` oraz to, czy `$_GET['action']` jest równe `move`. Następnie jest sprawdzana poprawność wartości ilości (liczba całkowita nie mniejsza niż 1).

Wyniki wszystkich procedur składowanych wywoływanych w skrypcie są przypisywane zmiennej `$r`, mimo że tak naprawdę jest używany wynik tylko jednej z nich (mowa o procedurze zwracającej zawartość koszyka). Mimo to lepiej przypisywać wyniki do zmiennej, ponieważ ułatwia to debugowanie.

Jeśli w czasie pracy nad skryptem pojawią się problemy, po wywołaniu procedury możesz wpisać następujący kod:

```
if (!$r) echo mysqli_error($dbc);
```

**11.** Sprawdź, czy został przesłany formularz:

```
} elseif (isset($_POST['quantity'] )) {
```

Wszystkie poprzednie warunki korzystały z żądań typu GET, natomiast ta strona może być również wywołana za pomocą żądania POST. Do tego typu sytuacji dojdzie, gdy użytkownik zmieni ilość produktów w koszyku i prześle formularz koszyka.

**12.** Przejdź w pętli przez wszystkie przesłane pozycje koszyka:

```
foreach ($_POST['quantity'] as $sku => $qty) {
   list($sp_type, $pid) = parse_sku($sku);
   if (isset($sp_type, $pid)) {
      $qty = (filter_var($qty, FILTER_VALIDATE_INT, array('min_range'=> 0))) ?
      ↪$qty : 1;
      $r = mysqli_query($dbc, "CALL update_cart('$uid', '$sp_type', $pid, $qty)");
   }
}
```

W `$_POST['quantity']` znajduje się tablica elementów w postaci *`SKU => ilość`*. Dla każdego elementu w `$_POST['quantity']` musi zostać zaktualizowany odpowiedni produkt w koszyku. W tym celu najpierw jest przetwarzany i sprawdzany numer SKU, a następnie sprawdzana jest ilość. Wartość ta zostanie użyta, jeśli jest to liczba całkowita równa co najmniej `0` (ponieważ klient może wpisać zero, by usunąć produkt z koszyka). Jeśli z jakiegoś powodu klient wprowadzi nieprawidłową wartość, zostanie zastosowana domyślna, wynosząca 1. Na końcu jest wywoływana procedura składowana `update_cart()` z prawidłowymi argumentami.

**13.** Zakończ główną instrukcję warunkową i pobierz zawartość koszyka:

```
} // Koniec głównej instrukcji warunkowej IF.
$r = mysqli_query($dbc, "CALL get_shopping_cart_contents('$uid')");
```

Bez względu na wykonaną akcję zawsze jest wyświetlana aktualna zawartość koszyka.

**14.** Dołącz właściwy plik widoku:

```
if (mysqli_num_rows($r) > 0) {
   include ('./views/cart.html');
} else { // Pusty koszyk!
   include ('./views/emptycart.html');
}
```

**15.** Zakończ kod strony:

```
include ('./includes/footer.html');
```

**16.** Zapisz plik.

# Tworzenie plików widoku

## Uwaga

Oba pliki widoków muszą być oczywiście zapisane w katalogu *views*.

Koszyk na zakupy używa dwóch plików widoku: jeden służy do wyświetlania produktów, a drugi informuje o tym, że koszyk jest pusty. Ten ostatni — *emptycart.html* — jest bardzo prosty (rysunek 9.3):

```
<!-- początek bloku -->
<div class="box alt"><div class="left-top-corner"><div class="right-top-corner">
➥<div class="border-top"></div></div></div><div class="border-left">
➥<div class="border-right"><div class="inner">
<h2>Twój koszyk</h2>
<p>Twój koszyk jest pusty.</p>
</div></div></div><div class="left-bot-corner"><div class="right-bot-corner">
➥<div class="border-bot"></div></div></div></div>
<!-- koniec bloku -->
```


**Rysunek 9.3**

Drugi plik widoku jest trochę bardziej skomplikowany. Musi wyświetlać wszystkie produkty zwrócone przez procedurę składowaną, czyli produkty dodane do koszyka. Jednak lista musi być formularzem HTML, tak by użytkownik mógł zaktualizować ilość. Ponadto każdy produkt powinien być uzupełniony linkami do funkcji usuwania z koszyka i przenoszenia do przechowalni. Poza tym muszą być obliczane i wyświetlane informacje o wartości poszczególnych pozycji koszyka oraz o wartości całego koszyka (na rysunku 9.4 jest przedstawiona początkowa wersja tego widoku, a końcową można zobaczyć na rysunku 9.1).


**Rysunek 9.4**

1. W edytorze lub IDE utwórz nowy plik HTML i zapisz go w katalogu *views* pod nazwą `cart.html`.
2. Rozpocznij blok HTML i nagłówek:

```
<div class="box alt"><div class="left-top-corner"><div class="right-top-corner">
<div class="border-top"></div></div></div><div class="border-left">
↪<div class="border-right"><div class="inner">
<h2>Twój koszyk</h2>
<p>Użyj tego formularza do zmiany zawartości koszyka. Możesz zmienić ilości,
↪całkowicie usunąć produkty z koszyka albo przenieść je do przechowalni,
↪by zamówić je później. Koszt dostawy i obsługi zamówienia jest podany nad
↪wartością koszyka. Jeśli chcesz złożyć zamówienie, kliknij przycisk Złóż
↪zamówienie. Zostaniesz przeniesiony do bezpiecznej strony, na której
↪sfinalizujesz zakupy.</p>
```

   Przygotowując tekst na stronę koszyka, musisz go odpowiednio wyważyć, by zawierał najważniejsze informacje, ale nie był zbyt długi. Pamiętaj, że podstawową funkcją strony koszyka na zakupy jest skłonienie klienta do złożenia zamówienia!

3. Rozpocznij kod formularza:

```
<form action="/cart.php" method="POST">
```

   Formularz odsyła dane z powrotem do skryptu *cart.php* za pomocą metody POST. Wartość atrybutu `action` rozpoczyna się od ukośnika, by wskazać, że skrypt *cart.php* znajduje się w głównym katalogu serwera WWW. Akurat w tym przypadku nie jest on niezbędny, ale w wielu innych tak, więc zastosowałem go w celu zachowania spójności i konsekwencji.

4. Rozpocznij definicję tabeli:

```
<table border="0" cellspacing="8" cellpadding="6">
   <tr>
```

```
      <th align="center">Produkt</th>
      <th align="center">Ilość</th>
      <th align="right">Cena</th>
      <th align="right">Wartość</th>
      <th align="center">Opcje</th>
   </tr>
```

Do wyświetlenia zawartości koszyka stosuję stary, sprawdzony sposób z tabelą. W tym przypadku potrzebnych jest pięć kolumn.

**5.** Rozpocznij blok kodu PHP:

```
<?php
$total = 0;
```

W bloku kodu PHP zmienna `$total` jest inicjowana wartością `0`, tak by całkowita wartość koszyka została prawidłowo obliczona.

**6.** Pobierz każdą pozycję koszyka:

```
while ($row = mysqli_fetch_array($r, MYSQLI_ASSOC)) {
   $price = get_just_price($row['price'], $row['sale_price'] );
   $subtotal = $price * $row['quantity'];
```

### Wskazówka

Ze względów bezpieczeństwa cena jest zawsze pobierana z bazy danych, tak by klient nie mógł jej w żaden sposób zmienić.

W pętli najpierw jest określana obowiązująca cena, za co odpowiada funkcja `get_just_price()`, której przekazuje się normalną i promocyjną cenę produktu. Następnie jest obliczana wartość danej pozycji koszyka jako iloczyn ceny i ilości.

**7.** Wyświetl wiersz tabeli:

```
echo '<tr>
<td>' . $row['category'] . '::' . $row['name'] . '</td>
<td align="center"><input type="text" name="quantity[' . $row['sku'] . ']"
➥value="' . $row['quantity'] . '" size="2" /></td>
<td align="right">' . $price . ' zł</td>
<td align="right">' . number_format($subtotal, 2) . ' zł</td>
<td align="right"><a href="/wishlist.php?sku=' . $row['sku'] . '&action=move&qty=' .
➥$row['quantity'] .'">Przenieś do przechowalni</a><br /><a href="/cart.php?sku=' .
➥$row['sku'] . '&action=remove">Usuń z koszyka</a></td>
</tr>
';
```

Dla każdej pozycji koszyka jest tworzony osobny wiersz tabeli. W pierwszej kolumnie jest wyświetlana nazwa, która składa się z nazwy kategorii i produktu, tak samo, jak było na stronach z produktami. W drugiej kolumnie znajduje się ilość, przy czym jest to wartość, którą można edytować, więc jest wyświetlana w polu formularza. Nazwa każdego pola jest tworzona według schematu *`quantity[sku]`*, więc po wysłaniu formularza będzie dostępna wartość zarówno SKU, jak i nowo ustawionej ilości (rysunek 9.5).

```
<tr><td>Kubki::"Czerwony smok"</td>
  <td align="center"><input type="text" name="quantity[I2]" value="1" size="2" class="small" /></td>
  <td align="right">21.00 zł</td>
  <td align="right">21.00 zł </td>
  <td align="right"><a href="/wishlist.php?sku=I2&action=move&qty=1">Przenieś do przechowalni</a><br />
  <a href="/cart.php?sku=I2&action=remove">Usuń z koszyka</a></td>
</tr>

<tr><td>Kona::1 kg - bezkof. - ziarnista</td>
  <td align="center"><input type="text" name="quantity[K9]" value="1" size="2" class="small" /></td>
  <td align="right">52.50 zł</td>
  <td align="right">52.50 zł </td>
  <td align="right"><a href="/wishlist.php?sku=K9&action=move&qty=1">Przenieś do przechowalni</a><br />
  <a href="/cart.php?sku=K9&action=remove">Usuń z koszyka</a></td>
</tr>
```

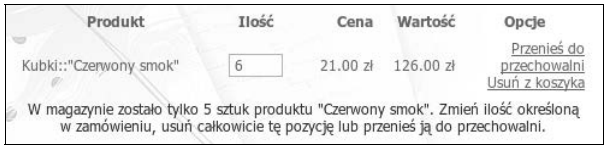
**Rysunek 9.5**

W dwóch kolejnych kolumnach jest wyświetlana cena i wartość. W piątej kolumnie są umieszczone dwa linki. Pierwszy służy do przenoszenia pozycji koszyka do przechowalni i przekazuje skryptowi *wishlist.php* wartość SKU, bieżącą ilość i akcję `move`. Drugi link prowadzi do tego samego skryptu (*cart.php*) i powoduje usunięcie pozycji z koszyka.

8. Dodaj komunikat o błędzie, jeśli w magazynie nie ma wystarczającej ilości produktu:

```
if ($row['stock'] < $row['quantity'] ) {
   echo '<tr class="error"><td colspan="5" align="center">W magazynie zostało
   ↪tylko ' . $row['stock'] . ' sztuk produktu ' . $row['name'] . '. Zmień ilość
   ↪określoną w zamówieniu, usuń całkowicie tę pozycję lub przenieś ją do
   ↪przechowalni.</td></tr>';
}
```

Jeśli klient chce zamówić więcej sztuk jakiegoś produktu, niż aktualnie jest w magazynie, musi zostać powiadomiony o problemie (rysunek 9.6). W takiej sytuacji klient poznaje stan magazynowy i jest proszony o samodzielne rozwiązanie problemu. W kolejnym rozdziale zmodyfikujesz kod, tak by produkty, których jest za mało, były automatycznie usuwane z zamówienia.




**Rysunek 9.6**

9. Do zmiennej `$total` dodaj wartość pozycji (`$subtotal`) i zakończ pętlę:

```
   $total += $subtotal;
} // Koniec pętli WHILE
```

10. W tabeli wyświetl całkowitą wartość koszyka (`$total`) i zakończ blok kodu PHP:

```
echo '<tr>
   <td colspan="3" align="right"><strong>Wartość</strong></td>
   <td align="right">$' . number_format($total, 2) . '</td>
   <td> </td>
</tr>
';
?>
```

**11.** Zakończ definicję tabeli i utwórz dwa przyciski:

```
</table><br /><p align="center">
<input type="submit" value="Aktualizuj koszyk" class="button" /></form></p><br />
<p align="center"><a href="https://<?php echo BASE_URL; ?>"checkout.php?session=
➥<?php echo $uid; ?> class="button">Złóż zamówienie</a></p></div>
```

Pierwszy przycisk służy do przesłania formularza (w celu zaktualizowania ilości). Kliknięcie drugiego przycisku powoduje rozpoczęcie procesu finalizowania zamówienia. Link ten kieruje do skryptu *checkout.php* poprzez bezpieczne połączenie HTTPS. Aby wygenerować adres, jest wymagana wartość stałej `BASE_URL`. Zwróć uwagę, że w adresie jest przekazywany identyfikator sesji użytkownika, tak więc w skrypcie *checkout.php* będzie można z niego skorzystać. O tym, że jest to niezbędne, przekonasz się w kolejnym rozdziale.

### Uwaga

Proces finalizowania zamówienia musi się odbywać na bezpiecznych stronach, tak by klient czuł się bezpiecznie!

**12.** Zakończ kod strony:

```
</div></div><div class="left-bot-corner"><div class="right-bot-corner">
➥<div class="border-bot"></div></div></div></div>
<!-- koniec bloku -->
```

**13.** Zapisz plik i przetestuj go w przeglądarce internetowej.

Cała funkcjonalność witryny, z wyjątkiem przechowalni, powinna już działać, więc możesz ją przetestować. W sklepie możesz już dodawać produkty do koszyka, a na stronie koszyka możesz aktualizować ilość produktów i je usuwać.

# BUDOWANIE PRZECHOWALNI

Procedury składowane używane w zarządzaniu przechowalnią są w zasadzie identyczne z tymi, które są stosowane w koszyku na zakupy. To samo dotyczy skryptu PHP i plików HTML — będą prawie wiernymi kopiami już napisanych. Przedstawię je tu w całości, więc jeśli chcesz poznać szczegóły, wróć do opisu ich odpowiedników dla koszyka na zakupy.

## Tworzenie skryptu PHP

Skrypt *wishlist.php* zapisany w głównym katalogu serwera WWW różni się od *cart.php* tylko czterema elementami:

- tytułem strony,
- nazwą wywoływanej procedury składowanej,
- innymi plikami widoków,
- brakiem instrukcji warunkowej sprawdzającej dodawanie produktów.

Tak naprawdę, poza ostatnią różnicą, skrypt dla przechowalni możesz przygotować metodą „wyszukaj i zamień”, traktując jako bazę kod skryptu *cart.php*. Ostatnia różnica polega na tym, że w tym skrypcie produkty są dodawane do przechowalni przez przeniesienie ich z koszyka na zakupy.

```
<?php // wishlist.php
require ('./includes/config.inc.php');
if (isset($_COOKIE['SESSION'] )) {
   $uid = $_COOKIE['SESSION'];
} else {
   $uid = md5(uniqid('biped',true));
}
setcookie('SESSION', $uid, time( )+(60*60*24*30));
$page_title = 'Kawy świata - przechowalnia';
include ('./includes/header.html');
require (MYSQL);
include ('./includes/product_functions.inc.php');
if (isset($_GET['sku'] )) {
   list($sp_type, $pid) = parse_sku($_GET['sku'] );
}
if (isset ($sp_type, $pid, $_GET['action'] ) && ($_GET['action'] == 'remove')) {
   $r = mysqli_query($dbc, "CALL remove_from_wish_list('$uid', '$sp_type', $pid)");
} elseif (isset ($sp_type, $pid, $_GET['action'], $_GET['qty'] ) && ($_GET['action']
↪== 'move') ) {
   $qty = (filter_var($_GET['qty'], FILTER_VALIDATE_INT, array('min_range' => 1)))
   ↪? $_GET['qty'] : 1;
   $r = mysqli_query($dbc, "CALL add_to_wish_list('$uid', '$sp_type', $pid, $qty)");
   $r = mysqli_query($dbc, "CALL remove_from_cart('$uid', '$sp_type', $pid)");
} elseif (isset($_POST['quantity'] )) {
   foreach ($_POST['quantity'] as $sku => $qty) {
      list($sp_type, $pid) = parse_sku($sku);
      if (isset($sp_type, $pid)) {
         $qty = (filter_var($qty, FILTER_VALIDATE_INT, array('min_range' => 0))) ?
         ↪$qty : 1;
         $r = mysqli_query($dbc, "CALL update_wish_list('$uid', '$sp_type', $pid,
         ↪$qty)");
      }
   }
}
$r = mysqli_query($dbc, "CALL get_wish_list_contents('$uid')");
if (mysqli_num_rows($r) > 0) {
   include ('./views/wishlist.html');
} else {
   include ('./views/emptylist.html');
}
include ('./includes/footer.html');
?>
```

# Tworzenie plików widoku

## Uwaga

Oczywiście oba pliki widoku muszą być zapisane w katalogu *views*.

Gdy przechowalnia klienta jest pusta, używany jest plik widoku *emptylist.html* (rysunek 9.7).


**Rysunek 9.7**

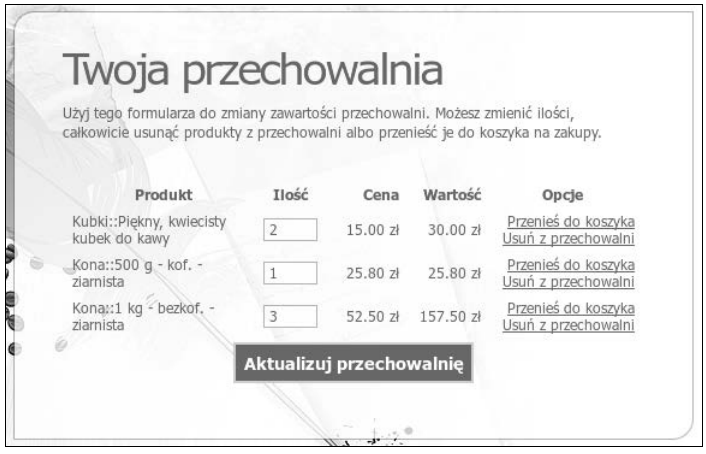
```
<!-- początek bloku -->
<div class="box alt"><div class="left-top-corner"><div class="right-top-corner">
➥<div class="border-top"></div></div></div><div class="border-left">
➥<div class="border-right"><div class="inner">
<h2>Twoja przechowalnia</h2>
<p>Twoja przechowalnia jest pusta.</p>
</div></div></div><div class="left-bot-corner"><div class="right-bot-corner">
➥<div class="border-bot"></div></div></div></div>
<!-- koniec bloku -->
```

Plik widoku *wishlist.html* wyświetla listę produktów w tabeli HTML. Podobnie jak w koszyku na zakupy, tu również zawartość listy można modyfikować na kilka sposobów:

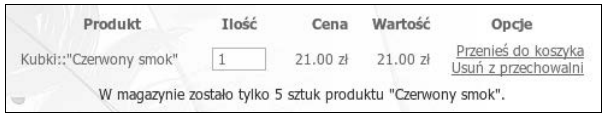
- zmieniając ilość produktów,
- przenosząc do koszyka na zakupy,
- usuwając z przechowalni.

Inaczej niż w skrypcie koszyka, na stronie przechowalni nie są wyświetlane ani informacje o wartości wszystkich produktów, ani link do strony zatwierdzania zamówienia (rysunek 9.8). Z kolei zamiast informowania o niewystarczającym stanie magazynowym na stronie przechowalni pojawia się jedynie komunikat z informacją o kończących się zapasach, co w zamyśle ma skłonić klienta do natychmiastowego zamówienia danego produktu (rysunek 9.9).

```
<div class="box alt"><div class="left-top-corner"><div class="right-top-corner">
➥<div class="border-top"></div></div></div><div class="border-left">
➥<div class="border-right"><div class="inner">
<h2>Twoja przechowalnia</h2>
<p>Użyj tego formularza do zmiany zawartości przechowalni. Możesz zmienić ilości,
➥całkowicie usunąć produkty z przechowalni albo przenieść je do koszyka na zakupy.</p>
<form action="/wishlist.php" method="POST">
<table border="0" cellspacing="8" cellpadding="6">
```




**Rysunek 9.8**




**Rysunek 9.9**

```
   <tr>
      <th align="center">Produkt</th>
      <th align="center">Ilość</th>
      <th align="right">Cena</th>
      <th align="right">Wartość</th>
      <th align="center">Opcje</th>
   </tr>
<?php
while ($row = mysqli_fetch_array($r, MYSQLI_ASSOC)) {
   $price = get_just_price($row['price'], $row['sale_price'] );
   $subtotal = $price * $row['quantity'];
   echo '<tr>
      <td>' . $row['category'] . '::' . $row['name'] . '</td>
      <td align="center"><input type="text" name="quantity[' . $row['sku'] . ']"
      ↪value="' . $row['quantity'] . '" size="2" /></td>
      <td align="right">$' . number_format($price, 2) . '</td>
      <td align="right">$' . number_format($subtotal, 2) . '</td>
      <td align="right"><a href="/cart.php?sku=' . $row['sku'] . '&action=move&qty=' .
      ↪$row['quantity'] .'">Przenieś do koszyka</a><br /><a href="/wishlist.php?
      ↪sku=' . $row['sku'] . '&action=remove">Usuń z przechowalni</a></td>
</tr>
';
// Sprawdź status:
if ( ($row['stock'] > 0) && ($row['stock'] < 10)) {
   echo '<tr class="error"><td colspan="5" align="center">W magazynie zostało
   ↪tylko ' . $row['stock'] . ' sztuk produktu ' . $row['name'] . '.</td></tr>';
}
```

```
} // Koniec pętli WHILE.
?> </table><p align="center"><input type="submit" value="Aktualizuj przechowalnię"
➥class="button" /></form></p></div></div></div><div class="left-bot-corner">
➥<div class="right-bot-corner"><div class="border-bot"></div></div></div></div>
<!--koniec bloku -->
```

# OBLICZANIE KOSZTÓW WYSYŁKI

Do koszyka warto dodać przydatną funkcję podającą koszt wysyłki. Dla wielu klientów opłaty za transport i obsługę zamówienia są istotnym czynnikiem, który może decydować o dokonaniu zakupu. Koszty te mogą być stałe lub można je uzależnić od:

- wagi przesyłanych produktów,
- odległości między nadawcą a odbiorcą,
- wymiarów paczki (przesłanie np. mebli wiąże się z dużymi kosztami),
- wielkości zamówienia.

W witrynie zostanie zastosowane ostatnie kryterium.

Przede wszystkim musisz zdefiniować funkcję obliczającą koszty wysyłki według ustalonej zależności. Załóżmy, że ustalamy pewien bazowy koszt, który trzeba przeznaczyć na wynagrodzenie pracownika kompletującego i pakującego zamówienie. Do tego należy doliczyć koszt częściowo zależny od wielkości zamówienia — co prawda większe zamówienia wiążą się z większą i cięższą przesyłką, jednak również więcej się na nich zarabia. Z tego względu koszty wysyłki zdefiniujemy tak, by były odwrotnie proporcjonalne do wielkości zamówienia (określanego na podstawie całkowitej wartości zamówienia). W związku z tym funkcja może wyglądać następująco:

```
function get_shipping($total = 0) {
   // Ustal bazowe koszty obsługi zamówienia:
   $shipping = 3;
   // Ustal współczynnik zależny od całkowitej wartości zamówienia:
   if ($total < 50) {
      $rate = .25;
   } elseif ($total < 100) {
      $rate = .20;
   } elseif ($total < 200) {
      $rate = .18;
   } elseif ($total < 500) {
      $rate = .16;
   } else {
      $rate = .15;
   }
   // Oblicz całkowity koszt przesyłki:
   $shipping = $shipping + ($total * $rate);
   // Zwróć całkowity koszt przesyłki:
   return number_format($shipping, 2);
} // Koniec funkcji get_shipping( ).
```

Jak się możesz domyślać, funkcję należy umieścić w pliku *product_functions.inc.php*, tak by była dostępna dla wielu skryptów (np. *cart.php* czy *checkout.php*, który zostanie utworzony w kolejnym rozdziale).

Aby skorzystać z tej funkcji w skrypcie *cart.php* (w przechowalni nie jest wyświetlana całkowita wartość zamówienia), dodaj poniższy kod przed wierszem wyświetlającym całkowitą wartość koszyka (rysunek 9.10).


**Rysunek 9.10**

```
$shipping = get_shipping($total);
$total += $shipping;
echo '<tr>
<td colspan="3" align="right"><strong>Transport i obsługa</strong></td>
<td align="right">' . $shipping . ' zł</td>
<td> </td>
</tr>
';
```

# SKOROWIDZ

## C

## D



## E



## F

## G



## H



## I

## J



## K

## L



## Ł



## M



## N

## O



## P

## R

## S

## S



## T



## U

## V



## W

## X



## Z

# E-commerce

Genialnie proste tworzenie serwisów

# w PHP i MySQL

Handel elektroniczny to znakomity sposób prowadzenia działalności zarobkowej. Twoje przedsięwzięcie może odnieść prawdziwy sukces — pod warunkiem że się do niego dobrze przygotujesz. Oto jedna z nielicznych na rynku książek dostarczających szczegółowych informacji na temat tworzenia serwisów e-commerce z wykorzystaniem PHP i MySQL. Bez względu na to, czy tworzysz dynamiczne strony internetowe od lat, czy dopiero od kilku tygodni, z pewnością znajdziesz tu mnóstwo bezcennych informacji.

Dzięki książce „E-commerce. Genialnie proste tworzenie serwisów w PHP i MySQL" dowiesz się, jak zaprojektować bazę danych, generować katalog produktów, zarządzać koszykiem zakupów, obsługiwać zamówienia i płatności oraz sprawić, by Twój sklep internetowy nie wymagał od Ciebie pracy ponad siły, a jednak był dochodowy. Podręcznik zawiera również omówienie tak istotnych zagadnień, jak zapewnienie bezpieczeństwa witryny oraz zadbanie o interfejs przyjazny użytkownikom, a także informacje dotyczące modułowego programowania, gotowego do dalszej rozbudowy. Przytoczone tu praktyczne przykłady pozwolą Ci spojrzeć na systemy e-commerce z możliwie jak najszerszej perspektywy.

- Wybór technologii internetowych
- Struktura i projekt witryny
- Zarządzanie zawartością witryny
- Tworzenie kont użytkowników
- Łączenie różnych systemów płatniczych
- Sprzedaż wirtualnych produktów
- Tworzenie bezpiecznego środowiska serwera i baz danych
- Tworzenie paneli administracyjnych
- Zasady składania zamówień

**Magia tworzenia** profesjonalnych serwisów e-commerce

helion.pl
księgarnia internetowa

Nr katalogowy: 6274

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Helion

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Cena 69,00 zł

ISBN 978-83-246-3213-8

Informatyka w najlepszym wydaniu