PHP и MySQL
СОЗДАНИЕ ИНТЕРНЕТ-МАГАЗИНА

2-е издание

titula-PHP MySQL-2ed.indd 1 25.12.2009 18:21:39

BEGINNING PHP AND MYSQL E-COMMERCE

FROM NOVICE TO PROFESSIONAL

SECOND EDITION

Cristian Darie and Emilian Balanescu

Apress®

titula-PHP MySQL-2ed.indd 2 25.12.2009 18:21:39

PHP и MySQL
СОЗДАНИЕ ИНТЕРНЕТ-МАГАЗИНА

2-е издание

Кристиан Дари и Эмилиан Баланеску



Москва · Санкт-Петербург · Киев
2011

titula-PHP MySQL-2ed.indd 3 25.12.2009 18:21:39

ББК 32.973.26-018.2.75
Д20
УДК 681.3.07

Издательский дом "Вильямс"

Зав. редакцией *С.Н. Тригуб*

Перевод с английского и редакция *А.С. Молявко*

По общим вопросам обращайтесь в Издательский дом "Вильямс" по адресу:
info@williamspublishing.com, http://www.williamspublishing.com

Дари, Кристиан, **Баланеску**, Эмилиан.

Д20 PHP и MySQL: создание интернет-магазина, 2-е изд. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2011. — 640 с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-1602-0 (рус.)

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства APress, Berkeley, CA.

Authorized translation from the English language edition published by APress, Copyright © 2008 by Cristian Darie and Emilian Balanescu.

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

Russian language edition is published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2011.

Научно-популярное издание

Кристиан Дари, Эмилиан Баланеску

PHP и MySQL: создание интернет-магазина

2-е издание

Литературный редактор *Л.Н. Красножон*
Верстка *М.А. Удалов*
Художественный редактор *В.Г. Павлютин*
Корректор *Л.А. Гордиенко*

Подписано в печать 24.12.2009. Формат 70x100/16.
Гарнитура Times. Печать офсетная.
Усл. печ. л. 51,6. Уч.-изд. л. 33,77.
Доп. тираж 1000 экз. Заказ № 0000.

Отпечатано по технологии CtP в ОАО "Печатный двор" им. А. М. Горького
197110, Санкт-Петербург, Чкаловский пр., 15.

ООО "И. Д. Вильямс", 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978-5-8459-1602-0 (рус.)
ISBN 978-1-59-059864-1 (англ.)

© Издательский дом "Вильямс", 2011
© by Cristian Darie and Emilian Balanescu, 2008

Оглавление

Введение 14

Часть I. Первый этап разработки 21

Глава 1. Начало разработки сайта электронного магазина 23

Глава 2. Закладываем фундамент 33

Глава 3. Начало реализации проекта TShirtShop 49

Глава 4. Создание каталога товаров: часть I 79

Глава 5. Создание каталога товаров: часть II 125

Глава 6. Атрибуты товаров 181

Глава 7. Оптимизация для поисковых систем 193

Глава 8. Поиск в каталоге 223

Глава 9. Прием платежей через PayPal 247

Глава 10. Администрирование каталога: отделы и категории 263

Глава 11. Администрирование каталога: товары и атрибуты 297

Часть II. Второй этап разработки 343

Глав12. Создание оригинальной корзины покупателя 345

Глава 13. Усовершенствование сайта с помощью AJAX 377

Глава 14. Прием заказов от покупателей 405

Глава 15. Предложение дополнительных товаров 431

Часть III. Третий этап разработки 443

Глава 16. Учетные записи клиентов 445

Глава 17. Хранение заказов 499

Глава 18. Реализация конвейера обработки заказов: часть I 523

Глава 19. Реализация конвейера обработки заказов: часть II 543

Глава 20. Работа с кредитными картами 567

Глава 21. Отзывы о товарах 593

Глава 22. Работа с веб-службами Amazon.com 601

Предметный указатель 622

Содержание

Об авторах 12
О технических редакторах 13
Благодарности 13

Введение 14

Для кого предназначена эта книга 14
Структура книги 15
- Этап I 15
- Этап II 17
- Этап III 18

Исходный код 19
Как связаться с авторами 20
От издательства 20

Часть I. Первый этап разработки 21

Глава 1. Начало разработки сайта электронного магазина 23

Решаем, нужно ли создавать сайт 23
- Привлечение новых клиентов 24
- Получение большего дохода от существующих клиентов 24
- Снижение стоимости выполнения заказов 25

Зарабатывание денег 25
Риски и угрозы 26
Создание веб-сайта 27
- Этап I. Базовая платформа сайта 27
- Этап II. Создание собственной корзины покупателя 28
- Этап III. Обработка заказов и дополнительные возможности 28

Сайт TShirtShop 30
Резюме 31

Глава 2. Закладываем фундамент 33

Дизайн для дальнейшего роста 34
- Выполнение долгосрочных требований с минимумом усилий 34
- Трехуровневая архитектура 35

Выбор технологий и инструментов 40
- Использование PHP для генерации динамического веб-контента 40
- Разделение кода и оформления с помощью Smarty 41
- Использование MySQL на уровне данных 42
- MySQL и трехуровневая архитектура 45

Выбор стандартов именования и оформления кода 46
Резюме 48

Глава 3. Начало реализации проекта TShirtShop 49

Редактор исходного кода 50
Установка XAMPP 50
Подготовка псевдонима TShirtShop 53

Установка Smarty 56
Создание каркаса сайта 58
Создание входной страницы TShirtShop 61
Обработка ошибок и сообщения о них 68
Подготовка базы данных 75
Загрузка исходного кода 77
Резюме 78

Глава 4. Создание каталога товаров: часть I 79

Демонстрация товаров посетителям сайта 79
Как выглядит каталог товаров 80
Предварительный просмотр каталога товаров 80
Последовательность действий в этой главе 83
Хранение информации из каталога 84
Концепция таблиц данных 85
Создание таблицы department 91
Работа с базой данных 93
Язык SQL 94
Хранимые процедуры в MySQL 97
Добавление управляющей логики в приложение 99
Установка соединения с MySQL 100
Написание кода уровня логики приложения 105
Отображение списка отделов 111
Использование подключаемых файлов Smarty 112
Создание фабрики ссылок 119
Резюме 122

Глава 5. Создание каталога товаров: часть II 125

Хранение новых данных 125
Из чего состоит реляционная база данных 125
Создание и заполнение новых таблиц 131
Добавление названий товаров и их привязка к категориям 133
Использование диаграмм баз данных 135
Запросы к новым данным 136
Получение кратких описаний товаров 136
Соединения таблиц данных 137
Постраничное отображение списков товаров 140
Создание хранимых процедур 141
Завершение кода уровня логики приложения 149
Определение констант для списков товаров и активизация сеансов 149
Реализация уровня представления 155
Отображение информации об отделе и категориях 156
Отображение списка категорий 161
Отображение списков товаров 164
Отображение содержимого главной страницы 170
Отображение подробных сведений о товарах 173
Резюме 179

Глава 6. Атрибуты товаров 181

Реализация уровня данных 182
Реализация уровня логики приложения 186
Реализация уровня представления 187
Резюме 192

Глава 7. Оптимизация для поисковых систем **193**
Оптимизация в TShirtShop 194
Поддержка URL с ключевыми словами 195
Модуль mod_rewrite и регулярные выражения 198
Генерация URL с ключевыми словами 202
Коррекция URL с помощью перенаправлений 206
Модификация заголовков страниц 210
Обновление разбивки списков на страницы 212
Корректные сообщения об ошибках 404 и 500 214
Резюме 221

Глава 8. Поиск в каталоге **223**
Методы поиска в каталоге 223
Реализация поиска в базе данных 225
Поиск с помощью ключевых слов WHERE и LIKE 225
Полнотекстовый поиск в MySQL 226
Реализация кода уровня логики приложения 233
Реализация уровня представления 237
Создание поля ввода 237
Отображение результатов поиска 241
Резюме 246

Глава 9. Прием платежей через PayPal **247**
Выбор платежной компании 248
Начинаем работу с PayPal 248
Интеграция корзины покупателя и приема платежей в сайт TShirtShop 249
Использование механизма покупки отдельных товаров в PayPal 260
Резюме 261

Глава 10. Администрирование каталога: отделы и категории **263**
Предварительное обсуждение интерфейса администратора 264
Создание страницы администрирования каталога 266
Использование защищенных соединений 267
Конфигурирование TShirtShop для SSL 269
Получение сертификата SSL 269
Принудительное использование соединений SSL 270
Аутентификация администраторов 271
Администрирование отделов 279
Реализация уровня представления 280
Реализация уровня логики приложения 285
Реализация уровня данных 286
Администрирование категорий 288
Резюме 295

Глава 11. Администрирование каталога: товары и атрибуты **297**
Администрирование атрибутов товаров 297
Администрирование товаров 310
Администрирование описаний товаров 315
Сведения о товарах: реализация уровня представления 315
Сведения о товарах: реализация уровня логики приложения 326
Сведения о товарах: реализация уровня данных 330
Создание ссылок для администрирования 334
Резюме 341

Часть II. Второй этап разработки 343

Глава 12. Создание оригинальной корзины покупателя 345

Проектирование корзины покупателя 346
Сохранение информации о корзинах покупателей 348
Реализация уровня данных 349
Реализация уровня логики приложения 353
Реализация уровня представления 358
Администрирование корзины покупателя 369
Удаление товаров, находящихся в корзинах покупателей 370
Создание страницы администрирования корзин покупателей 370
Резюме 375

Глава 13. Усовершенствование сайта с помощью AJAX 377

Краткое введение в AJAX 377
JavaScript 378
DOM 379
Объект XMLHttpRequest 379
Написание устойчивого к сбоям кода 386
Всегда ли полезен AJAX 387
Создание корзины покупателя с применением AJAX 389
Применение AJAX для добавления товаров в корзину 389
Модернизация корзины покупателя с помощью AJAX 400
Резюме 404

Глава 14. Прием заказов от покупателей 405

Реализация системы приема заказов 405
Сохранение деталей заказов 407
Реализация уровня данных 409
Реализация уровня логики приложения 411
Реализация уровня представления 412
Администрирование заказов 415
Отображение существующих заказов 419
Отображение деталей заказов 424
Резюме 430

Глава 15. Предложение дополнительных товаров 431

Увеличение продаж с помощью динамических рекомендаций 431
Выбор рекомендаций из базы данных 433
Реализация рекомендаций для товаров 439
Резюме 442

Часть III. Третий этап разработки 443

Глава 16. Учетные записи клиентов 445

Хранение учетных записей клиентов 446
Схема учетных записей клиентов TShirtShop 446
Создание таблиц customer и shipping_region 447
Реализация классов, отвечающих за безопасность 448
Реализация функциональности хеширования на уровне логики приложения 448
Реализация функциональности шифрования на уровне логики приложения 451
Хранение информации о кредитных картах с помощью класса SecureCard 456
Добавление учетных записей пользователей в TShirtShop 464
Реализация уровня данных 467

Реализация уровня логики приложения 469
Реализация уровня представления 472
Создание страницы оплаты 488
Защита конфиденциальных сведений с помощью SSL 496
Резюме 498

Глава 17. Хранение заказов 499

Добавление заказов в учетные записи пользователей 499
Администрирование заказов 503
Изменение уровня данных 503
Изменение уровня логики приложения 505
Изменение уровня представления 507
Налоги и стоимость доставки 512
Налоги 512
Стоимость доставки 513
Реализация учета налогов и стоимости доставки 514
Резюме 521

Глава 18. Реализация конвейера обработки заказов: часть I 523

Что такое конвейер обработки заказов 524
Проектирование конвейера обработки заказов 524
Закладываем основы 528
Обновление кода обработки заказов 537
Резюме 542

Глава 19. Реализация конвейера обработки заказов: часть II 543

Реализация этапов конвейера 543
Класс PsInitialNotification 544
Класс PsCheckFunds 545
Класс PsCheckStock 546
Класс PsStockOk 547
Класс PsTakePayment 548
Класс PsShipOk 548
Класс PsFinalNotification 549
Проверка работы конвейера 550
Обновление страницы оплаты 557
Обновление страницы администрирования заказов 560
Реализация уровня данных 560
Реализация уровня логики приложения 561
Реализация уровня представления 561
Резюме 565

Глава 20. Работа с кредитными картами 567

Основы транзакций с кредитными картами 567
Работа со шлюзами оплаты 568
Основы транзакций с кредитными картами 569
Работа с DataCash 569
Запрос преаутентификации 570
Ответ на запрос преаутентификации 571
Запрос завершения 572
Работа с Authorize.net 583
Резюме 592

Глава 21. Отзывы о товарах **593**
Планирование механизма отзывов 593
Реализация отзывов о товарах 595
Резюме 599

Глава 22. Работа с веб-службами Amazon.com **601**
Введение в веб-службы 601
Доступ к веб-службам Amazon.com 603
Создание учетной записи Amazon.com Web Services 603
Идентификатор сообщества Amazon 604
Доступ к Amazon.com E-Commerce Service с помощью REST 605
Доступ к Amazon.com E-Commerce Service с помощью SOAP 608
Интеграция A2S и TShirtShop 611
Реализация логики уровня приложения 612
Реализация уровня представления 616
Резюме 620

Предметный указатель **622**

Об авторах

Кристиан Дари (Christian Darie) — программист, занимающий должность программного архитектора в компании Netbridge Development S.R.L. и отвечающий за поддержку крупнейшего румынского сайта электронного магазина, `OKazii.ro`.

Кристиан написал множество книг о программировании, а сейчас изучает архитектуры распределенных приложений в Политехническом университете Бухареста, Румыния. Он участвует в нескольких исследовательских и коммерческих проектах. Если вы хотите с ним пообщаться, зайдите на его персональный веб-сайт по адресу `http://www.christiandarie.ro`.

Эмилиан Баланеску (Emilian Balanescu) — программист с опытом использования множества технологий, в том числе PHP, Java, .NET, PostgreSQL, MySQL и Microsoft SQL Server. Он получил сертификат Microsoft Certified Technology Specialist и сейчас является старшим веб-разработчиком в компании SoftNET Business Services S.R.L. и разрабатывает инструменты для упрощения совместной работы в малых и средних компаниях. С ним можно связаться через его персональный веб-сайт по адресу `http://www.emilianbalanescu.ro`.

О технических редакторах

Богдан Бринзареа-Иаманди (Bogdan Brinzarea-Iamandi) обладает обширными познаниями в различных связанных с компьютерами областях. Он получил ученую степень бакалавра на факультете автоматического управления и компьютеров Политехнического университета Бухареста и степень магистра в Политехнической школе в Париже. Его основная сфера интересов — новые веб-технологии, распределенные системы и мобильные вычисления.

Шэрон Дэмпси (Sharon Dempsey) — автор электронного журнала, посвященного финансовым вопросам. Желание создать сайт с возможностью поиска в базах данных и поддержкой платежей привело ее к участию в написании этой книги. Шэрон тестировала код из книги в Windows XP и вносила предложения как читатель, не являющийся экспертом.

Одра Хендрикс (Audra Hendrix) сейчас привыкает к жизни в Америке после пяти лет, проведенных во Франции. Хорошо зная французский, она работала независимым консультантом по разработке и развертыванию приложений с различными клиентами, в основном в области электронной торговли. Одра училась в Северозападном университете в Эванстоне, штат Иллинойс, и работала в компании Hewlett-Packard. Сейчас она является консультантом и советником небольших компаний. Поработав в компаниях из списка Fortune 500, она пришла к выводу, что небольшие фирмы нравятся ей гораздо больше. Кроме того, она разрабатывает маркетинговые стратегии для коммерческих сайтов и веб-служб.

Благодарности

Авторы благодарят следующих людей за их неоценимую помощь в подготовке к изданию данной книги.

Трэйси Браун Коллинс (Tracy Brown Collins), менеджера проекта, — за успешное руководство всеми, кто работал над книгой. Сложности, возникавшие перед нами за год написания книги, были бы непреодолимыми, если бы не ее талант организатора.

Хизер Лэнг (Heather Lang) и Ким Вимпсетт (Kim Wimpsett) — за умелое редактирование, в результате которого бессмысленные наборы слов превращались в понятные тексты.

Лору Эстерман (Laura Esterman) и остальных сотрудников издательства — за превращение файлов с текстами и рисунками в книгу, которую вы сейчас держите в руках.

Богдана Бринзареа-Иаманди, Шэрон Дэмпси и Одру Хендрикс — за тестирование кода, проверку точности приведенных в книге сведений и множество ценных предложений, значительно повысивших качество книги и устранивших потенциальные источники проблем для читателей.

Свои семьи и друзей — за неизменную лояльность и эмоциональную поддержку, оказанную в процессе работы над книгой.

Введение

Вы держите в руках второе издание книги о разработке сайтов электронной торговли на PHP.

Эта книга — практический пошаговый учебник по использованию PHP и MySQL для разработки реальных приложений. В нем рассматриваются все этапы проектирования и реализации приложений, и в конце концов вы создадите полнофункциональный, расширяемый сайт электронного магазина.

В каждой главе будут реализованы и протестированы определенные элементы сайта, и каждый раз мы будем рассматривать теоретические аспекты создаваемых элементов. Сложность материала возрастает от начала книги к концу, чтобы упростить его усвоение. Изучив книгу, вы будете понимать теоретические концепции и обладать практическими навыками создания веб-сайтов.

Те, кто читали первое издание книги, обнаружат, что книга существенно переработана и в нее добавлено много нового материала, отражающего последние новшества в разработке веб-сайтов. В частности, расширены разделы, посвященные использованию AJAX, оптимизации для поисковых систем и атрибутам товаров.

Учебный проект — создание электронного магазина — разбит на три этапа. На первом этапе мы с минимальными затратами сил и времени создадим работающий сайт с базовым набором функций. К концу этого этапа сайт действительно будет готов к развертыванию — в нем будут присутствовать работающий каталог товаров с поддержкой поиска и механизм приема платежей с помощью PayPal.

Цель второго этапа — повысить доходы электронного магазина. На этом этапе мы создадим механизмы, повышающие привлекательность сайта для посетителей и побуждающие делать дополнительные покупки, например механизм динамических рекомендаций. Кроме того, мы применим AJAX для того, чтобы ускорить реакцию сайта на действия посетителей, и заменим корзину покупателя, предоставляемую PayPal, корзиной собственной разработки.

На третьем этапе мы повысим эффективность работы с клиентами за счет автоматизации обработки заказов и работы с кредитными картами. Кроме того, вы узнаете, как интегрировать в сайт внешнюю функциональность через веб-службы и добавить поддержку отзывов о товарах.

Мы надеемся, что вам будет приятно читать эту книгу и что полученные из нее знания пригодятся вам в работе над реальными проектами.

Для кого предназначена эта книга

Эта книга адресована разработчикам, которых интересует пример разработки сайта электронного магазина — от формирования требований до развертывания. В книге рассмотрено большинство существенных моментов и все этапы реализации, но подразумевается, что у вас есть базовые навыки создания веб-сайтов с помощью PHP и MySQL. Если таких навыков у вас нет, поищите книгу попроще, например 3-е издание книги Мэтта Зандстры *Освой самостоятельно PHP за 24 часа* (ИД "Вильямс", 2008 г.).

Код в этой книге тестировался на PHP 5 и MySQL 5. Он *не* совместим с более ранними версиями PHP, поскольку только в PHP 5 появилась поддержка объектно-ориентированных механизмов, используемых в этом коде.

Информацию о совместимости кода с более новыми версиями PHP и MySQL можно найти на сайте автора книги, по адресу `http://www.christiandarie.ro/php-mysql-ecommerce-2/`.

Структура книги

Эта книга разделена на три части, содержащие в общей сложности 22 главы. Каждая из частей соответствует определенному этапу разработки веб-сайта. Мы рассмотрим множество тем, в том числе следующие.

- Создание каталога товаров с поддержкой поиска по ключевым словам и выборки по критериям
- Реализация страниц администрирования каталога, позволяющих добавлять товары в каталог, изменять сведения о товарах, категориях и отделах и выполнять другие операции
- Создание корзины покупателя и механизма приема платежей с помощью PHP
- Прием платежей с помощью PayPal, DataCash и Authorize.net
- Реализация системы учетных записей посетителей
- Интеграция с веб-службами Amazon.com для продажи товаров Amazon.com через наш веб-сайт

Работая над перечисленными задачами, вы научитесь следующему.

- Проектировать реляционные базы данных и составлять SQL-запросы и сохраненные процедуры
- Использовать возможности поиска MySQL для организации поиска товаров в каталоге
- Использовать систему шаблонов Smarty для создания структурированного и расширяемого кода на PHP
- Оптимизировать сайт для работы с поисковыми системами
- Ускорять реакцию сайта на действия пользователей с помощью AJAX
- Использовать внешние веб-службы

Далее кратко описано содержимое каждой из глав книги.

Этап I

На первом этапе будет создан базовый вариант сайта с работающим каталогом товаров. Сначала мы решим, какой должна быть общая архитектура сайта и как будут взаимодействовать между собой различные элементы. Затем мы займемся непосредственно созданием каталога и заполним его данными.

Глава 1. Начало разработки сайта электронного магазина

В этой главе мы обсудим общие принципы работы сайтов электронных магазинов. Вы узнаете, почему важно сосредоточиваться на краткосрочной прибыли и снижении рисков. Мы также рассмотрим три основных способа получения дохода в электронных магазинах и остановимся на каждом из них в отдельной части книги.

Глава 2. Закладываем фундамент

В главе 1 мы рассмотрели общие принципы работы электронных магазинов. Решив создать свой электронный магазин, нужно решить, какой будет его структура и какие технологии будут в нем использоваться.

Глава 3. Начало реализации проекта TShirtShop

В этой главе мы установим все инструменты, необходимые для создания учебного проекта — электронного магазина TShirtShop. Мы подробно опишем процесс установки всех нужных программ, включая веб-сервер Apache и сервер баз данных MySQL. Кроме того, мы напишем первые фрагменты кода проекта и создадим базу данных, в которой будут храниться все данные электронного магазина.

Глава 4. Создание каталога товаров: часть I

Познакомившись с трехуровневой архитектурой и создав заготовку главной страницы сайта, можно начать работу над каталогом товаров. Поскольку этот каталог будет состоять из множества компонентов, его созданию посвящены две главы. В главе 4 мы создадим первую таблицу базы данных, первую сохраненную процедуру и код на PHP, обращающийся к этой сохраненной процедуре. К концу главы у нас будет работающая веб-страница с динамически генерируемыми элементами.

Глава 5. Создание каталога товаров: часть II

В главе 4 мы создали список отделов, из которого пользователь может выбрать нужный отдел. Но каталог товаров — это не только список отделов. В главе 5 мы добавим в каталог все остальные элементы — страницы категорий товаров, списки товаров и страницы отдельных товаров. Создавая структуру данных, которая обеспечит работу всех этих элементов, вы узнаете, как создавать отношения между таблицами в базах данных и использовать параметризованные сохраненные процедуры в MySQL.

Глава 6. Атрибуты товаров

Многие электронные магазины позволяют покупателям выбирать атрибуты приобретаемых товаров. Например, если мы будем продавать футболки, можно позволить покупателю выбрать цвет или размер, или рисунок на футболке. В этой главе мы добавим в проект поддержку атрибутов товаров.

Глава 7. Оптимизация для поисковых систем

Оптимизация сайта для поисковых систем — это набор операций, которые позволяют повысить релевантность страниц сайта с точки зрения поисковых систем в Интернете. Сейчас поисковые системы являются основным инструментом, используемым людьми для получения информации, и получение вашим сайтом высоких рейтингов у поисковых систем резко увеличивает шансы на то, что ваши товары будут хорошо продаваться. В этой главе мы сделаем архитектуру TShirtShop более дружелюбной к поисковым системам.

Глава 8. Поиск в каталоге

Итак, в предыдущих главах мы создали каталог товаров. Но в нем нет важного механизма — он не позволяет выполнять поиск товаров по ключевым словам. В этой главе мы как раз и займемся созданием механизма поиска по ключевым словам.

Вы убедитесь, что добавлять новые элементы в проект с продуманной архитектурой совсем несложно.

Глава 9. Прием платежей через PayPal

Нашему электронному магазину абсолютно необходим механизм приема платежей от покупателей. Для крупных компаний удобнее всего открыть специальный счет в банке, но небольшие компании обычно предпочитают более простые решения, избавляющие их от необходимости работать с кредитными картами и чеками.

Существует множество компаний, предоставляющих услуги по приему платежей в Интернете. В этой главе мы рассмотрим работу с одной из самых известных компаний такого рода — PayPal.

Глава 10. Администрирование каталога: отделы и категории

Последний элемент, который нужно добавить на сайт, прежде чем он будет готов к работе, — это интерфейс администрирования. Хотя посетители никогда его не увидят, он абсолютно необходим для стабильной работы сайта. В этой и следующей главах мы займемся его созданием. В главе 10 мы сосредоточимся на администрировании отделов и категорий.

Глава 11. Администрирование каталога: товары и атрибуты

В этой главе мы закончим работу над механизмом администрирования, создав страницы администрирования товаров и атрибутов. К концу главы у администраторов сайта будет возможность добавлять в каталог новые товары, заносить товары в категории и отделы, создавать и удалять атрибуты товаров и т.д.

Этап II

Второй этап посвящен повышению дохода от магазина за счет улучшения впечатления, которое магазин производит на посетителей. Мы добавим в проект механизм динамических рекомендаций товаров, а также применим AJAX для ускорения реакции сайта на действия посетителей.

Глава 12. Создание оригинальной корзины покупателя

С этой главы начинается второй этап разработки. На этом этапе мы займемся улучшением и расширением функциональности уже работающего электронного магазина. В главе 12 мы создадим собственную версию корзины покупателя, которая будет сохранять все необходимые данные в нашей базе данных. Это даст нам возможности, которых нет у корзины, предоставляемой PayPal, в частности мы сможем анализировать покупки и генерировать рекомендации товаров по результатам анализа.

Глава 13. Усовершенствование сайта с помощью AJAX

В этой главе мы улучшим работу каталога товаров и корзины покупателя с помощью технологии AJAX. Данная технология позволяет ускорить обновление веб-интерфейса за счет передачи только изменяющихся частей веб-страниц.

Глава 14. Прием заказов от покупателей

Корзина покупателя на нашем сайте полностью работоспособна, за исключением одной маленькой детали: она не позволяет посетителям размещать заказы. Это делает ее полностью бесполезной для реального электронного магазина, и данную проблему необходимо устранить. В первой части главы 14 мы создадим клиентскую часть механизма приема заказов, а во второй — простую страницу администрирования заказов, которая позволит администратору сайта просматривать и обрабатывать поступающие заказы.

Глава 15. Предложение дополнительных товаров

Одно из основных преимуществ электронного магазина по сравнению с обычным — способность веб-страниц подстраиваться под требования посетителей, в частности исходя из статистики, собранной по действиям предыдущих посетителей. Если ваш электронный магазин сможет "угадывать", какие товары заинтересуют посетителей, ваш доход заметно увеличится. Вы наверняка видели реализации этого механизма в реальных электронных магазинах — и он действительно помогает увеличить продажи. В этой главе мы реализуем простой, но эффективный механизм динамических рекомендаций товаров.

Этап III

Последний этап посвящен повышению эффективности работы электронного магазина за счет автоматизации обработки заказов и приема платежей. Кроме того, на этом этапе мы рассмотрим возможности интеграции внешних веб-служб в электронный магазин, а также добавим систему отзывов покупателей о товарах.

Глава 16. Учетные записи клиентов

Наш сайт уже способен обрабатывать заказы посетителей и принимать платежи с помощью PayPal. На третьем этапе разработки мы продвинемся дальше. Создав собственный механизм приема платежей вместо предоставленного PayPal, мы получим возможность лучше контролировать платежи и уменьшить накладные расходы. Это не так сложно, как вам может показаться, если все делать аккуратно. В главе 16 мы начнем создание нового механизма приема платежей системы учетных записей клиентов. Кроме того, мы обсудим вопросы безопасного хранения ценной информации.

Глава 17. Хранение заказов

В главе 16 мы добавили в проект поддержку учетных записей клиентов, и теперь у нас есть возможность хранить адреса клиентов и сведения об их кредитных картах. Но пока что мы не используем эту информацию в системе обработки заказов, созданной на втором этапе разработки. Мы вообще не связываем сделанный заказ с учетной записью клиента, который его разместил.

В этой главе мы внесем в проект изменения, необходимые для связывания заказов с учетными записями клиентов. В частности, мы модифицируем таблицу заказов в базе данных, чтобы в каждом заказе был указан идентификатор клиента, разместившего этот заказ. В оставшейся части книги мы в основном будем так или иначе использовать эту информацию.

Связывание учетных записей клиентов с заказами закладывает основу для нового конвейера обработки заказов и системы приема платежей с кредитных карт, которые мы создадим в последующих главах.

Глава 18. Реализация конвейера обработки заказов: часть I

Конвейер для обработки заказов — чрезвычайно полезный механизм для сайта электронного магазина. Он позволяет отслеживать прохождение заказов через все этапы обработки и генерировать информацию для аудита, которая пригодится нам, если что-то пойдет не так. Еще лучше, если конвейер не будет полагаться на внешнюю систему учета, — это снизит наши накладные расходы.

Реализация конвейера обработки заказов — это первый шаг по направлению к профессиональной системе управления заказами. В этой и последующей главах мы создадим конвейер, включающий в себя систему авторизации для работы с кредитными картами, проверку запасов, контроль отправки, рассылку уведомлений и другие компоненты. Работой с кредитными картами мы займемся в главе 20, но здесь вы узнаете, как механизм для работы с ними будет использоваться в конвейере.

Глава 19. Реализация конвейера обработки заказов: часть II

В этой главе мы создадим элементы конвейера, которые позволят полностью отследить обработку заказа, хотя механизмом для работы с кредитными картами мы займемся только в следующей главе. Кроме того, мы поработаем над администрированием заказов, модифицировав ранее созданные страницы администрирования так, чтобы они работали с новым конвейером.

Глава 20. Работа с кредитными картами

Последний элемент, которого не хватает нашему электронному магазину, — это механизм для работы с кредитными картами. Мы рассмотрим создание такого механизма с помощью двух популярных шлюзов для работы с кредитными картами — DataCash и Authorize.net. К концу этой главы наш электронный магазин будет полностью готов к работе.

Глава 21. Отзывы о товарах

К данному моменту у нас есть работающий сайт электронного магазина. Но это не значит, что мы больше ничего не можем к нему добавить. Добавив систему отзывов о товарах, вы повысите шансы на то, что после покупки клиент вернется на ваш сайт, чтобы написать отзыв или почитать отзывы, оставленные другими клиентами.

Глава 22. Работа с веб-службами Amazon.com

В предыдущих главах мы уже интегрировали в наш сайт функциональность, предоставляемую внешними службами — PayPal, DataCash и Authorize.net. В этой главе будут рассмотрены возможности, которые предоставит интеграция в наш сайт веб-служб Amazon.com. Мы можем увеличить доход от нашего электронного магазина, организовав продажу через него товаров, предлагаемых Amazon.com, и в главе 22 мы рассмотрим интеграцию в наш сайт веб-служб, позволяющих организовать такую продажу.

Исходный код

Исходный код примеров из этой книги можно загрузить в виде архива в формате zip из раздела `Downloads` на сайте издательства Apress. Комментарии к коду, сообщения о найденных ошибках и другие ресурсы доступны на сайте автора книги по адресу `http://www.christiandarie.ro/php-mysql-ecommerce-2/`.

Как связаться с авторами

Связаться с Кристианом Дари можно через его веб-сайт по адресу `http://www.christiandarie.ro`, а с Эмилианом Баланеску — через сайт `http://www.emilianbalanescu.ro`.

От издательства

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо, либо просто посетить наш Web-сервер и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг. Наши координаты:

E-mail:	`info@williamspublishing.com`
WWW:	`http://www.williamspublishing.com`

Адреса для писем из:

России:	127055, г. Москва, ул. Лесная, д. 43, стр. 1
Украины:	03150, Киев, а/я 152

Часть I

Первый этап разработки

Глава 1

Начало разработки сайта электронного магазина

Термин “электронный магазин” за несколько последних лет заметно утратил окружавшую его ауру привлекательности. Когда-то одного предложения создать сайт электронного магазина было достаточно, чтобы заставить крупных бизнесменов немедленно раскошелиться. Однако сейчас уже мало одних только фраз вроде “За электронными магазинами будущее — работайте в Интернете или уходите с рынка”. Чтобы убедить кого-то в необходимости создать сайт электронного магазина, потребуются веские аргументы и хорошее знание предметной области.

Сегодня, прежде чем создать сайт электронного магазина, нужно ответить на несколько непростых вопросов. Среди них, прежде всего, следующие.

- Множество сайтов электронных магазинов закрылись, принеся создателям одни убытки. Что я могу выиграть от выхода в Интернет в современных условиях жесткой конкуренции?
- Большинству компаний, занимающихся электронной коммерцией, потребовались крупные инвестиции. Смогу ли я создать сайт в условиях ограниченного финансирования?
- Даже успешные сайты электронных магазинов нередко существуют несколько лет, прежде чем становятся прибыльными. Мой бизнес не может столько ждать. Что сделать, чтобы получить прибыль быстрее?

В этой главе мы попытаемся ответить на эти вопросы.

Решаем, нужно ли создавать сайт

Существует великое множество причин, по которым компания может решить выйти в Интернет, однако все эти причины можно разбить на несколько категорий.

- Сохранение существующих клиентов и обретение новых
- Получение большего дохода от существующих клиентов
- Уменьшение накладных расходов при выполнении заказов

Ниже мы рассмотрим каждую из этих категорий подробнее.

Привлечение новых клиентов

Эта причина — одна из наиболее распространенных. На первый взгляд кажется, что с помощью веб-сайта даже маленькая компания легко может привлечь клиентов со всего мира. Иногда это действительно так. Однако здесь есть несколько тонкостей, которые трудно сразу заметить, — часто люди считают, что любой сайт в Интернете автоматически привлекает к себе посетителей. Увы, это не так. В повседневной жизни ни один человек не пойдет в магазин, о существовании которого он не подозревает. Точно так же и в Интернете никто не зайдет на сайт, о котором никому не известно. Клиенты должны знать о вашем существовании, чтобы что-то у вас купить.

Решение этой проблемы в целом сводится к рекламе вашего сайта. Кроме рекламы, можно привлекать клиентов, зарегистрировав свой сайт в поисковых системах и службах директорий, оптимизировав сайт для поисковых систем, создав форумы и т.д.

В этой книге мы не будем рассматривать вопросы продажи сайтов — она посвящена продаже *товаров* с помощью сайта, но не самого сайта. Мы рассмотрим простейшие методики оптимизации для поисковых систем (для привлечения посетителей) и оформления сайта (для упрощения работы с ним).

Получение большего дохода от существующих клиентов

Если у вашей компании уже есть сформированная база клиентов, вы, вероятно, хотите зарабатывать на них больше денег. Что мешает им тратить больше? Если клиенты не хотят покупать какой-то товар, вряд ли они купят его и в электронном магазине. Однако есть и другие препятствия к росту продаж, которые можно устранить. К ним относятся, например, следующие.

- До вашего магазина физически трудно добраться, или клиенты не хотят отправлять заказы почтой.
- Товары, которые вы продаете, удобнее покупать в другом месте.
- Вы открыты, когда ваши клиенты в основном на работе.
- В обычном магазине трудно создать эффективную систему рекомендаций по товарам.

Качественный сайт электронного магазина может заметно увеличить ваш доход. Кроме того, сам факт наличия у вас такого сайта может заставить людей покупать у вас, а не у местных поставщиков. Если ваш сайт работает круглые сутки, а обычные магазины — только с 9 утра до 6 вечера, покупатели смогут совершать покупки на сайте в нерабочее время. Кроме того, многие люди посещают электронные магазины, сидя за компьютерами на работе.

Люди с доступом к Интернету часто обнаруживают, что заказать товар в электронном магазине гораздо проще, чем купить его в обычном магазине — зачастую в электронных магазинах люди покупают такие вещи, которые и не подумали бы покупать в обычных. Специальные предложения для постоянных клиентов, предложения дополнительных покупок во время оплаты, аксессуаров к выбранным товарам или более дорогих альтернатив этим товарам могут заставить покупателей тратить больше. Веб-сайт легко обновлять, добавляя сезонные предложения, анонсируя интересные новые товары или дополнения к товарам, которые клиент уже покупал.

Как использовать некоторые из этих методов, вы узнаете из данной книги; прочтя ее до конца, вы будете представлять себе, как добавлять на сайт новые возможности самостоятельно.

Наконец, изучать покупателей с помощью веб-сайта гораздо удобнее, чем в обычном магазине или при получении заказов по почте. Даже зная только адреса электронной почты своих покупателей, вы можете рассылать по этим адресам новости и анонсы новых товаров. Многие сайты электронной коммерции автоматически анализируют предпочтения покупателей и предлагают им дополнительные товары, которые могут этим покупателям понравиться.

Еще одна выгода электронных магазинов в том, что, если пользователь зашел на ваш сайт, но ничего не купил, вы не потратите на него денег. Вообще говоря, чем больше людей посещают сайт, тем лучше для магазина. Стоит подумать о том, чтобы добавить на сайт элементы, предназначенные именно для привлечения постоянных посетителей, например форумы или бесплатные материалы, относящиеся к продаваемым товарам.

Снижение стоимости выполнения заказов

Правильно спроектированный сайт электронного магазина обходится в эксплуатации гораздо дешевле обычного магазина. В обычной бизнес-модели кто-то из персонала должен передать заказ системе обработки заказов в компании. На сайте электронного магазина это может сделать сам покупатель — переход между веб-сайтом и системой обработки заказов может быть вообще незаметным.

Разумеется, когда сайт электронного магазина уже создан и работает, стоимость обработки заказов приближается к нулю — вам не нужно платить продавцам и охранникам, снимать помещение и т.д.

Если у вас есть хорошая бизнес-идея и вы хорошо организовали сайт, можете получить доход без больших инвестиций. Важно всегда сосредоточиваться именно на доходе — убедитесь, что ваш сайт, все его элементы, действительно помогают привлекать новых клиентов, сохранять существующих и заставлять их тратить больше или просто уменьшать ваши затраты на выполнение заказов.

А теперь давайте посмотрим на сайт, который будет служить нам рабочим примером в этой книге, и выясним, как все сказанное выше относится к этому сайту.

Зарабатывание денег

Мы будем создавать сайт электронного магазина, торгующего футболками. При создании таких сайтов всегда приходится выдерживать баланс между созданием впечатляющих сайтов, которые всем нравятся, и созданием сайтов, которые позволяют получать доход при ограниченных затратах. Обычно я склоняюсь именно к созданию первых, но суровая проза жизни всегда заставляет меня вспоминать о сроках сдачи проектов и ограниченности финансов. Если вы сами являетесь собственным клиентом, т.е. создаете сайт для себя, учтите, что важно создать работающий сайт, прежде чем вы утратите энтузиазм и бросите это дело.

В данной книге мы поочередно рассмотрим этапы создания сайта электронного магазина, в котором будет все, что нужно для получения прибыли. Однако при разработке собственного сайта нужно хорошо подумать о том, кем являются ваши клиенты, что им нужно, как они будут размещать заказы и что они, скорее всего, будут покупать. Прежде чем проектировать сайт и тем более начинать программировать, обдумайте следующие моменты.

- *Привлечение клиентов.* Как новые клиенты будут узнавать о вашем сайте?
- *Предложение товаров.* Какие товары вы будете предлагать и какие товары ваши клиенты будут покупать? Будут ли они делать крупные закупки? Часто ли они будут возвращаться за новыми покупками? Будут ли они заранее решать, что им нужно купить, или вам нужно помочь им сделать выбор? Ответы на эти вопросы определят, какие возможности и инструменты должны быть доступны на вашем сайте, как вы должны организовать каталог товаров и какой механизм обработки заказов вам следует использовать. Например, если покупатель заранее точно знает, что он хочет купить, можно будет разместить форму для заказа прямо на входной странице сайта.
- *Обработка заказов.* Как превратить заказ на сайте в сверток, готовый к отправке? Здесь основные проблемы — это эффективная обработка платежей и доставка заказов тем, кто управляет вашим складом или производством. Как вы убедите своих клиентов в вашей способности быстро и безошибочно выполнять их заказы?
- *Обслуживание клиентов.* Понадобятся ли покупателям какие-то дополнительные услуги, связанные с купленными товарами, — гарантия, сервисное обслуживание, ремонт, расходные материалы?
- *Повторное привлечение клиентов.* Как вы привлечете на сайт тех, кто уже его посещал? Будут ли люди посещать ваш сайт, только чтобы сделать покупку, или и с другими целями (чтение новостей, общение на форумах и т.д.)? Можете ли вы предсказать, когда покупателя перестанет устраивать купленный у вас товар, и он решит купить новый?

Обдумав все эти моменты, вы можете приступать к проектированию сайта, зная, что вы делаете это для клиентов, а не просто творите то, что вам кажется полезным в данный момент. Пример сайта, который мы будем создавать в этой книге, выбран с таким расчетом, чтобы продемонстрировать вам как можно больше разных методик, используемых в электронной торговле.

Чтобы выделиться на фоне конкурентов, вам не обязательно напичкивать сайт красивыми картинками или Flash-анимацией. Достаточно понимать, привлекать и обслуживать клиентов лучше, чем это смогут сделать конкуренты. В этой книге мы постараемся объяснить вам, как это делать.

Риски и угрозы

Хотя у вас могло сложиться впечатление, что риски и угрозы в электронной торговле минимальны, на самом деле они многочисленны и весьма серьезны. Некоторые из них являются общими для торговли во всех ее вариантах, но есть риски и угрозы, специфичные для ее онлайн-разновидности.

- Хакерские атаки
- Мошенничество с кредитными картами
- Отказы оборудования
- Ненадежные службы доставки
- Ошибки программного обеспечения
- Постоянно изменяющиеся законы

Полностью избежать этих рисков невозможно, однако если вы их учтете, то сможете свести к минимуму эффект от них.

В защите сайтов от всех угроз особое место занимает резервное копирование. Вы знаете, что резервные копии всегда нужны. Но если вы хоть в чем-то похожи на остальных, то для вас уйти с работы на пять минут раньше важнее, чем создать резервную копию. Увы, если вы управляете коммерческим веб-сайтом, вам придется постоянно создавать резервные копии, как минимум каждый рабочий день.

О юридической стороне электронной торговли мы не будем много говорить — в конце концов, мы программисты, а не юристы. Однако, если вы собираетесь заниматься ею в течение продолжительного времени, вам придется изучить законы своей страны или нанять того, кто их изучил.

Особо следует отметить проблемы, связанные с ненадежностью служб доставки. Эти проблемы могут нанести огромный ущерб вашей компании в целом и электронному магазину в частности. Если вы не сможете обеспечить надежную и своевременную доставку заказов покупателям, они как минимум никогда больше не будут иметь с вами дела и никому не будут вас рекомендовать. Обычно службу доставки нужно организовывать до запуска электронного магазина.

Создание веб-сайта

Создание сайта электронного магазина требует значительных сил и средств. Разделив процесс создания такого сайта на этапы, вы сможете уменьшить необходимый объем начальных инвестиций и свести к минимуму потери, если позже идея окажется неудачной. По результатам первого этапа вы сможете оценить, стоит ли добавлять на сайт новые возможности и/или даже использовать доход от него для финансирования дальнейшего развития. Поэтапное создание сайта позволит как минимум сделать сайт работающим и начать обрабатывать заказы гораздо раньше, чем разработка сайта со всеми функциями с самого начала.

Но даже по завершении всех запланированных этапов ваша работа не заканчивается. При планировании большого проекта важно учесть в планах возможности неизбежного дальнейшего развития. Из главы 2, когда мы начнем заниматься техническими вопросами создания сайтов электронной коммерции, вы узнаете, как проектировать архитектуру веб-сайтов для обеспечения гибкости дальнейшего развития.

Если вы создаете сайты для клиентов, они, скорее всего, захотят оставлять за собой возможность выбора. Разбиение процесса на этапы сделает работу с вами более комфортной для клиентов. Они будут видеть, что вы действительно добиваетесь определенных результатов, и если почему-то решат прекратить дальнейшую разработку, вы сможете легко завершить проект в конце текущего этапа.

Этап I. Базовая платформа сайта

Главы 2–11 посвящены созданию базовой платформы сайта и организации электронного каталога товаров. Сначала мы создадим базовую архитектуру сайта и решим, как будут взаимодействовать его компоненты. Затем мы встроим в эту архитектуру каталог товаров. На этом этапе вы узнаете, как

- создать базу данных для хранения каталога товаров, содержащего названия отделов, категорий и товаров;
- написать код на SQL (Structured Query Language) и PHP (Hypertext Preprocessor) для доступа к этой базе данных и обеспечения работы каталога;
- добавить в каталог товаров данные, определяющие атрибуты этих товаров, например цвет и размер;
- реализовать базовую функциональность поиска;

- обеспечить взаимодействие сайта с поисковыми системами и уменьшить количество ошибок в URL-ссылках и перенаправлениях;
- принять платежи с помощью PayPal Website Payments Standard;
- создать закрытый раздел сайта для администраторов, позволяющий им управлять каталогом.

Создав каталог товаров, вы увидите, как можно продавать товары из этого каталога с помощью корзины покупателя и системы обработки заказов PayPal, которая будет обрабатывать транзакции с кредитными картами и отправлять вам сведения о заказах. Сами заказы будут обрабатываться вручную, но на ранних стадиях развития электронного магазина на ручную обработку заказов будет уходить меньше времени, чем на разработку автоматизированной системы для их обработки.

Этап II. Создание собственной корзины покупателя

Использование механизмов, предоставляемых PayPal, не представляет сложностей, но такой подход лишает вас нескольких важных возможностей. Например, вы не можете задать визуальное оформление корзины покупателя в PayPal, а если вы реализуете собственную, то сможете интегрировать ее в свой сайт, как сочтете нужным.

Это важное преимущество, но есть еще более существенные моменты. Например, реализовав собственную корзину покупателя, вы сможете хранить заказы в собственной базе данных и анализировать их, чтобы узнать больше о своих клиентах. Кроме того, немного постаравшись, можно сделать корзину платформой для предложения покупателям дополнительных товаров. Сколько раз вы сами покупали в последний момент то, что предлагалось вам возле кассы магазина? Такой же подход действует и в электронных магазинах. Создав собственную корзину и кассу, вы можете делать покупателям дополнительные предложения. Можно даже анализировать содержимое корзины и делать предложения, исходя из результатов этого анализа.

Итак, из глав 12–15 вы узнаете, как

- создать собственную корзину покупателя;
- передать заказы PayPal для обработки транзакций с кредитными картами;
- добавить функциональность AJAX в каталог товаров и корзину для улучшения взаимодействия покупателей с сайтом;
- создать страницу управления заказами;
- реализовать систему рекомендации товаров.

В конце второго этапа ваш сайт будет полностью готов к работе. Вы можете оставить его в таком состоянии или добавить элементы в существующую систему платежей на основе PayPal. Но если ваш электронный магазин будет развиваться успешно, рано или поздно вам придется реализовать собственную систему работы с кредитными картами. Это непросто, и вам придется уделять существенное внимание вопросам безопасности и надежности своего сайта.

Этап III. Обработка заказов и дополнительные возможности

В основе электронной торговли лежат механизмы обработки заказов и работы с кредитными картами. Именно наличие этих механизмов отличает сайты электронных магазинов от обычных веб-сайтов. До сих пор мы использовали механиз-

мы, предоставляемые PayPal, но у такого подхода есть свои ограничения, и однажды вам понадобятся собственные реализации этих механизмов.

Вот в чем вы можете выиграть, отказавшись от использования PayPal.

- *Цена.* Использование PayPal обходится недорого, но за дополнительные услуги приходится доплачивать. Переход к более простой системе работы с кредитными картами может снизить цену каждой транзакции (хотя и не обязательно), однако на разработку такой системы потребуются время и деньги.
- *Свобода.* PayPal использует весьма строгий набор правил и условий и работает только с жителями ограниченного набора стран. Создав собственную систему для работы с кредитными картами, вы будете вольны устанавливать такие правила и условия, какие сочтете удобными. Например, вы можете реализовать прием платежей с помощью специальных карт наподобие английских Switch.
- *Интеграция.* Если вы обрабатываете транзакции и заказы с помощью собственной системы, можете интегрировать магазин и склад сколь угодно тесно. Вы можете даже реализовать автоматическое обращение к сторонним поставщикам и передачу им заказов на поставку товаров непосредственно покупателю.
- *Информативность.* Обрабатывая весь заказ самостоятельно, вы можете записать и рассортировать всю полученную вами информацию, а позже воспользоваться ею в маркетинговых или исследовательских целях.

Интегрировав обработку заказов со складом, центром формирования отправок или напрямую с поставщиками, вы можете заметно снизить накладные расходы, а также сократить штат сотрудников или позволить вашей компании увеличить обороты, не нанимая дополнительных работников.

Собирая информацию о покупателях, вы можете определить, что нужно сделать, чтобы продавать больше. В самом простом варианте вы можете рассылать по почтовым адресам покупателей сообщения о новых товарах. Кроме того, можно анализировать сочетания покупаемых товаров и на основании этих данных формировать направленные рекламные кампании.

Итак, из глав 16–22 вы узнаете, как:

- создать модуль профилей покупателей, который позволит покупателям заходить на ваш сайт под своими профилями и просматривать детали своих предыдущих заказов;
- позволить пользователям добавлять отзывы о товарах;
- интегрировать службы Amazon.com в свой сайт с помощью веб-служб XML;
- установить безопасные соединения с помощью SSL (Secure Sockets Layer), защищая данные пользователей от перехвата при передаче через Интернет;
- работать с кредитными картами с помощью DataCash, Authorize.net и PayPal Website Payments Pro (ранее известной как VeriSign Payflow Pro);
- безопасно хранить номера кредитных карт в базе данных.

Третий этап — самый сложный технически, он потребует долгой и аккуратной работы. Однако к его концу у вас будет работающий сайт электронной коммерции с каталогом товаров, поддерживающим поиск, с корзиной покупателя, кассой и готовой системой обработки заказов.

Сайт TShirtShop

Как говорилось выше, в качестве примера мы будем создавать сайт электронного магазина TShirtShop, торгующего футболками. На рис. 1.1 показано, как этот сайт будет выглядеть в какой-то момент второго этапа разработки.

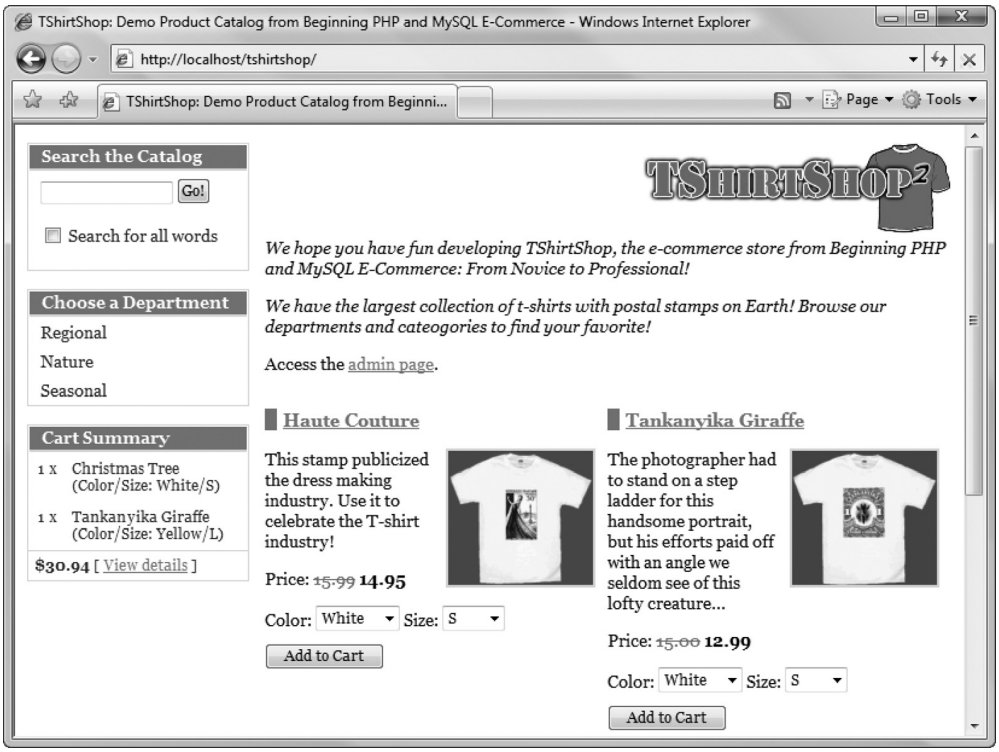


Рис. 1.1. Сайт TShirtShop на втором этапе развития

ПОДСКАЗКА

Ссылку на онлайн-версию TShirtShop можно найти по адресу `http://www.cristiandarie.ro/php-mysql-ecommerce-2/`. Огромная благодарность сотрудникам Going Postal, позволившим мне наполнить каталог на сайте TShirtShop данными об их продукции.

В этой книге мы будем считать, что клиент, для которого вы разрабатываете сайт, — это реальная компания, рассылающая заказы по почте и располагающая хорошей базой покупателей. Компания не является новичком в бизнесе и заказала вам сайт, чтобы упростить своим покупателям процесс составления заказов и, соответственно, заставить их тратить на покупки больше денег.

Зная это, мы предлагаем создавать и запускать сайт TShirtShop поэтапно по следующим причинам.

- Компания вряд ли сразу начнет получать массу заказов, а нам нужно, чтобы начальные затраты на сайт были минимальными.

- Компания уже приспособилась к обработке заказов вручную, поэтому ручная обработка заказов с помощью PayPal проблем не вызовет.
- Компания не хочет тратить кучу денег на создание крупного сайта, чтобы в конце концов выяснить, что покупатели предпочитают оформлять заказы по почте! Или, например, в конце первого этапа выяснится, что сайт — это не то, что нужно компании, и дальнейшая разработка нецелесообразна. В любом случае малые начальные затраты сделают ваше предложение более привлекательным для компании (и, возможно, позволят вам по окончании разработки получить от нее больше денег).

Поскольку эта компания уже работает, у нее, вероятно, есть банковские счета и механизмы для работы с кредитными картами. Поэтому для нее может оказаться полезным быстрый переход к третьему этапу — реализации эффективных механизмов работы с кредитными картами.

Резюме

В этой главе мы рассмотрели потенциальные выгоды для торговой компании от создания сайта электронного магазина. В реальном, часто враждебном мире коммерции, где важно сосредоточиваться на краткосрочной прибыли и минимизации рисков, электронный магазин может помочь вам в следующем.

- Увеличить количество покупателей
- Увеличить доход с одного покупателя
- Уменьшить накладные расходы на выполнение заказов

Мы составили состоящий из трех этапов план, позволяющий в конце каждого этапа получать работающий сайт электронного магазина, который можно расширять и дополнять новыми возможностями.

Итак, вы представили план разработки сайта клиенту. В следующей главе вы начнете создавать этот сайт (разумеется, при условии, что клиент принял этот план).

Глава 2

Закладываем фундамент

Итак, вы убедили клиента в том, что можете создать хороший веб-сайт, который существенно повысит эффективность его компании. Придя в себя после вечеринки по поводу заключения контракта, вам придется подумать, как воплотить в жизнь свои обещания. Как всегда, при попытке составить детальный план действий все начинает выглядеть сложнее, чем казалось на первый взгляд.

Чтобы обеспечить успех проекта, необходимо четко определить, как вы будете реализовывать то, что от вас требуется по контракту. Вам нужно, чтобы проект продвигался быстро и гладко, но самое главное — чтобы заказчик был доволен вашей работой. Соответственно, ваша задача — обеспечить сайту постоянно растущий поток посетителей. Для этого нужно сделать сайт удобным в использовании, функциональным и привлекательным.

Требования весьма жесткие, но для современных сайтов электронных магазинов это нормально. Чтобы довести до максимума вероятность успеха, мы постараемся учесть как можно больше технических требований и реализовать механизмы и элементы, которые позволят вносить дальнейшие усовершенствования с минимумом усилий. В целом наши задачи в этой главе таковы.

- Проанализировать проект с технической точки зрения
- Выбрать архитектуру для создаваемого приложения
- Решить, какие технологии, языки программирования и инструменты использовать
- Составить правила именования и оформления кода

ЗАМЕЧАНИЕ

Учтите, что в этой и нескольких последующих главах мы будем рассматривать материал весьма сжато, и они могут оказаться сложными для вас, если у вас нет опыта работы с PHP или MySQL. В таком случае мы можем только посоветовать предварительно почитать другие книги, например 3-е издание книги Мэтта Зандстры *Освой самостоятельно PHP за 24 часа* (ИД "Вильямс", 2008 г.).

Кроме того, мы очень рекомендуем вам использовать определенную методику управления проектом, чтобы максимизировать шансы на успешное завершение проекта в установленные сроки и без перерасхода денег. Большинство теорий управления проектами подразумевают, что вы как разработчик и ваш клиент подписали документ, содержащий начальные требования или спецификацию необходимого товара. Этот документ вы можете использовать в качестве ориентира в ходе разработки; если клиент позже выдвинет дополнительные требования, вы будете вправе потребовать дополнительную оплату и время на их реализацию.

Дизайн для дальнейшего роста

Термин "дизайн" применительно к веб-приложению может означать множество разных понятий. В самом распространенном случае он означает процесс создания пользовательского интерфейса и визуального оформления веб-страниц.

Этот аспект критически важен для проекта в целом. Давайте посмотрим правде в глаза: пользователю вряд ли понравится бесцветный, неряшливо оформленный или неустойчиво работающий сайт независимо от того, насколько хороша использованная в нем технологическая основа. Даже случайно зайдя на такой сайт, посетитель в дальнейшем вряд ли на него вернется.

К сожалению, этот момент заставляет многих неопытных разработчиков уделять все внимание только внешнему оформлению сайта, оставляя минимум времени на проработку самого приложения — управляющего кода, базы данных и всего остального. Да, красивое оформление может привлечь покупателя, но именно функциональность может убедить его вернуться на сайт. Часто веб-сайт удается создать очень быстро, если есть список требований к нему, но если при создании он был плохо структурирован, в дальнейшем его очень неудобно поддерживать в рабочем состоянии.

В проекте любого размера написанию кода должны предшествовать кое-какие действия. Но независимо от того, сколько подготовительной работы вы проведете, при написании кода всегда возникают неожиданные сложности, новые требования и изменяющиеся правила. Может показаться, что проект просто невозможно завершить в поставленные сроки. Даже если неприятных неожиданностей не случается, разработчиков часто просят добавить в проект не предусмотренную изначально функциональность — обычно уже после того, как проект завершен и товар успешно используется. Это случится и с нашим примером — проектом TShirtShop, который мы будем реализовывать в три этапа, как описано в главе 1.

Мы расскажем, как создать веб-сайт, который не развалится при попытках добавить в него новые элементы или функции. Поскольку это книга по программированию, мы сосредоточимся не на оформлении сайта или вопросах маркетинга, а на создании кода, который заставляет сайт работать.

Само по себе словосочетание "дизайн кода" может иметь разные значения, например в это понятие входит составление правил именования для переменных, функций и прочих элементов кода. Но самое важное для нашего проекта — это выбор архитектуры. К архитектуре относится и разбиение кода на функциональные элементы (например, механизм поиска в каталоге). Разумеется, можно создать работающий проект из нескольких крупных компонентов, но проект из множества мелких, простых компонентов модернизировать и поддерживать гораздо проще.

Прежде чем говорить о самой архитектуре, нужно решить, чего мы от нее хотим.

Выполнение долгосрочных требований с минимумом усилий

Помимо того, что нам нужен быстро работающий веб-сайт, на каждом этапе проектирования мы будем сталкиваться с новыми требованиями.

Каждый раз при переходе к новому этапу желательно, чтобы мы могли использовать большую часть ранее написанного кода. Будет очень плохо, если для реализации новой функции придется переделывать весь проект целиком — и визуальную часть, и код. Повысить эффективность повторного использования кода можно, хорошо спланировав и структурировав приложение, чтобы новая функция стала всего лишь еще одним модулем, а не потребовала переделки всего кода.

При создании веб-сайта реализация *гибкой архитектуры*, состоящей из подключаемых компонентов, позволяет легко добавлять в проект новые возможности, например корзину покупателя, список отделов или возможность поиска товаров по каталогу. Такие возможности пишутся в виде отдельных модулей, а затем подключаются к проекту. Достижение хорошей гибкости проекта — одна из основных задач при выборе архитектуры, и в этой главе мы постараемся показать вам, как достичь такой гибкости. Вы увидите, что проект тем гибче, чем больше времени потрачено на его дизайн и написание кода, так что мы попытаемся выдержать баланс между приемлемыми затратами времени и получением удобоваримого кода.

Еще одно важное требование, общее для всех онлайн-приложений — *масштабируемость*. Она определяется как способность приложений линейно наращивать производительность при линейном росте выделенных для них ресурсов. Другими словами, в идеально масштабирующейся системе соотношение между количеством обрабатываемых запросов в единицу времени и количеством выделенных ресурсов будет постоянным независимо от количества клиентов и других факторов. Плохо масштабирующаяся система не может справиться с ростом количества клиентов независимо от того, сколько аппаратных ресурсов мы для нее выделим. А поскольку мы надеемся, что покупателей у нас с течением времени будет все больше, мы должны сделать систему масштабируемой, чтобы сайт всегда работал быстро и плавно.

Надежность — еще одно критически важное требование к приложениям в области электронной торговли. Применив эффективные механизмы обработки ошибок и хорошую реляционную СУБД, вы сможете обеспечить целостность данных и гарантировать, что большинство ошибок не скажутся на стабильности работы сайта.

Трехуровневая архитектура

В целом под архитектурой понимают разбиение на части кода приложения. Мы постараемся выполнить это разбиение так, чтобы код, относящийся к одной части приложения, был объединен в один логический уровень.

В частности, трехуровневая архитектура подразумевает разделение кода на три уровня.

- Уровень представления (presentation layer)
- Уровень логики приложения (бизнес-уровень, или business layer)
- Уровень данных (data layer)

Уровень представления содержит код, отвечающий за интерфейс веб-сайта, и логику, обеспечивающую взаимодействие сайта с пользователями. Этот уровень "оживляет" сайт, и его качество критически важно для успеха сайта. Поскольку наше приложение — веб-сайт, его интерфейсом будет набор динамических веб-страниц.

Уровень логики приложения (иногда называемый просто *средним уровнем*, или бизнес-уровнем) получает запросы от уровня представления и возвращает уровню представления ответы на эти запросы, руководствуясь управляющей логикой, размещенной на бизнес-уровне. Почти каждое событие, происходящее на уровне представления, приводит к обращению к уровню логики приложения, за исключением событий, с которыми уровень представления может разобраться сам (например, провести простую проверку введенных пользователем данных). Если, к примеру, пользователь обращается к механизму поиска товара в каталоге, уровень представления обращается к уровню логики приложения и требует сообщить, какие товары соответствуют критериям поиска, которые задал пользователь. В большинстве слу-

чаев уровень логики приложения должен обращаться к уровню данных за информацией, необходимой для ответов на запросы с уровня представления.

Уровень данных (иногда называемый уровнем базы данных) отвечает за управление данными приложения и передачу этих данных уровню логики приложения при получении запросов. В проекте приложения TShirtShop нам понадобится хранить данные о товарах (включая сведения о категориях и отделах), о пользователях, о корзинах покупателей и т.д. Почти любой запрос пользователя в конечном итоге приводит к обращению приложения к уровню данных (если только данные ранее не были кэшированы на уровне логики приложения или уровне представления), поэтому уровень данных должен использовать быструю и эффективную базу данных. В главах 4 и 5 мы рассмотрим создание базы данных, оптимизированной для достижения максимальной производительности.

Сами по себе уровни являются сугубо логическими единицами — код, относящийся к разным уровням, не обязательно должен физически размещаться в разных файлах. Вы можете размещать все приложение, все его уровни, на одном сервере, или размещать разные уровни на разных машинах, если приложение это позволяет. Из главы 22 вы узнаете, как интегрировать в приложение функциональность с других веб-сайтов с помощью веб-служб XML. Эти веб-службы позволяют выполнять интеграцию с использованием минимума специального кода.



Рис. 2.1. Простое представление трехуровневой архитектуры

Важное ограничение трехуровневой архитектуры — то, что информация должна поочередно передаваться между соседними уровнями. Уровень представления может обращаться к уровню логики приложения, но не к уровню данных. Уровень логики приложения — это мозг приложения, который может общаться и с уровнем данных, и с уровнем представления, координируя потоки данных. Если уровень представления будет напрямую общаться с уровнем данных, правила трехуровневой архитектуры будут нарушены.

Возможно, эти правила покажутся вам искусственными ограничениями, но, чтобы получить выигрыш от использования архитектуры, вы должны следовать ее правилам. Правильное применение трехуровневой архитектуры обеспечит простоту модернизации сайта и добавления в него новых возможностей, а кроме того, позволит контролировать доступ к данным. Сейчас это может показаться ненужным и излишним, но в будущем такой контроль очень пригодится, когда придется менять функциональность сайта.

На рис. 2.1 изображена диаграмма потоков данных в приложении с трехуровневой архитектурой.

Простой пример использования трехуровневой архитектуры

Чтобы разобраться, как данные передаются между уровнями и преобразуются, давайте рассмотрим простой пример. А чтобы этот пример был полезен для нашего проекта TShirtShop, он будет основан на реальной ситуации, которая может возникнуть при общении пользователя с приложением.

Как и в большинстве приложений электронных магазинов, в TShirtShop будет корзина покупателя, которую мы подробно рассмотрим позже. А пока что вам достаточно знать, что покупатель сможет добавлять товары в корзину, щелкая на

кнопке Add to Cart на веб-странице. На рис. 2.2 показан обмен информацией, происходящий в приложении после нажатия этой кнопки.

На шаге 1 пользователь щелкает на кнопке Add to Cart для определенного товара. На шаге 2 уровень представления (реагирующий на щелчки на этой кнопке) передает уровню логики приложения запрос на добавление данного товара в корзину покупателя. На шаге 3 уровень логики приложения получает этот запрос и обрабатывает его, приказав уровню данных обновить содержимое корзины покупателя, добавив в нее выбранный товар. К уровню данных необходимо обращаться, потому что он отвечает за хранение данных веб-сайта (включая корзину покупателя) и доступ к ним.

На шаге 4 уровень данных обновляет содержимое базы данных и сообщает уровню логики приложения, что обновление прошло успешно. На шаге 5 уровень логики приложения обрабатывает сообщение от уровня данных и/или коды ошибок, полученные от него при обновлении базы данных. Затем уровень логики приложения сообщает результаты обработки уровню представления.

На шаге 6 уровень представления генерирует обновленное отображение корзины покупателя. На шаге 7 генерируется соответствующий результатам работы приложения HTML-код, который передается клиенту. Этот код отображает в окне браузера обновленное содержимое корзины покупателя.

Заметьте, что в этом примере уровень логики приложения не выполняет какой-то сложной обработки данных, и сама управляющая логика на уровне приложения довольно проста. Однако если вам понадобится изменить эту логику, достаточно будет модифицировать только уровень логики приложения. Например, если вам нужно, чтобы управляющая логика позволяла добавлять товары в корзину покупателя, только если они есть в запасе, уровень логики приложения делал бы еще один запрос к уровню данных, проверяя, есть ли в запасе выбранный товар. Второй запрос, с приказом обновить содержимое корзины покупателя, делался бы только при наличии товара. Но в любом случае уровень представления отвечает за уведомление пользователя о результатах щелчка на кнопке Add to Cart.

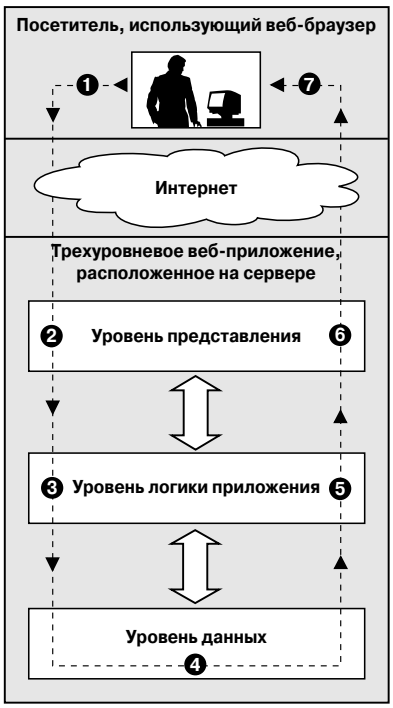


Рис. 2.2. Взаимодействие уровней в приложении с трехуровневой архитектурой

Смысл данных

Интересно посмотреть, как разные уровни приложения воспринимают одни и те же данные. Для уровня данных сами данные ничего не значат — это просто числа и строки, которые он сохраняет в базе данных или считывает из нее по запросам от уровня логики приложения. Он понятия не имеет о том, что такое склад, товары и корзина покупателя. Например, отсутствие товара на складе для него есть просто число 0 в одной из ячеек базы данных — 32-битовое целое число 0.

Данные обретают смысл только на уровене логики приложения. Когда этот уровень запрашивает у уровня данных количество единиц товара на складе и получает

ответ “0”, он интерпертирует этот ответ как “товара на складе нет”. Этот ответ преобразуется в форму, понятную уровню представления, и уровень представления докладывает о результатах запроса пользователю, например, сообщением “Извините, в данный момент заказать этот товар нельзя”.

Возможно, вы и не захотите запрещать пользователю добавлять в корзину товары, которых нет в запасе, но этот пример (показанный на рис. 2.3) достаточно хорошо иллюстрирует смысл данных на разных уровнях приложения с трехуровневой архитектурой.



Рис. 2.3. Смысл данных на разных уровнях приложения

Распределение функций между слоями

Поскольку в каждом уровне приложения содержится своя управляющая логика, иногда трудно четко разделить приложение на уровни. Например, в приведенном выше сценарии вместо организации взаимодействия между слоями можно было бы написать одну процедуру `add_product_to_cart`, которая добавила бы товар в корзину, только если он есть в запасе.

В этом сценарии часть управляющей логики перемещается из уровня логики приложения на уровень данных. В других случаях управляющая логика может размещаться и на уровене логики приложения, и на уровне данных. Для большинства реальных приложений не существует единственно верного способа разделения на уровни, и разделение приходится выполнять, исходя из личных предпочтений разработчиков или каких-то внешних ограничений.

Кроме того, бывают случаи, когда вы знаете *правильный* (с точки зрения архитектуры) способ реализации какой-то функции, но этот способ не является оптимальным с точки зрения производительности приложения. В целом, если за счет нарушения разбиения на уровни можно повысить производительность приложения, это следует сделать (например, можно добавить какие-то механизмы из уровня логики приложения на уровень данных), если такое нарушение не сделает при-

ложение плохо модернизируемым. В противном случае лучше держать всю управляющую логику на уровне приложения — это позволит сделать приложение более простым в обслуживании.

Еще один совет: никогда не пытайтесь напрямую обращаться к уровню данных из уровня представления. Эта ошибка приводит к получению запутанных, трудно модернизируемых и плохо отлаживаемых приложений. Во многих учебниках показывается, как простые программы напрямую обращаются к базам данных. В таких программах весь код обычно представляет собой один небольшой исходный файл, и этот код не делится на уровни. Но это всего лишь маленькие учебные программы, а не серьезные коммерческие приложения, которые должны быть гибкими и масштабируемыми.

Трехуровневая архитектура для TShirtShop

Использование трехуровневой архитектуры в проекте TShirtShop упростит выполнение задач, перечисленных в начале главы. Правила оформления кода, которые могут показаться излишне жесткими, в длительной перспективе обеспечат приложению гибкость и расширяемость.

Разделение больших частей приложения на маленькие отдельные компоненты сделает код более удобным для многократного использования. В дальнейшем, реализуя в приложении новые функции, вы будете обнаруживать, что эти функции можно собирать из уже существующих фрагментов кода.

Еще одно преимущество трехуровневой архитектуры — при правильной реализации она делает систему в целом устойчивой к изменениям. Когда изменяется часть одного из уровней, остальные уровни обычно остаются неизменными, даже если изменения весьма заметны. Например, если вы решите вместо MySQL реализовать уровень данных с помощью PostgreSQL, вам придется поменять только уровень данных (и, возможно, немного модифицировать уровень логики приложения).

А почему бы не использовать больше уровней?

Трехуровневая архитектура, которую мы рассматриваем, — это частный (хотя и наиболее популярный) вариант *n*-уровневой архитектуры, в которой приложение делится на *n* логических слоев. В больших и сложных проектах иногда есть смысл разделить уровень логики приложения на несколько более мелких уровней. Однако для нашего веб-сайта вполне достаточно трехуровневой архитектуры, которая позволит пользоваться преимуществами разбиения кода на уровни без лишней возни с написанием поддерживающего архитектуру кода.

Возможно, более сложная архитектура позволит сделать приложение еще более гибким и масштабируемым, но вам придется потратить гораздо больше времени на проектирование, прежде чем вы сможете приступить к написанию кода. Трехуровневая архитектура, пожалуй, оптимальна для проектов средней сложности наподобие веб-сайта TShirtShop.

Возможно, вы зададитесь противоположным вопросом “А почему бы не использовать меньше уровней?” Двухуровневая архитектура, более известная как *архитектура “клиент/сервер”*, вполне подходит для несложных проектов. В целом она требует меньше времени на планирование приложений и позволяет быстрее приступить к написанию кода; однако в дальнейшем такие приложения труднее обслуживать и модернизировать. А поскольку мы планируем в дальнейшем расширять возможности приложения, мы не будем использовать эту архитектуру.

Итак, выбрав базовую архитектуру приложения, давайте посмотрим, какие технологии и инструменты можно применить, чтобы воплотить ее в жизнь. Сейчас

мы кратко обсудим эти технологии и инструменты, а в главе 3 заложим основы уровня данных и уровня представления, создав первую страницу веб-сайта и базу данных нашего приложения. Реальную функциональность для каждого из трех уровней мы начнем создавать в главе 4, когда займемся каталогом товаров для электронного магазина.

Выбор технологий и инструментов

Независимо от выбранной архитектуры в каждом проекте по разработке приложений нужно выбирать какие-то технологии, языки программирования и инструменты, с помощью которых будут создаваться эти приложения. При этом нужно учитывать, что выбор может ограничиваться требованиями заказчиков.

В этой книге мы будем создавать веб-приложение с использованием PHP 5, MySQL 5 и связанных с ними технологий. Это распространенные технологии, но это не обязательно значит, что они будут наилучшим выбором для любого проекта в любых обстоятельствах. Кроме того, часто клиент требует, чтобы при разработке приложения использовались определенные технологии и инструменты. Этап анализа требований, который присутствует в большинстве проектов по разработке программ, как раз и предназначен для выбора технологий, которые будут использоваться.

Хотя в этой книге и подразумевается, что у вас есть опыт использования PHP и MySQL, мы кратко рассмотрим их, чтобы разобраться, как их можно применить в нашем проекте и в трехуровневой архитектуре в целом.

Использование PHP для генерации динамического веб-контента

PHP — это технология с открытым исходным кодом, предназначенная для генерации динамического, интерактивного веб-контента. Как говорится в ее официальном описании (на сайте `http://www.php.net`), "PHP — широко используемый интерпретируемый язык общего назначения, особенно удобный для веб-приложений, который может встраиваться в HTML".

"PHP" расшифровывается как "PHP": "Hypertext Preprocessor" (да, это рекурсивная аббревиатура), и его можно бесплатно загрузить с официального сайта. История PHP началась примерно с 1994 года и представляет собой пример впечатляющего успеха. Среди факторов, обусловивших этот успех, можно упомянуть следующие.

- PHP бесплатен. Если его использовать в сочетании с серверными программами под управлением Linux, он представляет собой очень экономически эффективную платформу для генерации динамического веб-контента.
- PHP легче изучать, чем большинство других интерпретируемых языков.
- Существует большое сообщество разработчиков, использующих PHP. Есть множество разнообразных библиотек для PHP (например, их можно найти в репозитории PEAR или на сайте `http://www.phpclasses.org`), и в них регулярно добавляются новые возможности.
- PHP хорошо работает с разными веб-серверами и операционными системами (семейств Unix, Windows или MacOS).

Однако PHP — отнюдь не единственный серверный интерпретатор, позволяющий генерировать динамические веб-страницы. Среди его конкурентов можно

упомянуть JavaServer Pages (JSP), Perl, ColdFusion и ASP.NET. У этих технологий есть свои особенности, но в целом они предназначены для решения одних и тех же задач. Например, страницы, сгенерированные с помощью любой из этих технологий, состоят из простого HTML, отвечающего за статичную часть страниц (шаблон) и кода, генерирующего динамическую часть.

ЗАМЕЧАНИЕ

Если вам интересно, прочтите 3-е издание книги Мэтью Мак-Дональда и др. *Microsoft ASP.NET 3.5 с примерами на C# 2008 и Silverlight 2 для профессионалов* ("Диалектика", 2009 г.). В ней рассматривается создание сайтов с помощью ASP.NET, C# и SQL Server.

Разделение кода и оформления с помощью Smarty

Из-за простоты PHP многие использующие его разработчики начинают писать код приложения, не продумав его архитектуру и не сформировав базовую платформу.

Еще хуже то, что PHP можно легко смешивать с HTML-кодом веб-страниц, поскольку PHP по умолчанию не содержит механизмов для четкого разделения кода на PHP и кода на HTML.

Смешивание кода на PHP и HTML приводит к двум довольно неприятным проблемам.

- В результате такого смешивания вы получаете объемный, запутанный и плохо поддающийся модернизации код. Возможно, вы видели длинные непонятные листинги с PHP и HTML, в которых сами их авторы не могут разобраться через неделю после написания.
- В таких файлах вынуждены копаться и программисты, и дизайнеры вебсайта, что существенно усложняет взаимодействие разработчиков. Кроме того, дизайнеры часто вносят в код ошибки, занимаясь оформлением вебстраниц.

Эти проблемы привели к появлению систем шаблонов, позволяющих разделять логику уровня представления и HTML-код. Smarty (`http://smarty.php.net`) — это одна из самых популярных и эффективных систем шаблонов для PHP. Ее основное назначение — именно разделение логики приложения (кода на PHP) и кода представления (HTML).

Такое разделение позволяет программистам и дизайнерам шаблонов независимо работать над одним и тем же приложением. Программист может менять логику в PHP-коде, не трогая файлы шаблонов, а дизайнер шаблонов может менять эти шаблоны, не беспокоясь о том, как работает код, в котором они используются.

На рис. 2.4 показаны взаимоотношения между шаблонами Smarty и соответствующими им подключаемыми файлами.

Дизайн-шаблон Smarty (файл `.tpl` с тегами HTML и тегами, специфичными для Smarty) и соответствующий ему подключаемый файл Smarty (файл `.php` с кодом, взаимодействующим с шаблоном) образуют *компонентный шаблон Smarty*.

На практике мы не будем создавать подключаемый файл Smarty для каждого дизайн-шаблона, как показано на рис. 2.4. Вместо этого мы создадим обобщенный подключаемый файл, который будет интегрироваться со всеми дизайн-шаблонами, загружая необходимые объекты представления. Объекты представления — это классы, снабжающие файлы шаблонов необходимыми данными.

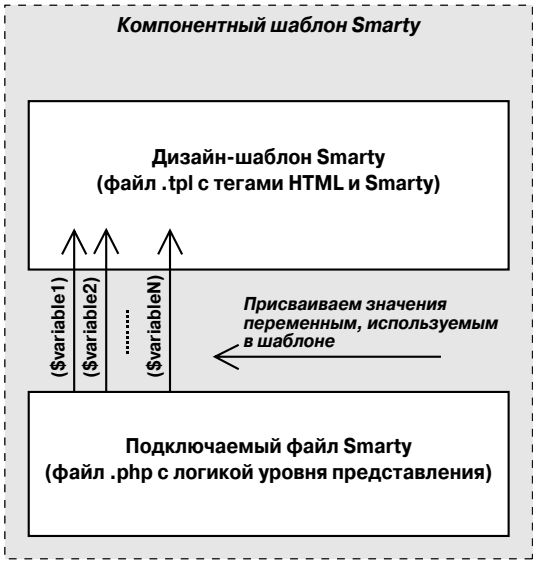


Рис. 2.4. Шаблоны Smarty и подключаемые файлы

Вы узнаете больше о Smarty, создавая наш сайт электронного магазина. Если вам нужен краткий вводный курс, прочтите Smarty Crash Course по адресу `http://smarty.php.net/crashcourse.php`. Если же вам требуется нечто более подробное, могу порекомендовать книгу *Smarty PHP Template Programming and Applications* (Hasin Hayder, J. P. Maia и Lucian Gheorghe; Packt Publishing, 2006).

ЗАМЕЧАНИЕ

Применение в проекте Smarty или другой системы шаблонов потребует написания кое-какого дополнительного кода и, разумеется, времени на обучение использованию этой системы. Но все же рекомендую вам попробовать это — в конечном счете вы немало выиграете от использования системы шаблонов.

Как насчет альтернатив?

Smarty — отнюдь не единственная существующая система шаблонов для PHP. Если вас интересуют альтернативы, попробуйте задать в Google поиск на тему "PHP template engines" или найти статью *Top 25 PHP Template Engines* (автор — Justin Silverton) с помощью того же Google.

В целом все системы шаблонов используют один и тот же базовый принцип, и мы выбрали Smarty для нашего проекта потому, что он обеспечивает очень хорошую производительность, развитую функциональность (например, компиляцию и кэширование шаблонов) и широко распространен.

Использование MySQL на уровне данных

Большую часть данных, которые будут видеть посетители нашего сайта, наше приложение должно будет извлекать из реляционной базы данных. Система управления реляционными базами данных (РСУБД) — это сложная программа, обеспечивающая максимально надежное хранение информации и быстрый доступ

к ней. На сайте TShirtShop в реляционной базе данных будут храниться сведения о товарах, отделах, пользователях, корзинах покупателей и т.д.

Для совместного использования с PHP подходит множество РСУБД, включая MySQL, PostgreSQL, Oracle и др. Однако, как показывает практика, чаще всего совместно с PHP используют именно MySQL.

MySQL — самая популярная в мире РСУБД с открытым исходным кодом, она бесплатна (для некоммерческого применения), быстро работает и весьма надежна. Еще одно ее преимущество — то, что многие провайдеры веб-хостинга предлагают доступ к базам данных MySQL, заметно упрощая жизнь разработчикам веб-приложений. Поэтому мы будем использовать именно MySQL при разработке электронного магазина TShirtShop.

Для обращения к реляционной базе данных мы будем использовать язык SQL (Structured Query Language — язык структурированных запросов). Каждая РСУБД использует свой диалект SQL, и, если вы решите использовать вместо MySQL другую РСУБД, вам, вероятно, придется модифицировать запросы, приведенные в листингах в этой книге.

Установка связи с MySQL

Чтобы обратиться к серверу РСУБД, нужно составить SQL-запрос, передать его серверу и получить от сервера ответ. SQL-запрос может содержать любое требование, касающееся базы данных, например "создать таблицу", "вывести список отделов", "удалить из базы данных товар 223" или "выдать всю информацию о желтых футболках в каталоге".

Независимо от того, что требует SQL-запрос от базы данных, нам нужно как-то передать его MySQL. MySQL поставляется с простым интерфейсом, использующим командную строку, который позволяет выполнять запросы и получать результаты их выполнения в текстовой форме. Если этот интерфейс покажется вам неудобным, не волнуйтесь: для него есть множество альтернатив. В частности, есть несколько товаров от сторонних разработчиков, позволяющих работать с базами данных MySQL с помощью удобного графического интерфейса. Многие провайдеры веб-хостинга предлагают работать с базами данных с помощью phpMyAdmin (это самый распространенный интерфейс для веб-клиентов MySQL). Очень советую вам познакомиться с этим инструментом. Вы можете также использовать какой-нибудь из графических клиентов, например Toad for MySQL (`http://www.quest.com/toad-for-mysql/`).

Интеграция с базой данных с помощью PDO

PDO (PHP Data Objects — объекты данных PHP) — это библиотека для доступа к данным, поставляемая вместе с PHP начиная с версии 5.1. Она также доступна в виде расширения PECL для PHP 5.0 (PECL — это репозиторий расширений для PHP, доступный по адресу `http://pecl.php.net/`). Официальное руководство по PDO, включая инструкции по установке, доступно по адресу `http://www.php.net/pdo`.

PDO предлагает единый способ доступа к различным источникам данных. Эта библиотека упрощает портирование приложений между разными платформами и повышает гибкость приложений, поскольку при изменениях уровня данных изменения в коде остаются минимальными (во многих случаях достаточно только изменить строку, создающую соединение с базой данных).

Познакомившись с уровнем абстрагирования данных в PDO, вы сможете использовать одни и те же приемы в разных проектах, предъявляющих разные требования к базам данных.

Чтобы продемонстрировать разницу между доступом к базе данных из обычного PHP-приложения и приложения, использующего PDO, давайте рассмотрим два фрагмента исходного кода.

ЗАМЕЧАНИЕ

Если вы не понимаете, что делает этот код, ничего страшного. В последующих главах мы подробно проанализируем его работу.

Вот код, обращающийся к базе данных MySQL с помощью встроенных функций PHP.

```
// Соединяемся с MySQL
$link = mysql_connect('localhost', $username, $password);

if (!$link)
{
   die ('Could not connect: ' . mysql_error());
}

$db_selected = mysql_select_db('tshirtshop', $link);

if (!$db_selected)
{
   die ('Could not select database : ' . mysql_error());
}

// Выполняем SQL-запрос
$queryString = 'SELECT * FROM product';

$result = mysql_query($queryString);
if (!$result)
{
  die ('Query failed : ' . mysql_error());
}

// Закрываем соединение
mysql_close($link);
```

ЗАМЕЧАНИЕ

Если вы все же хотите вместо PDO применять встроенные функции расширения PHP MySQL, подумайте о том, чтобы использовать улучшенную версию этого расширения (mysqli). Более подробную информацию об этом расширении можно найти по адресу `http://www.php.net/manual/en/ref.mysqli.php`.

А вот код, выполняющий те же действия, но использующий PDO.

```
try
{
   // Создаем новый экземпляр PDO
   $database_handler =
      new PDO('mysql:host=localhost;dbname=tshirtshop',
      $username, $password);

   // Составляем SQL-запрос
   $sqlQuery = 'SELECT * FROM product';

   // Выполняем SQL-запрос
```

```
    $statement_handler = $database_handler->query($sqlQuery);

    // Получаем данные
    $result = $statement_handler->fetchAll(PDO::FETCH_ASSOC);

    // Очищаем экземпляр объекта PDO
    $database_handler = null;
}
catch (PDOException $e)
{
    /* Если что-то пойдет не так, мы можем перехватить исключение
       из объекта, вывести сообщение и прекратить выполнение
       сценария */
    print 'Error! <br />' . $e->getMessage() . '<br />';
    exit;
}
```

Версия кода, использующая PDO, длиннее, но в ней используется эффективный механизм обработки ошибок — это очень полезно при отладке приложения. Если вам непонятно, о чем идет речь, подождите, пока мы не начнем реально использовать PDO в последующих главах и подробно анализировать работу кода.

Кроме того, если вы используете PDO, вам не придется менять код, отвечающий за доступ к данным, если вы, например, решите вместо MySQL использовать PostgreSQL. В то же время первый фрагмент кода пришлось бы переделывать целиком (в частности, использовать функции `pg_connect` и `pg_query` вместо `mysql_connect` и `mysql_query`). Кроме того, некоторые функции для работы с PostgreSQL используют параметры, которых нет в функциях для MySQL.

Применяя уровень абстрагирования данных (например, PDO), вам, вероятно, придется менять строку соединения при смене базы данных. Учтите, что пока что мы говорим только о PHP-коде, взаимодействующем с базой данных. Если новая база данных будет использовать другой диалект SQL, вам, вероятно, придется изменять и строки запросов.

ЗАМЕЧАНИЕ

Чтобы свести к минимуму необходимость изменения запросов, старайтесь делать их как можно более соответствующими стандарту SQL. Более подробно мы поговорим об SQL в главе 4.

MySQL и трехуровневая архитектура

Думаю, вам уже ясно, что MySQL относится к уровню данных. Но если до сих пор вам не приходилось работать с базами данных, вы могли не понять, что MySQL — это не просто хранилище для данных. Кроме самих данных MySQL может хранить и управляющую логику в виде хранимых процедур, а также контролировать целостность данных, проверять их корректность и многое другое.

Для общения с MySQL вам не придется применять SQL — язык, используемый в реляционных базах данных. SQL позволяет отдавать РСУБД приказы наподобие “выдай мне детали последних десяти заказов” или “удали товар номер 228”.

Хотя и можно составлять SQL-запросы непосредственно в коде на PHP и сразу же отправлять их на выполнение, это не лучшее решение, поскольку оно делает программу уязвимой к ошибкам, снижает производительность и усложняет последующие модернизации. В нашем проекте мы будем хранить управляющую логику уровня данных в *функциях базы данных* (database functions).

Код, приведенный в этой книге, проверялся на MySQL 5.0 и MySQL 5.1. Место сервера MySQL в трехуровневой архитектуре показано на рис. 2.5.

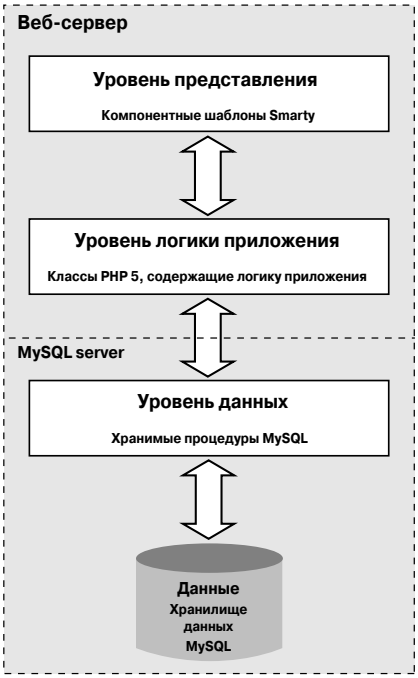


Рис. 2.5. Место MySQL в архитектуре приложения

Выбор стандартов именования и оформления кода

Возможно, на первый взгляд, стандарты именования и оформления кода покажутся вам не слишком важными, но их не стоит игнорировать. Если код писать, не следуя правилам оформления, в результате получится приложение, в котором будет очень сложно разобраться, не говоря уже о том, чтобы отлаживать его или модифицировать. Но если вы будете четко следовать правилам оформления, вы сможете с самого начала считать свое приложение наполовину задокументированным. Это важный вклад в будущее проекта, особенно если над ним одновременно работает много людей.

ПОДСКАЗКА

В одних компаниях есть официальные стандарты именования и оформления кода, а в других вы можете использовать такие стандарты, какие сочтете нужным. В любом случае самое главное — *единообразие в оформлении всего написанного кода*. Еще один важный совет: обязательно пишите комментарии к своему коду (чем больше, тем лучше).

Правила именования относятся ко многим элементам кода в проекте просто потому, что почти у всех элементов кода есть имена: сам проект, файлы, классы, переменные, методы, параметры методов, таблицы базы данных, столбцы этих

таблиц и т.д. Если вы не будете следовать каким-то правилам в задании имен, через неделю вы не поймете ни строчки из кода, который сами написали.

При разработке TShirtShop мы использовали набор правил именования, популярный у разработчиков на PHP. Некоторые правила из этого набора приведены ниже и проиллюстрированы фрагментом кода.

- Имена классов и методов записываются слитно, первые буквы слов в именах — заглавные (например, `WarZone`).
- Имена открытых атрибутов классов записываются по тем же правилам, но в начале этих имен ставится буква `m`, например `$mSomeSoldier`.
- Имена закрытых атрибутов классов записываются так же, как имена открытых, но в начале этих имен ставится знак подчеркивания, например `$_mSomeOtherSoldier`.
- Имена методов записываются слитно, слова в этих именах, кроме первого слова, записываются с прописной буквы, например `$someEnemy`, `$someOtherEnemy`.
- Имена переменных записываются строчными буквами, слова в них разделяются знаками подчеркивания, например `$master_of_war`.
- Элементы баз данных именуются по тем же правилам, что и переменные, например столбец `department_id`.
- Старайтесь использовать в коде отступы фиксированной величины (скажем, четыре символа на уровень, хотя в этой книге используются три символа из-за ограниченной ширины страниц).
- Вот пример кода, оформленного по этим правилам.

```
class WarZone
{
   public $mSomeSoldier;
   private $_mSomeOtherSoldier;

   function SearchAndDestroy($someEnemy, $someOtherEnemy)
   {
      $master_of_war = 'Soldier';
      $this->mSomeSoldier = $someEnemy;
      $this->_mSomeOtherSoldier = $someOtherEnemy;
   }
}
```

Кроме того, вам стоит сразу решить, какие кавычки вы будете использовать для ограничения строк. PHP, HTML и JavaScript позволяют использовать и одинарные кавычки (апострофы), и двойные. В этой книге одинарные кавычки используются в коде HTML и JavaScript, а двойные — в PHP. Хотя для JavaScript выбор кавычек — это дело вкуса (можете использовать одинарные, но только делайте это постоянно), в PHP одинарные кавычки обрабатываются быстрее, более безопасны и вызывают меньше ошибок при написании кода. Более подробно строки в PHP описаны в документации по адресу `http://php.net/types.string`. Кроме того, две полезные статьи по строкам в PHP можно найти по адресам `http://www.sitepoint.com/print/quick-php-tips` и `http://www.jeroenmulder.com/weblog/2005/04/php_single_and_double_quotes.php`.

Резюме

В этой главе мы коснулись многих тем. Мы обсудили трехуровневую архитектуру и разобрались, каким образом она помогает создавать гибкие и масштабируемые приложения. Кроме того, мы выяснили, какие места в этой архитектуре будут занимать используемые технологии.

Если вы чувствуете себя перегруженным информацией, не волнуйтесь. В следующей главе мы приступим к реальной работе по созданию первой части нашего сайта. Каждый шаг этой работы мы будем подробно описывать, и вы поймете, как работает каждый элемент приложения.

Глава 3

Начало реализации проекта TShirtShop

Итак, мы закончили теоретический этап создания проекта TShirtShop. Пришла пора проверить правильность принятых решений на практике. В этой главе мы создадим первую страницу сайта TShirtShop. Вы узнаете, как

- установить на свой компьютер и настроить все необходимые инструменты;
- создать общую структуру веб-сайта;
- реализовать механизмы обработки ошибок и сообщений об ошибках;
- создать базу данных, в которой будут храниться данные каталога, данные о заказах покупателей и прочая информация.

В последующих главах проект будет развиваться — мы добавим в него каталог товаров с информацией об отделах и категориях, списки товаров, страницы с информацией о товарах и многое другое.

ЗАМЕЧАНИЕ

Учтите, что в этой и нескольких последующих главах материал излагается весьма сжато, и если вы плохо разбираетесь в PHP и MySQL, то он будет для вас непростым. Если вам нужна подготовка базового уровня, могу порекомендовать, например, 3-е издание книги Мэтта Зандстры *Освой самостоятельно PHP за 24 часа* (ИД "Вильямс", 2008 г.).

В предыдущей главе мы уже определились, какое приложение нужно создать. Теоретические рассуждения, безусловно, полезны, но практическая работа гораздо интереснее. Код в этой книге тестировался на следующей платформе:

- PHP 5.2;
- Apache 2.2;
- MySQL 5.

ЗАМЕЧАНИЕ

Код из этой книги, вероятно, будет работать и с последующими версиями упомянутых товаров, но вряд ли он будет выполняться с версиями PHP, предшествовавшими 5.0.

Большая часть кода этого проекта должна работать и на других веб-серверах, при условии, что они совместимы с PHP (см. `http://www.php.net/manual/en/`

installation.php). Механизм изменения URL использует модуль Apache mod_rewrite. Если вы решите использовать другой веб-сервер, вам придется модифицировать код, изменяющий URL, который вы найдете в главе 7. Впрочем, Apache применяется в подавляющем большинстве проектов, использующих PHP.

Редактор исходного кода

Если на вашем компьютере не установлен редактор исходного кода, которым вы постоянно пользуетесь, выберите какой-нибудь из доступных в Интернете бесплатных редакторов и установите его. Если возможностей бесплатных редакторов вам мало, примите к сведению, что существует огромное множество коммерческих, платных программ. Какую из них выбрать — дело вкуса (или финансовых возможностей). Список редакторов, поддерживающих PHP, можно найти по адресу http://www.php-editors.com. Вот несколько наиболее популярных из них.

- Zend Studio (http://www.zend.com/products/zend_studio) — самая мощная интегрированная среда разработки (IDE) для создания PHP-приложений.
- phpEclipse (http://phpeclipse.net) — постепенно обретающая все большую популярность среда для разработки PHP-приложений. Zend входит в состав Eclipse foundation — разработчиков этой среды.
- Emacs (http://www.gnu.org/software/emacs/) описан на его веб-сайте как "расширяемый, настраиваемый, самодокументирующий редактор". Это очень мощный, бесплатный и кроссплатформенный редактор.
- SciTe (http://scintilla.sourceforge.net/SciTEDownload.html) — еще один бесплатный кроссплатформенный редактор.
- PSPad (http://www.pspad.com) — бесплатный редактор, популярный среди разработчиков в среде Windows. Он умеет выделять цветом элементы синтаксиса для множества разных форматов файлов. Кроме того, для него есть подключаемые модули, реализующие поддержку для редактирования CSS и проверку правописания.
- PHP Designer 2006 (http://www.mpsoftware.dk) — редактор для среды Windows со встроенным отладчиком.

Установка XAMPP

XAMPP — это пакет, созданный Apache Friends (http://www.apachefriends.org). В его состав входят Apache, PHP, MySQL и еще несколько компонентов. Если у вас нет этих компонентов, самый простой способ установить их — именно с помощью XAMPP. XAMPP существует в версиях для Windows, MacOS, Linux и Solaris.

Чтобы самостоятельно установить Apache, PHP и MySQL, придется установить и дополнительные модули, которые понадобятся на разных этапах работы над проектом. Это PDO, драйвер PDO для MySQL, cURL, mcrypt и mhash. Кроме того, на третьем этапе проектирования, когда мы начнем работать напрямую с кредитными картами покупателей, понадобится модуль поддержки SSL для Apache.

Следуйте инструкциям из следующего упражнения, чтобы установить XAMPP под управлением Windows. Инструкции по установке для Linux приведены ниже в отдельном упражнении. Чтобы установить XAMPP под управлением MacOS, следуйте инструкции по установке по адресу http://www.apachefriends.org/en/xampp-macosx.html.

За дополнительной информацией по поводу установки XAMPP можете также обратиться по адресу `http://www.installationwiki.org/XAMPP`.

Упражнение: установка XAMPP для Windows

Вот инструкции по установке XAMPP под управлением Windows.

1. Зайдите на страницу `http://www.apachefriends.org/en/xampp-windows.html` и загрузите установочный модуль XAMPP; это должен быть выполняемый файл с именем наподобие `xampp-win32-версия-installer.exe`.
2. Запустите загруженный файл. На первых нескольких экранах мастера установки оставьте заданные по умолчанию настройки, но мы рекомендуем вам установить XAMPP во вложенную папку корневой папки диска, например `C:\xampp`. Также выберите установку Apache и MySQL как служб, как показано на рис. 3.1. Затем щелкните на кнопке Install.

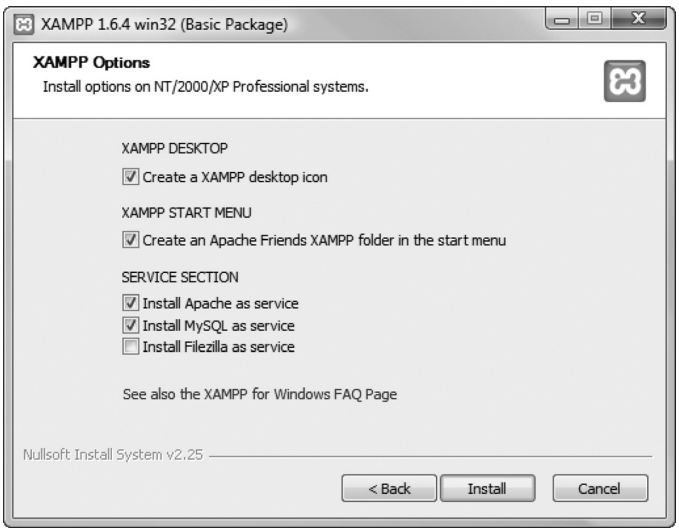


Рис. 3.1. Выбор настроек при установке XAMPP

3. Далее вас попросят подтвердить установку каждой из выбранных служб. Вам не понадобится FTP-сервер FileZilla, но установить Apache и MySQL как службы необходимо.

ЗАМЕЧАНИЕ

Нельзя одновременно установить на один компьютер несколько веб-серверов, работающих через порт 80 (это порт, используемый по умолчанию HTTP-серверами). Если на вашем компьютере уже есть веб-сервер, использующий порт 80 (например, IIS), вам придется изменить его настройки, отключить его или удалить из системы. В противном случае Apache не будет работать. Если вы хотите настроить Apache на использование другого порта (например, вместо 80 часто применяется порт 8080), откройте в текстовом редакторе файл `C:\xampp\apache\conf\httpd.conf`, найдите строки, в которых содержатся директивы `Listen 80` и `ServerName localhost:80`, и замените `80` номером выбранного вами порта.

4. В конце подтвердите запуск XAMPP Control Panel, с помощью которой можно управлять установленными службами. Окно этой программы показано на рис. 3.2.

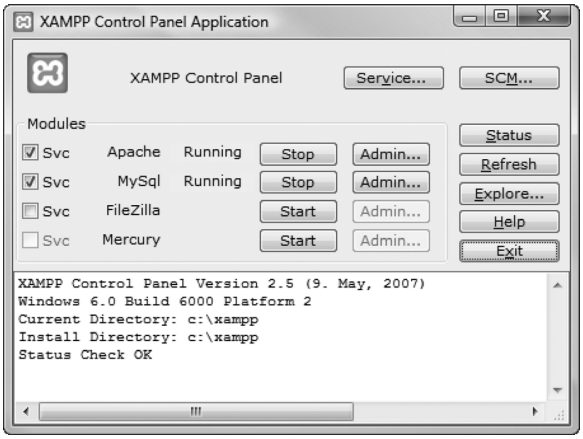


Рис. 3.2. Окно панели управления XAMPP: XAMPP Control Panel

5. Чтобы убедиться в том, что Apache установлен и работает, попробуйте в окне браузера открыть страницу `http://localhost:8080/` (если Apache использует порт 8080). Должна загрузиться страница приветствия XAMPP, показанная на рис. 3.3.



Рис. 3.3. Проверка успешности установки Apache

Упражнение: установка XAMPP для Linux

Вот инструкции по установке XAMPP под управлением Linux.

1. Зайдите на страницу `http://www.apachefriends.org/en/xampp-linux.html` и загрузите установочный модуль XAMPP; это должен быть архив с именем наподобие `xampp-linux-X.Y.Z.tar.gz`.
2. Зайдя в Linux с правами администратора, выполните из командной строки команду

   ```
   tar xvfz xampp-linux-X.Y.Z.tar.gz –C /opt
   ```

   Файлы из архива будут распакованы в папку `/opt`.

ЗАМЕЧАНИЕ

Нельзя одновременно установить на один компьютер несколько веб-серверов, работающих через порт 80 (это порт, используемый по умолчанию HTTP-серверами). Если на вашем компьютере уже есть веб-сервер, использующий порт 80, вам придется изменить его настройки, отключить его или удалить из системы. В противном случае Apache не будет работать. Если вы хотите настроить Apache на использование другого порта (например, вместо 80 часто используется порт 8080), откройте в текстовом редакторе файл `/opt/lampp/etc/httpd.conf`, найдите строки, в которых содержатся директивы `Listen 80` и `ServerName localhost:80`, и замените `80` номером выбранного вами порта.

3. Чтобы запустить XAMPP, используйте команду

   ```
   /opt/lampp/lampp start
   ```

   Чтобы перезапустить XAMPP, вместо `start` в этой команде используйте `restart`, а чтобы остановить его, используйте `stop`.
4. Чтобы убедиться в том, что Apache установлен и работает, попробуйте в окне браузера открыть страницу `http://localhost:8080/` (если Apache использует порт 8080). Должна загрузиться страница приветствия XAMPP, показанная на рис. 3.3.

Подготовка псевдонима TShirtShop

Одно из преимуществ использования платформенно-независимых программ с открытым исходным кодом — возможность выбора операционной системы. Вы можете создавать и использовать TShirtShop под управлением Windows, Linux или MacOS. Однако это значит, что вам придется немного повозиться, настраивая среду, особенно если вы начинающий разработчик.

Некоторые детали, существенные при создании папки проекта, меняются в зависимости от того, какую операционную систему вы выберете (в основном из-за разных путей к файлам), поэтому ниже мы рассмотрим отдельно варианты для Windows и Linux. Однако базовые шаги одинаковы для обеих платформ.

1. Создайте папку `tshirtshop` (мы будем использовать для имен папок строчные буквы). В ней будут находиться файлы нашего проекта (файлы с кодом на PHP, файлы рисунков и т.д.).
2. Отредактируйте файл конфигурации Apache (`httpd.conf`) — создайте псевдоним (alias) `tshirtshop`, который будет указывать на созданную вами папку

tshirtshop. Это позволит открывать наш проект при обращении по адресу http://localhost/tshirtshop в браузере. Такая функциональность реализуется в Apache с помощью псевдонимов, задаваемых в файле httpd.conf. Синтаксис псевдонима следующий:

```
Alias имя_псевдонима имя_настоящей_папки
```

ПОДСКАЗКА

Файл httpd.conf снабжен комментариями, но если этих комментариев вам недостаточно, более подробную информацию о содержащихся в нем настройках можете найти по адресу http://httpd.apache.org/docs-2.0/.

Если вы работаете под управлением Windows, используйте приведенные ниже инструкции для настройки рабочей папки tshirtshop. Инструкции для Linux приведены далее.

Упражнение: подготовка псевдонима tshirtshop для Windows

1. Создайте папку tshirtshop, в которой будут размещаться все рабочие файлы нашего проекта. Мы предполагаем, что вы создадите ее в корневой папке диска C: (поскольку в проекте мы используем относительные пути, вы можете размещать папку где угодно, но так, чтобы к ней был доступ из папки Apache).
2. По умолчанию Apache из XAMPP использует для обслуживания запросов клиентов папку C:\xampp\htdocs. Эта папка задается с помощью команды DocumentRoot в конфигурационном файле Apache: c:\xampp\apache\conf\httpd.conf.

   Поскольку мы хотим использовать созданную нами папку, а не заданную по умолчанию с помощью команды DocumentRoot, нужно создать псевдоним tshirtshop, указывающий на папку, созданную в п. 1. Откройте конфигурационный файл Apache (httpd.conf), найдите раздел псевдонимов (он обозначен меткой <IfModule alias_module>) и добавьте в него следующие строки.

```
<IfModule alias_module>
# ...
# Задаем псевдоним tshirtshop
Alias /tshirtshop/ "C:/tshirtshop/"
Alias /tshirtshop "C:/tshirtshop"
</IfModule>
<Directory "C:/tshirtshop">
Options Indexes FollowSymLinks
AllowOverride All
Order allow,deny
Allow from all
</Directory>
```

   Добавив выделенные полужирным начертанием строки и перезапустив Apache, вы сможете открывать наше приложение в браузере, введя URL http://localhost/tshirtshop/ или http://localhost/tshirtshop (разумеется, сначала нужно это приложение создать).
3. Создайте в папке tshirtshop файл с именем test.php со следующей строкой внутри:

```
<?php phpinfo(); ?>
```

4. Перезапустите Apache (проще всего это сделать с помощью XAMPP Control Panel). Перезапуск веб-сервера необходим, потому что мы внесли изменения в конфигурационный файл.
5. Попробуйте открыть URL `http://localhost/tshirtshop/test.php` (или `http://localhost:8080/tshirtshop/test.php`, если Apache работает через порт 8080) в окне браузера.

Упражнение: подготовка псевдонима `tshirtshop` для Linux

1. Создайте новую папку `tshirtshop`, в которой мы будем размещать все рабочие файлы нашего проекта. Вероятно, вам будет проще создать ее в своей домашней папке (в этом случае путь к ней будет выглядеть примерно как `/home/имя_пользователя/tshirtshop`), но в проекте мы будем использовать относительные пути, так что вы вольны создать ее там, где сочтете нужным.
2. По умолчанию Apache из XAMPP использует для обслуживания запросов клиентов папку с именем наподобие `/opt/lampp/htdocs`. Эта папка задается с помощью команды `DocumentRoot` в конфигурационном файле Apache, полный путь к которому — обычно `/opt/lampp/etc/httpd.conf`.

   Поскольку мы хотим использовать созданную нами папку, а не заданную по умолчанию с помощью команды `DocumentRoot`, нужно создать псевдоним `tshirtshop`, указывающий на нашу папку, созданную в п. 1. Откройте конфигурационный файл Apache (`httpd.conf`), найдите раздел псевдонимов (он обозначен меткой `<IfModule alias_module>`) и добавьте в него следующие строки.

```
<IfModule alias_module>
# ...
# Задаем псевдоним tshirtshop
Alias /tshirtshop/ "/home/username/tshirtshop/"
Alias /tshirtshop "/home/username/tshirtshop/"
</IfModule>

<Directory "/home/username/tshirtshop/">
Options Indexes FollowSymLinks
AllowOverride All
Order allow,deny
Allow from all
</Directory>
```

   Добавив выделенные полужирным начертанием строки и перезапустив Apache, вы сможете открывать наше приложение в браузере, введя URL `http://localhost/tshirtshop/` или `http://localhost/tshirtshop` (когда мы создадим это приложение).
3. Создайте в папке `tshirtshop` файл с именем `test.php` со следующей строкой внутри:

```
<?php phpinfo(); ?>
```

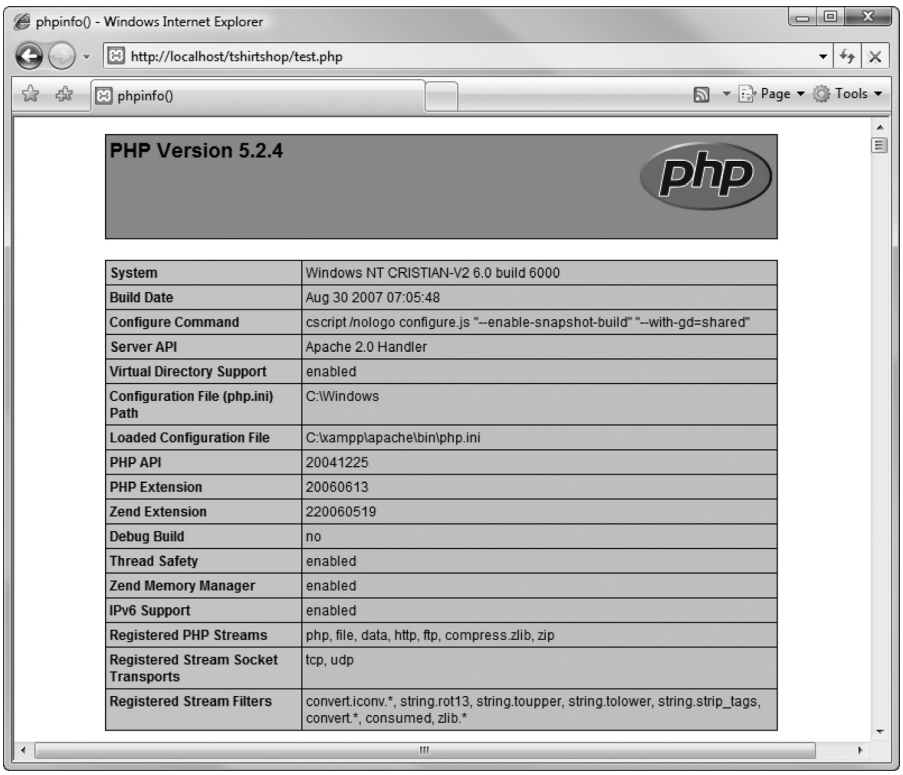
4. Перезапустите Apache (это необходимо, потому что мы внесли изменения в конфигурационный файл). Затем попробуйте открыть URL `http://localhost/tshirtshop/test.php` (или `http://localhost:8080/tshirtshop/test.php`, если Apache работает через порт 8080) в окне браузера.

Как это работает: подготовка псевдонима `tshirtshop` для Windows и Linux

Первый шаг к созданию электронного магазина TShirtShop — маленький, но очень важный, поскольку он позволяет проверить работоспособность Apache, PHP и псевдонима `tshirtshop`. Если при открытии тестовой страницы возникли проблемы, прежде всего убедитесь, что вы правильно установили XAMPP.

Независимо от того, работаете вы под управлением Windows или Linux, при открытии файла `test.php` в окне браузера должна отображаться информация о PHP, возвращаемая функцией `phpinfo()`, примерно такая, как на рис. 3.4.

Кроме того, вы убедились, что папка `tshirtshop` и ее содержимое доступны веб-серверу.



System	Windows NT CRISTIAN-V2 6.0 build 6000
Build Date	Aug 30 2007 07:05:48
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--with-gd=shared"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\xampp\apache\bin\php.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	enabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	php, file, data, http, ftp, compress.zlib, zip
Registered Stream Socket Transports	tcp, udp
Registered Stream Filters	convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, zlib.*

Рис. 3.4. Проверка успешного завершения установки Apache

Установка Smarty

Чтобы установить Smarty, достаточно просто скопировать его PHP-классы в папку вашего проекта. Многие компании, предлагающие веб-хостинг, предоставят вам эти классы бесплатно, но лучше выполнить их установку самостоятельно по двум причинам.

- Всегда лучше делать проект независимым от настроек сервера.
- Даже если компания, предоставляющая веб-хостинг, установит Smarty вместо вас, она может позже сменить его версию на новую, не предупредив вас, и ваше приложение может начать сбоить.

В упражнении, приведенном ниже, Smarty устанавливается во вложенную папку `libs` папки `tshirtshop`. Это упражнение остается неизменным независимо от того, какую операционную систему вы используете.

Упражнение: установка Smarty

1. Создайте в папке `tshirtshop` папку `libs`, а внутри `libs` — папку `smarty`.
2. Загрузите свежую стабильную версию Smarty с ее домашней страницы (`http://smarty.php.net/download.php`). Архив будет представлять собой файл `.tar.gz`. Чтобы распаковать его для Windows, вам понадобится программа наподобие WinRar (`http://www.rarlabs.com`) или WinZip (`http://www.winzip.com`).
3. Откройте загруженный архив и скопируйте из него содержимое папки `Smarty-2.X.Y/libs` в созданную в п. 1 папку `tshirtshop/libs/smarty`. Вам понадобится скопировать только содержимое папки `libs`, ничего больше (в версии Smarty 2.6.18, последней, с которой мы тестировали наш проект, в этой папке было четыре файла `.php` и две вложенные папки — `internals` и `plugins`).
4. Для работы Smarty требует трех папок, которые вам нужно будет создать, — `templates`, `templates_c` и `configs`. Внутри папки `tshirtshop` создайте папку `presentation`, а в ней — папки `templates` и `templates_c`. В папке `presentation` будут находиться файлы уровня представления нашего приложения.
5. В папке `tshirtshop` создайте папку `include`. В ней будут храниться конфигурационные файлы приложения. Внутри папки `include` создайте папку `configs` — мы будем ее использовать для хранения конфигурационных данных Smarty.
6. Если вы используете операционную систему Unix или Linux, убедитесь, что у Apache есть права записи в папку `templates_c`, в которой Smarty будет хранить файлы скомпилированных шаблонов (что это такое, вы узнаете немного позже). Чтобы гарантировать наличие у Apache этих прав, выполните из командной строки команду

```
chmod a+w /home/имя_пользователя/tshirtshop/presentation/templates_c
```

ЗАМЕЧАНИЕ

Задав права на запись командой, приведенной выше, вы позволите любому пользователю, у которого есть учетная запись на вашем компьютере, просматривать исходные коды во всех файлах в папке. В этом коде могут содержаться ценные сведения (пароли к базам данных, ключи шифрования для информации с кредитных карт и т.д.). Для более тонкой настройки прав обратитесь к своему системному администратору.

7. Зайдите на страницу этой книги в Интернете и просмотрите раздел обнаруженных ошибок (errata). Это до некоторой степени обезопасит вас от проблем на случай, если более новые версии Smarty вызывают проблемы в работе приложения TShirtShop. Страница, посвященная ошибкам, находится по адресу `http://www.cristiandarie.ro/php-mysql-ecommerce-2/`. Можете также поискать ее на сайте издательства Apress (`http://www.apress.com`).

Как это работает: установка Smarty

В этом упражнении вы создали три папки, необходимые Smarty для работы.

- Папку `templates`, в которой будут находиться шаблоны Smarty для нашего веб-сайта (файлы `.tpl`).
- Папку `templates_c`, в которой будут находиться скомпилированные шаблоны Smarty. Это файлы `.php`, которые Smarty генерирует автоматически, обрабатывая шаблоны `.tpl`. Скомпилированные шаблоны генерируются заново при каждом изменении исходных файлов `.tpl`.
- Папку `configs` для конфигурационных файлов, которые могут нам понадобиться для шаблонов.

После их добавления дерево папок должно выглядеть следующим образом.

```
tshirtshop/
   include/
      configs/
   libs/
      smarty/
         internals/
         plugins/
   presentation/
      templates/
      templates_c/
```

Создание каркаса сайта

Оформление сайта разработчики обычно обсуждают с клиентом или на этапе заключения контракта, или позже, в процессе работы над проектом. Обычно оформлением сайта занимаются не программисты, а профессиональные веб-дизайнеры, но вы можете купить готовый шаблон веб-сайта у какой-нибудь из множества компаний, специализирующихся на веб-дизайне.

Визуальному оформлению сайтов, удобству их использования, производительности и прочим смежным темам посвящено колоссальное количество литературы. Поскольку данная книга посвящена программированию, в ней основное внимание уделяется техническим аспектам создания сайта TShirtShop. Но как ответственный веб-разработчик вы должны по крайней мере представлять себе общее состояние проекта, над которым работаете. Если вас интересует веб-дизайн, осмелимся порекомендовать несколько книг на эту тему.

- *Don't Make Me Think: A Common Sense Approach to Web Usability*, Second Edition, Steve Krug (New Riders Press, 2005)
- *Prioritizing Web Usability*, Jakob Nielsen, Hoa Loranger (New Riders Press, 2006). Перевод на русский язык: Якоб Нильсен, Хоа Лоранжер, *Web-дизайн: удобство использования Web-сайтов*, ИД "Вильямс", 2007 г.
- *Designing Interfaces: Patterns for Effective Interaction Design*, Jenifer Tidwell (O'Reilly, 2005)
- *Web Accessibility:Web Standards and Regulatory Compliance*, Andrew Kirkpatrick, Richard Rutter, Christian Heilmann, Jim Thatcher, Cynthia Waddell (Friends of ED, 2006)
- *Ambient Findability*, Peter Morville (O'Reilly, 2005)
- *Bulletproof Web Design*, Second Edition, Dan Cederholm (New Riders Press, 2007)
- *Professional Search Engine Optimization with PHP: A Developer's Guide to SEO*, Cristian Darie, Jaimie Sirovich (Wrox Press, 2007). Перевод на русский язык:

Джейми Сирович, Кристиан Дари, *Поисковая оптимизация сайта на PHP для профессионалов. Руководство разработчика по SEO*, издательство "Диалектика", 2008 г.

Для TShirtShop мы применим относительно простую, но эффективную структуру сайта, которая позволит в дальнейшем легко вносить изменения и сосредоточиться на технических деталях. Чтобы сосредоточиться на этих деталях, для создания внешнего оформления сайта воспользуемся платформой, которая называется Yahoo User Interface Library (YUI). Это позволит реализовать простой интерфейс, который будет выглядеть одинаково во всех современных браузерах, даже если мы не являемся знатоками CSS. Официальный сайт YUI находится по адресу `http://developer.yahoo.com/yui/`. Все страницы TShirtShop, включая первую страницу, будут иметь структуру, показанную на рис. 3.5.

Хотя структуру каталога товаров подробно мы будем рассматривать только в следующей главе, сейчас нужно знать, что основной список отделов должен отображаться на каждой странице сайта. Когда пользователь щелкает на одном из отделов в списке, ниже этого списка отображается список категорий выбранного отдела. Кроме того, на каждой странице должно быть поле поиска, с помощью которого посетитель сможет найти нужный ему товар. В верхней части всех страниц должен находиться заголовок сайта.

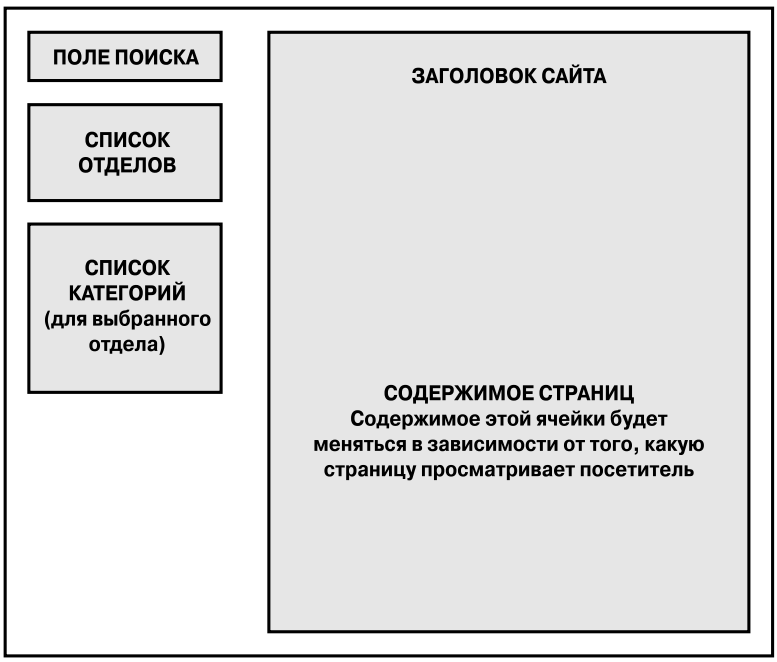


Рис. 3.5. Структура веб-страниц сайта TShirtShop

Чтобы как можно проще реализовать эту структуру, воспользуемся компонентными шаблонами Smarty (или простыми дизайн-шаблонами Smarty), как показано на рис. 3.6.

На рис. 3.6 показаны компонентные шаблоны Smarty, которые мы создадим в нескольких последующих главах. Часть страницы с содержимым сайта будет генерироваться простым дизайн-шаблоном с именем `store_front.tpl`, который мы

создадим позже в этой главе. В главе 4 мы превратим его в компонентный шаблон, который понадобится нам при генерации динамического содержимого.

ЗАМЕЧАНИЕ

Компонентными шаблонами Smarty мы называем сочетания дизайн-шаблонов Smarty (файлов `.tpl`) с соответствующими подключаемыми файлами Smarty (файлами `.php`). Мы добавляем подключаемые файлы, если шаблон должен будет отображать какие-то данные динамически — в этом случае в подключаемом файле будет логика представления, которая соберет нужные данные и передаст их шаблону. Если в каких-то случаях мы сможем обойтись отображением статичных данных, такая функциональность будет излишней и достаточно будет одного только дизайн-шаблона. Как работать с подключаемыми файлами Smarty, вы узнаете из главы 4. Официальная страница с документацией по подключаемым файлам находится по адресу `http://smarty.php.net/manual/en/plugins.php`.

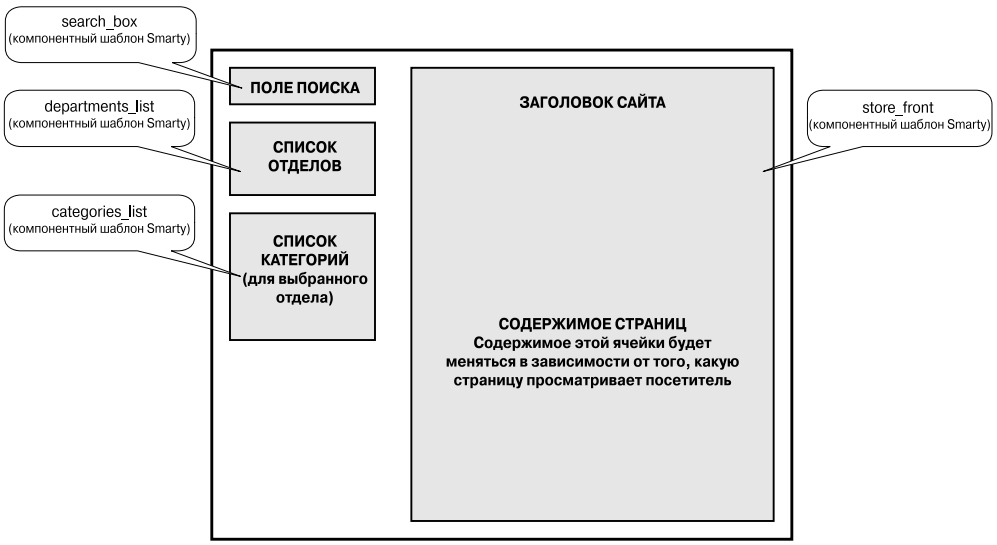


Рис. 3.6. Шаблоны Smarty для компонентов веб-страницы

Реализация элементов функциональности сайта с помощью шаблонов Smarty дает нам преимущества, перечисленные в главе 2. Разделение несвязанных между собой элементов функциональности позволяет модифицировать каждый из них независимо от других или использовать их на других страницах без необходимости писать их заново. Кроме того, элементы, оформленные в виде шаблонов Smarty, очень легко при необходимости переместить на другое место на странице.

Поле поиска и список отделов — это элементы, которые будут присутствовать на каждой странице сайта. Список категорий появляется, только если посетитель выберет какой-то отдел из списка. Ячейка с содержимым — это динамический элемент, содержимое которого будет меняться в зависимости от того, какую страницу сайта просматривает посетитель (начальную страницу, список товаров, страницу с информацией о компании и т.д.). Есть два основных варианта реализации этой ячейки: создать компонентный шаблон, изменяющий себя в зависимости от выбранной пользователем страницы, или набор разных компонентных шаблонов, каждый из которых используется для одной из страниц. В пользу каждого из вариантов можно привести свои доводы, и выбор определяется в основном спецификой

конкретного проекта. Для TShirtShop мы выберем второй вариант и создадим набор специализированных шаблонов.

В оставшейся части главы мы

- создадим входную страницу сайта TShirtShop;
- реализуем основы механизма обработки ошибок для приложения TShirtShop;
- создадим базу данных `tshirtshop`.

Создание входной страницы TShirtShop

Входная страница сайта TShirtShop будет генерироваться файлами `index.php` и `store_front.tpl`.

Шаблон Smarty `store_front.tpl` будет содержать заместители для трех основных элементов страниц — заголовка, списка отделов и ячейки с содержимым страниц. Как упоминалось выше, мы будем использовать YUI и его инструмент для разметки страниц, чтобы получить CSS-таблицу для нашего сайта.

Реализации входной страницы посвящено приведенное ниже упражнение, а технические детали мы рассмотрим в разделе "Как это работает: первая страница TShirtShop".

Упражнение: создание входной страницы

1. Создайте в папке `tshirtshop` вложенную папку `images`.
2. Скопируйте в только что созданную папку файлы из папки `image_folders/images` со страницы Source Code/Download данной книги (эта страница доступна на сайте `http://cristiandarie.ro`).
3. В папке `tshirtshop/include/configs` создайте файл `site.conf` и введите в него следующую строку:

   ```
   site_title = "TShirtShop: Demo Product Catalog from Beginning PHP and
   MySQL E-Commerce"
   ```

4. Создайте в папке `tshirtshop` вложенную папку `styles`.
5. Загрузите свежую версию YUI со страницы `http://developer.yahoo.com/yui/download`. Откройте загруженный архив и скопируйте из него в папку `tshirtshop/styles` файлы `yui_2.X.Y/build/reset/rest-min.css`, `yui_2.X.Y/build/base/base-min.css`, `yui_2.X.Y/build/fonts/fonts-min.css` и `yui_2.X.Y/build/grids/grids-min.css`.
6. Загрузите YUI Grid Builder по адресу `http://developer.yahoo.com/yui/grids/builder` и задайте параметры Toolbox, приведенные в табл. 3.1. После того как вы зададите эти параметры, страница должна выглядеть так, как на рис. 3.7.

Таблица 3.1. Параметры YUI CSS Grid Builder

Параметр	Значение
Body	`750 px`
Body columns	`Sidebar left 180 px`
Split content	Оставьте в одной строке значение `1 Column (100)` и щелкните на Add Another Row, чтобы добавить еще одну строку

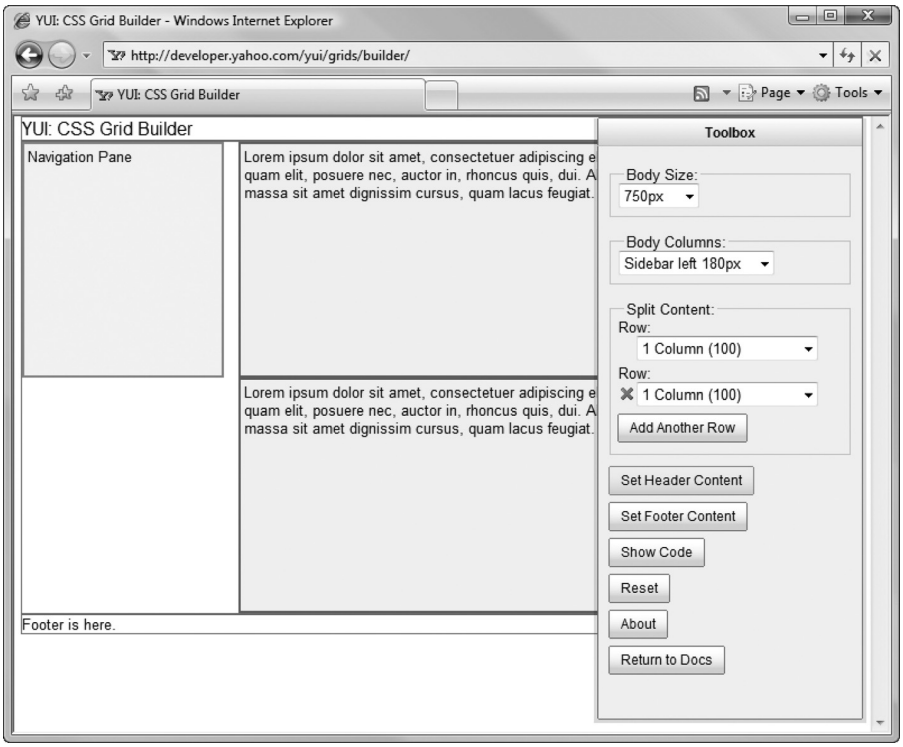


Рис. 3.7. Использование YUI CSS Grid Builder

7. Щелкните в окне Grid Builder на кнопке Show Code, чтобы инструмент сгенерировал код с заданными параметрами разметки. Этот код будет отображен в окне наподобие показанного на рис. 3.8.

ЗАМЕЧАНИЕ

С помощью YUI CSS Grid Builder можно легко генерировать HTML-разметку, но, если вы не хотите пользоваться этим инструментом, просто перейдите к следующему шагу упражнения. При желании вы сможете набрать требуемый HTML-код вручную.

8. Создайте в папке `tshirtshop/presentation/templates` файл `store_front.tpl` и вставьте в него следующий код (большую часть этого кода должен был сгенерировать YUI CSS Grid Builder, но вы можете набрать его вручную).

```
{* smarty *}
{config_load file="site.conf"}
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title>{#site_title#}</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <link type="text/css" rel="stylesheet" href="styles/tshirtshop.css" />
  </head>
  <body>
    <div id="doc" class="yui-t2">
```

```
      <div id="bd">
        <div id="yui-main">
          <div class="yui-b">
            <div id="header" class="yui-g">
              <a href="index.php">
                <img src="images/tshirtshop.tif" alt="tshirtshop logo" />
              </a>
            </div>
            <div id="contents" class="yui-g">
              Place contents here
            </div>
          </div>
        </div>
        <div class="yui-b">
          Place list of departments here
        </div>
      </div>
    </div>
  </body>
</html>
```

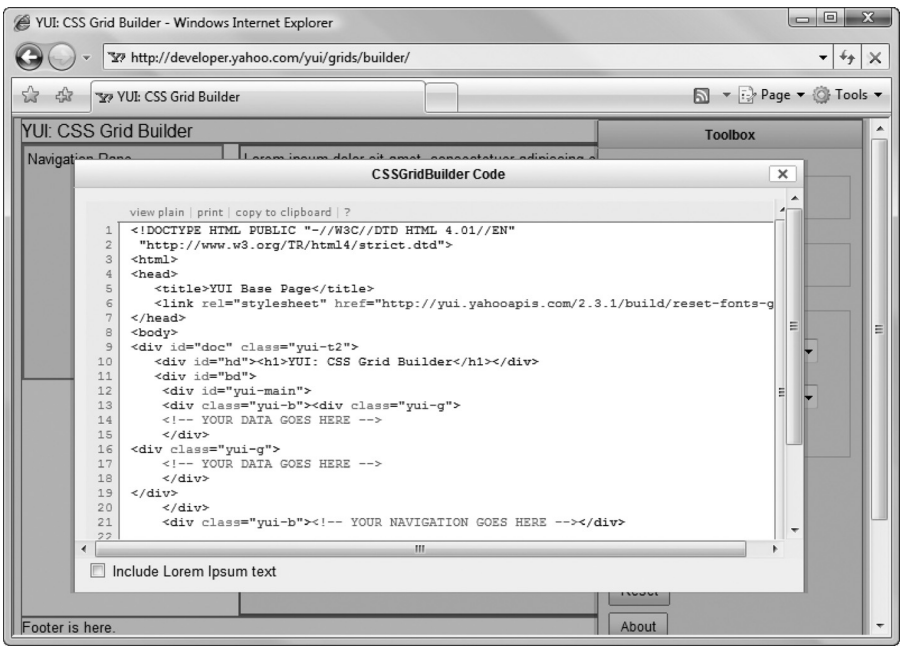


Рис. 3.8. Код, сгенерированный YUI CSS Grid Builder

9. Создайте в папке `tshirtshop/styles` файл `tshirtshop.css` и наберите в нем следующий код.

```
@import "reset-min.css";
@import "base-min.css";
@import "fonts-min.css";
@import "grids-min.css";

body {
  font-size: 85%;
  font-family: "georgia";
```

```
}

.yui-t2, #bd, #yui-main {
  z-index: -5;
}

.yui-b, .yui-g {
  z-index: auto;
}

#header {
  margin-top: 15px;
  text-align: right;
}

.error_box {
  background-color: #ffffcc;
  border: 1px solid #dc143c;
  color: #DC143C;
  margin: 0 auto;
  overflow: auto;
  padding: 5px;
  position: relative;
  text-align: left;
  width: 90%;
  z-index: 5;
}
```

10. Добавьте в папку `tshirtshop/include` файл `config.php` и введите в него следующий код.

```
<?php
// в SITE_ROOT содержится полный путь к папке tshirtshop
define('SITE_ROOT', dirname(dirname(__FILE__)));

// Папки приложения
define('PRESENTATION_DIR', SITE_ROOT . '/presentation/');
define('BUSINESS_DIR', SITE_ROOT . '/business/');

// Настройки, необходимые для конфигурирования Smarty
define('SMARTY_DIR', SITE_ROOT . '/libs/smarty/');
define('TEMPLATE_DIR', PRESENTATION_DIR . 'templates');
define('COMPILE_DIR', PRESENTATION_DIR . 'templates_c');
define('CONFIG_DIR', SITE_ROOT . '/include/configs');
?>
```

Прежде чем двигаться дальше, давайте разберемся, что происходит в этом файле. Функция `dirname(__FILE__)` возвращает имя папки, в которой находится указанный файл. Соответственно, `dirname(dirname((__FILE__))` возвращает папку, в которой находится папка с указанным файлом. Это позволяет поместить в `SITE_ROOT` полный путь к `tshirtshop`. С помощью константы `SITE_ROOT` мы задаем абсолютные пути к папкам Smarty.

11. В папке `tshirtshop/presentation` создайте файл с именем `application.php` и введите в него следующий код.

```
<?php
// Ссылка на библиотеку Smarty
require_once SMARTY_DIR . 'Smarty.class.php';
/* Класс, расширяющий Smarty, используется для обработки и
   отображения файлов Smarty */
class Application extends Smarty
```

```
{
   // Конструктор класса
   public function __construct()
   {
      // Вызов конструктора Smarty
      parent::Smarty();
      // Меняем папки шаблонов по умолчанию
      $this->template_dir = TEMPLATE_DIR;
      $this->compile_dir = COMPILE_DIR;
      $this->config_dir = CONFIG_DIR;
   }
}
?>
```

В этом файле мы расширяем класс Smarty классом-оберткой с именем `Application`, меняющим поведение Smarty по умолчанию. Конструктор класса `Application` конфигурирует Smarty на использование папок, созданных нами ранее.

ПОДСКАЗКА

Как уже говорилось, Smarty требует для работы трех папок: `templates`, `templates_c` и `configs`. В конструкторе класса `Application` мы задаем набор из трех папок, используемый нашим приложением. Если вы хотите включить кэширование, Smarty потребуется также папка с именем `cache`. Мы не будем использовать кэширование Smarty в проекте TShirtShop, но если эта возможность вас интересует, можете почитать о ней в руководстве по Smarty: зайдите по адресу `http://smarty.php.net/manual/en/caching.php`.

12. Добавьте в папку `tshirtshop` файл `index.php`. Задача этого файла — загружать файл `store_front.tpl` с помощью класса `Application`, созданного нами выше. Вот код, который нужно поместить в файл `index.php`.

```
<?php
// Подключаем служебные файлы
require_once 'include/config.php';

// Загружаем шаблон страницы приложения
require_once PRESENTATION_DIR . 'application.php';

// Загружаем файл шаблонов Smarty
$application = new Application();

// Отображаем страницу
$application->display('store_front.tpl');
?>
```

13. А теперь давайте посмотрим, как все это работает. Откройте в окне браузера ссылку `http://localhost/tshirtshop/`. Должна отобразиться страница, показанная на рис. 3.9.

ПОДСКАЗКА

Вероятно, будет неплохо, если вы создадите в браузере закладку для входной страницы проекта TShirtShop — это позволит в дальнейшем не вводить адрес вручную каждый раз, когда вам понадобится открыть проект и протестировать его.

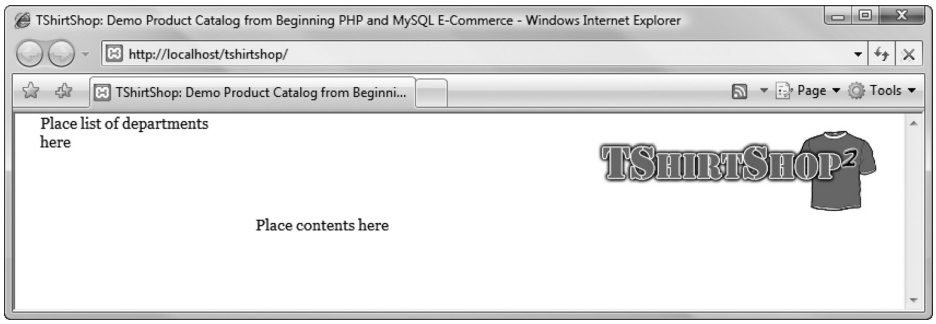


Рис. 3.9. Работающее приложение TShirtShop

Как это работает: первая страница TShirtShop

Созданная нами веб-страница состоит из трех основных разделов. В ней есть две ячейки, которые мы позже заполним компонентными шаблонами: одна — для списка отделов, другая — для содержимого страницы.

Обратите внимание на список отделов в левой части страницы, заголовок сверху и основную ячейку с содержимым, соответствующим данному этапу создания приложения. Как уже говорилось, основная ячейка — это единственная часть страницы, содержимое которой будет меняться при переходе в разные разделы сайта — все остальные части страницы будут выглядеть одинаково во всех разделах. Такая реализация упрощает жизнь нам как программистам и придает всем страницам сайта сходный внешний вид.

Прежде чем продолжить, нужно разобраться в том, как работает Smarty. Все начинается с файла `index.php`, так что давайте рассмотрим его подробнее. Вот его код.

```
<?php
// Подключаем служебные файлы
require_once 'include/config.php';

// Загружаем шаблон страницы приложения
require_once PRESENTATION_DIR . 'application.php';

// Загружаем файл шаблонов Smarty
$application = new Application();

// Отображаем страницу
$application->display('store_front.tpl');
?>
```

Пока что реализованная в этом файле функциональность очень проста. Сначала этот файл загружает конфигурационный файл, `config.php`, в котором задаются значения некоторых глобальных констант. Затем он загружает файл шаблона Smarty, который и займется генерацией HTML-разметки, когда посетитель попытается открыть файл `index.php`.

Стандартный способ создания и настройки страницы Smarty показан в следующем фрагменте кода.

```
<?php
// Загружаем библиотеку Smarty
require_once SMARTY_DIR . 'Smarty.class.php';

// Создаем новый экземпляр класса Smarty
$smarty = new Smarty();
$smarty->template_dir = TEMPLATE_DIR;
$smarty->compile_dir = COMPILE_DIR;
$smarty->config_dir = CONFIG_DIR;
?>
```

В приложении TShirtShop мы создали класс `Application`, наследующий Smarty. В конструкторе класса `Application` содержится процедура инициализации, которая упрощает использование шаблонов Smarty. Еще раз приведем код класса `Application`.

```
/* Класс, расширяющий Smarty, используется для обработки и
   отображения файлов Smarty */
class Application extends Smarty
{
   // Конструктор класса
   public function __construct()
   {
      // Вызов конструктора Smarty
      parent::Smarty();
      // Меняем папки шаблонов, заданные по умолчанию
      $this->template_dir = TEMPLATE_DIR;
      $this->compile_dir = COMPILE_DIR;
      $this->config_dir = CONFIG_DIR;
   }
}
```

ЗАМЕЧАНИЕ

Концепция конструктора специфична для объектно-ориентированного программирования. Конструктор класса — это специальный метод, выполняющийся автоматически при создании экземпляра этого класса. В PHP конструкторы классов называются `__construct()`. Код, помещенный в конструктор класса, гарантированно выполнится при создании каждого объекта этого класса.

Файл шаблона Smarty (с расширением `.tpl`) содержит, за небольшими исключениями, простую HTML-разметку, но нас интересуют именно эти исключения. Давайте рассмотрим их подробнее. В файле `store_front.tpl` перед началом HTML-разметки загружается файл `site.conf`.

```
{* smarty *}
{config_load file="site.conf"}
```

ПОДСКАЗКА

Комментарии в Smarty помещаются между символами `{*` и `*}`.

На данный момент единственная переменная, значение которой задается в файле `site.conf`, — это `site_title`, в которой содержится название веб-сайта. Значение этой переменной используется для генерации заголовка страницы в HTML.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title>{#site_title#}</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <link href="styles/tshirtshop.css" type="text/css" rel="stylesheet" />
  </head>
```

Переменные, загружаемые из конфигурационного файла, помечаются в HTML символами # или с помощью переменной `$smarty.config`, как в следующем фрагменте.

```
<head>
<title>{$smarty.config.site_title}</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <link href="styles/tshirtshop.css" type="text/css" rel="stylesheet" />
</head>
```

Мы загрузили конфигурационный файл `site.conf` с помощью `{config_load file="site.conf"}` и обратились к переменной `site_title` с помощью `#site_title#`. Такое обращение мы будем использовать везде, где потребуется получить заголовок сайта. Если понадобится сменить этот заголовок, достаточно будет изменить значение переменной `site_title` в файле `site.conf`.

И наконец, стоит отметить, что мы используем таблицы стилей CSS. Эти таблицы — мощный инструмент для управления оформлением HTML-страниц. Определения стилей CSS могут храниться в одном или нескольких файлах с расширением `.css` либо даже помещаться внутрь самого HTML-файла, в котором они используются. Такой подход позволяет веб-дизайнерам при необходимости отделить определения стилей от документов, в которых эти определения используются. Если CSS используется правильно, он позволяет легко менять внешний вид всего сайта — для этого достаточно отредактировать файл с таблицей стилей.

В нашем проекте мы использовали три файла CSS из библиотеки YUI и создали еще один файл собственноручно. Копаться в деталях файлов CSS нам не понадобится, но при желании вы можете сделать это. Для решения наших задач достаточно знать, что YUI позволит создать простую и удобную разметку, которая будет корректно отображаться во всех современных браузерах.

Если раньше вам никогда не приходилось использовать CSS, не волнуйтесь. Ничего особо сложного в нем нет, и для его использования не требуются особые познания. Но при более внимательном рассмотрении оказывается, что CSS — это обширная тема, по которой есть много литературы и учебных курсов, включая бесплатные, например `http://www.w3.org/Style/CSS/` и `http://www.csstutorial.net`. Изрядное количество полезных ресурсов по CSS можно найти на сайте `http://www.cssgarden.com`. Кроме того, в Википедии есть страница о CSS, на которой рассказано об истории его развития, текущем состоянии и ограничениях.

Вы будете встречаться с CSS весьма часто в следующей главе этой книги.

Обработка ошибок и сообщения о них

Хотя мы будем стараться писать код так, чтобы он выполнялся без неприятных сюрпризов, всегда есть вероятность того, что при обработке запроса пользователя что-то пойдет не так. В таких случаях лучше всего централизованно обработать возникшую ошибку и выполнить действия, которые нейтрализуют ее эффект.

К сожалению, PHP прославился своими маловразумительными сообщениями об ошибках. Если вы использовали другие языки программирования, то, вероятно, научились ценить информацию, которую можно получить из стека вызовов функций при отладке. В PHP информация о стеке вызовов по умолчанию не отображается, но такое положение дел можно изменить. На этапе разработки приложения информация из стека вызовов весьма пригодится вам для отладки, а в финальной версии приложения сообщения о возникших ошибках должны отправляться администраторам. Еще одна проблема — замысловатый тип сообщений об ошибках `E_WARNING`, который не позволяет сразу определить, достаточно ли серьезна ошибка, чтобы останавливать работу приложения.

ПОДСКАЗКА

Если вы не знаете или не помните, как выглядят сообщения об ошибках в PHP, попробуйте добавить в файл `index.php` нашего проекта строку

```
require_once 'inexistent_file.php';
```

После этого откройте сайт в окне браузера и полюбуйтесь искомым сообщением. Проведя такой эксперимент, не забудьте удалить эту строку из файла!

В контексте рабочего веб-приложения ошибки могут возникать неожиданно по разным причинам, например из-за сбоев программного обеспечения (сбои операционных систем и СУБД, действия вирусов и т.д.) или сбоев аппаратуры. Важно записывать сведения о возникающих ошибках и сообщать о них администраторам

сайта (например, с помощью электронной почты), чтобы как можно быстрее устранить последствия этих ошибок.

Поэтому начнем с создания эффективного механизма обработки ошибок в нашем приложении. Мы создадим класс `ErrorHandler`, который и будет обрабатывать ошибки. В этом классе мы создадим статический определенный пользователем метод обработки ошибок `Handler()`, который будет вызываться при возникновении ошибок в работе PHP-программы. В PHP мы определим обработчик ошибок с помощью функции `set_error_handler()`.

ПРЕДУПРЕЖДЕНИЕ

Как вы вскоре увидите, второй параметр функции `set_error_handler()` используется для указания типов ошибок, которые должна обрабатывать указанная функция-обработчик. Но этот второй параметр поддерживается только начиная с PHP 5. Более подробную информацию об этой функции можно найти по адресу `http://www.php.net/set_error_handler`. Кроме того, информацию об обработке ошибок и ведению журналов в PHP можно найти в руководстве, по адресу `http://www.php.net/manual/en/ref.errorfunc.php`.

Серьезные типы ошибок (`E_ERROR`, `E_PARSE`, `E_CORE_ERROR`, `E_CORE_WARNING`, `E_COMPILE_ERROR` и `E_COMPILE_WARNING`) нельзя перехватывать и обрабатывать с помощью метода `ErrorHandler::Handler()`, но другие типы ошибок (например, `E_WARNING`) — можно.

Метод обработки ошибок, `Handler()`, будет работать следующим образом.

- Он составит подробное сообщение об ошибке.
- Если он сконфигурирован на это, он отправит сообщение администратору сайта по электронной почте.
- Если он сконфигурирован на это, он запишет сообщение в журнал ошибок.
- Если он сконфигурирован на это, он выведет сообщение об ошибке на странице ответа.
- Серьезные ошибки приведут к прекращению выполнения приложения. Менее серьезные ошибки позволят приложению продолжить выполнение.

Созданию класса `ErrorHandler` посвящено следующее упражнение.

Упражнение: реализация класса `ErrorHandler`

1. Добавьте в файл `include/config.php` следующие переменные конфигурации.

```
<?php
// В SITE_ROOT содержится полный путь к папке tshirtshop
define('SITE_ROOT', dirname(dirname(__FILE__)));

// Папки приложения
define('PRESENTATION_DIR', SITE_ROOT . '/presentation/');
define('BUSINESS_DIR', SITE_ROOT . '/business/');

// Настройки, необходимые для конфигурирования Smarty
define('SMARTY_DIR', SITE_ROOT . '/libs/smarty/');
define('TEMPLATE_DIR', PRESENTATION_DIR . 'templates');
define('COMPILE_DIR', PRESENTATION_DIR . 'templates_c');
define('CONFIG_DIR', SITE_ROOT . '/include/configs');

// Эти значения должны быть равны true на этапе разработки
```

```
define('IS_WARNING_FATAL', true);
define('DEBUGGING', true);

// Типы ошибок, о которых должны составляться сообщения
define('ERROR_TYPES', E_ALL);

// Настройки отправки сообщений администраторам по электронной почте
define('SEND_ERROR_MAIL', false);
define('ADMIN_ERROR_MAIL', 'Administrator@example.com');
define('SENDMAIL_FROM', 'Errors@example.com');
ini_set('sendmail_from', SENDMAIL_FROM);

// По умолчанию мы не записываем сообщения в журнал
define('LOG_ERRORS', false);
define('LOG_ERRORS_FILE', 'c:\\tshirtshop\\errors_log.txt'); // Windows
// define('LOG_ERRORS_FILE', '/home/username/tshirtshop/errors.log'); // Linux

/* Общее сообщение об ошибке, которое должно отображаться вместо
подробной информации (если DEBUGGING равно false) */
define('SITE_GENERIC_ERROR_MESSAGE', '<h1>TShirtShop Error!</h1>');
?>
```

2. В папке `tshirtshop` создайте вложенную папку `business`.

3. В папке `business` создайте файл `error_handler.php` и наберите в нем следующий код.

```
<?php
class ErrorHandler
{
  // Закрытый конструктор, не позволяющий непосредственно
  // создавать объекты класса
  private function __construct()
  {
  }

  /* Выбираем метод ErrorHandler::Handler в качестве метода обработки ошибок */
  public static function SetHandler($errTypes = ERROR_TYPES)
  {
    return set_error_handler(array ('ErrorHandler', 'Handler'), $errTypes);
  }

  // Метод обработки ошибок
  public static function Handler($errNo, $errStr, $errFile, $errLine)
  {
  /* Первые два элемента массива трассировки нам неинтересны:
  - ErrorHandler.GetBacktrace
  - ErrorHandler.Handler */
  $backtrace = ErrorHandler::GetBacktrace(2);

  // Сообщения об ошибках, которые будут выводиться, отправляться по
  // электронной почте или записываться в журнал
  $error_message = "\nERRNO: $errNo\nTEXT: $errStr" .
                   "\nLOCATION: $errFile, line " .
                   "$errLine, at " . date('F j, Y, g:i a') .
                   "\nShowing backtrace:\n$backtrace\n\n";

  // Отправляем сообщения об ошибках, если SEND_ERROR_MAIL равно true
  if (SEND_ERROR_MAIL == true)
```

```
  error_log($error_message, 1, ADMIN_ERROR_MAIL, "From: " .
            SENDMAIL_FROM . "\r\nTo: " . ADMIN_ERROR_MAIL);

// Записываем сообщения в журнал, если LOG_ERRORS равно true
if (LOG_ERRORS == true)
  error_log($error_message, 3, LOG_ERRORS_FILE);

/* Выполнение не прекращается при предупреждениях,
   если IS_WARNING_FATAL равно false. Ошибки E_NOTICE и E_USER_NOTICE
   тоже не приводят к прекращению выполнения */
if (($errNo == E_WARNING && IS_WARNING_FATAL == false) ||
  ($errNo == E_NOTICE || $errNo == E_USER_NOTICE))

// Если ошибка не фатальная...
{
  // Выводим сообщение, только если DEBUGGING равно true
  if (DEBUGGING == true)
    echo '<div class="error_box"><pre>' . $error_message . '</pre></div>';
  }
  else
  // Если ошибка фатальная...
  {
    // Выводим сообщение об ошибке
    if (DEBUGGING == true)
      echo '<div class="error_box"><pre>'. $error_message . '</pre></div>';
    else
      echo SITE_GENERIC_ERROR_MESSAGE;
      // Останавливаем обработку запроса
      exit();
    }
  }

  // Составляем список вызовов
  public static function GetBacktrace($irrelevantFirstEntries)
  {
    $s = '';
    $MAXSTRLEN = 64;
    $trace_array = debug_backtrace();

    for ($i = 0; $i < $irrelevantFirstEntries; $i++)
      array_shift($trace_array);
    $tabs = sizeof($trace_array) - 1;

    foreach ($trace_array as $arr)
    {
      $tabs -= 1;
      if (isset ($arr['class']))
        $s .= $arr['class'] . '.';
      $args = array ();

      if (!empty ($arr['args']))
        foreach ($arr['args']as $v)
          {
          if (is_null($v))
            $args[] = 'null';
          elseif (is_array($v))
            $args[] = 'Array[' . sizeof($v) . ']';
          elseif (is_object($v))
            $args[] = 'Object: ' . get_class($v);
          elseif (is_bool($v))
            $args[] = $v ? 'true' : 'false';
```

```
            else
            {
              $v = (string)@$v;
              $str = htmlspecialchars(substr($v, 0, $MAXSTRLEN));
              if (strlen($v) > $MAXSTRLEN)
                $str .= '...';
              $args[] = '"' . $str . '"';
            }
          }

        $s .= $arr['function'] . '(' . implode(', ', $args) . ')';
        $line = (isset ($arr['line']) ? $arr['line']: 'unknown');
        $file = (isset ($arr['file']) ? $arr['file']: 'unknown');
        $s .= sprintf(' # line %4d, file: %s', $line, $file);
        $s .= "\n";
      }

    return $s;
  }
}
?>
```

ЗАМЕЧАНИЕ

Из последующих глав вы узнаете больше о PHP и объектно-ориентированном программировании. А пока что мы хотели бы обратить ваше внимание на конструктор класса, специальную функцию с именем `__construct()`. Если вы набираете код вручную, обратите внимание на два знака подчеркивания в начале имени функции. Пропуск одного знака подчеркивания — распространенная причина ошибок в сценариях на PHP 5.

4. Внесите изменения в файл `index.php`, чтобы задействовать только что созданный файл `error_handler.php` и выбрать метод-обработчик ошибок.

```
<?php
// Подключаем служебные файлы
require_once 'include/config.php';
require_once BUSINESS_DIR . 'error_handler.php';

// Задаем обработчик ошибок
ErrorHandler::SetHandler();

// Загружаем шаблон страницы приложения
require_once PRESENTATION_DIR . 'application.php';
```

5. Поздравляем! Вы только что закончили создавать новый код для обработки ошибок. Давайте его протестируем. Сначала загрузите веб-сайт в браузере, чтобы убедиться, что вы все набрали корректно. Если сайт не сообщит о найденных ошибках, протестируйте систему обработки ошибок, добавив в файл `index.php` следующий фрагмент кода.

```
<?php
// Подключаем служебные файлы
require_once 'include/config.php';
require_once BUSINESS_DIR . 'error_handler.php';

// Задаем обработчик ошибок
ErrorHandler::SetHandler();

// Загружаем шаблон страницы приложения
require_once PRESENTATION_DIR . 'application.php';
```

```
// Отображаем страницу
$application->display('store_front.tpl');

// Пытаемся загрузить несуществующий файл
require_once 'inexistent_file.php';
?>
```

А теперь попробуйте обновить страницу в окне браузера и полюбуйтесь свеженьким сообщением об ошибке, которое вы же сами только что создали (рис. 3.10).

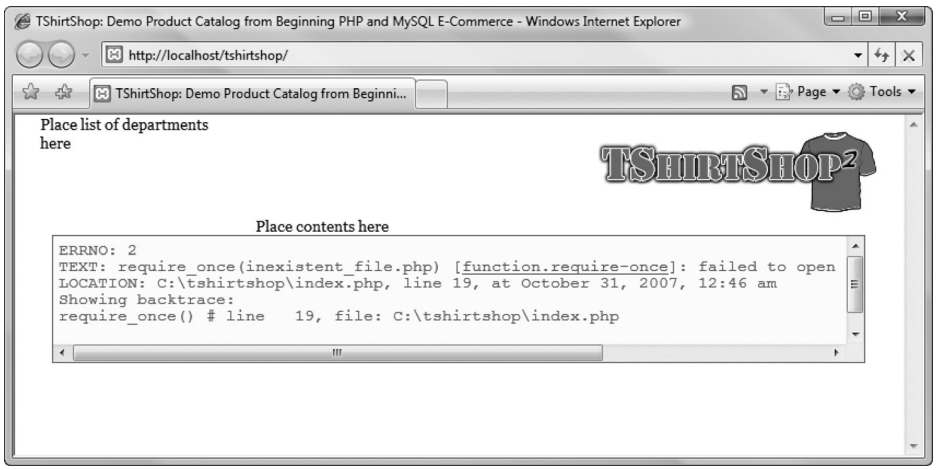


Рис. 3.10. Сообщение об ошибке с последовательностью вызовов функций

ПРЕДУПРЕЖДЕНИЕ

Не забудьте удалить вызывающую ошибку строку из файла `index.php`, прежде чем двигаться дальше.

Как это работает: обработка ошибок

Перехватом и обработкой ошибок в нашем веб-сайте занимается метод `ErrorHandler::Handler()`, который находится в файле `error_handler.php`. Код, регистрирующий этот метод в качестве обработчика ошибок, находится в методе `ErrorHandler::SetHandler()`, который вызывается из файла `index.php`.

```
/* Выбираем метод ErrorHandler::Handler в качестве метода обработки ошибок */
public static function SetHandler($errTypes = ERROR_TYPES)
{
return set_error_handler(array ('ErrorHandler', 'Handler'), $errTypes);
}
```

ЗАМЕЧАНИЕ

Второй параметр метода `set_error_handler()` указывает диапазон ошибок, которые должны перехватываться. Значение `E_ALL` указывает на все типы ошибок, включая ошибки `E_NOTICE`, о которых приложение должно сообщать на этапе разработки.

Метод ErrorHandler::Handler() составляет сообщение об ошибке с помощью метода ErrorHandler::GetBacktrace() и передает это сообщение клиентскому браузеру или записывает его в файл журнала, или отправляет администратору сайта по электронной почте. Поведение метода Handler() можно настроить, отредактировав файл config.php.

Метод GetBacktrace() получает информацию о стеке вызовов от функции debug_backtrace(), появившейся в PHP 4.3.0, и форматирует эту информацию, чтобы генерировать сообщения, похожие на сообщения об ошибках Java. Вам не обязательно понимать весь код метода GetBacktrace(), если вы не хотите изменить оформление сообщений об ошибках. Параметр 2, передаваемый GetBacktrace(), указывает, что из информации о стеке вызовов можно удалить сведения о двух последних вызовах (это вызовы ErrorHandler::Handler() и ErrorHandler::GetBacktrace()).

Подробное сообщение об ошибке составляет метод ErrorHandler::Handler().

```
$backtrace = ErrorHandler::GetBacktrace(2);

// Сообщения об ошибках, которые будут выводиться, отправляться по
// электронной почте или записываться в журнал
$error_message = "\nERRNO: $errNo\nTEXT: $errStr" .
                 "\nLOCATION: $errFile, line " .
                 "$errLine, at " . date('F j, Y, g:i a') .
                 "\nShowing backtrace:\n$backtrace\n\n";
```

В зависимости от настроек, заданных в файле config.php, метод решает, что сделать с сообщением — вывести в браузере, записать в журнал и/или отправить по электронной почте. Здесь мы используем функцию PHP error_log(), которая знает, как отправлять сообщения по электронной почте или записывать их в журнал.

```
// Отправляем сообщения об ошибках, если SEND_ERROR_MAIL равно true
if (SEND_ERROR_MAIL == true)
  error_log($error_message, 1, ADMIN_ERROR_MAIL, "From: " .
            SENDMAIL_FROM . "\r\nTo: " . ADMIN_ERROR_MAIL);
// Записываем сообщения в журнал, если LOG_ERRORS равно true
if (LOG_ERRORS == true)
  error_log($error_message, 3, LOG_ERRORS_FILE);
```

ЗАМЕЧАНИЕ

Если вы хотите отправлять сообщения об ошибках по электронной почте на локальный адрес (наподобие some_name@localhost), запустите на своей машине сервер Simple Mail Transfer Protocol (SMTP). В дистрибутивах Linux Red Hat (или Fedora) для запуска этого сервера достаточно консольной команды

```
service sendmail start
```

На компьютерах с Windows вам придется открыть IIS Manager и убедиться, что запущен Default SMTP Virtual Server.

В процессе разработки сайта константе DEBUGGING должно быть присвоено значение true, но после сдачи проекта ее стоит сбросить в false, чтобы вместо отладочной информации отображалось понятное рядовому пользователю сообщение об ошибке, а несущественные ошибки игнорировались.

Ошибки типа E_WARNING довольно замысловатые, поскольку трудно определить, при каких из них нужно останавливать обработку запросов. Значение константы IS_WARNING_FATAL задается в файле config.php и определяет, будут ли ошибки этого типа считаться фатальными. Ошибки типов E_NOTICE и E_USER_NOTICE фатальными не считаются.

```
/* Выполнение не прекращается при предупреждениях,
   если IS_WARNING_FATAL равно false. Ошибки E_NOTICE и E_USER_NOTICE
   также не приводят к прекращению выполнения */
if (($errNo == E_WARNING && IS_WARNING_FATAL == false) ||
  ($errNo == E_NOTICE || $errNo == E_USER_NOTICE))
// Если ошибка не фатальная...
{
```

```
// Выводим сообщение, только если DEBUGGING равно true
if (DEBUGGING == true)
  echo '<div class="error_box"><pre>' . $error_message . '</pre></div>';
}
else
// Если ошибка фатальная...
{
  // Выводим сообщение об ошибке
  if (DEBUGGING == true)
    echo '<div class="error_box"><pre>'. $error_message . '</pre></div>';
  else
    echo SITE_GENERIC_ERROR_MESSAGE;
    // Останавливаем обработку запроса
    exit();
  }
}
```

В последующих главах нам понадобится вручную вызывать ошибки с помощью функции PHP `trigger_error()`, которая позволяет выбирать тип генерируемой ошибки. По умолчанию эта функция генерирует ошибки типа `E_USER_NOTICE`, которые не считаются фатальными, но метод `ErrorHandler::Handler()` генерирует сообщения о них.

Подготовка базы данных

Последняя наша задача в этой главе — создать базу данных MySQL, которую мы будем использовать в следующей главе. Мы покажем вам, как создать эту базу данных и учетную запись пользователя с полными привилегиями с помощью phpMyAdmin, поставляемого в составе XAMPP; впрочем, вы можете использовать другие инструменты, например Toad (`http://www.quest.com/toad-for-mysql`) или даже текстовую консоль самого mySQL.

Прежде чем мы двинемся дальше, убедитесь, что MySQL запущен на вашем компьютере. Под управлением Windows вы можете запустить его с помощью XAMPP Control Panel, а под управлением Linux он обычно запускается автоматически при запуске XAMPP (для запуска XAMPP служит команда `/opt/lamp/lamp start`). Выполните следующее упражнение, чтобы создать базу данных и новую учетную запись пользователя.

Упражнение: создание базы данных MySQL и новой учетной записи пользователя

1. Загрузите в браузере страницу phpMyAdmin (`http://localhost/phpmyadmin`), введите `tshirtshop` в текстовом поле Create new database и выберите в списке MySQL connection collation пункт `utf8_unicode_ci`, как показано на рис. 3.11.
2. Щелкните на кнопке Create, чтобы создать базу данных. На открывшейся странице (рис. 3.12) будет показан SQL-запрос, с помощью которого phpMyAdmin создал базу данных.

ЗАМЕЧАНИЕ

Об SQL-запросах вы узнаете больше из главы 4. SQL — это язык, используемый для взаимодействия с базой данных. Однако многие операции, включая создание баз данных и таблиц в них, можно выполнять с помощью графических интерфейсов, например phpMyAdmin, которые будут генерировать SQL-запросы вместо вас.

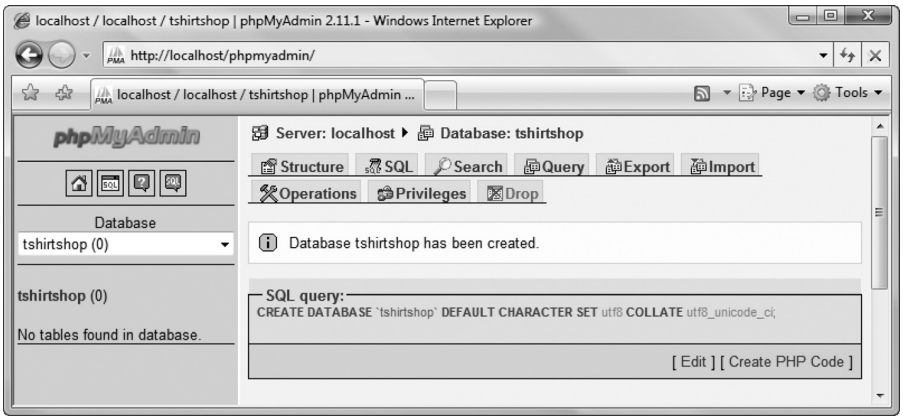


Рис. 3.11. Создание базы данных `tshirtshop` с помощью phpMyAdmin

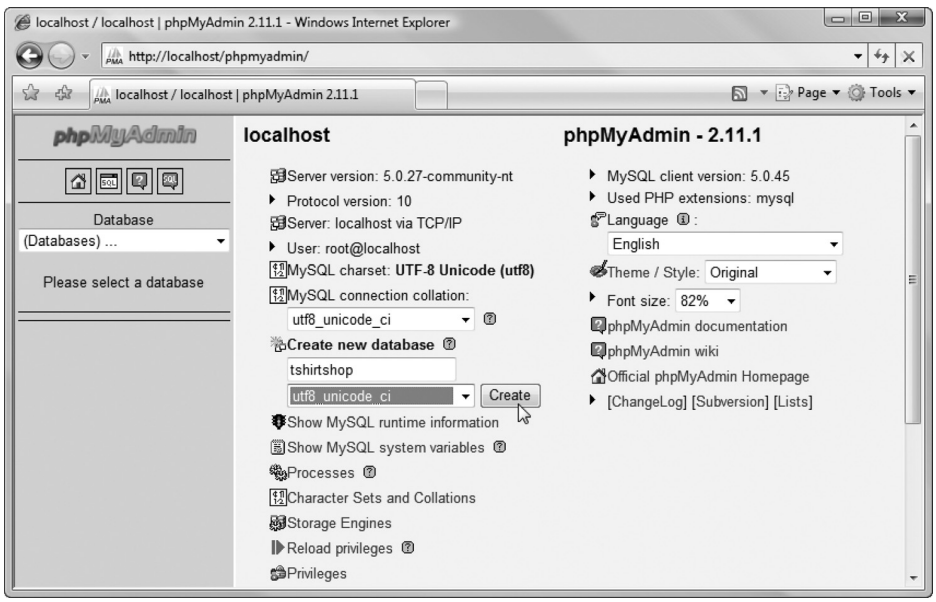


Рис. 3.12. Редактирование базы данных с помощью phpMyAdmin

3. Теперь добавьте в базу данных нового пользователя. Уровень данных в нашем приложении будет работать с базой данных, используя идентификатор этого пользователя. Мы создадим пользователя с именем `tshirtshopadmin` (и таким же паролем). У этого пользователя будет право полного доступа к базе данных TShirtShop, но не к другим базам данных. За более подробной информацией об учетных записях пользователей в MySQL можете обратиться к соответствующей документации по адресу `http://dev.mysql.com/doc/refman/5.1/en/account-management-sql.html`.

А теперь перейдите на вкладку SQL и введите с клавиатуры запрос, показанный на рис. 3.13.

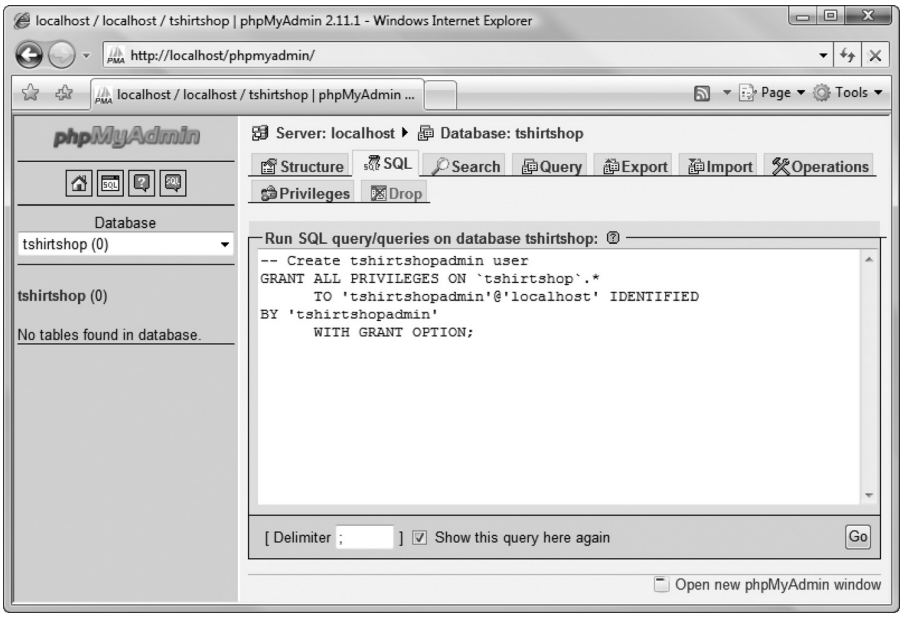


Рис. 3.13. Создание нового пользователя базы данных

4. Введя запрос, щелкните на кнопке Go. MySQL должен сообщить вам об успешном выполнении запроса (как на рис. 3.14).

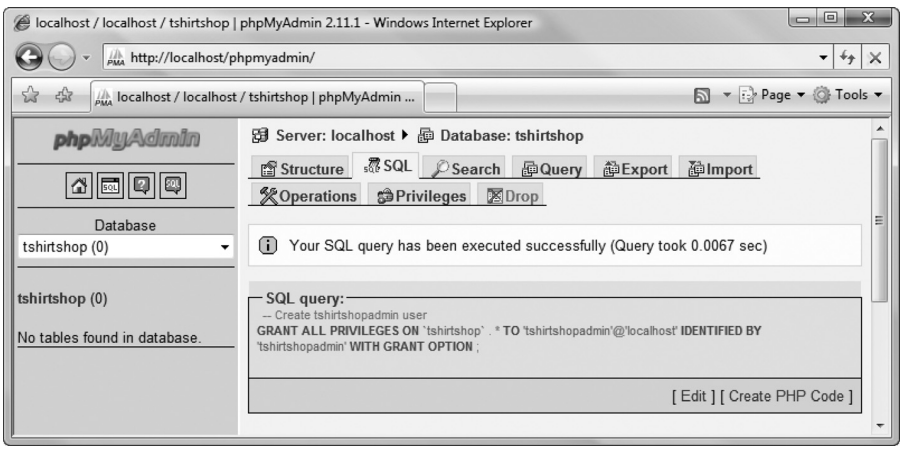


Рис. 3.14. Новый пользователь успешно создан

Загрузка исходного кода

Код, приведенный в листингах этой главы, можно загрузить с сайтов авторов книги по адресам `http://www.cristiandarie.ro` и `http://www.emilianbalanescu.ro` или из раздела Downloads сайта издательства Apress (`http://www.apress.com`). Вы

можете создавать приложения, просто последовательно выполняя приложения из этой книги, но если вы хотите проверить что-то на версии, которая наверняка работает, можете зайти по указанным адресам. Инструкции по загрузке кода для отдельных глав есть в файле `welcome.html` из загружаемого архива.

Резюме

Можете положить ноги на стол и наслаждаться сознанием хорошо выполненной работы. В этой главе вы увидели результаты применения в приложении эффективной архитектуры. Пока что наше приложение не снабжено никакой сложной функциональностью, и эти результаты станут более очевидными в последующих главах, когда мы станем добавлять в него новые элементы.

В этой главе мы создали базовую, статичную часть уровня представления, реализовали часть механизма обработки ошибок и создали базу данных `tshirtshop`, которая будет использоваться на уровне данных. В следующей главе мы займемся каталогом товаров, и вы немало узнаете о динамической генерации визуального содержимого из данных, получаемых из базы данных. В генерации этого содержимого будут участвовать и управляющая логика из уровня приложения, и элементы управления, и компоненты из уровня представления.

Глава 4

Создание каталога товаров: часть I

К данному моменту вы познакомились с трехуровневой архитектурой и реализовали небольшую часть главной страницы нашего веб-сайта. Приступим к следующему этапу — созданию каталога товаров.

Поскольку этот каталог состоит из множества компонентов, мы сочли целесообразным посвятить его созданию две главы. В этой главе мы создадим первую таблицу с данными в базе данных, реализуем методы доступа на уровне логики приложения и обсудим, как пользоваться уровнем данных. К концу этой главы в составе нашей страницы появится первое динамически генерируемое содержимое. В следующей главе мы закончим создание каталога товаров, добавив поддержку категорий, списков товаров, страниц с подробной информацией о товарах и многое другое.

Основные темы, которые мы рассмотрим в этой главе, таковы.

- Анализ структуры каталога товаров и функциональности, которую он должен поддерживать
- Создание структур в базе данных для самого каталога и уровня данных этого каталога
- Создание объектов для работы с каталогом на уровне логики приложения
- Реализация функционального пользовательского интерфейса для каталога товаров

Демонстрация товаров посетителям сайта

Одна из самых важных частей любого электронного магазина — это механизм, позволяющий посетителям просматривать предлагаемые товары. Представьте, как выглядел бы сайт Amazon.com без своего каталога товаров!

Независимо от того, ищет ли посетитель конкретный товар или просматривает все предлагаемые, вы должны сделать этот процесс простым и приятным. В конце концов, вы заинтересованы в том, чтобы ваши посетители как можно быстрее находили нужные им товары и покупали их. Именно для этого сайты электронных магазинов снабжаются механизмом поиска и структурирования товаров по категориям.

Для небольшого электронного магазина достаточно рассортировать предлагаемые товары по категориям, но если товаров много, понадобятся дополнительные инструменты для структурирования их каталога.

Определение размеров каталога — одна из важнейших наших задач в этой главе. Учтите, что при профессиональном подходе этот параметр должен быть определен задолго до написания кода — еще на этапе составления требований к проекту. Однако в учебной книге наподобие этой лучше осваивать каждую тему по отдельности.

Создав общую структуру каталога, мы напишем код, который оживит его и заставит работать.

Как выглядит каталог товаров

Сегодня посетители электронных магазинов гораздо требовательнее, чем в минувшие времена. Они желают быстро находить всю доступную информацию по интересующим их товарам или услугам, иначе они, скорее всего, уйдут к вашим конкурентам, не дав вам второго шанса. Разумеется, такой результат вас вряд ли устроит, поэтому нужно сделать все, что в наших силах, чтобы упростить работу пользователей с каталогом.

Поскольку на момент запуска электронного магазина в нем будет порядка 100 товаров, а в дальнейшем — еще больше, недостаточно просто рассортировать их по категориям. Кроме того, в магазине есть несколько отделов, в каждом из которых используются свои категории. В свою очередь, в каждой категории может быть произвольное количество товаров.

ЗАМЕЧАНИЕ

Позже мы создадим специальный раздел сайта для администраторов — своего рода панель управления сайтом. Этот раздел позволит обновлять данные об отделах, категориях и товарах. А пока что нам придется заполнять базу данных вручную (хотя вы можете "смошенничать", воспользовавшись SQL-сценариями, которые есть среди доступного для загрузки кода для этой книги на сайте `http://www.apress.com`).

Еще одна важная деталь, о которой необходимо подумать, — может ли одна категория относиться к нескольким отделам и может ли один товар фигурировать в нескольких категориях. Как вы, вероятно, понимаете, ответы на эти вопросы сильно повлияют на реализацию каталога товаров, так что по этому поводу лучше проконсультироваться с заказчиком проекта.

В проекте TShirtShop каждая категория может использоваться только в одном отделе, но конкретные товары могут относиться более чем к одной категории: например, товар Kat Over New Moon будет относиться к категориям Animal и Christmas. Такое положение дел заметно повлияет на структуру базы данных, и мы обратим ваше внимание на это, когда доберемся до создания соответствующей части базы данных.

И наконец, кроме товаров, рассортированных по категориям, мы хотим реализовать поддержку для специальных предложений. Специальные предложения могут относиться как к магазину в целом, так и к определенным отделам. В следующем разделе представлено несколько рисунков, поясняющих эту идею.

Предварительный просмотр каталога товаров

Хотя полностью работающий каталог товаров мы получим только в конце следующей главы, сейчас мы посмотрим, как он будет выглядеть, чтобы вы представляли себе, к чему мы движемся. На рис. 4.1 показана основная страница сайта TShirtShop с двумя товарами, относящимися к специальным предложениям.

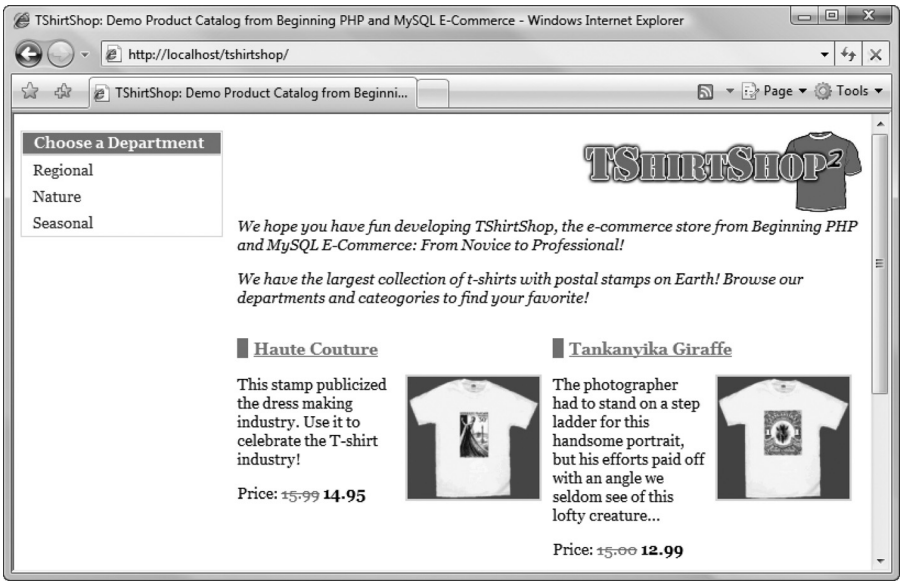


Рис. 4.1. Главная страница сайта с двумя специальными предложениями

Обратите внимание на список отделов в верхней левой части страницы. Он генерируется динамически на основе данных, получаемых из базы данных (мы составим его позже в этой главе).

Когда посетитель щелкает на названии отдела в списке, в браузере открывается главная страница данного отдела. При этом список специальных предложений магазина заменяется информацией, относящейся к выбранному отделу, включая его список специальных предложений. Например, на рис. 4.2 показан сайт в окне браузера после того, как посетитель щелкнет на имени отдела Seasonal.

Ниже списка отделов теперь отображается список категорий выбранного отдела. В правой части окна представлены название выбранного отдела, его описание и специальные предложения. Если на какой-то странице должно отображаться больше товаров, чем некоторое пороговое количество, страница разбивается на несколько подстраниц, и отображаются элементы навигации, позволяющие пользователю перемещаться между подстраницами. Эти элементы видны на рис. 4.2.

Мы решили отображать на главной странице отдела только специальные предложения, а остальные товары отображать на страницах категорий. На странице отдела текст над списком товаров представляет собой описание выбранного отдела. Это значит, что описание должно храниться в базе данных наряду с названием отдела. При выборе категории из списка должны отображаться все товары этой категории, а также название категории и ее описание. После щелчка на названии или изображении товара должна открываться страница с его описанием. Пример такой страницы показан на рис. 4.3. В списках отделов и категорий при просмотре описания товаров должны сохраняться выделения для выбранных пунктов; это позволит посетителю лучше ориентироваться на сайте. Кроме того, на странице с описанием товара есть ссылка `Continue shopping`, возвращающая посетителя на страницу, с которой была открыта страница с описанием товара.

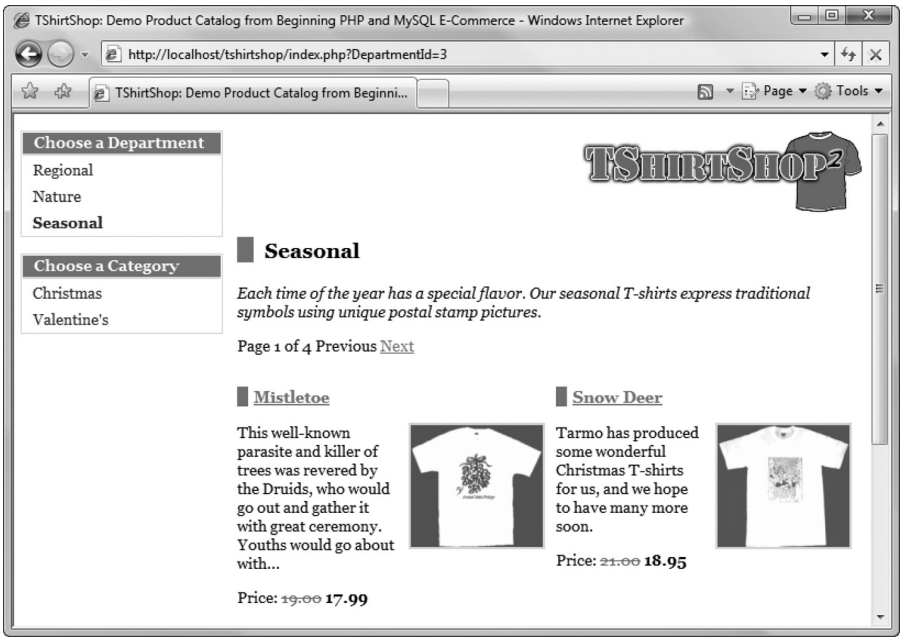


Рис. 4.2. Страница отдела Seasonal

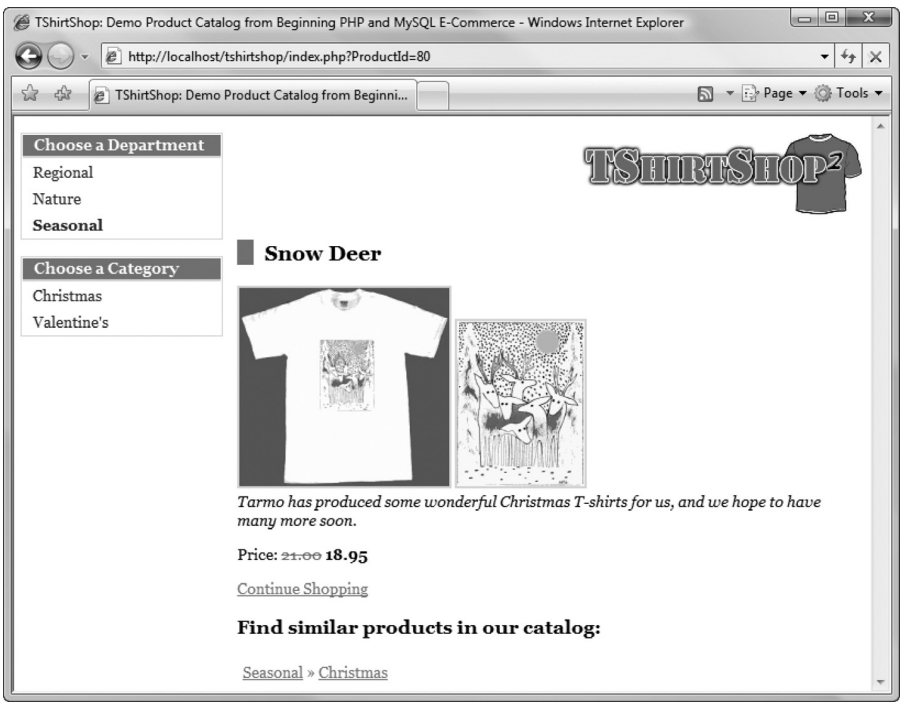


Рис. 4.3. Пример страницы с описанием товара

При выборе определенной категории выводится только список товаров этой категории — специальные предложения больше не отображаются. Кроме того, меняется текст описания. Теперь этот текст относится к выбранной категории.

Последовательность действий в этой главе

Как видите, каталог товаров — не слишком сложная часть программы, но в нем есть много частей, каждую из которых нам придется реализовать. В этой главе мы ограничимся созданием списка отделов (рис. 4.4).

Список отделов будет первым динамически генерируемым элементом нашего сайта (названия отделов будут извлекаться из базы данных).

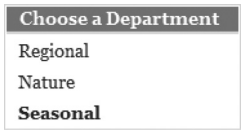


Рис. 4.4. Список отделов

Здесь мы составим список отделов. Разобравшись, как этот список работает, вы сможете в главе 5 быстро реализовать остальные части каталога товаров.

В главе 2 мы обсуждали трехуровневую архитектуру, которая будет применяться в веб-приложении. Каталог товаров вполне вписывается в эту архитектуру, и его компоненты (включая список категорий) будут разнесены по разным уровням. На рис. 4.5 показано, что мы создадим на каждом из уровней, чтобы получить работающий список категорий.

До сих пор мы затрагивали только уровень представления и уровень логики приложения. Сейчас, создавая каталог, мы затронем последний уровень, уровень данных, и будем работать с самой базой данных `tshirtshop` (специалисты расходятся во мнениях по поводу того, относится ли к трехуровневой архитектуре хранилище данных).

На рис. 4.5 показаны компоненты, которые нужно создать, чтобы на сайте появился динамически генерируемый список отделов. Мы начнем с базы данных и постепенно доберемся до уровня презентации.

1. Создадим в базе данных таблицу `department`. В ней будут храниться данные об отделах магазина. Прежде чем добавить эту таблицу, мы рассмотрим основные концепции работы с реляционными базами данных.
2. Напишем хранимую процедуру MySQL с именем `catalog_get_departments_list`, которая возвращает ID и названия отделов из таблицы `departments`. PHP-сценарии будут вызывать эту хранимую процедуру, чтобы сгенерировать список отделов для посетителя сайта. Хранимые процедуры MySQL относятся к уровню данных приложения. На этом этапе вы научитесь обращаться к базе данных на языке SQL.
3. Создадим класс `DatabaseHandler` — вспомогательный класс, выполняющий основные операции по взаимодействию с базой данных. Это будет класс-обертка для нескольких методов PDO, содержащий механизмы обработки ошибок, возникающих при работе с базой данных.
4. Создадим компоненты уровня логики приложения для списка отделов (класс `Catalog` и метод `GetDepartments()`). Вы увидите, как используется для работы с базой данных класс `DatabaseHandler`.
5. Реализуем компонентный шаблон Smarty `departments_list` и подключаемый модуль Smarty `presentation_object`, объединяющий шаблоны и соответствующие им объекты представления. Шаблону `departments_list` соответствует объект представления `DepartmentsList`.

Итак, первый шаг — создание таблицы `department` в базе данных.

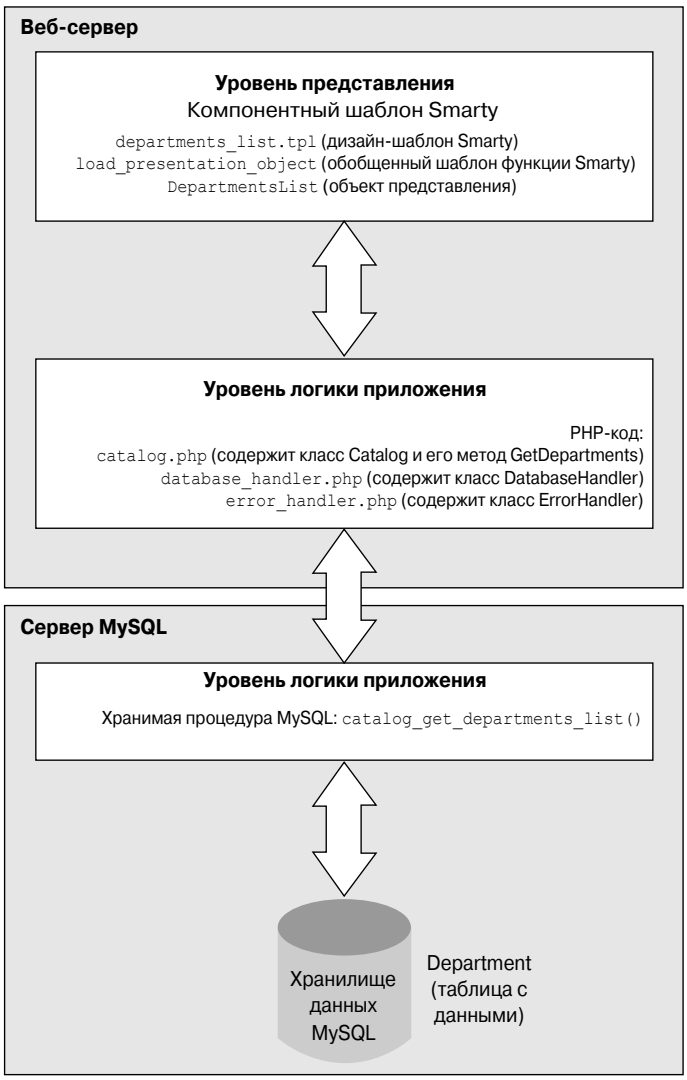


Рис. 4.5. Компоненты, необходимые для генерации списка отделов

Хранение информации из каталога

Подавляющее большинство веб-приложений, в том числе приложения электронных магазинов, основываются на источниках данных. Четкое понимание того, какие данные должно хранить и обрабатывать приложение, — важное требование к разработчикам.

Чаще всего для хранения данных в веб-приложениях используются реляционные базы данных, однако это отнюдь не единственный вариант. Ничто не мешает вам создать собственный уровень доступа к данным и такие структуры для их хранения, какие вы сочтете удобными.

ЗАМЕЧАНИЕ

В некоторых случаях удобнее хранить данные в виде простых текстовых или XML-файлов, а не в базах данных, но такие решения плохо подходят для электронных магазинов, и мы не будем рассматривать их в этой книге.

Хотя реляционные базы данных и не являются основной темой данной книги, ниже мы расскажем все, что понадобится вам для создания каталога товаров.

Можно считать, что реляционная база данных состоит из *таблиц данных* (data tables) и *отношений* (relations) между этими таблицами. Поскольку в этой главе нам понадобится только одна таблица данных, мы коснемся только той части теории баз данных, которая описывает эти таблицы как отдельные элементы. В следующей главе, когда мы добавим в базу новые таблицы, мы рассмотрим дополнительную часть теории, описывающую отношения между таблицами.

ЗАМЕЧАНИЕ

В реальности обычно проектируется сразу вся база данных (или по крайней мере все таблицы, необходимые для запуска первого варианта проекта). В этой книге мы разделили создание базы данных на несколько этапов, чтобы лучше поддерживать баланс между теорией и практикой.

Итак, для начала — немного теории.

Концепция таблиц данных

Этот раздел представляет собой краткое введение в базы данных, в котором содержится вся информация, необходимая для создания простых таблиц данных. Мы кратко рассмотрим все элементы, из которых состоит таблица данных.

- Первичные ключи (primary keys)
- Типы данных MySQL (data types)
- Столбцы (columns) с ограничениями UNIQUE
- Столбцы с ограничениями NOT NULL и значениями по умолчанию
- Столбцы с автоинкрементом
- Индексы (indexes)

ЗАМЕЧАНИЕ

Если у вас есть опыт работы с MySQL, можете без зазрения совести пропустить этот раздел и сразу перейти к разделу “Создание таблицы department”.

Таблица данных состоит из строк и столбцов. Иногда строки называют *записями* (records), а столбцы — *полями* (fields).

Поскольку в этой главе мы будем создавать только список отделов, нам хватит одной таблицы — `department`. В ней будут храниться данные об отделах (она, вероятно, самая простая из таблиц в нашем проекте).

Если вы точно знаете, какие данные должны храниться в таблице, можете легко создать ее с помощью консольного клиентского интерфейса MySQL. Но, проектируя таблицу, вы должны хорошо подумать о том, какие поля в ней должны быть и какие типы данных нужно использовать в этих полях. Кроме типов данных, у полей есть еще несколько свойств, которые мы рассмотрим ниже.

Чтобы определить, какие поля должны быть в таблице `department`, давайте запишем на бумаге несколько примеров записей, которые нужно будет в ней хранить. Из предыдущих примеров вы, вероятно, помните, что нам нужно хранить немного информации об отделе — только его название и краткое описание. Таблица, в которой хранится эта информация, может выглядеть примерно так, как на рис. 4.6 (в базе данных мы реализуем ее позже, после того, как рассмотрим всю нужную теорию).

name	description
Regional	Proud of your country? Wear a T-shirt with a national symbol stamp!
Nature	Find beautiful T-shirts with animals and flowers in our Nature department!
Seasonal	Each time of the year has a special flavor. Our seasonal T-shirts express traditional symbols using unique postal stamp pictures.

Рис. 4.6. Данные в таблице `department`

Из такой таблицы можно извлекать названия отделов для заполнения списка в верхней левой части веб-страницы и описания отделов для использования в качестве заголовков в списке специальных предложений.

Первичные ключи

С таблицами в электронной базе данных нужно обращаться немного не так, как с обычными, бумажными. Основное требование к таблицам в базе данных — каждая строка в таблице должна быть однозначно определяемой. Это вполне понятно, если учесть, что после добавления строки в таблицу вам рано или поздно понадобится ее прочитать. А если вы не сможете однозначно сообщить базе данных, какая строка вам нужна, вы не сможете ее получить. Поэтому в каждой строке должно быть что-то, что делает ее уникальной — отличающейся от остальных строк. Предположим, например, что вы добавите в таблицу `department`, показанную на рис. 4.6, еще одну строку. После этого таблица будет выглядеть так, как на рис. 4.7.

name	description
Regional	Proud of your country? Wear a T-shirt with a national symbol stamp!
Nature	Find beautiful T-shirts with animals and flowers in our Nature department!
Seasonal	Each time of the year has a special flavor. Our seasonal T-shirts express traditional symbols using unique postal stamp pictures.
Seasonal	Ooops! Don't try this at home!

Рис. 4.7. Два отдела с одинаковыми именами

Представьте себе, что из этой таблицы вам нужно извлечь описание отдела Seasonal. Возникает проблема — у нас есть два отдела с таким именем, т.е. значения в столбце `name` перестали быть уникальными. Если отправить к этой таблице запрос, выбирая данные по значениям в столбце `name`, можно получить две строки. Иногда это именно то, что требуется, но в данном случае мы ожидаем получить только одну строку, однозначно определяемую по значению в столбце `name`.

При проектировании реляционных баз данных эта проблема решается с помощью *первичного ключа* (primary key), позволяющего однозначно выбрать конкретную строку из множества строк. С технической точки зрения сам первичный ключ столбцом не является. Первичный ключ — это *ограничение* (constraint). Если оно применяется к столбцу, оно гарантирует, что значения в этом столбце будут разными во всех строках таблицы.

Ограничения — это правила, применяемые к таблицам базы данных. Они являются частью механизма, обеспечивающего *целостность данных* (data integrity). База данных следит за соблюдением этих правил и не позволяет нарушить целостность данных. Например, если вы попытаетесь поместить два одинаковых значе-

ния в столбец с ограничением PRIMARY KEY, база данных сообщит вам об ошибке и откажется выполнять эту операцию. Позже в настоящей главе мы убедимся в этом, проведя соответствующий эксперимент.

ЗАМЕЧАНИЕ

Первичный ключ — это не столбец, а ограничение, накладываемое на значения из этого столбца; однако для удобства мы будем называть первичным ключом именно столбец, к которому применено ограничение PRIMARY KEY.

Вернемся к нашему примеру. Сделав столбец name первичным ключом для таблицы department, мы решим проблему — база данных не позволит добавить в таблицу вторую строку со значением в столбце name, которое уже есть в таблице. Соответственно, выбор строки по значению столбца name будет давать в результате максимум одну строку (или ни одной, если заданного значения в столбце name нет в таблице).

ПОДСКАЗКА

Полагаю, об этом следует напомнить, хотя это и очевидно: в столбце первичного ключа не может быть NULL-значений.

В качестве альтернативы можно добавить в таблицу еще один столбец со значениями-идентификаторами, как на рис. 4.8. Этот столбец будет использоваться как первичный ключ.

department_id	name	description
1	Regional	Proud of your country? Wear a T-shirt with a national symbol stamp!
2	Nature	Find beautiful T-shirts with animals and flowers in our Nature department!
3	Seasonal	Each time of the year has a special flavor. Our seasonal T-shirts express traditional symbols using unique postal stamp pictures.

Рис. 4.8. Столбец department_id будет первичным ключом

В дальнейшем мы будем использовать суффикс _id в именах всех столбцов, которые будут первичными ключами в таблицах базы данных. Используя дополнительный столбец, можно хранить в таблице сведения о нескольких отделах с одинаковыми названиями, если у них будут разные идентификаторы (чтобы гарантировать уникальность значений в столбцах, не являющихся первичными ключами, можно использовать ограничение UNIQUE, которое мы рассмотрим чуть позже).

Есть два довода в пользу создания дополнительного столбца с номерами и использования его в качестве первичного ключа.

- *Производительность.* Ядро СУБД выполняет операции сортировки и поиска по числовым значениям гораздо быстрее, чем по строкам. Этот довод становится еще важнее при работе с несколькими таблицами, которые часто приходится соединять (соединения мы рассмотрим в главе 5).
- *Изменение названий отделов.* Если мы создадим дополнительный столбец с номерами-идентификаторами, то сможем свободно менять названия отделов, не изменяя их номера.

На рис. 4.8 первичный ключ состоит из одного столбца, но это необязательное требование. Можно образовать первичный ключ из нескольких столбцов при условии, что в каждой строке таблицы сочетания значений в этих столбцах будут уникальными (даже если значения в отдельных столбцах повторяются). В главе 5 мы

рассмотрим пример первичного ключа из нескольких столбцов, а пока что вам достаточно знать, что такие первичные ключи существуют.

ЗАМЕЧАНИЕ

Применение ограничения `PRIMARY KEY` к столбцу по умолчанию приводит к генерации для него индекса. Индексы — это объекты, предназначенные для ускорения работы базы данных и, значит, использующих ее приложений. Мы поговорим об индексах позже, в соответствующем разделе этой главы.

Ограничение UNIQUE

`UNIQUE` — это еще одна разновидность ограничений, которые можно применять к столбцам таблиц. Данное ограничение похоже на `PRIMARY KEY` в том, что оно запрещает повторение значений в столбце. Но между этими ограничениями есть разница. Для таблицы можно использовать только одно ограничение `PRIMARY KEY`, но сколько угодно ограничений `UNIQUE`.

Столбцы с ограничениями `UNIQUE` полезны, если в таблице уже есть первичный ключ, а вам нужно обеспечить уникальность значений в каком-то другом столбце (или группе столбцов). Например, чтобы запретить одинаковые названия отделов в таблице `department`, можно применить к столбцу `name` ограничение `UNIQUE`.

Вот что нужно помнить об ограничениях `UNIQUE`.

- Ограничение `UNIQUE` запрещает повторяющиеся значения в столбце, к которому оно применено.
- В таблице может быть несколько столбцов с ограничениями `UNIQUE`.
- В столбце с ограничением `UNIQUE` может быть сколько угодно значений `NULL`.
- Для столбцов с ограничением `UNIQUE`, как и `PRIMARY KEY`, по умолчанию автоматически генерируются индексы.

Столбцы и типы данных

В каждом столбце таблицы хранятся данные определенного типа. Взглянув на рис. 4.8, вы можете определить, что в столбце `department_id` хранятся цифровые данные, а в столбцах `name` и `description` — текстовые.

MySQL поддерживает обширный набор типов данных, и из этого набора нужно выбрать типы, которые мы будем использовать в таблице. В табл. 4.1 перечислены некоторые из поддерживаемых MySQL типов данных — те, которые, скорее всего, понадобятся в нашем проекте. Более подробное описание всех типов можно найти в документации к MySQL по адресу `http://www.mysql.org/doc/refman/5.1/en/data-types.html`.

ПОДСКАЗКА

За более подробной информацией обо всех аспектах MySQL, включая типы данных, можете обратиться к книге Тима Конверса, Джойса Парка и Кларка Моргана *PHP 5 и MySQL. Библия пользователя* ("Диалектика", 2006г).

Чтобы не переполнять книгу ненужной вам информацией, в табл. 4.1 в столбце "Тип данных" перечислено несколько типов, но в ячейках столбца "Описание" для каждого типа упомянуты похожие типы. Запоминать все сведения из таблицы вам не обязательно, но общее представление о типах данных в MySQL вы получить должны.

Таблица 4.1. Типы данных MySQL, используемые в проекте TShirtShop

Тип данных	Размер, байт	Описание
`int`	4	Позволяет хранить целые числа в диапазоне от −2 147 483 648 до 2 147 483 647. Похожие типы — `bigint`, `mediumint`, `smallint` и `tinyint`. Кроме того, есть тип `bit`, позволяющий хранить только значения 0 и 1
`decimal (M, N)`	M+2, если N > 0	По одному символу на каждую десятичную цифру, точку (если есть дробная часть) и знак "минус" (для отрицательных чисел)
	M+1, если N = 0	`decimal` — это числовой тип данных, используемый для хранения денежных сумм из-за его абсолютной точности. Чтобы сохранить точность значений этого типа, MySQL хранит данные значения в виде строк, а не чисел. M — это точность (количество разрядов, которые будут храниться), а N — масштаб (количество разрядов после запятой). Если N = 0, будут храниться только целые числа
`datetime`	8	Хранит значения даты и времени в диапазоне от 1000-01-01 00:00:00 до 9999-12-31 23:59:59
`varchar`	Переменный	Хранит строки переменной длины от 0 до 65 535 символов. Можно задать предельный размер хранимых в столбце строк
`text (blob)`	L+2, где $L<2^{16}$	Значения с максимальной длиной 65 535 ($2^{16}-1$) символов

Помните, что названия типов данных нечувствительны к регистру, поэтому могут в разных приложениях записываться и строчными, и прописными буквами.

А теперь давайте вернемся к таблице `department` и решим, какие типы данных мы будем в ней использовать. Не волнуйтесь о том, что этой таблицы еще нет в базе данных: мы создадим ее немного позже. В табл. 4.2 перечислены столбцы таблицы `department` и их свойства. Для столбца `department_id` мы используем тип `int`, а для столбцов `name` и `description` — тип `varchar`.

Таблица 4.2. Свойства столбцов таблицы `department`

Столбец	Тип	Кодовая таблица	NULL-значения	Дополнительные параметры
`department_id`	`int(11)`		Нет	`auto_increment`
`name`	`varchar(100)`	`utf8_unicode_ci`	Нет	
`description`	`varchar(1000)`	`utf8_unicode_ci`	Нет	

Для типа `varchar` число в скобках, например `varchar(100)`, означает максимальную длину строки, которую можно хранить в ячейке. Для названий отделов мы задали предел длины 100 символов, а для описаний — 1000 символов. Значения типа `integer`, как указано в табл. 4.1, всегда занимают 4 байт.

Ограничения NOT NULL и значения по умолчанию

Для каждого столбца в таблице можно указать, разрешены ли в его ячейках значения `NULL`. Проще всего воспринимать значения `NULL` как "неопределенные". Например, в таблице `department` обязательно должны быть значения в столбцах `department_id` и `name`, а значение в столбце `description` необязательно — мы можем добавлять в таблицу новые отделы, не задавая описаний для них. Если мы добавим

новую строку, не задавая значений для столбцов, в которых допустимы значения NULL, именно значения NULL будут автоматически вставлены в эти столбцы.

Для строк есть малозаметное, но существенное различие между пустыми строками и значениями NULL. Если вы добавите в таблицу товар и зададите в качестве его описания пустую строку, значит, описание товара есть — в нем просто нет текста, но это не значение NULL.

В столбцах, входящих в состав первичного ключа, значения NULL не допускаются. Для других столбцов вы можете сами решать, допустимы ли в них значения NULL.

В некоторых случаях вместо значений NULL лучше задать значения по умолчанию. При этом, если в таблицу будет добавлена строка, в которой нет значения для соответствующего столбца, в этот столбец будет вставлено значение по умолчанию. Значения по умолчанию могут быть непосредственными величинами (например, 0 или "неизвестно"), системными значениями или результатами вызова функций.

Столбцы с автоинкрементом

Столбцы с автоинкрементом — это просто столбцы, содержащие последовательно увеличивающиеся номера строк. Если столбец обозначить как столбец с автоинкрементом, MySQL автоматически будет присваивать значения ячейкам этого столбца при добавлении новых строк в таблицу. Обычно инкремент выполняется с шагом в единицу, т.е. если раньше максимальным значением в столбце было x, то при добавлении новой строки максимальным станет значение x+1.

Такой механизм обеспечивает уникальность всех значений в столбце, поэтому его удобно использовать в качестве первичного ключа. Вы уже знаете, что значения первичного ключа позволяют однозначно выбирать каждую строку в таблице. Если столбцу первичного ключа назначить свойство автоинкремента, MySQL будет автоматически заполнять ячейки этого столбца уникальными значениями при добавлении строк в таблицу, таким образом снабжая строки уникальными идентификаторами.

При задании автоинкремента для столбца первое значение, используемое MySQL в этом столбце, по умолчанию равно 1, но вы можете задать другое начальное значение перед добавлением данных в таблицу с помощью оператора SQL наподобие

```
ALTER TABLE имя_таблицы AUTO_INCREMENT = 1234;
```

Этот оператор прикажет MySQL генерировать значения автоинкремента начиная с 1234.

В табл. 4.2 в структуре таблицы указано, что столбец department_id использует автоинкремент.

ЗАМЕЧАНИЕ

В отличие от других СУБД, MySQL позволяет вручную задавать значения для ячеек столбца с автоинкрементом.

За дополнительной информацией о столбцах с автоинкрементом можете обратиться к документации по MySQL (http://www.mysql.org/doc/refman/5.1/en/example-auto-increment.html).

Индексы

Индексы предназначены для оптимизации производительности базы данных, так что мы только кратко их упомянем. Они представляют собой объекты базы

данных, позволяющие ускорить выполнение некоторых операций. Они работают при условии, что подавляющее большинство операций с базой данных — это операции чтения. Индексы ускоряют операции поиска в таблицах, но замедляют операции вставки, удаления и обновления. Впрочем, в большинстве случаев преимущества от использования индексов значительно перевешивают недостатки.

Для таблицы можно создать один или несколько индексов, при этом каждый индекс будет работать с одним или несколькими столбцами. Если таблица индексируется по конкретному столбцу, то для строк создается указатель или они физически переупорядочиваются по значениям в этом столбце и типу индекса. Это позволяет в дальнейшем очень быстро выполнять операции поиска по значениям из данного столбца. Например, если индекс существует для столбца `department_id` и вы ищете отдел со значением в этом столбце 934, то поиск СУБД выполнит очень быстро.

Недостаток индексов состоит в том, что они замедляют добавление в таблицы новых строк или изменение уже существующих, поскольку при каждой такой операции индексы должны обновляться (или строки таблиц должны переупорядочиваться).

Об индексах вам нужно помнить следующее.

- Индексы очень заметно ускоряют операции поиска в базе данных, но замедляют операции изменения данных (т.е. операции удаления, обновления и вставки).
- Слишком большое количество индексов может замедлить работу базы данных в целом. Обычно индексы создаются только для столбцов, часто используемых в разделах `WHERE`, `ORDER BY` и `GROUP BY` операторов, или для столбцов, используемых в соединениях таблиц.
- По умолчанию индексы автоматически создаются для столбцов, используемых в первичных ключах.

Существуют специальные инструменты для тестирования производительности баз данных под высокой нагрузкой с использованием индексов и без их использования; при серьезном подходе к проектированию базы данных эти инструменты нужно использовать, чтобы определить, какой набор индексов будет оптимальным.

ЗАМЕЧАНИЕ

В этом разделе вы познакомились с некоторыми свойствами данных. За более подробной информацией по этой теме можете обратиться к документации по MySQL (`http://www.mysql.org/doc/`) или к книге Тима Конверса, Джойса Парка и Кларка Моргана *PHP 5 и MySQL. Библия пользователя* ("Диалектика", 2006г.).

Создание таблицы `department`

В главе 3 мы создали базу данных `tshirtshop`. В следующем упражнении мы добавим в нее таблицу `department` с помощью веб-интерфейса phpMyAdmin.

ЗАМЕЧАНИЕ

Если хотите, можете воспользоваться готовыми SQL-сценарии с веб-страницы этой книги на сайте Apress, которые автоматически создадут таблицу `department` и заполнят ее данными. Сценарии для создания базы данных находятся в папке `Database` для этой главы в загружаемом архиве файлов для книги. Эти файлы можно также найти на сайтах авторов книги.

Упражнение: создание таблицы `department`

1. Откройте в веб-браузере страницу phpMyAdmin (`http://localhost/phpmyadmin/`), как вы это делали в предыдущей главе при создании базы данных `tshirtshop`.
2. Выберите базу данных `tshirtshop` из раскрывающегося списка Database в левой части страницы. В разделе страницы Create a new table on the database tshirtshop, в поле Name, введите `department`, а в поле Number of fields введите `3`, как на рис. 4.9.

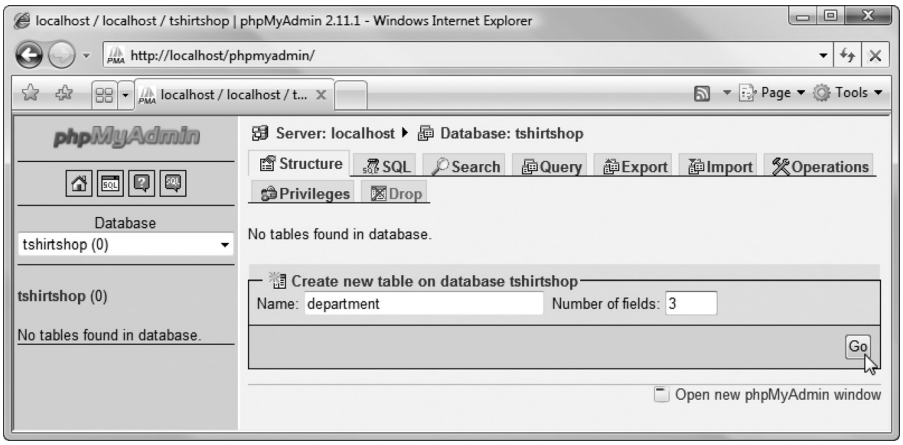


Рис. 4.9. Добавление в базу данных таблицы `department`

3. Щелкните на кнопке Go. Откроется страница, на которой вам нужно будет задать параметры каждого из трех столбцов таблицы, как показано на рис. 4.10.

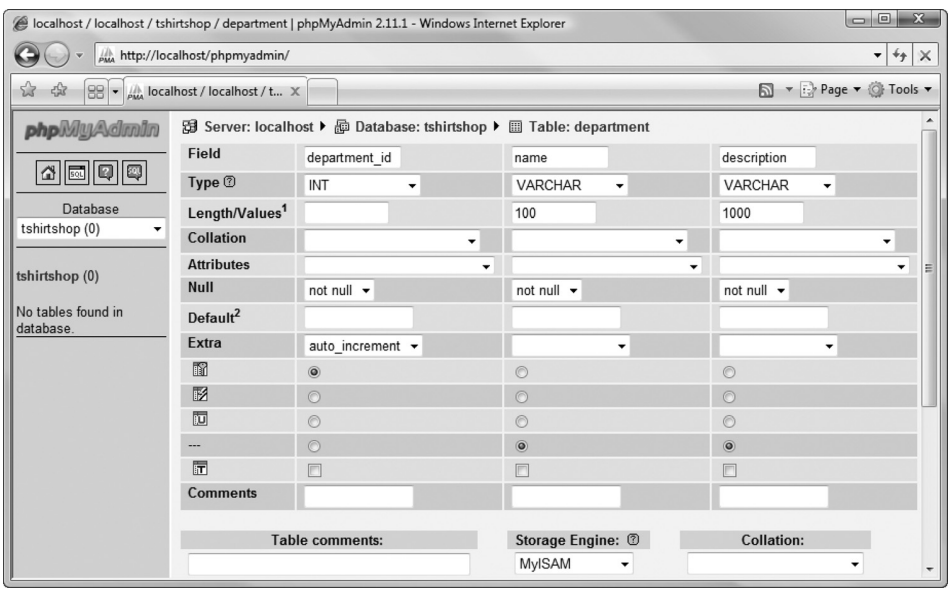


Рис. 4.10. Задание параметров столбцов таблицы

Если вы предпочитаете набрать SQL-код вручную, а не возиться с визуальным интерфейсом phpMyAdmin, вот код, который вам нужен (он есть в загружаемом архиве с исходным кодом для книги).

```
-- Создаем таблицу deparment
CREATE TABLE `department` (
  `department_id` INT             NOT NULL AUTO_INCREMENT,
  `name`          VARCHAR(100)    NOT NULL,
  `description`   VARCHAR(1000),
  PRIMARY KEY (`department_id`)
) ENGINE=MyISAM;
```

4. Щелкните на кнопке Save. Откроется страница с подробной информацией о только что созданной таблице. На этой странице будет приведен код оператора, использованного для создания таблицы, а также ряд других деталей, например подтверждение создания индекса для столбца первичного ключа.
5. Теперь в созданную таблицу можно добавить примеры данных. Перейдите на вкладку SQL, введите приведенный ниже оператор и щелкните на кнопке Go, чтобы MySQL выполнил его. Этот оператор добавит в таблицу `department` три строки.

```
-- Заполняем данными таблицу department
INSERT INTO `department` (`department_id`, `name`, `description`) VALUES
       (1, 'Regional',
'Proud of your country? Wear a T-shirt with a national symbol stamp!'),
       (2, 'Nature',
'Find beautiful T-shirts with animals and flowers in our Nature department!'),
       (3, 'Seasonal', 'Each time of the year has a special flavor. Our
seasonal T-shirts express traditional symbols using unique postal stamp
pictures.');
```

Как это работает: создание таблиц в MySQL

Вы только что создали свою первую таблицу в базе данных! Кроме того, вы поместили в нее несколько строк данных с помощью оператора `INSERT`, предназначенного для вставки строк в таблицы. Чуть позже мы рассмотрим этот оператор подробнее.

Как видите, четко представив себе таблицу, можно легко создать ее в базе данных с помощью phpMyAdmin. Давайте двигаться дальше.

Работа с базой данных

Создав таблицу и заполнив ее данными, давайте попробуем как-то ее использовать. Эта таблица предназначена для формирования списка отделов и заполнения шаблона Smarty этим списком.

Чтобы прочитать данные из базы данных, нужно знать, как с ней работать. Реляционные базы данных понимают разные диалекты языка SQL. Самый простой способ общения с базой данных, использующей MySQL, — составить оператор SQL, передать его серверу MySQL и получить от сервера результаты выполнения этого оператора.

На практике обычно удобнее взаимодействовать с базой данных с помощью набора *хранимых процедур* (stored procedures), но, прежде чем разбираться, что это такое, нужно познакомиться с основами SQL.

Язык SQL

Аббревиатура "SQL" расшифровывается как "Structured Query Language" — язык структурированных запросов. Этот язык используется для работы с подавляющим большинством современных реляционных систем управления базами данных (РСУБД, или RDBMS). Но нам неизвестны РСУБД, поддерживающие SQL именно в том виде, в каком он описан стандартами, — SQL 99 и SQL 2003. Это значит, что во многих случаях операторы SQL, написанные для одной РСУБД, не будут работать на других РСУБД. На момент написания этой книги MySQL полностью соответствовал стандарту SQL 92 и частично соответствовал стандарту SQL 99.

Наиболее часто используемые операторы SQL — `SELECT`, `INSERT`, `UPDATE` и `DELETE`. Они позволяют выполнять основные действия с данными в базах данных.

Как вы убедитесь немного позже, в базовом варианте синтаксис этих операторов очень прост. Но не забывайте, что SQL — очень гибкий и мощный язык, и с его помощью можно составлять запросы, которые намного сложнее используемых в этой книге. Из последующих глав вы узнаете больше, а пока что давайте рассмотрим базовые варианты синтаксиса этих операторов. За дополнительной информацией можете обратиться к соответствующей документации по следующим адресам.

- `http://www.mysql.org/doc/refman/5.1/en/select.html`
- `http://www.mysql.org/doc/refman/5.1/en/insert.html`
- `http://www.mysql.org/doc/refman/5.1/en/update.html`
- `http://www.mysql.org/doc/refman/5.1/en/delete.html`

Оператор SELECT

С помощью оператора `SELECT` можно извлекать из таблиц данные, соответствующие заданным критериям. Базовый синтаксис этого оператора такой.

```
SELECT <список столбцов>
  [FROM <имя таблицы (или таблиц)>]
  [WHERE <критерии>]
```

ЗАМЕЧАНИЕ

Чтобы упростить понимание кода, в этой книге операторы SQL записываются заглавными буквами, хотя сам язык SQL нечувствителен к регистру. Разделы `FROM` и `WHERE` в приведенном выше примере заключены в квадратные скобки, поскольку они не являются обязательными.

Например, следующий оператор возвращает название отдела с `department_id 1`. В нашем случае он вернет название `Regional`, но, если бы в таблице не было отдела с таким значением `department_id`, оператор не вернул бы никаких результатов.

```
SELECT name FROM department WHERE department_id = 1;
```

ПОДСКАЗКА

Если хотите, можете выполнить этот оператор с помощью phpMyAdmin или консольного интерфейса MySQL и убедиться, что он действительно работает.

Если вы хотите получить данные из нескольких столбцов таблицы, просто перечислите их после ключевого слова `SELECT`, разделяя имена столбцов запятыми. Можно также использовать символ `*`, который означает "все столбцы". Однако для

достижения максимальной производительности лучше считывать только значения из нужных столбцов. Использовать символ * не рекомендуется, даже если в каком-то случае вам нужны значения из всех столбцов, — в будущем вы можете добавить в таблицу новые столбцы, и ваш запрос начнет возвращать больше данных, чем вам на самом деле нужно. Кроме того, символ * не позволяет предсказать, в каком порядке будут следовать столбцы в ответе на запрос, и данный порядок может меняться (хотя это и маловероятно). Поэтому мы не будем использовать символ * в запросах.

Для нашей текущей таблицы `department` два следующих запроса вернут одинаковые результаты.

```
SELECT department_id, name, description
FROM   department
WHERE  department_id = 1;

SELECT * FROM department WHERE department_id = 1;
```

ПОДСКАЗКА

Можете безбоязненно разбивать SQL-запросы на несколько строк — MySQL возражать не будет.

Если вы не хотите добавлять в запрос критерии отбора строк, просто уберите из оператора раздел `WHERE`, и запрос вернет все строки таблицы. Например, приведенный ниже запрос вернет все строки и столбцы из таблицы `department`:

```
SELECT * FROM department;
```

ПОДСКАЗКА

Если вы не можете дождаться окончания главы, тестируйте запросы, которые мы рассматриваем, с помощью MySQL на нашей базе данных. Но будьте аккуратны и осторожны — в последующих главах мы будем полагать, что в таблицах этой базы данных находятся строго определенные данные.

Если в операторе `SELECT` не указан способ сортировки результатов, то невозможно предсказать, в каком порядке будут следовать строки в результатах запроса. Поэтому, например, выполнив один и тот же оператор два раза подряд, вы можете получить разные результаты. Для сортировки результатов предназначен раздел `ORDER BY` оператора `SELECT`. Например, список отделов, возвращаемый следующим запросом, будет отсортирован в алфавитном порядке по названиям отделов.

```
SELECT   department_id, name, description
FROM     department
ORDER BY name;
```

Оператор INSERT

Этот оператор предназначен для вставки строк в таблицы базы данных. Его синтаксис выглядит так:

```
INSERT INTO <имя таблицы> [(список столбцов)] VALUES (значения для вставки)
```

ПОДСКАЗКА

Хотя наличие списка столбцов и не является обязательным (если его нет в операторе, значения присваиваются столбцам в том порядке, в каком эти столбцы расположены в таблице), очень рекомендуется все же его задавать. Это гарантирует, что при изменении таблицы ранее созданные операторы `INSERT` будут продолжать корректно выполняться.

Следующий оператор INSERT добавляет в таблицу department сведения об отделе Zodiac T-Shirts Department:

```
INSERT INTO department (name) VALUES ('Zodiac T-Shirts Department');
```

В этом операторе не задано значение для поля description, поскольку при создании таблицы department мы указали, что для данного поля допустимы значения NULL. Это значит, что мы не обязаны задавать значения для этого столбца, добавляя в таблицу новые строки. Кроме того, в операторе нет значения для столбца department_id, поскольку для него задана опция AUTO_INCREMENT, и MySQL автоматически сгенерирует для него значения. Однако ничто не мешает вам задать для этого столбца значение в операторе INSERT.

ПОДСКАЗКА

Поскольку department_id — столбец первичного ключа, попытка добавить в таблицу строки с одинаковыми значениями в этом столбце приведет к ошибке: база данных не позволит присутствовать в столбце первичного ключа повторяющимся значениям.

Позволяя MySQL генерировать значения для столбцов с опцией AUTO_INCREMENT, вы можете получать последнее сгенерированное значение с помощью функции LAST_INSERT_ID(). Вот пример использования этой функции:

```
INSERT INTO department (name) VALUES ('Some New Department');
SELECT LAST_INSERT_ID();
```

ПОДСКАЗКА

В MySQL символ “точка с запятой” (;) используется для разделения операторов SQL.

Оператор UPDATE

Оператор UPDATE предназначен для обновления уже присутствующих в таблицах строк и имеет следующий синтаксис.

```
UPDATE <имя таблицы>
SET <имя столбца> = <новое значение> [, <имя столбца> = <новое значение>
... ]
[WHERE <ограничивающий критерий>]
```

Приведенный ниже оператор изменяет название отдела с ID 43 на Cool Department. Если бы таких отделов было несколько, были бы изменены все строки с ID 43 в таблице department, но, как вы помните, столбец department_id является первичным ключом, и в таблице может быть только одна строка с ID 43.

```
UPDATE department SET name='Cool Department' WHERE department_id = 43;
```

При использовании оператора UPDATE нужно быть внимательным — с его помощью можно очень легко перепутать данные в таблице. Если в операторе UPDATE не будет раздела WHERE, изменению подвергнутся все строки таблицы, а вам вряд ли это нужно. MySQL с радостью внесет изменения в каждую строку, даже если это будет значить, что у всех отделов будут одинаковые названия и описания. С точки зрения MySQL это все равно будут разные отделы, потому что у них будут разные значения в столбце department_id.

Оператор DELETE

Синтаксис оператора DELETE, удаляющего строки из таблицы, очень прост:

```
DELETE FROM <имя таблицы>
[WHERE <ограничивающее условие>]
```

Обычно условие в разделе WHERE формируется так, чтобы оператор удалял только одну строку из таблицы, например

```
DELETE FROM department WHERE department_id = 43;
```

Как и UPDATE, оператор DELETE требует осторожности при использовании: если в нем не задать раздел WHERE, он просто удалит все строки из таблицы. Сама таблица при этом не будет удалена — для удаления таблиц используется специальный оператор DROP TABLE (http://dev.mysql.com/doc/refman/5.0/en/drop-table.html). Например, следующий оператор удалит все строки из таблицы department:

```
DELETE FROM department;
```

Хранимые процедуры в MySQL

Хранимая процедура — это просто набор операторов SQL, сохраненный под определенным именем, по которому этот набор можно вызывать. Хранимые процедуры в MySQL, аналогично функциям в PHP, могут принимать аргументы и возвращать значения. Хранимые процедуры в MySQL соответствуют стандарту SQL 2003. Официальная документация по ним доступна по адресу http://www.mysql.org/doc/refman/5.1/en/stored-procedures.html.

Вам не обязательно использовать хранимые процедуры для работы с базой данных — ничто не мешает вам передавать MySQL операторы SQL из внешних приложений (например, PHP-сценариев из приложения TShirtShop). Однако пользоваться хранимыми процедурами проще — достаточно указать имя нужной процедуры и задать значения для ее параметров. Использование хранимых процедур дает следующие преимущества.

- Повышение производительности. При первом вызове хранимой процедуры MySQL генерирует план ее выполнения, а затем может его использовать для быстрого выполнения этой процедуры при ее повторных вызовах.
- Упрощение кода для доступа к данным, который хранится централизованно, и упрощение реализации трехуровневой архитектуры (уровень данных как раз и образуется хранимыми процедурами, хранящимися в базе данных).
- Упрощение обеспечения безопасности. MySQL позволяет задавать настройки безопасности отдельно для каждой процедуры.
- SQL-запросы, генерируемые “на лету” в PHP-коде, более уязвимы к инъекционным атакам. Это серьезный источник угроз (он рассматривается во многих статьях; если вам интересно, задайте в Google статью поиска SQL injection attack).
- Последнее преимущество, возможно, субъективно, однако отделение SQL от PHP делает PHP-код более аккуратным и простым для понимания: вызывать хранимые процедуры проще, чем генерировать SQL-операторы “на лету” в PHP-сценариях.

При разработке приложения TShirtShop мы будем обращаться к данным исключительно с помощью хранимых процедур, хранящихся в базе данных tshirtshop. Синтаксис для создания хранимых процедур имеет вид

```
DELIMITER $$
CREATE PROCEDURE <name>(<param1 type>, <param2 type> ... )
BEGIN
<code>
END$$

DELIMITER;
```

Учтите, что в качестве ограничителя (`DELIMITER`) не обязательно использовать символы `$$`. Главное, чтобы этот ограничитель отличался от используемой по умолчанию точки с запятой.

В базе данных нельзя создать две хранимые процедуры с одинаковыми именами. Чтобы удалить существующую хранимую процедуру, используется оператор `DROP PROCEDURE`. Изменить текст процедуры нельзя — можно только удалить ее и создать заново. Хотя в MySQL и поддерживается оператор `ALTER PROCEDURE`, с его помощью нельзя изменить тело или набор параметров существующей процедуры.

Первая хранимая процедура, которую мы создадим, будет называться `catalog_get_departments_list`. Она будет возвращать список ID и названий отделов и вызываться методом из уровня логики приложения TShirtShop, которому понадобятся эти данные. Созданию этой процедуры посвящено следующее упражнение.

Упражнение: создание хранимой процедуры MySQL

1. Откройте в веб-браузере страницу phpMyAdmin (`http://localhost/phpmyadmin/`). Выберите базу данных `tshirtshop` из раскрывающегося списка Database в левой части страницы.
2. Перейдите на вкладку SQL и задайте ограничитель `$$`, как показано на рис. 4.11.

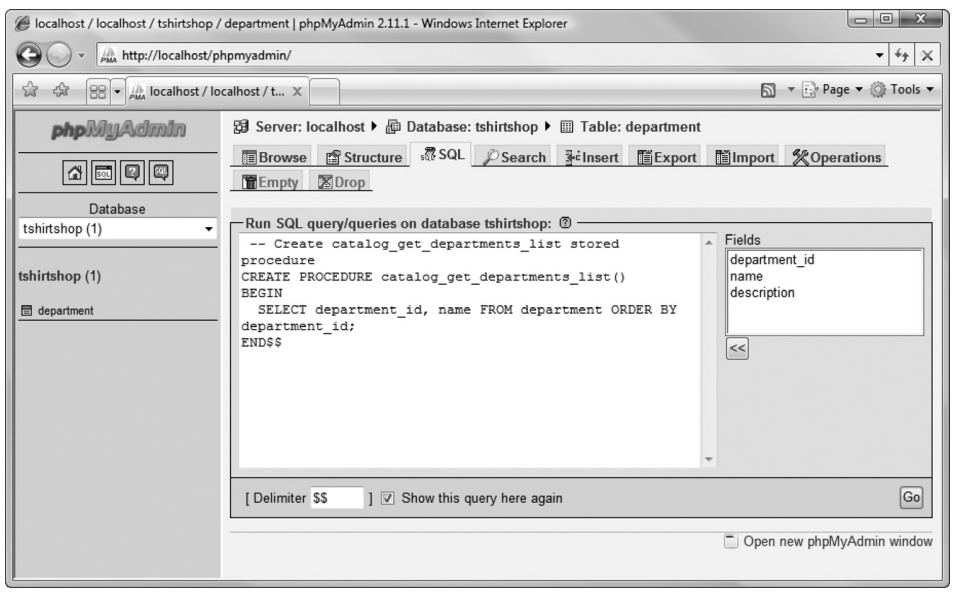


Рис. 4.11. Задание символов-ограничителей

3. Выполните следующий код, который и создаст хранимую процедуру `catalog_get_departments_list`.

```
-- Создание хранимой процедуры catalog_get_departments_list
CREATE PROCEDURE catalog_get_departments_list()
BEGIN
  SELECT department_id, name FROM department ORDER BY department_id;
END$$
```

Как это работает: хранимые процедуры MySQL

Давайте подробно рассмотрим хранимую процедуру `catalog_get_departments_list`. В первой строке мы задаем ее имя:

```
PROCEDURE catalog_get_departments_list()
```

Тело процедуры помещается между строками `BEGIN` и `END$$`. В фрагменте приведенного ниже кода полужирным шрифтом выделен фрагмент, который делает все, что нас интересует, — все остальное представляет собой просто вспомогательные элементы, обозначающие тело хранимой процедуры:

```
BEGIN
SELECT department_id, name FROM department ORDER BY department_id;
END$$
```

Возможно, на первый взгляд этот код покажется непривычным, но на самом деле все, что делает данная процедура, — это выполняет оператор `SELECT` и возвращает результаты выполнения.

Добавление управляющей логики в приложение

Уровень логики приложения (иногда именуемый просто средним уровнем) обычно считается мозговым центром — именно в нем размещается управляющая логика. Однако для простых задач наподобие получения списка отделов от уровня данных на уровне логики приложения не надо реализовывать никакой особой логики. Уровень логики приложения просто запрашивает данные у уровня данных и передает их уровню представления. Обычно эти данные принимает специальный объект на уровне представления, но приемником может быть и метод на уровне логики приложения, которому эти данные нужны для принятия каких-либо решений.

В этой главе мы закладываем основы уровня логики приложения, включая функциональность, открывающую и закрывающую соединения с базой данных, использующую хранимые процедуры MySQL и обеспечивающую доступ к этим процедурам из PHP.

Для отображения списка отделов нам понадобится реализовать на уровне логики приложения два класса.

- Класс `DatabaseHandler` будет хранить общую функциональность, отвечающую за доступ к базе данных. Этот класс мы будем в дальнейшем постоянно использовать, обращаясь к базе данных, чтобы избежать ошибок и нестыковок, связанных с реализацией одновременно нескольких механизмов доступа к данным.
- Класс `Catalog` будет содержать функциональность, специфичную для каталога товаров, например метод `GetDepartments()`, который будет извлекать из базы данных список отделов.

Установка соединения с MySQL

SQL-запросы, которые мы составим, необходимо будет как-то передавать MySQL для выполнения. Как уже говорилось в главе 2, для доступа к MySQL мы будем использовать PHP PDO.

Прежде чем писать код уровня логики приложения, нужно проанализировать возможные варианты реализации этого уровня. Вот важные вопросы, на которые нужно ответить до написания кода.

- Какую стратегию мы будем использовать для открытия и закрытия соединений с MySQL при выполнении запросов?
- Какие методы PHP PDO мы будем применять для выполнения хранимых процедур и получения результатов?
- Как мы будем обрабатывать возможные ошибки и интегрировать механизмы их обработки с кодом обработки ошибок, который написали в главе 3?

Рассмотрим каждый из этих вопросов по очереди, прежде чем писать код.

Открытие и закрытие соединений с сервером MySQL

Есть два основных подхода к работе с соединениями. Первый из них иллюстрирует приведенная ниже последовательность действий, которую нужно выполнять при каждом обращении к базе данных.

1. *Открываем соединение* с базой данных, когда нужно выполнить запрос к ней.
2. *Выполняем запрос SQL* (или хранимую процедуру) с помощью открытого соединения и получаем результаты выполнения. На этом этапе также нужно обработать ошибки (если таковые возникли).
3. *Закрываем соединение* с базой данных сразу же после выполнения запроса.

Преимущество такого подхода — кратковременность существования открытых соединений (это хорошо, поскольку открытые соединения потребляют ресурсы сервера). Кроме того, на некоторых маломощных серверах устанавливается ограничение на количество одновременно открытых соединений. Недостаток — разумеется, накладные расходы на частое открытие и закрытие соединений. Впрочем, его можно несколько нивелировать, используя постоянные соединения.

ЗАМЕЧАНИЕ

"Постоянные соединения" (persistent connections) — это название технологии, пытающейся повысить эффективность открытия и закрытия соединений с базой данных. Более подробно она описана, например, по адресу `http://www.php.net/manual/en/features.persistent-connections.php`.

Альтернативный подход, который мы будем использовать в TShirtShop, выглядит следующим образом.

1. *Открываем соединение* с базой данных, когда нам впервые требуется обратиться к ней для выполнения запроса.
2. *Выполняем все необходимые запросы* (или хранимые процедуры) через это соединение, не закрывая его. На этом же этапе осуществляется обработка возникших ошибок.
3. *Закрываем соединение* с базой данных, когда клиент завершает работу.

При использовании этого подхода все операции с базой данных для выполнения одного клиентского запроса (поступающего, когда пользователь открывает новую веб-страницу на нашем сайте) осуществляются через одно соединение с базой данных — мы не открываем и не закрываем соединение для каждой операции. Этот подход также допускает использование постоянных соединений для повышения эффективности.

Использование PHP PDO для работы с базой данных

А теперь мы поговорим о практической реализации выбранного подхода — открытии и закрытии соединений и выполнении запросов с помощью PHP PDO.

Как упоминалось в главе 2, мы не будем обращаться к MySQL через функции расширения PHP, а воспользуемся уровнем абстрагирования базы данных (PHP PDO). Классы PDO позволяют обращаться к различным источникам данных с помощью одного и того же API (Applications Programming Interface — интерфейса программирования приложений). Это значит, что вам не придется переделывать код, отвечающий за доступ к данным, если вы решите вместо MySQL использовать другую СУБД (хотя в таком случае вам, вероятно, придется модифицировать сами операторы SQL). Использование PHP PDO — удобный и современный способ взаимодействия с базами данных, и он заметно упрощает нашу жизнь как программистов.

Важный класс PHP PDO, с которым мы будем работать, так и называется `PDO`. В нем содержатся методы для выполнения различных операций с базами данных. Мы можем воспользоваться этими методами для обработки данных, установки соединений с базами данных и решения других типичных задач; нам не придется самим придумывать и писать код для их решения, поскольку он уже есть в классе `PDO`. Стоит потратить время на ознакомление с методами этого класса — вам необязательно точно знать, как они работают, но знать об их существовании вы должны, чтобы не тратить время на изобретение велосипеда.

ЗАМЕЧАНИЕ

В этой книге вы познакомитесь только с той частью функциональности PHP PDO, которая используется в проекте TShirtShop. Более полный обзор этой функциональности можно найти, например, в руководстве по PHP по адресу `http://www.php.net/manual/en/ref.pdo.php`.

Функциональность класса PDO позволяет устанавливать соединения с серверами MySQL и выполнять SQL-запросы. За открытие соединения отвечает конструктор класса PDO, который в качестве параметров получает строку соединения с сервером базы данных и (необязательно) аргумент, указывающий, будет ли создаваемое соединение постоянным. В строке соединения содержится вся информация, необходимая для установки соединения с сервером MySQL. Объект PDO создается конструктором примерно следующим образом.

```
$dbh = new PDO('mysql:dbname=' . $db_name . ';host=' . $db_host,
               $db_user,
               $db_pass,
               array(PDO::ATTR_PERSISTENT => $persistent));
```

ЗАМЕЧАНИЕ

Конструктор класса PDO возвращает проинициализированный объект соединения с базой данных (класс этого объекта определяется типом базы данных, например `mysql`), если соединение успешно установлено; в противном случае генерируется исключение.

Как видно из приведенного выше фрагмента кода, для установки соединения конструктору нужно передать пять переменных.

- `$db_user` с именем пользователя
- `$db_pass` с паролем
- `$db_host` с именем хоста сервера MySQL
- `$db_name` с именем базы данных, к которой вы хотите обращаться
- `$persistent` (`true`, если мы хотим создать постоянное соединение, или `false` — в противном случае)

Чтобы закрыть соединение с базой данных, нужно присвоить переменной `$dbh` значение `null` (просто с помощью оператора `$dbh = null`).

Приведенный ниже фрагмент кода демонстрирует, как можно создать, открыть и закрыть соединение с базой данных MySQL, а также перехватить любые генерируемые при этом исключения.

```
try
{
  // Открываем соединение
  $dbh = new PDO('mysql:dbname=' . $db_name . ';host=' . $db_host,
  $db_user, $db_pass);
  // Закрываем соединение
  $dbh = null;
}
catch (PDOException $e)
{
  echo 'Connection failed: ' . $e->getMessage();
}
```

Ключевые слова `try` и `catch` используются в *обработке исключений* (exception handling).

Обработка исключений в PHP 5

В главе 3 мы написали код, который перехватывает и обрабатывает ошибки в приложении TShirtShop (и в конечном итоге сообщает о них). *Ошибки PHP* (PHP Errors) — это стандартный механизм, позволяющий реагировать на возникновение ошибок в работающем PHP-приложении. Когда ошибка возникает, выполнение приложения приостанавливается; однако мы можем определить функцию для обработки ошибок, которая будет вызвана, прежде чем выполнение приостановится. Мы создали такую функцию в главе 3 — она получала максимум информации о возникшей ошибке и сохраняла эту информацию. Позже программист мог просмотреть сохраненную информацию и исправить приложение, чтобы эта ошибка в дальнейшем не возникала.

В PHP 5 вместе с другими объектно-ориентированными элементами появился новый механизм обработки ошибок — обработка исключений. Исключения — это современный способ реагирования на ошибки при выполнении приложений, гораздо более гибкий и эффективный, чем ошибки PHP. Исключения являются очень важным элементом объектно-ориентированных языков, и в PHP реализована модель исключений, очень похожая на применяемую в таких языках, например, как Java и C#. Однако исключения PHP мирно сосуществуют со стандартными ошибками PHP, и полагаться только на исключения не стоит. Одни расширения PHP, например PDO, можно настроить на генерацию исключений при ошибках, а другие лишены такой возможности.

Основное преимущество исключений перед ошибками — в большей гибкости их обработки. Когда генерируется исключение, вы можете обработать его локально и позволить вашему сценарию продолжать выполняться или можете передать исключение другому классу для обработки. Генерация исключения не приводит к завершению выполнения сценария, как генерация ошибки. При использовании исключений нужно поместить код, в котором вы предполагаете возможное их возникновение, в блок `try`, и перехватывать возникающие исключения блоком `catch`, например, так.

```
try
{
  // Код, в котором могут генерироваться
```

```
  // исключения, которые мы хотим перехватить
}
catch (Exception $e)
{
  // Код, который выполняется при генерации исключения
  // (подробности исключения доступны в объекте $e)
}
```

Когда в коде, помещенном в блок `try`, генерируется исключение, сразу же начинает выполняться код из блока `catch`. Если код в блоке `catch` не генерирует исключение повторно, считается, что это исключение успешно обработано и выполнение сценария продолжается нормально. Это позволяет решать многие проблемы, которые могут приводить к неработоспособности приложений. Поверьте, вы очень скоро станете ценить возможности, которые вам дает обработка исключений.

В PHP 5 исключения представлены классом `Exception`, в объектах которого хранятся детали исключений. Вы можете генерировать исключения явно, с помощью ключевого слова `throw`. Объект `Exception` после его создания передается по стеку вызовов, пока не будет перехвачен блоком `catch`. *Стек вызовов* (call stack) — это список выполняемых в данный момент методов в том порядке, в каком они вызывали друг друга. Например, если метод `C()` был вызван методом `B()`, в свою очередь, вызванным методом `A()`, то эти три метода и образуют стек вызовов. В таком сценарии возникшее исключение может быть обработано в том же методе, в котором оно возникло, при условии, что код, сгенерировавший исключение, находится в структуре `try-catch`. Если это не так, исключение из метода `C()` распространится вверх по стеку вызовов к методу `B()`, который также получит шанс обработать это исключение. Если исключение не обработает ни один метод в стеке вызовов, в конце концов это сделает интерпретатор PHP, который преобразует исключение в фатальную ошибку — PHP Fatal Error.

В нашем коде, работающем с базой данных, мы перехватываем потенциальные исключения, которые может генерировать PDO. Хотя по умолчанию PDO не генерирует исключений, его можно настроить на их генерацию, если что-то идет не так при выполнении операторов SQL или открытии соединения с базой данных. Например, это можно сделать так.

```
// Создаем новый экземпляр класса PDO
$handler = new PDO( ... );

// Настраиваем PDO на генерацию исключений
self::$_mHandler->setAttribute(PDO::ATTR_ERRMODE,
                               PDO::ERRMODE_EXCEPTION);
```

Затем мы перехватываем и обрабатываем исключения, которые могут генерироваться в коде доступа к данным, и передаем информацию об ошибках коду, который мы написали в главе 3. В следующем фрагменте кода показан короткий метод, в котором реализован такой механизм.

```
// Метод-обертка для PDOStatement::fetch()
public static function GetRow($sqlQuery, $params = null,
                               $fetchStyle = PDO::FETCH_ASSOC)
{
  // Инициализируем возвращаемое значение
  $result = null;

  // Пытаемся выполнить SQL-запрос или хранимую процедуру
  try
  {
    // Получаем дескриптор базы данных
    $database_handler = self::GetHandler();

    // Готовим запрос к выполнению
    $statement_handler = $database_handler->prepare($sqlQuery);

    // Выполняем запрос
    $statement_handler->execute($params);
```

```
    // Получаем результат
    $result = $statement_handler->fetch($fetchStyle);
  }

  // Генерируем ошибку, если при выполнении SQL-запроса возникло исключение
  catch(PDOException $e)
  {
    // Закрываем дескриптор базы данных и генерируем ошибку
    self::Close();
    trigger_error($e->getMessage(), E_USER_ERROR);
  }
  // Возвращаем результаты выполнения запроса
  return $result;
}
```

Выполнение запросов с помощью открытого соединения

Открыв соединение, мы достигаем момента, к которому стремились с самого начала: мы можем выполнять через это соединение операторы SQL.

Выполнять данные операторы можно несколькими способами. Выбор конкретного способа определяется обстоятельствами, например тем, должен ли запрос возвращать данные, и если должен, то в каком формате. Вот методы PDO, которые мы будем использовать для выполнения SQL-запросов.

- `PDOStatement::execute()`. Используется для выполнения операторов `INSERT`, `UPDATE` и `DELETE`, не возвращающих данные.
- `PDOStatement::fetch()`. Используется для получения из базы данных одной строки данных.
- `PDOStatement::fetchAll()`. Используется для получения множества строк данных из базы данных.
- `PDOStatement::prepare()`. Подготавливает SQL-оператор к выполнению, создавая так называемый подготовленный оператор.

Подготовленный оператор (prepared statement) — это параметризованный SQL-запрос, значения параметров которого заменены или метками параметров (`?`), или именованными переменными (`:имя_переменной`), как в следующих примерах:

```
$query1 = "SELECT name FROM department WHERE department_id = ?"
$query2 = "SELECT name FROM department WHERE department_id = :dept_id"
```

Чтобы выполнить подготовленный оператор, нужно передать значения параметров функциям, которые выполняют запрос, и они займутся генерацией полного SQL-оператора. Чтобы получить список отделов из базы данных, параметры нам не понадобятся, но из главы 5 вы узнаете, как с ними обращаться.

В этой книге мы всегда будем использовать подготовленные операторы, поскольку они дают два преимущества.

- Параметры подготовленных операторов проверяются, что предотвращает атаки инъекцией.
- Подготовленные операторы выполняются быстрее, поскольку сервер базы данных после первого их выполнения может заново использовать подготовленный план выполнения.

Чтобы создать централизованный механизм обработки ошибок в коде приложения, мы не будем использовать методы PDO непосредственно в коде уровня

логики приложения. Вместо этого мы "завернем" функциональность PDO в класс DatabaseHandler и будем использовать его в других классах уровня логики приложения.

Написание кода уровня логики приложения

Ну ладно, хватит теории. Давайте, наконец, напишем реальный код. Первое, что мы создадим, — это класс DatabaseHandler, вспомогательный класс, в котором будет сосредоточена функциональность, используемая другими механизмами из уровня логики приложения. Затем мы создадим еще один класс уровня логики приложения, Catalog, использующий функциональность класса DatabaseHandler для реализации функциональности, используемой на уровне представления. В классе Catalog будет, например, содержаться метод GetDepartments() (он будет использоваться для генерации списка отделов), а также GetCategories() и некоторые другие. В этой главе из класса Catalog нам понадобится только метод GetDepartments().

Хотя нам и не понадобится вся созданная функциональность, к концу главы класс DatabaseHandler будет полностью готов. В нем будут содержаться следующие методы.

- Execute() — метод, выполняющий хранимую процедуру, не возвращающую данных из базы данных, например с операторами INSERT, UPDATE и DELETE.
- GetAll() — метод, используемый для выполнения хранимых процедур, возвращающих множество строк данных, например список отделов.
- GetRow() — метод, используемый для выполнения запросов, возвращающих одну строку данных.
- GetOne() — метод, используемый для выполнения запросов, возвращающих единственное значение. Этот метод, например, можно использовать, чтобы получать из базы данных количество единиц товара определенного вида.

Упражнение: создание и использование класса DatabaseHandler

1. Добавьте имя пользователя и пароль для доступа к базе данных в конец файла tshirtshop/config.php, заменив значения из приведенного ниже фрагмента листинга значениями для своего компьютера. Этот фрагмент подразумевает, что вы создали учетную запись пользователя так, как описано в главе 3.

```
// Параметры соединения с базой данных
define('DB_PERSISTENCY', 'true');
define('DB_SERVER', 'localhost');
define('DB_USERNAME', 'tshirtshopadmin');
define('DB_PASSWORD', 'tshirtshopadmin');
define('DB_DATABASE', 'tshirtshop');
define('PDO_DSN', 'mysql:host=' . DB_SERVER . ';dbname=' . DB_DATABASE);
```

2. В папке tshirtshop/business создайте новый файл с именем database_handler.php и создайте в нем класс DatabaseHandler, листинг которого приведен ниже. На данный момент в классе будут только конструктор (объявленный как private, что делает невозможным создание экземпляров класса) и статический метод GetHandler(). Этот метод создает новое соединение с базой данных, сохраняет его в переменную-член класса $m_Handler и возвращает объект класса (более подробно работа этого класса рассматривается ниже, в разделе "Как это работает: код уровня логики приложения").

```
<?php
// Класс, предоставляющий базовую функциональность доступа к данным
class DatabaseHandler
{
  // Переменная для хранения экземпляра класса PDO
  private static $_mHandler;

  // private-конструктор, не позволяющий напрямую создавать объекты класса
  private function __construct()
  {
  }

  // Возвращает проинициализированный дескриптор базы данных
  private static function GetHandler()
  {
    // Создаем соединение с базой данных, только если его еще нет
    if (!isset(self::$_mHandler))
    {
      // Выполняем код, перехватывая потенциальные исключения
      try
      {
        // Создаем новый экземпляр класса PDO
        self::$_mHandler =
          new PDO(PDO_DSN, DB_USERNAME, DB_PASSWORD,
                  array(PDO::ATTR_PERSISTENT => DB_PERSISTENCY));

        // Настраиваем PDO на генерацию исключений
        self::$_mHandler->setAttribute(PDO::ATTR_ERRMODE,
                                       PDO::ERRMODE_EXCEPTION);
      }
      catch (PDOException $e)
      {
        // Закрываем дескриптор и генерируем ошибку
        self::Close();
        trigger_error($e->getMessage(), E_USER_ERROR);
      }
    }

    // Возвращаем дескриптор базы данных
    return self::$_mHandler;
  }
}
?>
```

3. Добавьте в класс `DatabaseHandler` метод `Close()`, который мы будем вызывать, когда понадобится закрыть соединение с базой данных. Вот листинг этого метода.

```
// Очищаем экземпляр класса PDO
public static function Close()
{
  self::$_mHandler = null;
}
```

4. Добавьте в класс `DatabaseHandler` метод `Execute()`. Он использует метод `Execute()` класса `PDOStatement` для выполнения запросов, не возвращающих данные (запросов `INSERT`, `DELETE` и `UPDATE`). Вот его листинг.

```
// Метод-обертка для PDOStatement::execute()
public static function Execute($sqlQuery, $params = null)
{
  // Пытаемся выполнить SQL-запрос или хранимую процедуру
  try
```

```
  {
    // Получаем дескриптор базы данных
    $database_handler = self::GetHandler();

    // Подготавливаем запрос к выполнению
    $statement_handler = $database_handler->prepare($sqlQuery);

    // Выполняем запрос
    $statement_handler->execute($params);
  }
  // Генерируем ошибку, если при выполнении SQL-запроса возникло исключение
  catch(PDOException $e)
  {
    // Закрываем дескриптор базы данных и генерируем ошибку
    self::Close();
    trigger_error($e->getMessage(), E_USER_ERROR);
  }
}
```

5. Следующий метод, `GetAll()`, являющийся оберткой для `PDOStatement::fetchAll()`. Этот метод позволяет получить все строки, возвращаемые запросом `SELECT`.

```
// Метод-обертка для PDOStatement::fetchAll()
public static function GetAll($sqlQuery, $params = null,
                               $fetchStyle = PDO::FETCH_ASSOC)
{
  // Инициализируем возвращаемое значение в null
  $result = null;
  // Пытаемся выполнить SQL-запрос или хранимую процедуру
  try
  {
    // Получаем дескриптор базы данных
    $database_handler = self::GetHandler();

    // Подготавливаем запрос к выполнению
    $statement_handler = $database_handler->prepare($sqlQuery);

    // Выполняем запрос
    $statement_handler->execute($params);

    // Получаем результат
    $result = $statement_handler->fetchAll($fetchStyle);
  }
  // Генерируем ошибку, если при выполнении SQL-запроса возникло исключение
  catch(PDOException $e)
  {
    // Закрываем дескриптор базы данных и генерируем ошибку
    self::Close();
    trigger_error($e->getMessage(), E_USER_ERROR);
  }
  // Возвращаем результаты запроса
  return $result;
}
```

6. Очередной метод, `GetRow()`, представляет собой обертку для `PDOStatement::fetch()`. Он используется для выполнения запросов `SELECT`, возвращающих одну строку. Вот его код.

```
// Метод-обертка для PDOStatement::fetch()
public static function GetRow($sqlQuery, $params = null,
                               $fetchStyle = PDO::FETCH_ASSOC)
```

```
{
  // Инициализируем возвращаемое значение
  $result = null;
  // Пытаемся выполнить SQL-запрос или хранимую процедуру
  try
  {
    // Получаем дескриптор базы данных
    $database_handler = self::GetHandler();

    // Готовим запрос к выполнению
    $statement_handler = $database_handler->prepare($sqlQuery);

    // Выполняем запрос
    $statement_handler->execute($params);
    // Получаем результат

    $result = $statement_handler->fetch($fetchStyle);
  }
  // Генерируем ошибку, если при выполнении SQL-запроса возникло исключение
  catch(PDOException $e)
  {
    // Закрываем дескриптор базы данных и генерируем ошибку
    self::Close();
    trigger_error($e->getMessage(), E_USER_ERROR);
  }
  // Возвращаем результаты выполнения запроса
  return $result;
}
```

7. Последний нужный нам метод, `GetOne()`, похож на `GetRow()`, но возвращающий только одно значение из запроса `SELECT`. Вот его листинг.

```
// Возвращает значение первого столбца из строки
public static function GetOne($sqlQuery, $params = null)
{
  // Инициализируем возвращаемое значение
  $result = null;
  // Пытаемся выполнить SQL-запрос или хранимую процедуру
  try
  {
    // Получаем дескриптор базы данных
    $database_handler = self::GetHandler();

    // Готовим запрос к выполнению
    $statement_handler = $database_handler->prepare($sqlQuery);

    // Выполняем запрос
    $statement_handler->execute($params);

    // Получаем результат
    $result = $statement_handler->fetch(PDO::FETCH_NUM);

    /* Сохраняем первое значение из множества (первый столбец первой строки)
    в переменной $result */
    $result = $result[0];
  }
  // Генерируем ошибку, если при выполнении SQL-запроса возникло исключение
  catch(PDOException $e)
  {
    // Закрываем дескриптор базы данных и генерируем ошибку
    self::Close();
    trigger_error($e->getMessage(), E_USER_ERROR);
```

```
  }
  // Возвращаем результаты выполнения запроса
  return $result;
}
```

8. В папке business создайте файл catalog.php. Добавьте в него следующий код.

```
<?php
// Класс уровня логики приложения для считывания информации
// о каталоге товаров
class Catalog
{
  // Получаем список отделов
  public static function GetDepartments()
  {
    // Составляем SQL-запрос
    $sql = 'CALL catalog_get_departments_list()';

    // Выполняем запрос и получаем результаты
    return DatabaseHandler::GetAll($sql);
  }
}
?>
```

9. Чтобы сделать класс DatabaseHandler доступным приложению, файл database_handler.php нужно включить в index.php. Чтобы сделать это, добавьте в код файла index.php выделенные полужирным шрифтом строки из следующего фрагмента.

```
<?php
// Включаем вспомогательные файлы
require_once 'include/config.php';
require_once BUSINESS_DIR . 'error_handler.php';

// Задаем обработчик ошибок
ErrorHandler::SetHandler();

// Загружаем шаблон страницы приложения
require_once PRESENTATION_DIR . 'application.php';

// Загружаем дескриптор базы данных
require_once BUSINESS_DIR . 'database_handler.php';
```

10. В конец файла index.php добавьте строки, выделенные полужирным в следующем фрагменте кода. Эти строки обеспечивают закрытие соединения с базой данных.

```
// Загружаем файл шаблона Smarty
$application = new Application();

// Отображаем страницу
$application->display('store_front.tpl');

// Закрываем соединение с базой данных
DatabaseHandler::Close();
?>
```

Как это работает: код уровня логики приложения

В предыдущем упражнении мы добавили в файл include/config.php информацию, необходимую программе для установки соединения с базой данных, а затем создали класс DatabaseHandler. В этом классе содержится ряд методов-оберток для методов PDO, обеспечивающих выполнение запросов к базе данных.

Класс DatabaseHandler предоставляет функциональность, используемую другими классами уровня логики приложения.

В классе DatabaseHandler конструктор объявлен как private-метод, поэтому объекты этого класса невозможно создавать напрямую. Однако вы можете свободно вызывать *статические методы* (static methods) класса. Статические элементы класса — поля и методы — не принадлежат конкретным экземплярам класса, а существуют в единственном экземпляре во всей программе. Например, для выполнения SQL-запроса к базе данных с помощью метода GetAll() не нужно создавать экземпляр класса, как в приведенном ниже фрагменте кода. Впрочем, мы при всем желании не смогли бы этого сделать не только потому, что private-конструктор не позволит нам создать объект, но и потому, что нет версии метода GetAll(), привязанной к определенному объекту класса.

```
$myHandler = new DatabaseHandler();
$results = $myHandler->GetAll($sql);
```

Вместо этого статические методы вызываются напрямую с использованием только имени класса (отделенного символами ::), а не имени объекта класса (отделенного символами ->):

```
DatabaseHandler::GetAll($sql);
```

Статические элементы класса PHP хранит, используя внутренний глобальный экземпляр класса. При использовании PDO преимущество хранения соединения с базой данных в статическом элементе (private static $_mHandler) в том, что все операции с базой данных во время одного сетевого запроса будут выполняться именно через это единственное соединение. Как уже говорилось выше, такой подход позволяет увеличить производительность, поскольку избавляет нас от необходимости заново создавать соединение для выполнения каждого оператора SQL.

ЗАМЕЧАНИЕ

Статические поля и методы — это концепции из объектно-ориентированного программирования, которые не поддерживались в PHP до появления версии 4. Очень хорошее введение в объектно-ориентированные возможности PHP 5 можно найти по адресу http://php.net/manual/en/language.oop5.php.

Все методы, используемые для выполнения запросов к базе данных, имеют сходную структуру, в которой используется возможность настроить PDO на генерацию исключений. В качестве примера давайте рассмотрим метод GetRow().

```
// Метод-обертка для PDOStatement::fetch()
public static function GetRow($sqlQuery, $params = null,
                               $fetchStyle = PDO::FETCH_ASSOC)
{
  // Инициализируем возвращаемое значение
  $result = null;

  // Пытаемся выполнить SQL-запрос или хранимую процедуру
  try
  {
    // Получаем дескриптор базы данных
    $database_handler = self::GetHandler();

    // Готовим запрос к выполнению
    $statement_handler = $database_handler->prepare($sqlQuery);

    // Выполняем запрос
    $statement_handler->execute($params);

    // Получаем результат
    $result = $statement_handler->fetch($fetchStyle);
  }
  // Генерируем ошибку, если при выполнении SQL-запроса возникло исключение
```

```
  catch(PDOException $e)
  {
    // Закрываем дескриптор базы данных и генерируем ошибку
    self::Close();
    trigger_error($e->getMessage(), E_USER_ERROR);
  }
  // Возвращаем результаты выполнения запроса
  return $result;
}
```

Этот метод генерирует ошибку (с помощью функции `trigger_error()`), если запрос к базе данных не удалось выполнить. Эта ошибка в дальнейшем перехватывается механизмом обработки ошибок, который мы создали в главе 3.

Код, который мы написали в главе 3, останавливает выполнение запроса при генерации ошибки типа `E_USER_ERROR`, а также записывает в журнал и/или отправляет по электронной почте сведения об ошибке и выводит вежливое сообщение пользователю наподобие "Пожалуйста, попробуйте еще раз немного позже".

Заметьте, что перед генерацией ошибки мы закрываем соединение с базой данных, чтобы гарантировать, что ресурсы сервера не останутся задействованными сценарием, который аварийно завершился.

Если функции `trigger_error()` не указывать, какой тип ошибки сгенерировать, то по умолчанию генерируется ошибка `E_USER_NOTICE`, которая не прерывает нормальное выполнение запроса (сведения об ошибке заносятся в журнал, но сценарий продолжает работать нормально).

Функциональность класса `DatabaseHandler` предназначена для использования другими классами уровня логики приложения, например классом `Catalog`. На данный момент в классе `Catalog` есть только один метод — `GetDepartments()`.

```
// Класс уровня логики приложения для считывания информации о каталоге товаров
class Catalog
{
  // Получаем список отделов
  public static function GetDepartments()
  {
    // Составляем SQL-запрос
    $sql = 'CALL catalog_get_departments_list()';

    // Выполняем запрос и получаем результаты
    return DatabaseHandler::GetAll($sql);
  }
}
```

Поскольку этот класс полагается на функциональность, уже реализованную в методах класса `DatabaseHandler`, код его метода очень прост и прямолинеен. Метод `GetDepartments()` будет вызываться из уровня представления, который отвечает за отображение данных. Вначале этот метод составляет строку SQL-запроса, а затем вызывает метод класса `DatabaseHandler`, чтобы передать этот запрос СУБД для выполнения. В данном случае используется метод `GetAll()`, поскольку составленный запрос должен вернуть несколько строк — список отделов.

В текущей версии приложения соединение с базой данных открывается в файле `index.php`, когда начинается обработка пользовательского запроса, и закрывается, когда обработка этого запроса заканчивается. Все операции с базой данных выполняются через это соединение.

Отображение списка отделов

Итак, все, что нам осталось сделать, чтобы список отделов отображался на вебстранице, — это создать уровень представления. Как вы, вероятно, помните, при открытии сайта в браузере список отделов должен выглядеть примерно так, как на рис. 4.12.

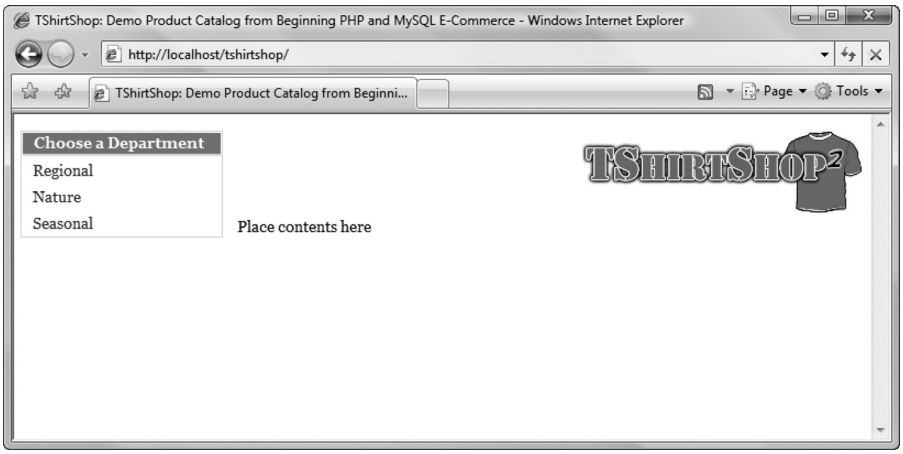


Рис. 4.12. Сайт TShirtShop с динамически генерируемым списком отделов

Функциональность уровня представления мы реализуем в виде отдельного компонентного шаблона — `departments_list`. Затем мы просто включим файл этого шаблона — `departments_list.tpl` — в основной шаблон Smarty (`templates/store_front.tpl`).

Сам компонентный шаблон `departments_list` будет разделен на три файла: дизайн-шаблон Smarty (`templates/departments_list.tpl`), объект представления (`presentation/departments_list.php`) и подключаемый файл Smarty (`presentation/smarty_plugins/function.load_presentation_object.php`). Подключаемый файл Smarty используется всеми шаблонами Smarty для загрузки объектов представления.

Использование подключаемых файлов Smarty

Подключаемые файлы Smarty (Smarty plug-ins) — это механизм в Smarty, с помощью которого мы будем реализовывать управляющую логику для дизайн-шаблонов Smarty (файлов с расширением `.tpl`). Это не единственный возможный способ реализации управляющей логики, но именно он рекомендован в документации по Smarty (`http://smarty.php.net/manual/en/tips.componentized.templates.php`).

Структура веб-страниц в нашем проекте генерируется с помощью дизайн-шаблонов Smarty. Если какой-то компонент страницы сложнее и требует PHP-кода для снабжения его данными или дополнительной функциональностью, мы используем компонентные шаблоны Smarty, в которые входит следующее.

- *Дизайн-шаблон Smarty.* Это файл `.tpl`, в котором содержатся теги HTML и теги, специфичные для Smarty. Для списка отделов используется дизайн-шаблон `departments_list.tpl`.
- *Подключаемая функция Smarty.* Эта функция используется в дизайн-шаблоне Smarty и должна снабжать шаблон данными для отображения. В нашем проекте одна и та же подключаемая функция Smarty (файл `function.load_presentation_object.php`) будет использоваться всеми дизайн-шаблонами Smarty. Однако эта функция будет возвращать разные результаты в зависимости от того, с какими параметрами она вызывается.
- *Объект представления.* Это класс, возвращающий данные, необходимые для дизайн-шаблона Smarty. В случае списка отделов этот класс будет называться

DepartmentsList. Он считывает список отделов и сохраняет его в открытой переменной, к которой может обращаться дизайн-шаблон Smarty.

Подключаемые файлы и функции Smarty должны соответствовать строгим правилам именования, чтобы Smarty смог их найти и использовать. Подключаемые файлы должны применять имена вида тип.имя.php (в нашем случае — function.load_presentation_object.php), а функции внутри этих файлов — имена вида smarty_имя_типа (в нашем случае — smarty_function_load_presentation_object). Официальные правила именования подключаемых файлов и функций изложены на странице http://smarty.php.net/manual/en/plugins.naming.conventions.php. Об использовании подключаемых файлов и функций можно узнать больше на странице http://smarty.php.net/manual/en/plugins.php.

После того как подключаемый файл Smarty создан, на него можно ссылаться из дизайн-шаблона Smarty (departments_list.tpl) с помощью, например, строки

```
{load_presentation_object filename="departments_list" assign="obj"}
```

Если правила именования были соблюдены, этой строки достаточно, чтобы Smarty загрузил подключаемый файл, который в свою очередь загрузит объект представления, указанный в переменной filename, и присвоит загруженный объект переменной шаблона (в нашем случае имя этой переменной — obj). Затем дизайн-шаблон сможет обращаться к переменным, заполненным подключаемой функцией. Это может выглядеть примерно так:

```
{$obj->mDepartments[i].name}
```

Чтобы понять, как работает весь механизм, давайте создадим компонентный шаблон departments_list и все остальные элементы, необходимые для его работы. Мы продолжим использовать CSS для задания визуального оформления веб-страниц. Хотя CSS обладает обширными возможностями, на базовом уровне он очень прост.

Упражнение: создание компонентного шаблона departments_list

1. Откройте файл tshirtshop.css в папке tshirtshop/styles и добавьте в него стили, приведенные ниже. Они описывают вид названий отделов в списке, когда они не выбраны пользователем, когда над ними находится указатель мыши и когда они выбраны.

```
div.yui-b div.box {
  color: #333333;
  border: 1px solid #c6e1ec;
  margin-top: 15px;
}
div.yui-b div p.box-title {
  background: #0590C7;
  border-bottom: 2px solid #c6e1ec;
  color: #FFFFFF;
  display: block;
  font-size: 93%;
  font-weight: bold;
  margin: 1px;
  padding: 2px 10px;
}
a {
  color: #0590C7;
}
a:hover {
```

```
  color: #ff0000;
}
a.selected {
  font-weight: bold;
}
div.yui-b div ul {
  margin: 0;
}
 div.yui-b div ul li {
  border-bottom: 1px solid #fff;
  list-style-type: none;
}
div.yui-b div ul li a {
  color: #333333;
  display: block;
  text-decoration: none;
  padding: 3px 10px;
}
div.yui-b div ul li a:hover {
  background: #c6e1ec;
  color: #333333;
}
```

2. Отредактируйте файл `presentation/application.php` и добавьте в конструктор класса `Application` две строки, выделенные полужирным шрифтом в фрагменте следующего кода. Эти строки конфигурируют папки с подключаемыми файлами Smarty. Первая папка предназначена для внутренних подключаемых файлов, а папке `smarty_plugins` будут находиться подключаемые файлы, которые мы создадим для TShirtShop.

```
/* Класс, расширяющий Smarty, используется для обработки и
   отображения файлов Smarty */
class Application extends Smarty
{
  // Конструктор класса
  public function __construct()
  {
    // Вызов конструктора Smarty
    parent::Smarty();
    // Меняем папки шаблонов по умолчанию
    $this->template_dir = TEMPLATE_DIR;
    $this->compile_dir = COMPILE_DIR;
    $this->config_dir = CONFIG_DIR;
    $this->plugins_dir[0] = SMARTY_DIR . 'plugins';
    $this->plugins_dir[1] = PRESENTATION_DIR . 'smarty_plugins';
  }
}
```

3. Теперь создайте файл шаблона Smarty для компонентного шаблона `departments_list`. Он должен находиться в папке `presentation/templates` и называться `departments_list.tpl`. Поместите в него следующий код.

```
{* departments_list.tpl *}
{load_presentation_object filename="departments_list" assign="obj"}
{* Начало списка отделов *}
<div class="box">
  <p class="box-title">Choose a Department</p>
  <ul>
  {* Перебираем элементы списка отделов *}
  {section name=i loop=$obj->mDepartments}
    {assign var=selected value=""}
```

```
    {* Проверяем, выделен ли отдел, чтобы определить, какой
       стиль CSS использовать *}
    {if ($obj->mSelectedDepartment ==
         $obj->mDepartments[i].department_id)}
      {assign var=selected value="class=\"selected\""}
    {/if}
    <li>
      {* Генерируем ссылку для нового отдела в списке *}
      <a {$selected} href="{$obj->mDepartments[i].link_to_department}">
        {$obj->mDepartments[i].name}
      </a>
    </li>
  {/section}
  </ul>
</div>
{* Конец списка отделов *}
```

Этот файл отвечает за генерацию визуального оформления списка отделов.

4. Создайте в папке `presentation` папку `smarty_plugins`. В ней будут храниться подключаемые файлы Smarty.

5. Внутри папки `smarty_plugins` создайте файл `function.load_presentation_object.php` и добавьте в него следующий код.

```
<?php
// Подключаемые функции из подключаемых файлов должны
// именоваться smarty_имя_типа
function smarty_function_load_presentation_object($params, $smarty)
{
  require_once PRESENTATION_DIR . $params['filename'] . '.php';

  $className = str_replace(' ', '',
                           ucfirst(str_replace('_', ' ',
                                               $params['filename'])));
  // Создаем объект представления
  $obj = new $className();

  if (method_exists($obj, 'init'))
  {
    $obj->init();
  }

  // Присваиваем переменную шаблона
  $smarty->assign($params['assign'], $obj);
}
?>
```

6. В папке `presentation` создайте файл `departments_list.php` и поместите в него следующий код.

```
<?php
// Отвечает за список отделов
class DepartmentsList
{
  /* Public-переменные, доступные в шаблоне Smarty departments_list.tpl */
  public $mSelectedDepartment = 0;
  public $mDepartments;

  // Конструктор считывает строку запроса как параметр
  public function __construct()
  {
    /* Если в строке запроса есть DepartmentId, мы посещаем отдел */
```

```
    if (isset ($_GET['DepartmentId']))
      $this->mSelectedDepartment = (int)$_GET['DepartmentId'];
  }

  /* Вызывает метод уровня логики приложения для считывания списка отделов
  /* и создания
     соответствующих ссылок*/
  public function init()
  {
    // Получаем список отделов из уровня логики приложения
    $this->mDepartments = Catalog::GetDepartments();

    // Создаем ссылки на отделы
    for ($i = 0; $i < count($this->mDepartments); $i++)
      $this->mDepartments[$i]['link_to_department'] =
        'index.php?DepartmentId=' .
         $this->mDepartments[$i]['department_id'];
  }
}
?>
```

7. Внесите в файл `index.php` ссылку на класс `Catalog` из уровня логики приложения.

```
// Загружаем шаблон страницы приложения
require_once PRESENTATION_DIR . 'application.php';

// Загружаем уровень логики приложения
require_once BUSINESS_DIR . 'catalog.php';

// Загружаем файл шаблонов Smarty
$application = new Application();
```

8. Внесите изменения в файл `presentation/templates/store_front.tpl`, чтобы созданный компонентный шаблон `departments_list` загружался. Найдите код

```
<div class="yui-b">
  Place list of departments here
</div>
```

и замените его кодом

```
<div class="yui-b">
  {include file="departments_list.tpl"}
</div>
```

9. Полюбуйтесь результатами своих трудов в окне браузера — откройте в нем страницу `http://localhost/tshirtshop/index.php` (в качестве эталона можете использовать рис. 4.12). Протестируйте страницу — посмотрите, что получится, если подвести указатель мыши к названию отдела или щелкнуть на этом названии.

ЗАМЕЧАНИЕ

Если в окне браузера вы не видите ожидаемых результатов, прежде всего проверьте, правильно ли настроен ваш компьютер и все ли нужные модули PHP загружены (прежде всего — PDO). О многих ошибках можно узнать из файла журнала Apache (по умолчанию — `C:\xampp\apache\logs\error.log` под управлением Windows или `/opt/lampp/logs/error_log` под управлением Linux). Кроме того, зайдите на сайт этой книги и просмотрите раздел, посвященный обнаруженным ошибкам (Errata), — в нем приведены решения проблем, с которыми вы можете столкнуться.

Как это работает: шаблон Smarty `departments_list`

Если ваш список отделов работает как полагается, вам очень повезло. В подавляющем большинстве случаев его приходится долго отлаживать, вылавливая опечатки и пропуски строк в коде. Кроме того, часто встречаются проблемы с доступом к базе данных, так что убедитесь в том, что вы настроили базу данных `tshirtshop` и учетную запись пользователя `tshirtshopadmin` в точности так, как описано в главе 3. В любом случае мы снабдили наш проект механизмом обработки ошибок именно для упрощения отладки. На рис. 4.13 показано сообщение об ошибке, которое я получил, неправильно указав пароль доступа к базе данных в файле `config.php`. Сообщение об ошибке выводится в поле, для которого оно сгенерировано (чтобы это сообщение было удобно читать, скопируйте его в какой-нибудь текстовый редактор).

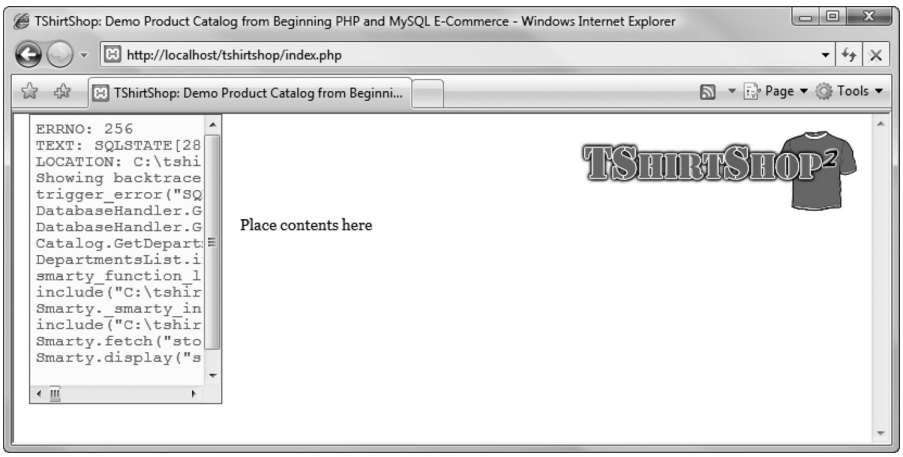


Рис. 4.13. Сообщение об ошибке, которое сгенерировал написанный нами ранее код

Однако, если все работает как положено, вы получите аккуратную страницу со списком отделов, сгенерированным с помощью шаблона Smarty. Название каждого отдела представляет собой ссылку на страницу этого отдела. На самом деле это ссылка на страницу `index.php` с параметром `DepartmentID` в строке запроса, указывающим, какой отдел выбран. Вот пример такой ссылки:

```
http://localhost/tshirtshop/index.php?DepartmentId=3
```

После щелчка на названии отдела в списке этот отдел выделяется визуально — к нему применяется другой стиль CSS (рис. 4.14).



Рис. 4.14. Название выбранного отдела выделено в списке другим стилем

Важно понимать, как файл шаблона Smarty (presentation/templates/department_styles.tpl) и подключаемый файл (presentation/smarty_plugins/function.load_presentation_object.php) генерируют список отделов и применяют нужный стиль к названию выбранного отдела.

Работа начинается с файла function.load_presentation_object.php, который подключается в файле store_front.tpl. Первая строка в файле departments_list.tpl загружает объект представления DepartmentsList через подключаемую функцию Smarty:

```
{load_presentation_object filename="departments_list" assign="obj"}
```

Подключаемая функция smarty_function_load_presentation_object() создает и инициализирует объект DepartmentsList (класс этого объекта находится в файле presentation/departments_list.php), после чего этот объект присваивается переменной obj, доступной для файла дизайн-шаблона Smarty.

```
// Подключаемые функции из подключаемых файлов должны
// именоваться smarty_имя_типа
function smarty_function_load_presentation_object($params, $smarty)
{
  require_once PRESENTATION_DIR . $params['filename'] . '.php';

  $className = str_replace(' ', '',
                           ucfirst(str_replace('_', ' ',
                                               $params['filename'])));
  // Создаем объект представления
  $obj = new $className();

  if (method_exists($obj, 'init'))
  {
    $obj->init();
  }

  // Присваиваем переменную шаблона
  $smarty->assign($params['assign'], $obj);
}
```

Метод init() класса DepartmentsList заполняет данными public-элемент класса ($mDepartments), помещая в него список отделов. В другой public-элемент класса, $mDepartments, этот метод помещает индекс выбранного в данный момент пункта списка.

Вернемся к коду Smarty. Внутри HTML-кода, формирующего структуру шаблона Smarty (presentation/templates/departments_list.tpl), можно увидеть теги, которые и выполняют все нужные действия.

```
{* Перебираем элементы списка отделов *}
{section name=i loop=$obj->mDepartments}
  {assign var=selected value=""}
  {* Проверяем, выделен ли отдел, чтобы определить, какой
     стиль CSS использовать *}
  {if ($obj->mSelectedDepartment ==
       $obj->mDepartments[i].department_id)}
    {assign var=selected value="class=\"selected\""}
  {/if}
  <li>
    {* Генерируем ссылку для нового отдела в списке *}
    <a {$selected} href="{$obj->mDepartments[i].link_to_department}">
      {$obj->mDepartments[i].name}
    </a>
  </li>
{/section}
```

Разделы (теги section) шаблона Smarty используются для перебора элементов массивов. В данном случае мы перебираем элементы массива $obj->mDepartments:

```
{section name=i loop=$obj->mDepartments}
  ...
{/section}
```

В этом цикле мы проверяем, был ли ID конкретного отдела (`$obj->mDepartments[i].department_id`) упомянут в строке запроса (`$obj->mSelectedDepartment`). В зависимости от этого мы решаем, имя какого стиля (`selected` или стиля по умолчанию) нужно сохранить в переменной `selected`.

Затем эта переменная используется для генерации ссылки:

```
<a {$selected} href="{$obj->mDepartments[i].link_to_department}">
  {$obj->mDepartments[i].name}
</a>
```

Создание фабрики ссылок

В правильно спроектированных веб-сайтах все ссылки оформлены по одним и тем же правилам. Например, в PHP параметры запросов обычно считываются по именам, а не по порядку их следования в строках, поэтому две приведенные ниже ссылки обычно эквивалентны:

```
http://localhost/index.php?DepartmentId=3&CategoryId=5
http://localhost/index.php?CategoryId=5&DepartmentId=3
```

Во многих случаях порядок следования частей URL можно менять, не влияя на результат активизации ссылки. Дабы придать единый стиль всем ссылкам на нашем сайте, создадим специальный класс, который будет заниматься генерацией ссылок.

Этот класс в дальнейшем может оказаться очень полезным. В главе 7 мы обновим формат URL на нашем сайте, чтобы они были более удобными для поисковых систем и посетителей. Наличие централизованного механизма генерации URL сильно упростит эту задачу.

Кроме того, позже нам понадобится сделать некоторые страницы сайта доступными только через защищенные соединения HTTPS, чтобы гарантировать конфиденциальность данных, которыми обмениваются сервер и клиент. Защищенными должны быть, например, страницы, на которых пользователь вводит свои имя и пароль, страницы с информацией о кредитных картах и некоторые другие. Пока что мы не будем погружаться в детали, но вам следует знать, что страницы, доступные только через HTTPS, отнимают заметную часть ресурсов сервера, и использовать защищенные соединения для доступа к обычным страницам не стоит. Поэтому фабрика ссылок может весьма нам пригодиться — ее можно будет настроить на генерацию ссылок HTTPS только для частей сайта, требующих повышенного уровня защиты.

Наша фабрика ссылок будет генерировать только абсолютные ссылки. Часто внутри сайта удобнее использовать относительные ссылки, например рисунок на главной странице может содержать короткую ссылку на файл `index.php`, а не полную абсолютную ссылку наподобие `http://www.example.com/index.php`. Но в этом случае пользователь, щелкнув на таком рисунке на странице, открытой через защищенное соединение, откроет `index.php` тоже через защищенное соединение. Это приведет просто к бесполезному расходованию ресурсов сервера.

Во избежание этой проблемы и еще нескольких похожих на нее, мы напишем код, который будет генерировать только абсолютные ссылки.

Упражнение: создание фабрики ссылок

1. В папке `presentation` создайте файл с именем `link.php` и поместите в него следующий код.

```
<?php
class Link
{
  public static function Build($link)
  {
    $base = 'http://' . getenv('SERVER_NAME');
    // Если константа HTTP_SERVER_PORT определена и значение отличается
    // от используемого по умолчанию...
    if (defined('HTTP_SERVER_PORT') && HTTP_SERVER_PORT != '80')
    {
      // Добавляем номер порта
      $base .= ':' . HTTP_SERVER_PORT;
    }
    $link = $base . VIRTUAL_LOCATION . $link;
    // Escape-символы для html
    return htmlspecialchars($link, ENT_QUOTES);
  }

  public static function ToDepartment($departmentId)
  {
    $link = 'index.php?DepartmentId=' . $departmentId;
    return self::Build($link);
  }
}
?>
```

2. Добавьте в файл `include/config.php` две константы.

```
// Порт HTTP-сервера (можно пропустить, если используется порт 80)
define('HTTP_SERVER_PORT', '80');

/* Имя виртуальной директории, в которой располагается сайт, например:
  '/tshirtshop/' если сайт работает из папки
http://www.example.com/tshirtshop/
  '/' если сайт работает из папки http://www.example.com/ */
define('VIRTUAL_LOCATION', '/tshirtshop/');
```

3. Модифицируйте метод `init()` из класса `DepartmentsList` в файле `presentation/departments_list.php` так, как показано в следующем листинге (изменения выделены полужирным шрифтом).

```
  /* Вызывает метод уровня логики приложения для считывания списка
  /* отделов и создания
     соответствующих ссылок*/
  public function init()
  {
    // Получаем список отделов из уровня логики приложения
    $this->mDepartments = Catalog::GetDepartments();
    // Создаем ссылки на отделы
    for ($i = 0; $i < count($this->mDepartments); $i++)
      $this->mDepartments[$i]['link_to_department'] =
        Link::ToDepartment($this->mDepartments[$i]['department_id']);
  }
```

4. Создайте в папке `presentation` файл `store_front.php` и поместите в него следующий фрагмент кода. Это объект уровня представления, который мы будем

использовать в шаблоне store_front, и он отвечает за создание ссылки на главную страницу сайта.

```
<?php
class StoreFront
{
  public $mSiteUrl;

  // Конструктор класса
  public function __construct()
  {
    $this->mSiteUrl = Link::Build('');
  }
}
?>
```

5. Вставьте в файл presentation/templates/store_front.tpl строки, выделенные в следующем листинге полужирным шрифтом.

```
{* smarty *}
{config_load file="site.conf"}
{load_presentation_object filename="store_front" assign="obj"}
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title>{#site_title#}</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <link type="text/css" rel="stylesheet"
    href="{$obj->mSiteUrl}styles/tshirtshop.css" />
  </head>
  <body>
    <div id="doc" class="yui-t2">
      <div id="bd">
        <div id="yui-main">
          <div class="yui-b">
            <div id="header" class="yui-g">
              <a href="{$obj->mSiteUrl}">
                <img src="{$obj->mSiteUrl}images/tshirtshop.png"
                 alt="tshirtshop logo" />
              </a>
            </div>
            <div id="contents" class="yui-g">
              Place contents here
            </div>
          </div>
        </div>
        <div class="yui-b">
          {include file="departments_list.tpl"}
        </div>
      </div>
    </div>
  </body>
</html>
```

6. Откройте файл index.php и добавьте в него ссылку на класс Link, выделенную в следующем листинге полужирным шрифтом.

```
<?php
// Включаем вспомогательные файлы
require_once 'include/config.php';
require_once BUSINESS_DIR . 'error_handler.php';
```

```
// Задаем обработчик ошибок
ErrorHandler::SetHandler();

// Загружаем шаблон страницы приложения
require_once PRESENTATION_DIR . 'application.php';
require_once PRESENTATION_DIR . 'link.php';

// Загружаем дескриптор базы данных
require_once BUSINESS_DIR . 'database_handler.php';
```

7. Попробуйте открыть приложение TShirtShop в браузере и убедитесь, что оно работает как надо. Это упражнение не должно менять уже существующую функциональность — его целью является добавление нового механизма, который пригодится нам при расширении сайта в последующих главах.

Как это работает: использование фабрики ссылок

Прежде всего убедитесь, что новая запись в файле `config.php` воспринимается приложением корректно. Если для запуска нашего приложения вы используете порт, отличный от 80 (например, 8080), укажите требуемый номер порта в константе `HTTP_SERVER_PORT`. А теперь давайте посмотрим, как работает фабрика ссылок. Объект представления `Link` используется так, как показывают изменения, внесенные в файлы `store_front.tpl` и `departments_list.tpl`, — он преобразует полученные относительные ссылки в абсолютные.

ЗАМЕЧАНИЕ

Если вы не используете псевдоним `tshirtshop` так, как описано в главе 3, вам нужно будет изменить значение константы `VIRTUAL_LOCATION` в файле `config.php`, чтобы в ней было указано реальное место размещения приложения на вашем компьютере.

Заметьте, что метод `Build()` не добавляет в ссылку номер порта, если константа `HTTP_SERVER_PORT` не объявлена или ее значение равно `80`.

```
// Если константа HTTP_SERVER_PORT определена и значение отличается
// от используемого по умолчанию...
if (defined('HTTP_SERVER_PORT') && HTTP_SERVER_PORT != '80')
{
  // Добавляем номер порта
  $base .= ':' . HTTP_SERVER_PORT;
}
```

Однако константу `HTTP_SERVER_PORT` стоит добавить в файл `config.php` в любом случае — это упростит перенастройку приложения, если вы перенесете его на сервер, работающий через другой порт. Если бы, например, `HTTP_SERVER_PORT` имела значение `8080`, то ссылка на файл `index.php` приобрела бы вид

```
<a href="http//www.example.com:8080/index.php">
```

Резюме

Это была длинная глава, правда? В ней вы узнали немало нового и создали солидную часть будущего электронного магазина. Говоря конкретнее, в этой главе вы сделали следующее.

- Создали таблицу `department` и заполнили ее данными.
- Узнали, как обращаться к этим данным из уровня данных с помощью PDO и к уровню данных из уровня логики приложения.

- Узнали, как использовать исключения PHP 5.
- Реализовали интерфейс пользователя с помощью шаблонов Smarty.

В следующей главе мы закончим создание каталога товаров. После этого на страницах приложения будут отображаться категории товаров и отдельные товары! Затем мы сможем рассмотреть в целом структуру нашего веб-сайта и теоретические сведения, полученные вами к тому времени.

Глава 5

Создание каталога товаров: часть II

В предыдущей главе мы создали в приложении элемент, позволяющий отображать списки отделов и выбирать нужные отделы. Однако каталог товаров — это не только список отделов. В этой главе мы добавим в TShirtShop много новых элементов каталога — списки товаров и подробные сведения об этих товарах.

Чтобы приблизительно представить себе, какую функциональность мы добавим в приложение в этой главе, можете еще раз взглянуть на рис. 4.1–4.3. Если говорить точнее, то в этой главе вы

- узнаете больше о данных, хранящихся в реляционных базах данных, а также об отношениях между таблицами;
- узнаете, как создавать новые структуры данных в базах данных;
- узнаете, что такое соединения баз данных и как с ними работать;
- дополните уровень логики приложения функциональностью, использующей новые механизмы и новые хранимые процедуры MySQL;
- завершите уровень представления, дополнив его механизмами отображения категорий товаров и сведений об отдельных товарах.

Хранение новых данных

Учитывая, какую функциональность мы хотим добавить в наше приложение в этой главе, думаю, неудивительно, что в базу данных нужно добавить новые таблицы. Однако новыми таблицами изменения в базе данных не ограничатся. Вы узнаете о том, что такое отношения между таблицами, и как эти отношения можно использовать для получения данных из нескольких таблиц сразу.

Из чего состоит реляционная база данных

Вряд ли вы удивитесь, узнав, что в базе данных хранятся данные. Однако современные системы управления реляционными базами данных (РСУБД), например MySQL, PostgreSQL, SQL Server, Oracle, DB 2 и другие, основываются не просто на данных, а на реляционных данных. Этот момент заслуживает более подробного обсуждения.

Итак, что же такое реляционные данные? Легко можно заметить, что все данные, записываемые в базу данных, каким-то образом связаны с данными, записанными в нее ранее. Товары как-то связаны с отделами и категориями, заказы связаны с клиентами и товарами, и т.д. Реляционная база данных не только хранит данные в своих таблицах, но и отслеживает отношения между этими таблицами.

В целом именно взаимосвязанные таблицы и образуют реляционную базу данных, делая ее чем-то большим, чем просто набор отдельных таблиц. Говорят, что данные превращаются в информацию, когда они обретают смысл (можно считать, что они обретают смысл, когда мы устанавливаем отношения между ними).

Давайте посмотрим на каталог товаров и определим, какие элементы данных в нем содержатся и как эти элементы превратить в информацию. Для каталога товаров понадобятся как минимум три таблицы: по одной для отделов, категорий и товаров. Важно отметить, что физически каждая таблица является независимым объектом в базе данных, даже если логически она входит в более крупный объект. Другими словами, хотя мы и говорим, что товары относятся к категории, таблица товаров вовсе не входит в состав таблицы категорий. Это утверждение не противоречит теории реляционных баз данных.

На рис. 5.1 показано простое представление трех таблиц, содержащих определенные данные.

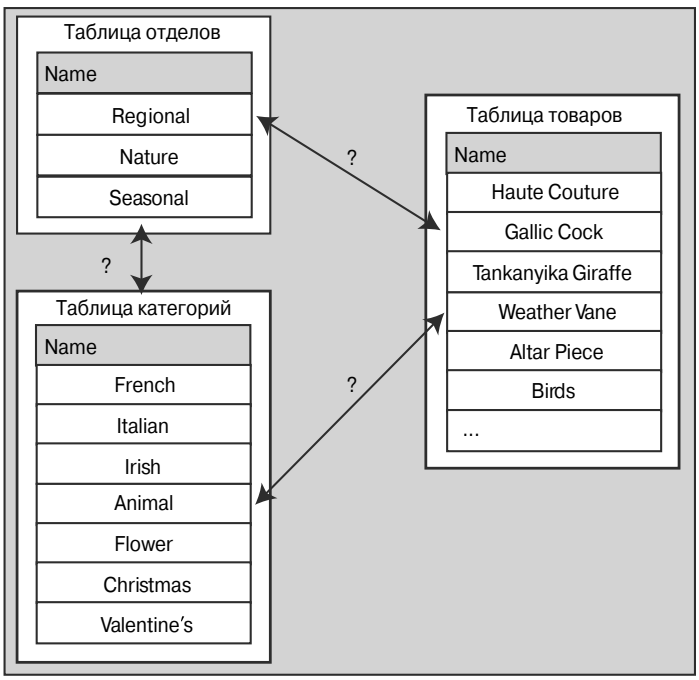


Рис. 5.1. Несвязанные между собой таблицы отделов, категорий и товаров

Когда говорят, что между двумя таблицами есть отношение, на самом деле подразумевают, что отношение есть между отдельными записями этих таблиц. Поэтому, если есть отношение между таблицей товаров и таблицей категорий, значит, каждая запись в таблице товаров как-то связана с определенными записями в таблице категорий.

На рис. 5.1 не показано физическое представление базы данных, а также настоящие имена таблиц. Диаграммы наподобие рис. 5.1 позволяют только решить, *что* будет храниться в базе данных. Когда вы решите, что будет храниться, нужно решить, *как* хранящиеся данные будут взаимосвязаны. Здесь становится существенной логическая структура базы данных. Хотя на рис. 5.1 показаны три таблицы, на самом деле для реализации этой структуры нам понадобится еще одна таблица.

Итак, решив, какие данные мы будем хранить, давайте подумаем, как таблицы будут связаны одна с другой — какие типы *отношений* (relations) будут между ними. Давайте посмотрим, какие вообще типы отношений бывают в реляционных базах данных.

Реляционные данные и отношения между таблицами

Чтобы продолжить наше исследование мира реляционных баз данных, давайте подробнее рассмотрим три логические таблицы, показанные на рис. 5.1. Для ясности дадим этим таблицам имена: таблицу с товарами назовем `products`, таблицу с категориями — `category`, а таблицу с отделами, как и в предыдущей главе, — `department`. Между этими тремя таблицами будут отношения двух часто встречающихся видов — *"один-ко-многим"* (one-to-many) и *"многие-ко-многим"* (many-to-many).

ЗАМЕЧАНИЕ

Существует несколько вариантов этих двух отношений, а также менее распространенные отношения *"один-к-одному"* (one-to-one). Например, в базе данных, отвечающей за распределение пациентов по палатам больницы, именно отношение "один-к-одному" должно описывать размещение пациентов по койкам. СУБД обычно не поддерживают такой тип отношений напрямую, но для его соблюдения достаточно одновременно добавлять данные в две таблицы. Кроме того, две таблицы с отношением "один-к-одному" можно просто объединить в одну.

Отношения "один-ко-многим"

Отношение "один-ко-многим" имеет место, если одну запись из какой-то таблицы можно сопоставить нескольким записям из другой таблицы, но не наоборот. В нашем случае между таблицами отделов и категорий существует именно отношение "один-ко-многим": одному отделу может соответствовать несколько категорий, но каждая категория относится только к одному отделу. Это отношение показано на рис. 5.2.

Еще один случай применения отношения "один-ко-многим" — таблицы `order` и `order_detail`. В таблице `order` содержатся общие сведения о каждом заказе (дата, общая сумма, имя клиента и т.д.), а в `order_detail` — сведения о каждом из товаров, входящих в отдельный заказ.

Когда в базе данных требуется реализовать отношение "один-ко-многим", в таблицу на стороне "многим" добавляется вспомогательный столбец, ссылающийся на значения ID из таблицы на стороне "один". Например, в таблице `category` будет столбец `department_id`. В нем для каждой категории будет указан ID отдела, к которому относится эта категория. Все это мы реализуем в базе данных немного позже, после того, как вы познакомитесь с отношениями "многие-ко-многим" и ограничениями `FOREIGN KEY`.

Отношения "многие-ко-многим"

Это еще один часто встречающийся вид отношений. Он имеет место, когда записи в каждой из таблиц могут соответствовать несколько записей из другой таблицы в отношении. В нашем случае такое отношение есть между таблицами `prod-`

uct и category, поскольку одновременно и товар может относиться к нескольким категориям, и в категорию может входить несколько товаров.

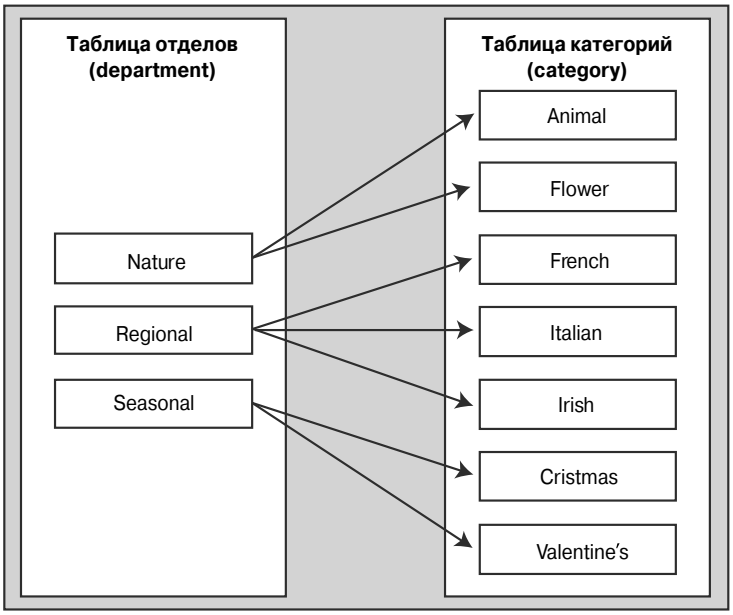


Рис. 5.2. Отношение "один-ко-многим" между таблицами отделов и категорий

Это произошло, поскольку мы решили, что товар может относиться к нескольким категориям. Если бы товар мог относиться только к одной, у нас было бы еще одно отношение "один-ко-многим" наподобие существующего между таблицами department и category (где категория не может относиться более чем к одному отделу).

Графически отношение "многие-ко-многим" можно представить примерно так, как на рис. 5.3.

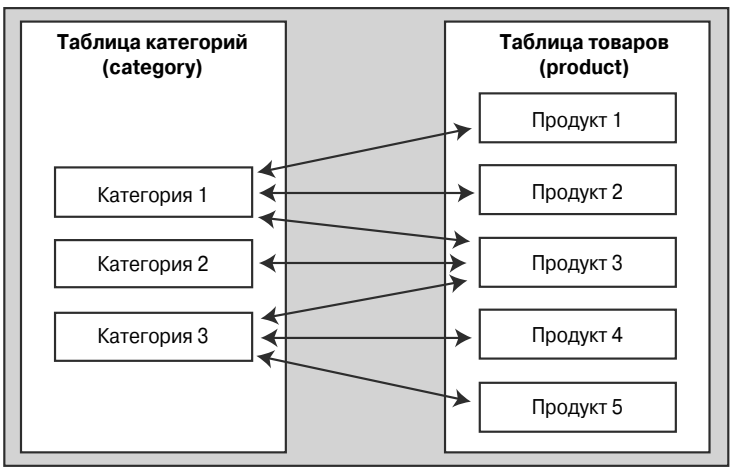


Рис. 5.3. Отношение "многие-ко-многим" между таблицами категорий и товаров

Хотя отношение "многие-ко-многим" часто встречается в базах данных, его нельзя физически реализовать с помощью только двух таблиц. Поэтому в большинстве РСУБД (включая MySQL) для реализации такого отношения используется дополнительная таблица, которая называется *таблицей соединения* (junction table) или *связующей таблицей* (linking table). Она позволяет представить отношение "многие-ко-многим" как два отношения "один-ко-многим". Например, она позволит связать каждый товар с произвольным количеством категорий, а каждую категорию — с произвольным количеством товаров, как показано на рис. 5.4. При этом неважно, сколько у нас категорий и товаров.

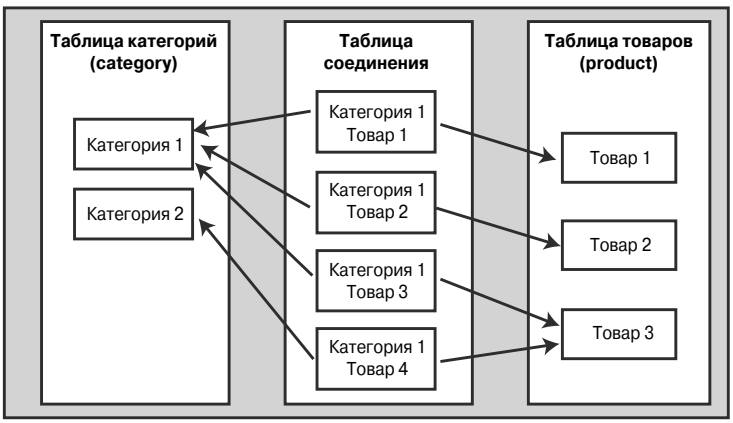


Рис. 5.4. Отношение "многие-ко-многим", преобразованное в два отношения "один-ко-многим"

Как видите, каждая запись в таблице соединения связывает одну категорию с одним товаром. В таблицу соединений можно добавить сколько угодно записей, связывая любой товар с любой категорией. В этой таблице должно быть два поля: идентификатор категории, `category_id`, и идентификатор товара, `product_id`. Добавляя новые записи в таблицу соединений, мы можем связывать новые товары с уже существующими категориями или новые категории с уже существующими товарами.

Поскольку отношение "многие-ко-многим" реализуется через вспомогательную таблицу, в связываемые таблицы не нужно добавлять новые столбцы, как при реализации отношения "один-ко-многим".

Имя для таблицы соединений можно выбирать произвольно, однако обычно его образуют, просто соединяя имена двух соединяемых таблиц — в нашем случае оно будет выглядеть как `product_category`.

Отношения между таблицами и внешние ключи

Соблюдение отношений между таблицами в базе данных можно гарантировать с помощью ограничения `FOREIGN KEY`, или *внешнего ключа*.

В предыдущей главе мы рассматривали ограничения `PRIMARY KEY` и `UNIQUE`, которые относились к отдельно взятой таблице. Внешний ключ относится к отношению между двумя таблицами — таблице, в которой он определен (ссылающаяся таблица), и таблице, на которую он ссылается.

ПОДСКАЗКА

На самом деле можно сделать так, что таблица будет ссылаться с помощью внешнего ключа сама на себя. Это не слишком распространенный случай, но он тоже встречается. Например, в таблице со сведениями о работниках каждая строка может содержать ссылку на строку руководителя данного работника (в этом случае строка самого главного начальника будет, вероятно, ссылаться сама на себя).

Внешний ключ — это столбец или несколько столбцов, используемых для контроля связи между двумя таблицами (обычно при отношении "один-ко-многим"). Внешние ключи используются и для обеспечения целостности данных, и для установки отношения между таблицами.

Для обеспечения целостности данных внешние ключи позволяют задавать определенные ограничения. В отличие от ограничений `PRIMARY KEY` и `UNIQUE`, относящихся к отдельно взятой таблице, ограничение `FOREIGN KEY` накладывается на обе участвующие в нем таблицы. Например, если мы решим обеспечивать соблюдение отношения "один-ко-многим" между таблицами `department` и `category` с помощью ограничения `FOREIGN KEY`, это ограничение будет включено в систему контроля целостности данных. При этом мы не сможем добавить категорию в несуществующий отдел или удалить отдел, в котором есть категории.

К сожалению, использование ограничений `FOREIGN KEY` в MySQL связано с некоторыми сложностями. Проблема в том, что используемая по умолчанию в MySQL система хранения данных — MyISAM — не поддерживает контроля ограничений `FOREIGN KEY`. Альтернатива MyISAM — InnoDB — поддерживает этот контроль, но в ней нет возможности полнотекстового поиска, который нам понадобится (в главе 8).

К счастью, мы можем использовать разные системы хранения данных для разных таблиц в одной и той же базе данных, поэтому мы будем использовать MyISAM для таблиц, в которых нам понадобится выполнять текстовый поиск, и InnoDB для других таблиц. Однако при манипуляциях данными в таблицах, использующих MyISAM, придется соблюдать особую осторожность, поскольку в них не гарантируется соблюдение ограничений.

ЗАМЕЧАНИЕ

В системах, не поддерживающих внешние ключи, их можно реализовать программно с помощью *триггеров* (triggers). Эта методика описана в статье, доступной по адресу `http://dev.mysql.com/tech-resources/articles/mysql-enforcing-foreign-keys.html`.

Прежде чем двигаться дальше, нужно разобраться в разных системах хранения данных, используемых в таблицах MySQL.

Типы таблиц в MySQL

MySQL поддерживает несколько систем хранения данных, позволяющих работать с таблицами. При создании новой таблицы по умолчанию используется система MyISAM. Вот краткое описание поддерживаемых систем.

- MyISAM — система хранения данных, используемая по умолчанию при создании новых таблиц, начиная с MySQL 3.23 (в более ранних версиях использовалась система ISAM). Это самая быстрая из всех доступных систем, но она не поддерживает внешние ключи, ограничения `CHECK`, транзакции и некоторые другие возможности. Однако, в отличие от других типов таблиц, она поддерживает полнотекстовый поиск, который очень пригодится нам при реализации веб-сайта.

- InnoDB — популярная альтернатива MyISAM, которая, помимо всего прочего, поддерживает транзакции, позволяет одновременно выполнять множество операций обновления и обеспечивает соблюдение ограничений FOREIGN KEY. Эта система разрабатывается независимо от MySQL, ее домашняя страница находится по адресу http://www.innodb.com.
- HEAP — это особый тип таблиц, создаваемый в оперативной памяти. Таблицы этого типа нельзя использовать для долговременного хранения данных (при сбое системы данные в них будут безвозвратно потеряны), но они позволяют очень быстро выполнять операции с данными, которые в случае потери легко восстановить или создать заново.

Если вы хотите узнать больше о системах хранения данных, поддерживаемых MySQL, попробуйте начать с посвященной им страницы в документации (http://dev.mysql.com/doc/refman/5.0/en/storage-engines.html).

ЗАМЕЧАНИЕ

В каталоге товаров проекта TShirtShop мы в основном будем использовать таблицы MyISAM, поскольку нам понадобится возможность полнотекстового поиска. Если вам нужно будет изменить тип таблицы после ее создания, это можно сделать с помощью оператора ALTER TABLE. Например, для преобразования таблицы department в тип InnoDB достаточно строки

```
ALTER TABLE department ENGINE=InnoDB;
```

Создание и заполнение новых таблиц

Сейчас мы создадим три таблицы для каталога товаров.

- category
- product
- product_category

Добавление категорий

Процесс создания таблицы категорий не слишком отличается от такового для таблицы department, которую мы создали в главе 4. В таблице category будет четыре столбца, описанных в табл. 5.1.

Таблица 5.1. Столбцы таблицы category

Имя столбца	Тип данных	Описание
category_id	int	Целочисленный столбец с автоинкрементом, содержащий уникальные идентификаторы категорий. Этот столбец — первичный ключ таблицы, и в нем не могут присутствовать значения NULL
department_id	int	Целочисленный столбец, указывающий, к какому отделу принадлежит категория. В этом столбце не могут присутствовать значения NULL
name	varchar(100)	Столбец, в котором хранятся названия категорий. В этом столбце не могут присутствовать значения NULL
description	varchar(1000)	Столбец, в котором хранятся описания категорий. В этом столбце допустимы значения NULL

Вам доступны два способа создания этой таблицы: можно просто выполнить сценарий, загруженный с веб-страницы книги на сайте Apress (`http://www.apress.com/`), а можно последовательно выполнить инструкции из приведенного ниже упражнения.

Упражнение: создание таблицы `category`

1. Откройте базу данных `tshirtshop` с помощью phpMyAdmin.
2. Щелкните на кнопке SQL в верхнем меню и с помощью открывшейся формы выполните следующий запрос SQL (он создаст таблицу `category`). Если такой подход вас не устраивает, можете создать таблицу, указав параметры столбцов с помощью графического интерфейса, как мы делали это раньше для таблицы `department`.

```
-- Создаем таблицу category
CREATE TABLE 'category' (
  'category_id'   INT            NOT NULL  AUTO_INCREMENT,
  'department_id' INT            NOT NULL,
  'name'          VARCHAR(100)   NOT NULL,
  'description'   VARCHAR(1000),
  PRIMARY KEY ('category_id'),
  KEY 'idx_category_department_id' ('department_id')
) ENGINE=MyISAM;
```

3. Заполните таблицу данными, выполнив следующий SQL-оператор.

```
-- Заполняем таблицу category данными
INSERT INTO 'category' ('category_id', 'department_id', 'name',
'description') VALUES
      (1, 1, 'French', 'The French have always had an eye for beauty. One
look
at the T-shirts below and you''ll see that same appreciation has been applied
abundantly to their postage stamps. Below are some of our most beautiful and
colorful T-shirts, so browse away! And don't forget to go all the way to the
bottom - you don''t want to miss any of them!'),
      (2, 1, 'Italian', 'The full and resplendent treasure chest of art,
literature, music, and science that Italy has given the world is reflected
splendidly in its postal stamps. If we could, we would dedicate hundreds of
T-shirts to this amazing treasure of beautiful images, but for now we will
have to live with what you see here. You don't have to be Italian to love
these gorgeous T-shirts, just someone who appreciates the finer things in
life!'),
      (3, 1, 'Irish', 'It was Churchill who remarked that he thought the
Irish
most curious because they didn''t want to be English. How right he was! But
then, he was half-American, wasn''t he? If you have an Irish genealogy you
will want these T-shirts! If you suddenly turn Irish on St. Patrick's Day,
you too will want these T-shirts! Take a look at some of the coolest T-shirts
we have!'),
      (4, 2, 'Animal', ' Our ever-growing selection of beautiful animal
Tshirts
represents critters from everywhere, both wild and domestic. If you
don't see the T-shirt with the animal you''re looking for, tell us and
we''ll find it!'),
      (5, 2, 'Flower', 'These unique and beautiful flower T-shirts are just
the item for the gardener, flower arranger, florist, or general lover of
things beautiful. Surprise the flower in your life with one of the beautiful
botanical T-shirts or just get a few for yourself!'),
      (6, 3, 'Christmas', ' Because this is a unique Christmas T-shirt that
```

```
you''ll only wear a few times a year, it will probably last for decades
(unless
some grinch nabs it from you, of course). Far into the future, after you're
gone, your grandkids will pull it out and argue over who gets to wear it.
What
great snapshots they''ll make dressed in Grandpa or Grandma''s incredibly
tasteful and unique Christmas T-shirt! Yes, everyone will remember you for-
ever
and what a silly goof you were when you would wear only your Santa beard and
cap so you wouldn''t cover up your nifty T-shirt.'),
      (7, 3, 'Valentine''s', 'For the more timid, all you have to do is wear
your heartfelt message to get it across. Buy one for you and your sweetie(s)
today!');
```

Как это работает: заполнение таблицы данными

Добавление в таблицу новых данных не представляет сложностей, если вы знаете, какие данные хотите добавить. Как упоминалось выше, SQL-сценарии для этой книги вы можете найти в разделе Source Code/Download на веб-сайте издательства Apress (`http://www.apress.com`). На рис. 5.5 показано, как данные из таблицы `category` отображаются в phpMyAdmin.

←T→	category_id	department_id	name	description
	1	1	French	The French have always had an eye for beauty. One ...
	2	1	Italian	The full and resplendent treasure chest of art, li...
	3	1	Irish	It was Churchill that remarked that he thought the...
	4	2	Animal	Our ever-growing selection of beautiful animal T-...
	5	2	Flower	These unique and beautiful flower T-shirts are jus...
	6	3	Christmas	Because this is a unique Christmas T-shirt that y...
	7	3	Valentine's	For the more timid, all you have to do is wear you...

Рис. 5.5. Данные из таблицы `category`, отображаемые `phpMyAdmin`

Обратите внимание на использование специальных последовательностей символов в SQL-операторе. Например, апострофы в этом операторе повторяются дважды, чтобы MySQL понял, что их нужно вставить в текстовую строку, а не воспринять как знак ее завершения.

Добавление названий товаров и их привязка к категориям

Сейчас мы создадим таблицу несколько более сложную, чем в предыдущем упражнении, но все основные шаги останутся неизменными. Таблица `products` будет состоять из столбцов, перечисленных в табл. 5.2.

Таблица 5.2. Столбцы таблицы `products`

Имя столбца	Тип данных	Описание
`product_id`	`int`	Целочисленный столбец с автоинкрементом, содержащий уникальные идентификаторы категорий. Этот столбец — первичный ключ таблицы, и в нем не могут присутствовать значения `NULL`
`name`	`varchar(100)`	Столбец, в котором хранятся названия товаров. В нем не могут присутствовать значения `NULL`
`description`	`varchar(1000)`	Столбец, в котором хранятся описания товаров. В нем допустимы значения `NULL`

Окончание табл. 5.2

Имя столбца	Тип данных	Описание
price	numeric(10, 2)	Столбец для хранения цен товаров
discounted_price	numeric(10, 2)	Столбец для хранения цен со скидками. Если в данный момент скидок на какой-то товар нет, то для товара будет храниться значение 0.00
image	varchar(150)	В этом столбце хранятся имена файлов с изображениями товаров (или, точнее, имена файлов и пути к ним). Эти изображения используются на страницах с подробной информацией о товарах. Можно хранить изображения непосредственно в базе данных, но обычно гораздо эффективнее хранить файлы с изображениями отдельно и записывать в базу данных только их имена. Если ваш сайт часто посещаемый, можно даже переместить файлы изображений на отдельный носитель (например, на другой жесткий диск), чтобы повысить производительность. В этом столбце могут присутствовать значения NULL
image_2	varchar(150)	Столбец для хранения имени второго файла с изображением товара. В нем могут присутствовать значения NULL
thumbnail	varchar(150)	Столбец для хранения имен файлов с уменьшенными изображениями товаров. Эти изображения используются в списках товаров в каталоге
display	smallint	Значение из этого столбца указывает, в каких разделах каталога должен отображаться соответствующий товар. Возможные значения: 0 (по умолчанию) — только на странице категории, к которой относится этот товар; 1 — еще и на главной странице каталога; 2 — на странице отдела, к которому товар относится; 3 — и на главной странице, и на странице отдела. Этот столбец помогает администраторам сайта выделять товары, которые могут быть особенно привлекательными для посетителей, например специальные предложения. Кроме того, с помощью этого столбца можно выделять товары со скидками

Таблица product_category предназначена для поддержки отношения "многие-ко-многим" между таблицами product и category и состоит из двух столбцов: product_id и category_id. Они образуют первичный ключ.

Выполните следующее упражнение, чтобы создать в базе данных таблицу product.

Упражнение: создание таблицы product

1. С помощью phpMyAdmin выполните следующий оператор SQL, чтобы создать таблицу product.

```
-- Создаем таблицу product
CREATE TABLE `product` (
  `product_id`       INT            NOT NULL AUTO_INCREMENT,
  `name`             VARCHAR(100)   NOT NULL,
  `description`      VARCHAR(1000)  NOT NULL,
  `price`            NUMERIC(10, 2) NOT NULL,
  `discounted_price` NUMERIC(10, 2) NOT NULL DEFAULT 0.00,
```

```
  `image`             VARCHAR(150),
  `image_2`           VARCHAR(150),
  `thumbnail`         VARCHAR(150),
  `display`           SMALLINT       NOT NULL DEFAULT 0,
PRIMARY KEY (`product_id`)
) ENGINE=MyISAM;
```

2. Создайте таблицу `product_category` с помощью следующего запроса.

```
-- Создание таблицы product_category
CREATE TABLE `product_category` (
  `product_id`  INT NOT NULL,
  `category_id` INT NOT NULL,
  PRIMARY KEY (`product_id`, `category_id`)
) ENGINE=MyISAM;
```

3. Запустите на выполнение сценарий `populate_product.sql` из загружаемых материалов к книге, чтобы заполнить данными таблицу `product`.

4. С помощью сценария `populate_product_category.sql` из загружаемых материалов к книге заполните данными таблицу `product_category`.

Как это работает: отношения "многие-ко-многим"

Для реализации отношений "многие-ко-многим" используется вспомогательная таблица, которая называется таблицей соединений. В данном случае она называется `product_category`. В ней содержатся пары значений `product_id` и `category_id`, связывающие определенные товары и категории. Например, запись наподобие `(1, 4)` в таблице `product_category` связывает товар с ID `1` и категорию с ID `4`.

Таблица `product_category` — первая в нашем проекте, использующая в качестве первичного ключа не один столбец, а группу. В этой таблице первичный ключ формируется из пар значений в столбцах `product_id` и `category_id`. Это значит, что отдельные значения в столбцах `product_id` и `category_id` могут повторяться, но все сочетания этих значений в строках таблицы должны быть разными. Данное требование вполне логично: любой товар можно связать с любой категорией, но связать его с этой категорией несколько раз нельзя.

Использование диаграмм баз данных

Теоретическое описание отношений между таблицами может быть запутанным и непонятным, но постепенно вы в них разберетесь. Чтобы представить себе эти отношения более ясно, можно воспользоваться графическими диаграммами баз данных.

Существуют инструменты, позволяющие строить диаграммы баз данных, физически генерировать по ним готовые базы данных и составлять необходимые SQL-операторы. Хотя в этой книге мы и не будем использовать такие инструменты, вам следует знать, что они существуют. Список наиболее популярных товаров этого класса можно найти по адресу `http://www.databaseanswers.com/modelling_tools.htm`.

На диаграммах баз данных можно легко показывать отношения между таблицами. Например, диаграмма с таблицами и отношениями, которые мы создаем в базе данных `tshirtshop`, показана на рис. 5.6.

На диаграмме первичные ключи таблиц обозначаются буквами "PK" (Primary Key), а внешние — "FK" (Foreign Key). Поскольку внешних ключей в каждой таблице может быть несколько, эти ключи нумеруются, например FK1, FK2 и т.д.

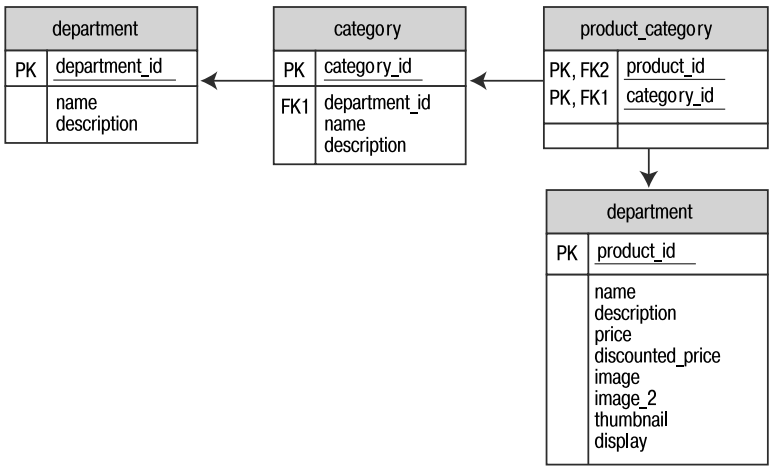


Рис. 5.6. Диаграмма базы данных `tshirtshop` с таблицами и отношениями

Запросы к новым данным

Итак, у нас есть база данных, данные из которой должно обрабатывать наше приложение. Однако, прежде чем мы начнем писать код, который будет этим заниматься, нужно разобраться в кое-каких тонкостях.

В этой главе уровень данных будет несколько сложнее, чем в предыдущей, потому что он должен будет обрабатывать более сложные запросы наподобие "Покажите мне вторую страницу списка футболок из категории Cartoons" или "Покажите мне товары со скидками для отдела X". Прежде чем создавать хранимые процедуры, которые будут работать с данными, давайте рассмотрим теоретические вопр осы.

- Как извлекать краткие описания товаров из базы данных
- Как соединять в запросах разные таблицы
- Как реализовывать поддержку страниц в списках товаров

Начнем с первого вопроса.

Получение кратких описаний товаров

На странице со списком товаров не стоит отображать полные описания каждого товара — они слишком длинны, чтобы их можно было аккуратно разместить на этой странице. Поэтому мы будем отображать на странице списка только начало каждого описания — по 150 символов. Если описание длиннее 150 символов, мы будем отображать вместо остальных символов троеточие. Разумеется, вместо 150 можно выбрать другое количество символов — главное здесь, чтобы страница со списком товаров выглядела аккуратной. Меняя количество отображаемых символов из описания, просматривайте страницу списка в браузере, чтобы убедиться в том, что она нормально отображается.

Для выделения заданного количества символов из описаний товаров мы будем использовать функцию MySQL `LEFT(str, len)`. Ее параметр `str` — это строка, из которой извлекаются символы, а `len` — количество символов, которое нужно извлечь. Например, следующий оператор `SELECT` возвращает первые 30 символов из

описаний товаров с символами ... вместо оставшихся символов (если описание длиннее 30 символов).

```
SELECT   name,
         IF(LENGTH(description) <= 30, description,
                   CONCAT(LEFT(description, 30), '...')) AS description
FROM     product
ORDER BY name;
```

Новый столбец, генерируемый выражением `CONCAT(LEFT(description, 30), '...'))`, по умолчанию не имеет имени, поэтому мы присваиваем ему псевдоним `description` с помощью ключевого слова `AS`. Для данных, которые в текущий момент должны находиться в таблице `product`, этот запрос вернет примерно такой результат.

name	description
A Partridge in a Pear Tree	The original of this beautiful...
Adoration of the Kings	This design is from a miniatur...
Afghan Flower	This beautiful image was issue...
Albania Flower	Well, these crab apples starte...
Alsace	It was in this region of Franc...
...	...

Соединения таблиц данных

Поскольку нужные нам данные хранятся в нескольких таблицах, необходимо каким-то образом объединять их для обработки. Например, ниже показаны данные, выбранные из таблиц `department` и `category`.

Department Name	Category Name
Regional	French
Regional	Italian
Regional	Irish
Nature	Animal
Nature	Flower
Seasonal	Christmas
Seasonal	Valentine's

В других случаях нужные данные могут содержаться в одной таблице, но для выбора требуемых строк следует задавать условия с использованием данных из другой таблицы. Для получения таких данных нужны более сложные запросы, чем те, которые мы использовали до сих пор. В большинстве случаев при работе с данными из нескольких таблиц используются *соединения таблиц* (table joins).

Запрос, извлекающий из базы данных товары определенной категории, сильно отличается от запроса для извлечения категорий определенного отдела, потому что товары и категории связывает между собой специальная таблица product_category.

Чтобы получить список товаров из определенной категории, сначала нужно будет обратиться к таблице product_category и извлечь из нее все пары значений (product_id, category_id) с требуемым значением category_id. Список этих пар будет содержать идентификаторы всех нужных нам товаров. Затем по этим идентификаторам мы сможем извлечь данные о товарах из таблицы product. Возможно, это кажется вам сложным, но на самом деле все это легко можно сделать с помощью одного SQL-запроса. Именно в этом и заключается привлекательность SQL — он позволяет выполнять сложные действия с большими объемами данных с помощью простых запросов.

Соединение двух таблиц — это соединение столбцов (но не строк) этих таблиц в одну общую таблицу. Чтобы таблицы можно было соединить, в них должен быть общий столбец, по которому можно выполнить соединение. Для соединения таблиц используется раздел JOIN операторов SQL. Сейчас вы узнаете, как выполнять соединения, на примере таблиц product и product_category. В результате соединения мы сможем получить список товаров, относящихся к определенной категории.

Предположим, что нам нужно получить все товары с category_id = 5. Запрос, который объединит таблицы product и product_category, будет выглядеть так.

```
SELECT     product_category.product_id,
           product_category.category_id,
           product.name
FROM       product_category
INNER JOIN product
        ON product.product_id = product_category.product_id
ORDER BY   product.product_id;
```

Результат выполнения этого оператора будет выглядеть примерно так (чтобы сэкономить место, ниже приведена только часть строк результата).

Product_id	category_id	name
1	1	Arc d'Triomphe
2	1	Chartres Cathedral
3	1	Coat of Arms
4	1	Gallic Cock
5	1	Marianne
6	1	Alsace
7	1	Apocalypse Tapestry
8	1	Centaur
9	1	Corsica
10	1	Haute Couture
11	1	Iris
12	1	Lorraine
13	1	Mercury
14	1	County of Nice
15	1	Notre Dame
16	1	Paris Peace Conference

Product_id	category_id	name
17	1	Sarah Bernhardt
18	1	Hunt
19	2	Italia
20	2	Torch
…		

Таблица, полученная в результате выполнения оператора, состоит из столбцов двух таблиц, синхронизированных по столбцу product_id, который был выбран в качестве общего для соединения. Как видите, в этой таблице товары, относящиеся к нескольким категориям, упоминаются по одному разу для каждой категории, к которой они относятся. Однако это не будет проблемой для нас — в дальнейшем мы все равно выберем из таблицы товары только одной категории, и повторений не будет.

Заметьте, что в разделе SELECT перед именами столбцов указаны имена таблиц, к которым принадлежат эти столбцы. Это необходимо, если в соединяемых таблицах есть столбцы с одинаковыми именами. Например, в нашем случае в обеих таблицах есть столбец product_id. Если столбец с каким-то именем есть только в одной из таблиц, указывать перед его именем имя таблицы необязательно, однако это упрощает отладку программы.

Чтобы получить, например, только товары категории 5, можно использовать запрос вида

```
SELECT     product.product_id, product.name
FROM       product_category
INNER JOIN product
ON         product.product_id = product_category.product_id
WHERE      product_category.category_id = 5;
```

Вот результаты выполнения этого запроса.

product_id	name
65	Afghan Flower
66	Albania Flower
67	Austria Flower
68	Bulgarian Flower

Еще один момент, о котором стоит упомянуть, — использование *псевдонимов* (alias). Псевдонимы бывают очень полезными (а иногда и необходимыми) в соединениях таблиц, поскольку позволяют задавать новые (обычно более короткие) имена для используемых таблиц. Псевдонимы необходимы при соединении таблицы с самой собой, поскольку результату соединения понадобится новое имя, чтобы его можно было отличить в коде от исходной таблицы. Например, приведенный ниже запрос возвращает те же результаты, что и предыдущий, но с помощью псевдонимов.

```
SELECT p.product_id, p.name
FROM product_category pc
INNER JOIN product p
ON p.product_id = pc.product_id
WHERE pc.category_id = 5;
```

Постраничное отображение списков товаров

Если в ассортименте электронного магазина будет много различных товаров, то нам понадобится возможность постраничного вывода списков этих товаров (так посетителям будет гораздо удобнее просматривать каталог). Количество товаров, отображаемых на одной странице, может быть фиксированным (заданным в коде) или регулироваться посетителем.

Есть три основных способа реализации постраничного отображения.

- *Реализация на уровне данных.* В этом случае база данных передает верхним уровням список товаров постранично.
- *Реализация на уровне логики приложения.* Уровень логики приложения запрашивает полный список товаров, выполняет фильтрацию и передает уровню представления только ту страницу списка, которая должна отображаться.
- *Реализация на уровне представления.* Уровень представления получает полный список товаров и выбирает из него страницу, которую нужно показать посетителю.

Реализация постраничного отображения на уровне логики приложения и уровне представления чревата потерей производительности, особенно при работе с большими объемами данных, поскольку требует дополнительных передач данных из базы данных на уровень представления. Кроме того, при такой реализации в памяти сервера нужно хранить дополнительные данные, и на их хранение будут тратиться ресурсы сервера.

В нашем приложении постраничное отображение будет обеспечивать уровень данных — не только из-за лучшей производительности, но и из-за возможности познакомить вас с некоторыми тонкостями разработки веб-приложений.

Чтобы обеспечить поддержку постраничного отображения на уровне данных, необходимо знать, как составить запрос `SELECT`, возвращающий только нужную часть данных из таблицы. В каждом диалекте SQL такая возможность реализована по-своему. Например, в MySQL для этой цели используется ключевое слово `LIMIT`, за которым следует один или два аргумента. Первый аргумент — это порядковый номер первой нужной записи, а второй — количество нужных записей.

Приведенный ниже запрос приказывает MySQL вернуть строки 15–19 из списка товаров, отсортированного по значениям `product_id` (не забывайте, что нумерация строк начинается с нуля, поэтому строка 15 будет иметь номер 14).

```
SELECT   name
FROM     product
ORDER BY product_id
LIMIT    14, 5;
```

Результаты выполнения такого запроса в нашей базе данных будут следующими.

Name
Notre Dame
Paris Peace Conference
Sarah Bernhardt
Hunt
Italia

Более подробную информацию о ключевом слове LIMIT можно найти в документации к MySQL по адресу `http://dev.mysql.com/doc/refman/5.1/en/select.html`.

Создание хранимых процедур

Теперь вы знаете, как составить нужные нам запросы к базе данных, и мы можем приступить к созданию хранимых процедур, образующих уровень данных нашего приложения. Сначала создадим хранимые процедуры, извлекающие информацию об отделах и категориях.

- `catalog_get_department_details`
- `catalog_get_categories_list`
- `catalog_get_category_details`

Затем мы перейдем к хранимым процедурам, работающим с товарами. Непосредственно запрашивают данные о товарах только четыре процедуры, но мы создадим еще три вспомогательные (`catalog_count_products_in_category`, `catalog_count_products_on_department` и `catalog_count_products_on_catalog`) для реализации постраничного отображения списков товаров.

Итак, вот полный список хранимых процедур, которые мы создадим в этой главе.

- `catalog_count_products_in_category`
- `catalog_get_products_in_category`
- `catalog_count_products_on_department`
- `catalog_get_products_on_department`
- `catalog_count_products_on_catalog`
- `catalog_get_products_on_catalog`
- `catalog_get_product_details`
- `catalog_get_product_locations`

Имя каждой процедуры кратко описывает ее назначение. Это упростит дальнейшую отладку программы — у нас будет меньше возможностей запутаться — и позволит быстрее разобраться в ней тем, кому, возможно, придется с ней работать. Поверьте, через несколько месяцев после написания программы вы уже не будете помнить, что делает каждая функция, хотя и написали ее сами. "Говорящие" имена функций и других элементов программы позволят быстрее вспомнить, как она работает.

В последующих разделах мы поочередно рассмотрим код каждой из хранимых процедур. Мы не будем посвящать созданию каждой хранимой процедуры отдельное упражнение. Создайте эти процедуры с помощью вкладки SQL в phpMyAdmin, задавая в качестве разделителя символы `$$` (как на рис. 5.7).

ПРЕДУПРЕЖДЕНИЕ

Символы разделителя нужно задавать заново для каждой хранимой процедуры, которую вы создаете.
`catalog_get_department_details`

Эта хранимая процедура должна возвращать название и описание отдела, идентификатор которого передан ей в качестве аргумента. Она будет использоваться, когда пользователь будет выбирать отдел в каталоге и сведения об этом отделе нужно будет извлечь из базы данных.

Вот SQL-код, создающий хранимую процедуру `catalog_get_department_details`. Выполните его с помощью phpMyAdmin, но предварительно не забудьте задать символы разделителя `$$`.

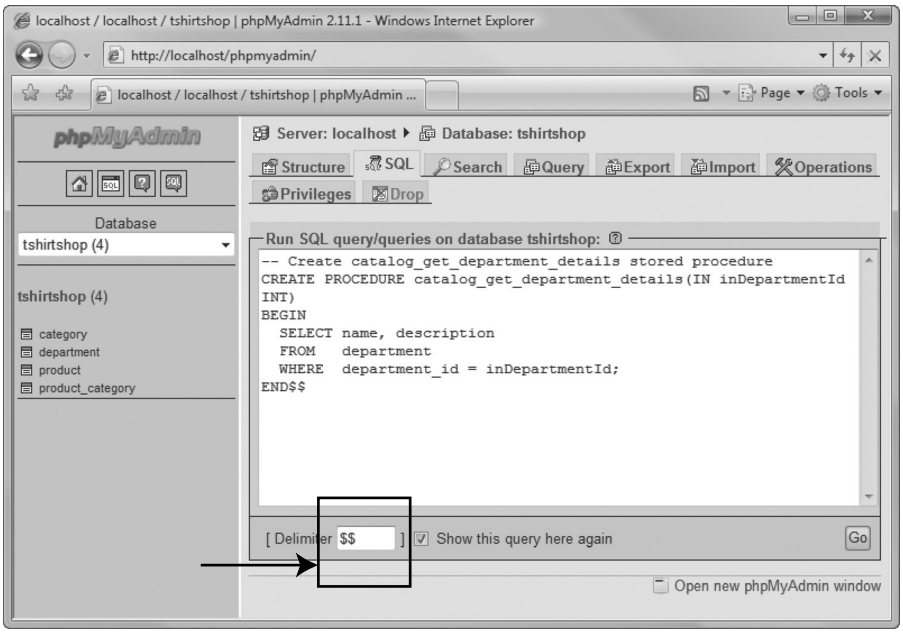


Рис. 5.7. Задание символов разделителя в phpMyAdmin

```
-- Создаем хранимую процедуру catalog_get_department_details
CREATE PROCEDURE catalog_get_department_details(IN inDepartmentId INT)
BEGIN
  SELECT name, description
  FROM   department
  WHERE  department_id = inDepartmentId;
END$$
```

Как видите, сама хранимая процедура напоминает функцию PHP — она получает входные данные в аргументах, выполняет какие-то действия и возвращает результаты. В данном случае у процедуры один входной (`IN`) параметр типа `int` с именем `inDepartmentId`. Обратите внимание на оформление имени параметра — в качестве префикса используется тип параметра и имя образуется из слов, начинающихся с заглавных букв.

MySQL поддерживает не только входные, но и выходные (`OUT`) и универсальные (`INOUT`) параметры. Примеры их использования вы увидите немного позже. Официальная документация по оператору `CREATE PROCEDURE`, создающему хранимые процедуры, доступна по адресу `http://dev.mysql.com/doc/refman/5.1/en/create-procedure.html`.

Хранимая процедура `catalog_get_department_details` возвращает названия и описания всех отделов, значения `department_id` которых совпадают с указанным в параметре `inDepartmentId`. Для выбора нужного отдела служит раздел `WHERE department_id = inDepartmentId`.

catalog_get_categories_list

Когда посетитель выбирает отдел, нам нужно отобразить, относящиеся к нему категории. Эти категории выбирает из базы данных хранимая процедура `catalog_get_categories_list`. Она принимает в качестве аргумента идентификатор отдела,

категории которого нужно извлечь из базы данных. Вот оператор для создания этой процедуры.

```
-- Создаем хранимую процедуру catalog_get_categories_list
CREATE PROCEDURE catalog_get_categories_list(IN inDepartmentId INT)
BEGIN
  SELECT   category_id, name
  FROM     category
  WHERE    department_id = inDepartmentId
  ORDER BY category_id;
END$$
```

catalog_get_category_details

Когда посетитель выбирает отдел, должны отображаться его название и описание. За их получение из базы данных отвечает процедура `catalog_get_category_details`. Чтобы создать ее, выполните следующий оператор, предварительно задав символы разделителя `$$`.

```
-- Создаем хранимую процедуру catalog_get_category_details
CREATE PROCEDURE catalog_get_category_details(IN inCategoryId INT)
BEGIN
  SELECT name, description
  FROM   category
  WHERE  category_id = inCategoryId;
END$$
```

catalog_count_products_in_category

Эта процедура возвращает количество товаров в заданной категории. Нам понадобится знать это количество, когда мы будем разбивать список товаров на страницы для отображения.

Заметьте, что предыдущие хранимые процедуры возвращали строки из таблиц базы данных. Процедура `catalog_count_products_in_category` будет возвращать не строку, а единственное значение — число, являющееся результатом работы функции `COUNT`. Эта функция возвращает количество строк, соответствующих критериям в запросе. Вот оператор для создания процедуры `catalog_count_products_in_category`.

```
-- Создаем хранимую процедуру catalog_count_products_in_category
CREATE PROCEDURE catalog_count_products_in_category(IN inCategoryId INT)
BEGIN
  SELECT     COUNT(*) AS categories_count
  FROM       product p
  INNER JOIN product_category pc
               ON p.product_id = pc.product_id
  WHERE      pc.category_id = inCategoryId;
END$$
```

catalog_get_products_in_category

Эта хранимая процедура возвращает товары, принадлежащие к заданной категории. Чтобы получить этот список товаров, нам нужно соединить таблицы `product` и `product_category` так, как мы это делали ранее в этой главе. Кроме того, данная хранимая процедура генерирует сокращенные версии описаний товаров. Она использует четыре входных параметра.

- inCategoryId — идентификатор категории, товары которой нас интересуют.
- inShortProductDescriptionLength — максимальная допустимая длина описания товара. Если описание длиннее, чем задано в этом параметре, процедура вернет только заданное количество символов, добавив в конце возвращаемой части троеточие. Заметьте, что сокращенная версия описания будет использоваться только на страницах со списками товаров — на страницах с описаниями отдельных товаров эти описания будут отображаться полностью. Значение этой переменной позже будет задаваться в файле config.php, но пока что вам достаточно знать, что эта переменная существует и ее значение равно 150.
- inProductsPerPage — максимальное количество товаров, отображаемых на странице каталога. Если общее количество товаров в категории больше, чем значение этого параметра, мы возвращаем пользователю только страницу с информацией о части товаров. Значение этой переменной тоже будет задаваться в файле config.php, но пока что вам достаточно знать, что для нашего проекта мы зададим это значение равным 4.
- inStartItem — индекс (порядковый номер) первого товара, который должна вернуть процедура. Например, если посетитель хочет просмотреть вторую страницу списка товаров, а на странице помещается четыре товара, то inProductsPerPage будет иметь значение 4, и inStartItem должен иметь то же значение. Получив эти значения, catalog_get_products_in_category вернет товары 5–9 в качестве результата выполнения запроса.

Вот оператор, создающий хранимую процедуру catalog_get_products_in_category.

```
-- Создаем хранимую процедуру catalog_get_products_in_category
CREATE PROCEDURE catalog_get_products_in_category(
  IN inCategoryId INT, IN inShortProductDescriptionLength INT,
  IN inProductsPerPage INT, IN inStartItem INT)
BEGIN
  -- Подготавливаем оператор
  PREPARE statement FROM
    "SELECT    p.product_id, p.name,
               IF(LENGTH(p.description) <= ?,
                  p.description,
                  CONCAT(LEFT(p.description, ?),
                         '...')) AS description,
               p.price, p.discounted_price, p.thumbnail
    FROM       product p
    INNER JOIN product_category pc
                 ON p.product_id = pc.product_id
    WHERE      pc.category_id = ?
    ORDER BY   p.display DESC
    LIMIT      ?, ?";

  -- Определяем параметры запроса
  SET @p1 = inShortProductDescriptionLength;
  SET @p2 = inShortProductDescriptionLength;
  SET @p3 = inCategoryId;
  SET @p4 = inStartItem;
  SET @p5 = inProductsPerPage;

  -- Выполняем оператор
  EXECUTE statement USING @p1, @p2, @p3, @p4, @p5;
END$$
```

Эта процедура — практический пример использования *подготовленного оператора* (prepared statement), представляющего собой шаблон запроса с параметрами, значения которых задаются непосредственно перед выполнением этого запроса.

В данном случае подготовленный оператор нам нужен потому, что он позволяет использовать параметры в разделе `LIMIT` оператора `SELECT`. На момент написания этой книги MySQL 5 не позволял использовать параметры в данном разделе без применения подготовленных операторов.

С помощью оператора `PREPARE` мы создаем подготовленный оператор, извлекающий данные о нужных товарах. Переменные в этом операторе обозначаются знаками вопроса (`?`). Когда подготовленный оператор запускается на выполнение с помощью оператора `EXECUTE`, значения для переменных передаются с помощью параметров оператора `EXECUTE`. Поскольку в нашем подготовленном операторе пять параметров, мы передаем пять параметров оператору `EXECUTE`.

Подготовленные операторы полезны с точки зрения производительности (при необходимости их можно быстро выполнить несколько раз подряд) и безопасности (поскольку типы данных для параметров проверяются на совместимость). Однако в нашем случае мы используем подготовленный оператор, только чтобы применить параметр в разделе `LIMIT` оператора `SELECT`.

Тот же прием мы будем использовать и в других случаях, когда понадобится применить параметр в разделе `LIMIT`.

catalog_count_products_on_department

Эта хранимая процедура подсчитывает товары, которые должны отображаться на странице заданного отдела. Если помните, на странице отдела должны отображаться не все его товары, а только те, в столбце `display` которых указано значение 2 или 3.

Вот оператор, создающий хранимую процедуру `catalog_count_products_on_department`.

```
-- Создаем хранимую процедуру catalog_count_products_on_department
CREATE PROCEDURE catalog_count_products_on_department(IN inDepartmentId
INT)
BEGIN
  SELECT DISTINCT COUNT(*) AS products_on_department_count
  FROM            product p
  INNER JOIN      product_category pc
                    ON p.product_id = pc.product_id
  INNER JOIN      category c
                    ON pc.category_id = c.category_id
  WHERE           (p.display = 2 OR p.display = 3)
                  AND c.department_id = inDepartmentId;
END$$
```

Эта процедура похожа на `catalog_get_products_on_department`, которую мы рассмотрим следующей.

catalog_get_products_on_department

Когда посетитель выбирает определенный отдел, мы должны вывести не только его название, описание и список категорий (код, необходимый для этого, мы написали раньше), но и список рекламируемых товаров этого отдела.

Процедура `catalog_get_products_on_department` возвращает все товары, относящиеся к заданному отделу, со значениями в столбце `display` 2 или 3 (т.е. акционные товары).

В этой процедуре нам понадобится соединять таблицы — это необходимо, чтобы выбрать товары, принадлежащие к определенной категории. При этом соединение будет несколько более сложным, чем в предыдущем примере, поскольку мы не можем напрямую определить, какие товары относятся к выбранному отделу.

Вы знаете, как выбрать из базы данных категории, относящиеся к определенному отделу (это делает хранимая процедура `catalog_get_categories_list`), и как выбрать товары, принадлежащие к определенной категории (за это отвечает процедура `catalog_get_products_in_category`). Объединив эти две процедуры, мы можем составлять списки товаров по отделам. В общей сложности для этого понадобятся два соединения таблиц.

Кроме того, мы будем использовать ключевое слово `DISTINCT`, чтобы избежать повторения товаров в списке (это может произойти, если товар принадлежит к нескольким категориям одного отдела). В этом случае товар будет фигурировать в списке несколько раз, если не использовать в запросе `DISTINCT`.

Вот оператор, создающий хранимую процедуру `catalog_get_products_on_department`.

```
-- Создаем хранимую процедуру catalog_get_products_on_department
CREATE PROCEDURE catalog_get_products_on_department(
  IN inDepartmentId INT, IN inShortProductDescriptionLength INT,
  IN inProductsPerPage INT, IN inStartItem INT)
BEGIN
  PREPARE statement FROM
    "SELECT DISTINCT p.product_id, p.name,
                     IF(LENGTH(p.description) <= ?,
                        p.description,
                        CONCAT(LEFT(p.description, ?),
                               '...')) AS description,
                     p.price, p.discounted_price, p.thumbnail
     FROM            product p
     INNER JOIN      product_category pc
                       ON p.product_id = pc.product_id
     INNER JOIN      category c
                       ON pc.category_id = c.category_id
     WHERE           (p.display = 2 OR p.display = 3)
                     AND c.department_id = ?
     ORDER BY        p.display DESC
     LIMIT           ?, ?";

  SET @p1 = inShortProductDescriptionLength;
  SET @p2 = inShortProductDescriptionLength;
  SET @p3 = inDepartmentId;
  SET @p4 = inStartItem;
  SET @p5 = inProductsPerPage;

  EXECUTE statement USING @p1, @p2, @p3, @p4, @p5;
END$$
```

ПОДСКАЗКА

Если соединения таблиц кажутся вам слишком сложными, попробуйте разобраться в них с помощью диаграммы, показанной на рис. 5.6. Чем больше вы будете работать с базами данных и запросами, тем проще вам будет в них разбираться. Не волнуйтесь — часто случается, что даже профессионалы не могут сразу составить правильный запрос или разобраться в отношениях между таблицами. Главное — не спешите, и со временем у вас все получится.

catalog_count_products_on_catalog

Эта хранимая процедура возвращает количество товаров, которое должно отображаться на главной странице каталога. Это товары, в столбце display которых установлено значение 1 (для отображения только на главной странице) или 3 (для отображения на главной странице и на страницах отделов).

Вот оператор, создающий хранимую процедуру catalog_count_products_on_catalog.

```
-- Создаем хранимую процедуру catalog_count_products_on_catalog
CREATE PROCEDURE catalog_count_products_on_catalog()
BEGIN
  SELECT COUNT(*) AS products_on_catalog_count
  FROM   product
  WHERE  display = 1 OR display = 3;
END$$
```

catalog_get_products_on_catalog

Эта хранимая процедура возвращает непосредственно товары, которые должны отображаться на главной странице каталога. Это товары, в столбце display которых установлено значение 1 (для отображения только на главной странице) или 3 (для отображения на главной странице и на страницах отделов). Описания товаров обрезаются до указанного размера. Механизм постраничного отображения здесь реализован так же, как и в предыдущих хранимых процедурах, возвращавших списки товаров.

Вот оператор, создающий хранимую процедуру catalog_get_products_on_catalog.

```
-- Создаем хранимую процедуру catalog_get_products_on_catalog
CREATE PROCEDURE catalog_get_products_on_catalog(
  IN inShortProductDescriptionLength INT,
  IN inProductsPerPage INT, IN inStartItem INT)
BEGIN
  PREPARE statement FROM
    "SELECT product_id, name,
    IF(LENGTH(description) <= ?,
       description,
       CONCAT(LEFT(description, ?),
              '...')) AS description,
             price, discounted_price, thumbnail
    FROM     product
    WHERE    display = 1 OR display = 3
    ORDER BY display DESC
    LIMIT    ?, ?";

  SET @p1 = inShortProductDescriptionLength;
  SET @p2 = inShortProductDescriptionLength;
  SET @p3 = inStartItem;
  SET @p4 = inProductsPerPage;

  EXECUTE statement USING @p1, @p2, @p3, @p4;
END$$
```

catalog_get_product_details

Эта хранимая процедура возвращает подробную информацию о выбранном товаре и вызывается, когда нужно отобразить эту информацию на странице товара.

Вот оператор, создающий хранимую процедуру catalog_get_product_details.

```
-- Создаем хранимую процедуру catalog_get_product_details
CREATE PROCEDURE catalog_get_product_details(IN inProductId INT)
BEGIN
  SELECT product_id, name, description,
         price, discounted_price, image, image_2
  FROM   product
  WHERE  product_id = inProductId;
END$$
```

catalog_get_product_locations

Эта хранимая процедура возвращает отделы и категории, к которым принадлежит указанный товар. Она понадобится в механизме поиска в каталоге похожих товаров (этот механизм доступен на странице сведений о товаре).

В этой процедуре мы впервые встретимся с подзапросами. В приведенном ниже коде процедуры подзапросы выделены полужирным шрифтом. Как видите, сами по себе подзапросы — это просто запросы, находящиеся внутри другого запроса.

С подзапросами связан один интересный момент. Их часто можно использовать вместо соединений таблиц и получать те же результаты. Выбор между подзапросами и соединениями чаще всего определяется личными предпочтениями разработчиков, но иногда использование подзапросов или соединений диктуется соображениями производительности. Впрочем, это слишком сложный вопрос для этой книги, и, если он вас интересует, советуем обратиться к специальным учебникам по базам данных.

Итак, вот оператор, создающий хранимую процедуру `catalog_get_product_locations`.

```
-- Создаем хранимую процедуру catalog_get_product_locations
CREATE PROCEDURE catalog_get_product_locations(IN inProductId INT)
BEGIN
  SELECT c.category_id, c.name AS category_name, c.department_id,
         (SELECT name
          FROM department
          WHERE department_id = c.department_id) AS department_name
          -- Подзапрос возвращает название отдела заданной категории
  FROM   category c
  WHERE  c.category_id IN
           (SELECT category_id
            FROM   product_category
            WHERE  product_id = inProductId);
            -- Этот подзапрос возвращает идентификаторы категорий,
            -- к которым принадлежит товар
END$$
```

Это последняя из хранимых процедур, которую нам требовалось создать в данной главе. К этому моменту в базе данных есть все данные, которые нужны проекту, и все хранимые процедуры для работы с этими данными. Чтобы убедиться, что вы не пропустили ни одной хранимой процедуры, можете выполнить команду, которая отобразит их список:

```
SHOW PROCEDURE STATUS
```

Создав хранимые процедуры, вы уже реализовали солидную часть каталога товаров. Теперь мы можем переходить к следующему этапу — реализации уровня логики приложения.

Завершение кода уровня логики приложения

В этой главе мы добавим на уровень логики приложения несколько методов, предназначенных для вызова только что созданных нами хранимых процедур. Если помните, мы начали работать над классом `Catalog` (расположенным в файле `business/catalog.php`) в главе 4. Вот список методов, которые мы добавим в него в этой главе.

- `GetDepartmentDetails()`
- `GetCategoriesInDepartment()`
- `GetCategoryDetails()`
- `HowManyPages()`
- `GetProductsInCategory()`
- `GetProductsOnDepartment()`
- `GetProductsOnCatalog()`
- `GetProductDetails()`
- `GetProductLocations()`

Определение констант для списков товаров и активизация сеансов

Прежде чем написать методы уровня логики приложения, давайте обновим содержимое файла `include/config.php`, добавив в него константы `SHORT_PRODUCT_DESCRIPTION_LENGTH` и `PRODUCTS_PER_PAGE`. Первая из них задает длину сокращенного описания товара, а вторая определяет, сколько товаров должно помещаться на странице списка. Значения этих переменных будут передаваться методами уровня логики приложения хранимым процедурам, например `catalog_get_products_on_catalog`.

```
// Порт HTTP-сервера (можно пропустить, если используется порт 80)
define('HTTP_SERVER_PORT', '80');
/* Имя виртуальной директории, в которой располагается сайт, например:
  '/tshirtshop/' если сайт работает из папки
http://www.example.com/tshirtshop/
  '/' если сайт работает из папки http://www.example.com/ */
define('VIRTUAL_LOCATION', '/tshirtshop/');

// Задаем параметры, используемые при генерации списков товаров
define('SHORT_PRODUCT_DESCRIPTION_LENGTH', 150);
define('PRODUCTS_PER_PAGE', 4);
?>
```

Кроме того, нужно внести изменения в файл `index.php`.

```
<?php
// Активизируем сеанс
session_start()

// Включаем вспомогательные файлы
require_once 'include/config.php';
require_once BUSINESS_DIR . 'error_handler.php';
// Задаем обработчик ошибок
ErrorHandler::SetHandler();
...
```

Константа SHORT_PRODUCT_DESCRIPTION_LENGTH указывает, сколько символов из описания товара можно отображать в списке товаров. Полностью описание отображается только на странице товара, которую мы создадим в конце этой главы.

Константа PRODUCTS_PER_PAGE задает максимальное количество товаров, которое можно отображать на одной странице списка. Если товаров, которые нужно отобразить, больше, чем указано в PRODUCTS_PER_PAGE, список разбивается на страницы, между которыми можно перемещаться с помощью навигационных элементов управления.

Кроме того, мы активизируем сессию PHP, чтобы повысить производительность при перемещении между страницами товаров.

ЗАМЕЧАНИЕ

Поддержка сеансов — замечательная возможность PHP, позволяющая сохранять значения различных переменных, которые должны быть разными для разных пользователей, посещающих веб-сайт. Пока посетитель просматривает каталог товаров, переменные сеанса хранятся сервером и привязываются к уникальному идентификатору сеанса (который по умолчанию хранится на машине посетителя в cookie-файле). Объект сеанса для конкретного посетителя хранит переменные на сервере в виде набора пар "имя–значение". В этой главе мы будем использовать сеансы для повышения производительности нашего приложения. При реализации постраничного отображения списков мы сначала будем запрашивать у базы данных общее количество товаров, которое нужно будет включить в список, чтобы показать это число посетителю. Оно будет сохранено в сеансовой переменной, чтобы нам не пришлось несколько раз выполнять один и тот же запрос к базе данных, когда пользователь будет листать страницы списка (эту функциональность мы реализуем в методе HowManyPages(), который напишем позже). Кроме того, в этой главе мы с помощью сессий реализуем кнопки Continue Shopping, позволяющие посетителю вернуться к списку товаров со страниц с информацией об отдельных товарах.

Давайте рассмотрим по очереди каждый метод уровня логики приложения. Все эти методы нужно добавить в класс Catalog, в файле business/catalog.php, который мы начали писать в главе 4.

GetDepartmentDetails()

Этот метод вызывается из уровня представления, когда посетитель щелкает на названии отдела в списке отделов. Он отображает название отдела и его описание. Уровень представления передает идентификатор выбранного посетителем отдела, и метод должен вернуть название и описание этого отдела.

Требуемые данные извлекает из базы данных хранимая процедура catalog_get_department_details, которую мы написали ранее. Как и планировалось, в данном случае уровень логики приложения играет роль посредника между уровнем данных и уровнем представления.

```
// Возвращает подробные сведения о выбранном отделе
public static function GetDepartmentDetails($departmentId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_department_details(:department_id)';

  // Создаем массив параметров
  $params = array (':department_id' => $departmentId);

  // Выполняем запрос и возвращаем результат
  return DatabaseHandler::GetRow($sql, $params);
}
```

GetCategoriesInDepartment()

Этот метод предназначен для получения из базы данных списка категорий, относящихся к заданному отделу. Добавьте его код в класс Catalog.

```
// Возвращает список категорий, относящихся к выбранному отделу
public static function GetCategoriesInDepartment($departmentId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_categories_list(:department_id)';
  // Создаем массив параметров
  $params = array (':department_id' => $departmentId);
  // Выполняем запрос и возвращаем результат
  return DatabaseHandler::GetAll($sql, $params);
}
```

GetCategoryDetails()

Этот метод вызывается из уровня представления, когда посетитель щелкает на названии категории в списке категорий, и должен возвращать извлеченное из базы данных описание категории. Уровень представления передает идентификатор выбранной посетителем категории, и метод должен вернуть название и описание этой категории. Добавьте код этого метода в класс Catalog.

```
// Возвращает название и описание выбранной категории
public static function GetCategoryDetails($categoryId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_category_details(:category_id)';
  // Создаем массив параметров
  $params = array (':category_id' => $categoryId);
  // Выполняем запрос и возвращаем результат
  return DatabaseHandler::GetRow($sql, $params);
}
```

HowManyPages()

Как вам известно, наш каталог будет отображать фиксированное количество товаров на каждой странице. Если необходимо отобразить больше товаров, чем помещается на одной странице, мы отображаем навигационные элементы управления, которые позволят посетителю перемещаться между страницами списка товаров. Эти элементы управления видны на рис. 4.2.

Отображая навигационные элементы управления, нужно знать, сколько всего страниц понадобится для отображения списка товаров. Именно для этого и предназначен вспомогательный метод HowManyPages().

Этот метод получает в качестве аргумента запрос SELECT, запрашивающий общее количество товаров ($countSql), и возвращает количество страниц, которое нам понадобится. Он просто делит общее количество товаров на количество, которое можно отобразить на одной странице (если помните, оно задается с помощью константы PRODUCTS_PER_PAGE в файле include/config.php).

Чтобы повысить производительность, сохраним вычисленное количество страниц в сеансовой переменной. Это позволит обращаться к базе данных только один раз, когда нужно считать из нее общее количество товаров в списке.

Методом HowManyPages() будут пользоваться другие методы уровня логики приложения (GetProductsInCategory(), GetProductsOnDepartment() и GetProductsOnCatalog()), которые мы рассмотрим позже.

Вот код метода HowManyPages(), который нужно добавить в класс Catalog().

```
/* Вычисляет, сколько страниц понадобится для отображения всех товаров -
количество товаров возвращает запрос $countSql */
private static function HowManyPages($countSql, $countSqlParams)
{
  // Создаем хеш для SQL-запроса
  $queryHashCode = md5($countSql . var_export($countSqlParams, true));

  // Проверяем, есть ли результаты выполнения запроса в кэше
  if (isset ($_SESSION['last_count_hash']) &&
      isset ($_SESSION['how_many_pages']) &&
      $_SESSION['last_count_hash'] === $queryHashCode)
  {
    // Извлекаем кэшированное значение
    $how_many_pages = $_SESSION['how_many_pages'];
  }
  else
  {
    // Выполняем запрос
    $items_count = DatabaseHandler::GetOne($countSql, $countSqlParams);

    // Вычисляем количество страниц
    $how_many_pages = ceil($items_count / PRODUCTS_PER_PAGE);

    // Сохраняем данные в сеансовых переменных
    $_SESSION['last_count_hash'] = $queryHashCode;
    $_SESSION['how_many_pages'] = $how_many_pages;
  }

  // Возвращаем количество страниц
  return $how_many_pages;
}
```

Давайте разберемся, как этот метод делает свое дело.

Метод объявлен как `private`, потому что он не должен быть доступен из других классов, — это вспомогательный метод для класса `Catalog`.

Итак, сначала метод проверяет, был ли предыдущий выполненный запрос тем запросом, который используется в этом методе. Если был, метод возвращает результат, кэшированный после его предыдущего вызова. Этот маленький фокус повышает производительность, если посетитель просто перемещается между страницами списка товаров (таким образом мы избегаем лишних запросов к базе данных).

```
// Создаем хеш для SQL-запроса
$queryHashCode = md5($countSql . var_export($countSqlParams, true));

// Проверяем, есть ли результаты выполнения запроса в кэше
if (isset ($_SESSION['last_count_hash']) &&
    isset ($_SESSION['how_many_pages']) &&
    $_SESSION['last_count_hash'] === $queryHashCode)
{
  // Извлекаем кэшированное значение
  $how_many_pages = $_SESSION['how_many_pages'];
}
```

Количество страниц, связанное с текущим запросом, хранится в сеансе текущего пользователя в переменной `how_many_pages`. Если условия не выполняются, т.е. результатов запроса в кэше нет, мы выполняем запрос и сохраняем его результаты в сеансовой переменной.

```
else
{
  // Выполняем запрос
```

```
  $items_count = DatabaseHandler::GetOne($countSql, $countSqlParams);

  // Вычисляем количество страниц
  $how_many_pages = ceil($items_count / PRODUCTS_PER_PAGE);

  // Сохраняем данные в сеансовых переменных
  $_SESSION['last_count_hash'] = $queryHashCode;
  $_SESSION['how_many_pages'] = $how_many_pages;
}
```

В конце концов метод возвращает количество страниц независимо от того, было ли это значение прочитано из кэша или получено с помощью запроса к базе данных.

```
  // Возвращаем количество страниц
  return $how_many_pages;
}
```

GetProductsInCategory()

Этот метод возвращает список товаров, принадлежащих к заданной категории. Добавьте его код в класс `Catalog` в файле `business/catalog.php`.

```
// Возвращает список товаров, принадлежащих к заданной категории
public static function GetProductsInCategory(
                        $categoryId, $pageNo, &$rHowManyPages)
{
  // Запрос, возвращающий количество товаров в категории
  $sql = 'CALL catalog_count_products_in_category(:category_id)';
  // Создаем массив параметров
  $params = array (':category_id' => $categoryId);

  // Определяем, сколько страниц понадобится для отображения товаров
  $rHowManyPages = Catalog::HowManyPages($sql, $params);
  // Определяем, какой товар будет первым
  $start_item = ($pageNo - 1) * PRODUCTS_PER_PAGE;

  // Получаем список товаров
  $sql = 'CALL catalog_get_products_in_category(
                :category_id, :short_product_description_length,
                :products_per_page, :start_item)';

  // Создаем массив параметров
  $params = array (
    ':category_id' => $categoryId,
    ':short_product_description_length' =>
      SHORT_PRODUCT_DESCRIPTION_LENGTH,
    ':products_per_page' => PRODUCTS_PER_PAGE,
    ':start_item' => $start_item);

  // Выполняем запрос и возвращаем результат
  return DatabaseHandler::GetAll($sql, $params);
}
```

Две основные задачи этого метода таковы.

- Вычислить необходимое количество страниц и вернуть это значение через параметр `&$howManyPages`. Чтобы вычислить его, мы используем ранее созданный метод `HowManyPages()`. SQL-запрос, с помощью которого мы получаем общее количество товаров, вызывает хранимую процедуру `catalog_count_products_in_category`, которую мы создали ранее в этой главе.
- Вернуть список товаров, принадлежащих к заданной категории.

ЗАМЕЧАНИЕ

Символ "амперсанд" (&) перед именем параметра метода означает, что этот параметр передается по ссылке. Если метод позже изменит значение этого параметра, новое значение станет известно и вызывающему методу. Параметры, передаваемые по ссылке, — часто используемая альтернатива возвращаемым значениям, особенно удобная, если нужно вернуть из метода несколько значений сразу. Например, метод `CreateSubpageQuery()` возвращает текст SQL-запроса через возвращаемое значение и количество подстраниц — через передаваемый по ссылке параметр `&$HowManyPages`.

GetProductsOnDepartment()

Этот метод возвращает список рекламируемых товаров заданного отдела. Товары должны отображаться на странице отдела. Поместите код этого метода в класс `Catalog`.

```
// Возвращает список товаров для страницы отдела
public static function GetProductsOnDepartment(
                          $departmentId, $pageNo, &$rHowManyPages)
{
  // Запрос, возвращающий количество товаров для страницы отдела
  $sql = 'CALL catalog_count_products_on_department(:department_id)';
  // Создаем массив параметров
  $params = array (':department_id' => $departmentId);

  // Определяем, сколько страниц понадобится для отображения товаров
  $rHowManyPages = Catalog::HowManyPages($sql, $params);
  // Определяем, какой товар будет первым
  $start_item = ($pageNo - 1) * PRODUCTS_PER_PAGE;

  // Получаем список товаров
  $sql = 'CALL catalog_get_products_on_department(
                :department_id, :short_product_description_length,
                :products_per_page, :start_item)';

  // Создаем массив параметров
  $params = array (
    ':department_id' => $departmentId,
    ':short_product_description_length' =>
      SHORT_PRODUCT_DESCRIPTION_LENGTH,
    ':products_per_page' => PRODUCTS_PER_PAGE,
    ':start_item' => $start_item);

  // Выполняем запрос и возвращаем результат
  return DatabaseHandler::GetAll($sql, $params);
}
```

GetProductsOnCatalog()

Этот метод возвращает список товаров, которые должны отображаться на главной странице каталога. Вот его код.

```
// Возвращает список товаров для главной страницы каталога
public static function GetProductsOnCatalog($pageNo, &$rHowManyPages)
{
  // Запрос, возвращающий количество товаров для главной страницы каталога
  $sql = 'CALL catalog_count_products_on_catalog()';

  // Определяем, сколько страниц понадобится для отображения товаров
  $rHowManyPages = Catalog::HowManyPages($sql, null);
```

```
  // Определяем, какой товар будет первым
  $start_item = ($pageNo - 1) * PRODUCTS_PER_PAGE;

  // Получаем список товаров
  $sql = 'CALL catalog_get_products_on_catalog(
                :short_product_description_length,
                :products_per_page, :start_item)';

  // Создаем массив параметров
  $params = array (
    ':short_product_description_length' =>
      SHORT_PRODUCT_DESCRIPTION_LENGTH,
    ':products_per_page' => PRODUCTS_PER_PAGE,
    ':start_item' => $start_item);

  // Выполняем запрос и возвращаем результат
  return DatabaseHandler::GetAll($sql, $params);
}
```

GetProductDetails()

Это метод, возвращающий описание товара для страницы с информацией об отдельном товаре. Вот его код.

```
// Возвращает подробную информацию о товаре
public static function GetProductDetails($productId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_product_details(:product_id)';

  // Создаем массив параметров
  $params = array (':product_id' => $productId);

  // Выполняем запрос и возвращаем результат
  return DatabaseHandler::GetRow($sql, $params);
}
```

GetProductLocations()

И наконец, метод `GetProductLocations()`, вызывающий хранимую процедуру `catalog_get_product_locations`, возвращает категории и отделы, к которым принадлежит товар.

```
// Возвращает список отделов и категорий, к которым принадлежит товар
public static function GetProductLocations($productId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_product_locations(:product_id)';

  // Создаем массив параметров
  $params = array (':product_id' => $productId);

  // Выполняем запрос и возвращаем результат
  return DatabaseHandler::GetAll($sql, $params);
}
```

Реализация уровня представления

К данному моменту мы сделали все, что хотели сделать в этой главе с уровнем данных и уровнем логики приложения. Все, что осталось, — задействовать новую функциональность на уровне представления. В этом, последнем, разделе данной

главы мы создадим несколько новых шаблонов Smarty и интегрируем их в уже существующий проект.

Запустите приложение TShirtShop (или откройте в браузере страницу `http://localhost/tshirtshop/`) и снова попробуйте щелкнуть на названии отдела в списке. Главная страница (`index.php`) загрузится заново, но на этот раз в конце URL появится строка запроса, например

```
http://localhost/tshirtshop/index.php?DepartmentId=1
```

Параметр `DepartmentId` указывает, какой отдел выбрал посетитель, и позволяет приложению извлечь из базы данных информацию, относящуюся к этому отделу, — его название, описание, список категорий, список рекламируемых товаров и т.д. В последующих разделах мы создадим элементы управления, отображающие эту информацию.

Отображение информации об отделе и категориях

Компонентный шаблон, отвечающий за отображение содержимого отдела, называется `department`, и мы создадим его в следующем упражнении. Сначала мы создадим этот компонентный шаблон, а затем модифицируем другой, `store_front`, чтобы он загружал шаблон `department`, если в строке запроса присутствует параметр `DepartmentId`. После этого упражнения щелчок на имени отдела в списке должен давать результат, подобный представленному на рис. 5.8.

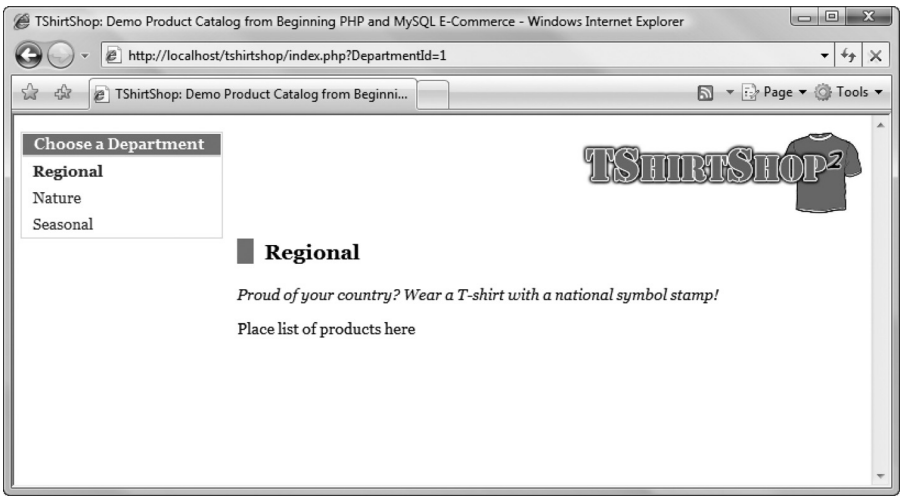


Рис. 5.8. Информация о выбранном отделе

Упражнение: отображение информации об отделе

1. Добавьте в файл стилей `tshirtshop.css` из папки `styles` два следующих стиля. Они понадобятся для отображения названия отдела и его описания.

```
.title {
  border-left: 15px solid #0590C7;
  padding-left: 10px;
}
.description {
```

```
  font-style: italic;
}
```

2. Создайте файл шаблона с именем `blank.tpl` в папке `presentation/templates` и поместите в этот файл строку

```
{* Smarty blank page *}
```

Это файл пустого шаблона Smarty, в котором есть только комментарий. Он понадобится нам немного позже. Обязательно поместите в файл приведенную выше строку комментария. Если вы этого не сделаете, то получите сообщение об ошибке при попытке использовать шаблон.

3. Создайте файл шаблона с именем `department.tpl` в папке `presentation/templates` и поместите в него следующий код.

```
{* department.tpl *}
{load_presentation_object filename="department" assign="obj"}
<h1 class="title">{$obj->mName}</h1>
<p class="description">{$obj->mDescription}</p>
Place list of products here
```

Два элемента объекта представления, `$obj->mName` и `$obj->mDescription`, содержат название и описание выбранного отдела. Объект представления `Department` создается подключаемым модулем `presentation/smarty_plugins/function.load_presentation_object.php`.

4. Создайте объект представления файл `Department` в файле `department.tpl`. Создайте этот файл в папке `presentation` и поместите в него такой код.

```
<?php
// Занимается извлечением сведений об отделе
class Department
{
  // Public-переменные для шаблонов Smarty
  public $mName;
  public $mDescription;

  // Private-элементы
  private $_mDepartmentId;
  private $_mCategoryId;

  // Конструктор класса
  public function __construct()
  {
    // В строке запроса должен присутствовать параметр DepartmentId
    if (isset ($_GET['DepartmentId']))
      $this->_mDepartmentId = (int)$_GET['DepartmentId'];
    else
      trigger_error('DepartmentId not set');

    /* Если CategoryId есть в строке запроса, мы сохраняем его значение
      (преобразуя его в integer для защиты от некорректных значений) */
    if (isset ($_GET['CategoryId']))
      $this->_mCategoryId = (int)$_GET['CategoryId'];
  }

  public function init()
  {
    // Если посещаем отдел...
    $department_details =
      Catalog::GetDepartmentDetails($this->_mDepartmentId);
```

```
      $this->mName = $department_details['name'];
      $this->mDescription = $department_details['description'];

      // Если посещаем категорию...
      if (isset ($this->_mCategoryId))
      {
        $category_details =
          Catalog::GetCategoryDetails($this->_mCategoryId);
        $this->mName = $this->mName . ' &raquo; ' .
          $category_details['name'];
        $this->mDescription = $category_details['description'];
      }
    }
}
?>
```

5. Теперь давайте модифицируем файлы `presentation/templates/store_front.tpl` и `presentation/store_front.php`, чтобы только что созданный нами компонентный шаблон загружался, когда в строке запроса присутствует параметр `DepatmentId`. Если посетитель просматривает отдел, мы присваиваем компонентный шаблон элементу `$mContentsCell` объекта представления `StoreFront`, чтобы на странице отобразились рекламируемые товары отдела.

Вставьте в файл `presentation/store_front.php` строки, выделенные полужирным шрифтом в следующем листинге.

```
<?php
class StoreFront
{
  public $mSiteUrl;
  // Определяем файл шаблона для содержимого страницы
  public $mContentsCell = 'blank.tpl';

  // Конструктор класса
  public function __construct()
  {
    $this->mSiteUrl = Link::Build('');
  }

  // Инициализируем объект представления
  public function init()
  {
    // Загружаем подробные сведения об отделе на страницу отдела
    if (isset ($_GET['DepartmentId']))
    {
      $this->mContentsCell = 'department.tpl';
    }
  }
}
?>
```

6. Откройте файл `presentation/templates/store_front.tpl` и замените в нем текст `Place contents here` строкой

```
{include file=$obj->mContentsCell}
```

7. Загрузите свой веб-сайт в браузере и выберите один из отделов в списке, чтобы убедиться, что все работает как полагается.

ВНИМАНИЕ

В некоторых версиях PHP 5 есть ошибка, вследствие которой при загрузке сайта отображается сообщение об ошибке наподобие `General error: 2014 Cannot execute queries while other unbuffered queries are active`. Ее описание доступно по адресу `http://bugs.php.net/bug.php?id=39858`. Как справиться с этой проблемой, можно прочесть на сайте книги, на странице, посвященной ошибкам.

Как это работает: компонентный шаблон отдела

Поздравляем! Если список отделов в вашем приложении работает корректно, значит, вы успешно одолели самую сложную часть этой книги. Но прежде чем двигаться дальше, убедитесь, что вы четко понимаете, как работает весь код в приложении.

После того как мы создали уровень данных и уровень логики приложения, реализация уровня представления была несложной задачей. Добавив новые стили CSS и создав пустой файл шаблона, мы создали шаблон Smarty `department.tpl`, в котором содержится HTML-разметка для отображения информации об отделе. Этот шаблон загружается в ячейку содержимого, ниже заголовка, в шаблоне `store_front.tpl`.

```
<div id="yui-main">
  <div class="yui-b">
    <div id="header" class="yui-g">
      <a href="{$obj->mSiteUrl}">
        <img src="{$obj->mSiteUrl}images/tshirtshop.png"
             alt="tshirtshop logo" />
      </a>
    </div>
    <div id="contents" class="yui-g">
      {include file=$obj->mContentsCell}
    </div>
  </div>
</div>
```

Поле `$mContentsCell` объекта представления заполняется в файле `store_front.php` в зависимости от параметров из строки запроса. Если в строке запроса есть параметр `DepartmentId`, это поле заполняет шаблон `department.tpl`. В противном случае (например, если открыта первая страница сайта), используется шаблон `blank.tpl`, поскольку мы еще не реализовали шаблон ячейки содержимого первой страницы (вскоре мы его создадим).

Вот код из файла `store_front.php`, присваивающий значение полю `$mContentsCell` при выборе отдела.

```
// Инициализируем объект представления
public function init()
{
  // Загружаем подробные сведения об отделе на страницу отдела
  if (isset ($_GET['DepartmentId']))
  {
    $this->mContentsCell = 'department.tpl';
  }
}
```

Первое, на что стоит обратить внимание, — как `department.tpl` загружает объект представления `Department` с помощью подключаемой функции Smarty `load_presentation_object`:

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

Это позволяет нам обращаться к экземпляру класса `Department` (который мы рассмотрим позже) и его `public`-элементам (`$mName` и `$mDescription`) из файла шаблона (`department.tpl`) следующим образом:

```
<h1 class="title">{$obj->mName}</h1>
<p class="description">{$obj->mDescription}</p>
Place list of products here
```

Следующее, в чем нужно разобраться, — как объект представления из файла `department.php` получает название и описание отдела. В этом файле содержится класс `Department`, в котором есть два `public`-элемента, доступных из шаблона Smarty (в них хранятся название и описание). Класс должен присвоить этим элементам соответствующие значения, которые используются для генерации веб-страницы:

```
// Занимается извлечением сведений об отделе
class Department
{
  // Public-переменные для шаблонов Smarty
  public $mName;
  public $mDescription;
```

Кроме того, в классе `Department` есть два `private`-элемента, используемых для внутренних задач. Это элементы `$_mDepartmentId` и `$_mCategoryId`, в которых хранятся значения параметров `DepartmentId` и `CategoryId` из строки запроса.

```
// Private-элементы
private $_mDepartmentId;
private $_mCategoryId;
```

Следующим в коде указан конструктор. В любом объектно-ориентированном языке конструктор выполняется при создании объекта (экземпляра) класса, и в коде конструктора выполняется инициализация создаваемого объекта. В нашем случае конструктор класса `Department` считывает значения параметров `DepartmentId` и `CategoryId` из строки запроса и помещает эти значения в элементы `$_mDepartmentId` и `$_mCategoryId`. Это необходимо, потому что, если в строке запроса есть параметр `CategoryId`, значит, вместо названия и описания отдела мы должны отображать название и описание категории.

```
// Конструктор класса
public function __construct()
{
  // В строке запроса должен присутствовать параметр DepartmentId
  if (isset ($_GET['DepartmentId']))
    $this->_mDepartmentId = (int)$_GET['DepartmentId'];
  else
    trigger_error('DepartmentId not set');

  /* Если CategoryId есть в строке запроса, мы сохраняем его значение
    (преобразуя его в integer для защиты от некорректных значений) */
  if (isset ($_GET['CategoryId']))
    $this->_mCategoryId = (int)$_GET['CategoryId'];
}
```

Большая часть функциональности класса содержится в методе `init()`. В нашем проекте этот метод всегда выполняется сразу после конструктора, поскольку мы вызываем его сразу после создания объекта класса — в подключаемой функции Smarty `load_presentation_object` (как вы знаете из главы 4, эта подключаемая функция используется всеми шаблонами Smarty для загрузки соответствующих объектов представления).

Метод `init()` заполняет элементы `$mName` и `$mDescription` значениями, полученными от уровня логики приложения. Метод `GetDepartmentDetails()` класса `Catalog` из уровня логики приложения получает из уровня данных информацию об отделе; при необходимости для получения информации о категории вызывается метод `GetCategoryDetails()` того же класса `Catalog` (сведения об отделе нужно извлекать в любом случае, потому что на странице со сведениями о категории должно отображаться название отдела).

```
public function init()
{
  // Если посещаем отдел...
  $department_details =
    Catalog::GetDepartmentDetails($this->_mDepartmentId);
```

```
  $this->mName = $department_details['name'];
  $this->mDescription = $department_details['description'];

  // Если посещаем категорию...
  if (isset ($this->_mCategoryId))
  {
    $category_details =
      Catalog::GetCategoryDetails($this->_mCategoryId);
    $this->mName = $this->mName . ' &raquo; ' .
      $category_details['name'];
    $this->mDescription = $category_details['description'];
  }
}
```

Отображение списка категорий

Когда посетитель выбирает отдел из списка, отображается список категорий этого отдела. Чтобы список категорий мог отображаться, нужно создать шаблон Smarty `categories_list`. Он похож на ранее созданный `department_list` и содержит раздел, перебирающий элементы списка категорий (с названиями и идентификаторами категорий). В этом разделе будут содержаться ссылки на `index.php`, но они будут создавать запросы с параметром `CategoryId`, сообщая, на какой категории щелкнул посетитель, например

`http://localhost/tshirtshop/index.php?DepartmentId=1&CategoryId=2`

Действия, которые вы должны выполнить в следующем упражнении, очень похожи на аналогичные действия при создании шаблона `departments_list` (созданного в конце главы 4).

Упражнение: создание компонентного шаблона `categories_list`

1. В файл `categories_list.tpl` в папке `presentation/templates` введите следующий код.

```
{* categories_list.tpl *}
{load_presentation_object filename="categories_list" assign="obj"}
{* Start categories list *}
<div class="box">
  <p class="box-title">Choose a Category</p>
  <ul>
  {section name=i loop=$obj->mCategories}
    {assign var=selected value=""}
    {if ($obj->mSelectedCategory == $obj->mCategories[i].category_id)}
      {assign var=selected value="class=\"selected\""}
    {/if}
    <li>
      <a {$selected} href="{$obj->mCategories[i].link_to_category}">
        {$obj->mCategories[i].name}
      </a>
    </li>
  {/section}
  </ul>
</div>
{* End categories list *}
```

2. Создайте в папке `presentation` файл `categories_list.php` и поместите в него следующий код.

```
<?php
// Отвечает за списки категорий
class CategoriesList
{
  // Public-переменные для шаблона smarty
  public $mSelectedCategory = 0;
  public $mSelectedDepartment = 0;
  public $mCategories;

  // Конструктор считывает параметр из строки запроса
  public function __construct()
  {
    if (isset ($_GET['DepartmentId']))
      $this->mSelectedDepartment = (int)$_GET['DepartmentId'];
    else
      trigger_error('DepartmentId not set');

    if (isset ($_GET['CategoryId']))
      $this->mSelectedCategory = (int)$_GET['CategoryId'];
  }

  public function init()
  {
    $this->mCategories =
      Catalog::GetCategoriesInDepartment($this->mSelectedDepartment);

    // Генерируем ссылки для страниц категорий
    for ($i = 0; $i < count($this->mCategories); $i++)
      $this->mCategories[$i]['link_to_category'] =
        Link::ToCategory($this->mSelectedDepartment,
                         $this->mCategories[$i]['category_id']);
  }
}
?>
```

3. Откройте файл `presentation/link.php` и добавьте в класс `Link` метод `ToCategory()`, приведенный ниже:

```
public static function ToCategory($departmentId, $categoryId)
{
  $link = 'index.php?DepartmentId=' . $departmentId .
          '&CategoryId=' . $categoryId;

  return self::Build($link);self::Build($link);
}
```

4. Вставьте в файл `presentation/store_front.tpl` строки, выделенные полужирным шрифтом в приведенном ниже листинге.

```
<?php
class StoreFront
{
  public $mSiteUrl;
  // Определяем файл шаблона для содержимого страницы
  public $mContentsCell = 'blank.tpl';
  // Определяем файл шаблона для ячеек категорий
  public $mCategoriesCell = 'blank.tpl';

  // Конструктор класса
```

```
  public function __construct()
  {
    $this->mSiteUrl = Link::Build('');
  }

  // Инициализируем объект представления
  public function init()
  {
    // Загружаем подробные сведения об отделе на страницу отдела
    if (isset ($_GET['DepartmentId']))
    {
      $this->mContentsCell = 'department.tpl';
      $this->mCategoriesCell = 'categories_list.tpl';
    }
  }
}
?>
```

5. Включите компонентный шаблон `categories_list` в файл `presentation/templates/store_front.tpl` сразу после списка отделов. Заметьте, что мы включаем `$Obj->mCategoriesCell`, а не `categories_list.tpl`, потому что список категорий должен отображаться, только когда выбран отдел. Если отдел не выбран, элементу `$Obj->mCategoriesCell` присваивается файл `blank.tpl`.

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

6. Откройте TShirtShop в браузере. Когда страница загрузится, щелкните на одном из отделов в списке. При этом должен отобразиться список категорий. Выбор категории приводит к отображению информации о ней, как показано на рис. 5.9.



Рис. 5.9. Отображение сведений о категории Animal

Как это работает: компонентный шаблон `categories_list`

Шаблон `categories_list` работает почти так же, как `department_list`, созданный нами в предыдущей главе. Класс `CategoriesList` (находящийся в файле объекта представления `presentation/categories_list.php`) содержит три `public`-элемента, доступных из файла шаблона (`categories_list.tpl`).

```
// Public-переменные для шаблона smarty
public $mSelectedCategory = 0;
public $mSelectedDepartment = 0;
public $mCategories;
```

В элементе `$mSelectedCategory` хранится идентификатор выбранной категории — для вывода этой категории в списке используется особый стиль. Это верно и для `$mSelectedDepartment`. Элемент `$mCategories` содержит список категорий, получаемый при обращении к уровню приложения.

Ссылки в списке генерируются методом `Link::ToCategory()` — это гарантирует их правильное оформление.

Отображение списков товаров

Независимо от того, открыта главная страница сайта или страница категории, вместо текста `Place list of products here` должны отображаться краткие сведения о каких-то товарах. Сейчас мы создадим компонентный шаблон `products_list`, который будет отвечать за отображение списков с информацией о товарах. Если товаров, которые нужно отобразить, больше, чем может поместиться на странице, будут отображены навигационные ссылки (рис. 5.10).

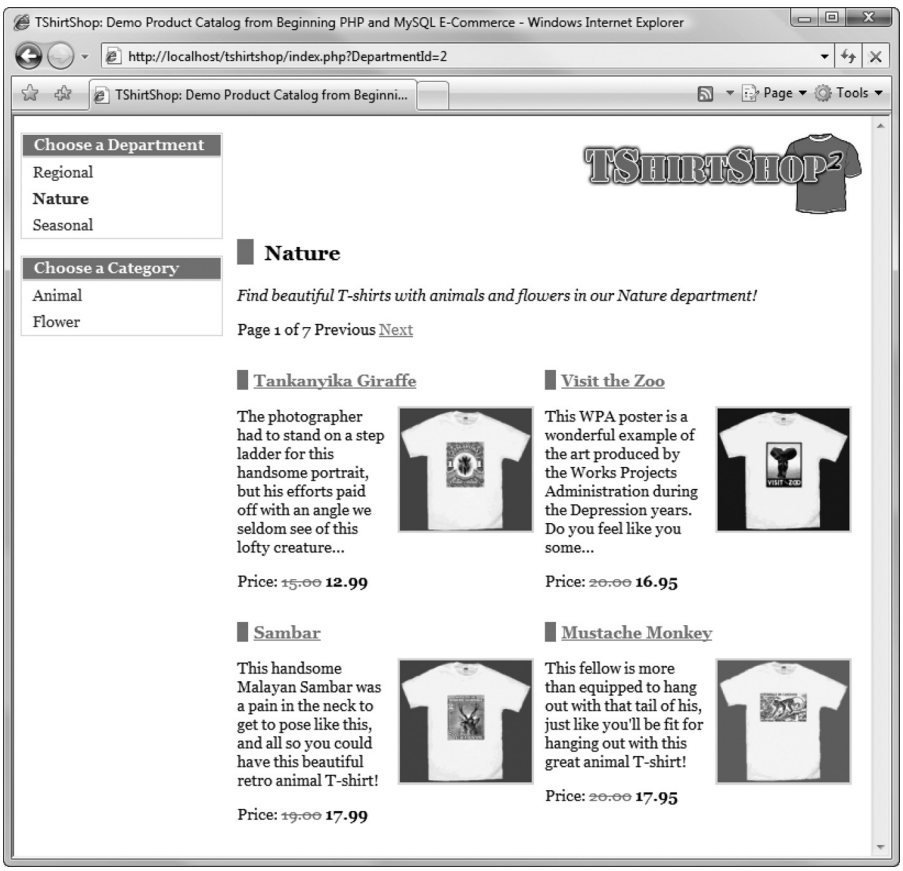


Рис. 5.10. Результаты применения компонентного шаблона products_list

Этот компонентный шаблон будет использоваться на нашем сайте в нескольких местах. На главной странице он будет обеспечивать отображение товаров, в базе данных в столбце `display` которых задано значение `1` или `3`. Когда посетитель выберет отдел, шаблон `products_list` будет отвечать за отображение рекламируемых товаров из этого отдела. И наконец, при выборе категории этот шаблон просто будет отображать товары из этой категории. Наша база данных реализована таким образом, что можно отображать товар на главной странице или на странице отдела, но не на обеих этих страницах одновременно. Если товар относится к нескольким отделам, он будет отображаться на странице каждого из этих отделов.

Компонентный шаблон определяет, какие товары нужно отображать, анализируя строку запроса. Если в строке запроса присутствуют параметры `DepartmentId` и `CategoryId`, значит, нужно отображать товары из выбранной категории. Если присутствует только параметр `DepartmentId`, значит, посетитель просматривает страницу отдела, и нужно отобразить рекламируемые товары этого отдела. Если и параметра `DepartmentId` в запросе нет, значит, посетитель просматривает главную страницу, и нужно отображать только предназначенные для нее товары.

Чтобы интегрировать шаблон `products_list` с главной страницей, понадобится создать дополнительный файл шаблона (`first_page_contents.tpl`) (мы создадим его немного позже). После того как в следующем упражнении мы создадим шаблон `products_list`, вы сможете просматривать товары по отделам и категориям. Затем мы займемся отображением товаров на главной странице.

Упражнение: создание компонентного шаблона `products_list`

1. В архиве исходных кодов для этой книги, который можно загрузить с ее веб-сайта, найдите папку `product_images` и скопируйте ее в папку проекта TShirtShop.
2. Добавьте в файл `tshirtshop.css`, находящийся в папке `styles`, следующие стили.

```
.product-list tbody tr td {
  border: none;
  padding: 0;
  width: 50%;
}

.product-list tbody tr td p img {
  border: 2px solid #c6e1ec;
  float: right;
  margin: 0 10px;
  vertical-align: top;
}

.product-title {
  border-left: 10px solid #0590C7;
  padding-left: 5px;
}

.section {
  display: block;
}

.price {
  font-weight: bold;
}

.old-price {
```

```
  color: #ff0000;
  font-weight: normal;
  text-decoration: line-through;
}
```

3. В папке `presentation/templates` создайте новый файл дизайн-шаблона Smarty с именем `products_list.tpl`, и поместите в этот файл следующий код:

```
{* products_list.tpl *}
{load_presentation_object filename="products_list" assign="obj"}
{if $obj->mrTotalPages > 1}
<p>
  Page {$obj->mPage} of {$obj->mrTotalPages}
  {if $obj->mLinkToPreviousPage}
  <a href="{$obj->mLinkToPreviousPage}">Previous</a>
  {else}
  Previous
  {/if}
  {if $obj->mLinkToNextPage}
  <a href="{$obj->mLinkToNextPage}">Next</a>
  {else}
  Next
  {/if}
</p>
{/if}
{if $obj->mProducts}
<table class="product-list" border="0">
  <tbody>
    {section name=k loop=$obj->mProducts}
      {if $smarty.section.k.index % 2 == 0}
      <tr>
      {/if}
        <td valign="top">
          <h3 class="product-title">
            <a href="{$obj->mProducts[k].link_to_product}">
              {$obj->mProducts[k].name}
            </a>
          </h3>
          <p>
            {if $obj->mProducts[k].thumbnail neq ""}
            <a href="{$obj->mProducts[k].link_to_product}">
              <img src="{$obj->mProducts[k].thumbnail}"
                alt="{$obj->mProducts[k].name}" />
            </a>
            {/if}
            {$obj->mProducts[k].description}
          </p>
        <p class="section">
          Price:
          {if $obj->mProducts[k].discounted_price != 0}
            <span class="old-price">{$obj->mProducts[k].price}</span>
            <span class="price">{$obj->mProducts[k].discounted_price}</span>
          {else}
            <span class="price">{$obj->mProducts[k].price}</span>
          {/if}
        </p>
      </td>
    {if $smarty.section.k.index % 2 != 0 && !$smarty.section.k.first ||
        $smarty.section.k.last}
    </tr>
    {/if}
  {/section}
```

```
  </tbody>
</table>
{/if}
```

4. Для шаблона `products_list.tpl` создайте объект представления. Создайте файл с именем `products_list.php` в папке `presentation` и поместите в него следующий код.

```
<?php
class ProductsList
{
  // Public-переменные, доступные из шаблона Smarty
  public $mPage = 1;
  public $mrTotalPages;
  public $mLinkToNextPage;
  public $mLinkToPreviousPage;
  public $mProducts;

  // Private-переменные
  private $_mDepartmentId;
  private $_mCategoryId;

  // Конструктор класса
  public function __construct()
  {
    // Получаем DepartmentId из строки запроса и преобразуем его в int
    if (isset ($_GET['DepartmentId']))
    $this->_mDepartmentId = (int)$_GET['DepartmentId'];

    // Получаем CategoryId из строки запроса и преобразуем его в int
    if (isset ($_GET['CategoryId']))
      $this->_mCategoryId = (int)$_GET['CategoryId'];

    // Получаем номер страницы из строки запроса и преобразуем его в int
    if (isset ($_GET['Page']))
      $this->mPage = (int)$_GET['Page'];

    if ($this->mPage < 1)
      trigger_error('Incorrect Page value');
  }

  public function init()
  {
  /* Если посетитель просматривает категорию, получаем список ее товаров,
     вызывая метод уровня логики приложения GetProductsInCategory() */
  if (isset ($this->_mCategoryId))
  $this->mProducts = Catalog::GetProductsInCategory(
  $this->_mCategoryId, $this->mPage, $this->mrTotalPages);

  /* Если посетитель просматривает отдел, получаем список его товаров,
     вызывая метод уровня логики приложения GetProductsOnDepartment() */
  elseif (isset ($this->_mDepartmentId))
    $this->mProducts = Catalog::GetProductsOnDepartment(
      $this->_mDepartmentId, $this->mPage, $this->mrTotalPages);

  /* Если список товаров разбит на несколько страниц, отображаем
     навигационные элементы управления */
  if ($this->mrTotalPages > 1)
  {
    // Создаем ссылку Next
    if ($this->mPage < $this->mrTotalPages)
```

```
    {
      if (isset($this->_mCategoryId))
        $this->mLinkToNextPage =
          Link::ToCategory($this->_mDepartmentId, $this->_mCategoryId,
                           $this->mPage + 1);
      elseif (isset($this->_mDepartmentId))
        $this->mLinkToNextPage =
          Link::ToDepartment($this->_mDepartmentId, $this->mPage + 1);
    }

    // Создаем ссылку Previous
    if ($this->mPage > 1)
      {
        if (isset($this->_mCategoryId))
          $this->mLinkToPreviousPage =
            Link::ToCategory($this->_mDepartmentId, $this->_mCategoryId,
                             $this->mPage - 1);
        elseif (isset($this->_mDepartmentId))
          $this->mLinkToPreviousPage =
            Link::ToDepartment($this->_mDepartmentId, $this->mPage - 1);
      }
    }

    // Генерируем ссылки на страницы товаров
    for ($i = 0; $i < count($this->mProducts); $i++)
    {
      $this->mProducts[$i]['link_to_product'] =
        Link::ToProduct($this->mProducts[$i]['product_id']);

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

5. В файле `presentation/link.php` вставьте в класс `Link` новый метод и добавьте параметр `$Page` и код поддержки страниц в методы `ToDepartment()` и `ToCategory()` (в следующем листинге код, который нужно вставить, выделен полужирным шрифтом).

```
public static function ToDepartment($departmentId, $page = 1)
{
  $link = 'index.php?DepartmentId=' . $departmentId;

  if ($page > 1)
    $link .= '&Page=' . $page;

  return self::Build($link);
}

public static function ToCategory($departmentId, $categoryId, $page = 1)
{
  $link = 'index.php?DepartmentId=' . $departmentId .
          '&CategoryId=' . $categoryId;

  if ($page > 1)
    $link .= '&Page=' . $page;
```

```
    return self::Build($link);
  }

  public static function ToProduct($productId)
  {
    return self::Build('index.php?ProductId=' . $productId);
  }
}
?>
```

6. Откройте файл `presentation/templates/department.tpl` и замените в нем текст

```
Place list of products here
```

текстом

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

7. Загрузите наше приложение в браузере и перейдите на страницу какого-нибудь отдела, а затем выберите одну из категорий этого отдела. Найдите категорию с более чем четырьмя товарами, чтобы проверить, как работает постраничное отображение списков товаров (все должно выглядеть примерно так, как на рис. 5.10).

Как это работает: компонентный шаблон `products_list`

Большую часть функциональности, связанной со списком товаров, мы уже реализовали на уровне данных и уровне логики приложения, поэтому шаблон делает относительно немногое. Файл дизайн-шаблона Smarty (`products_list.tpl`) содержит разметку, используемую для отображения списка товаров, а соответствующий ему файл объекта представления (`presentation/products_list.php`) отвечает за заполнение этого списка требуемыми товарами.

Конструктор класса `ProductsList` (в файле `products_list.php`) создает новый экземпляр класса уровня логики приложения `Catalog` и получает значения `DepartmentId`, `CatalogId` и `Page` из строки запроса, преобразуя эти значения в тип `int` для обеспечения их корректности. По этим значениям принимается решение о том, какие товары нужно отображать в списке.

```
// Конструктор класса
public function __construct()
{
  // Получаем DepartmentId из строки запроса и преобразуем его в int
  if (isset ($_GET['DepartmentId']))
  $this->_mDepartmentId = (int)$_GET['DepartmentId'];

  // Получаем CategoryId из строки запроса и преобразуем его в int
  if (isset ($_GET['CategoryId']))
    $this->_mCategoryId = (int)$_GET['CategoryId'];

  // Получаем номер страницы из строки запроса и преобразуем его в int
  if (isset ($_GET['Page']))
    $this->mPage = (int)$_GET['Page'];

  if ($this->mPage < 1)
    trigger_error('Incorrect Page value');
}
```

Метод `init()`, продолжающий работу конструктора, сначала получает требуемый список товаров. Он решает, какой из методов уровня логики приложения вызвать, анализируя значения элементов `$_mCategoryId` и `$_mDepartmentId` (конструктор поместил в эти элементы значения параметров строки запроса `CategoryId` и `DepartmentId`).

```
public function init()
{
/* Если посетитель просматривает категорию, получаем список ее товаров,
   вызывая метод уровня логики приложения GetProductsInCategory() */
if (isset ($this->_mCategoryId))
$this->mProducts = Catalog::GetProductsInCategory(
$this->_mCategoryId, $this->mPage, $this->mrTotalPages);

/* Если посетитель просматривает отдел, получаем список его товаров,
   вызывая метод уровня логики приложения GetProductsOnDepartment() */
elseif (isset ($this->_mDepartmentId))
  $this->mProducts = Catalog::GetProductsOnDepartment(
    $this->_mDepartmentId, $this->mPage, $this->mrTotalPages);
```

Далее метод обеспечивает постраничное отображение списка товаров. Если при обращении к уровню логики приложения мы выяснили, что в списке будет больше одной страницы (т.е. товаров больше, чем указано в константе `PRODUCTS_PER_PAGE`), значит, нужно показывать посетителю только текущую страницу списка, снабдив ее навигационными ссылками Next и Previous. Комментарии в коде достаточно подробны, чтобы не было необходимости приводить его здесь.

Последняя часть метода добавляет в записи товаров данные о ссылках на страницы этих товаров и уменьшенные копии их изображений. Для генерации ссылок используются методы `Link::ToProduct()` и `Link::Build()`.

```
// Генерируем ссылки на страницы товаров
for ($i = 0; $i < count($this->mProducts); $i++)
{
  $this->mProducts[$i]['link_to_product'] =
    Link::ToProduct($this->mProducts[$i]['product_id']);

  if ($this->mProducts[$i]['thumbnail'])
    $this->mProducts[$i]['thumbnail'] =
      Link::Build('product_images/' . $this->mProducts[$i]['thumbnail']);
}
```

Кроме того, мы модифицировали методы класса `Link`, чтобы они добавляли в генерируемые ссылки параметр `Page`, если номер страницы списка больше единицы. Это необходимо, поскольку для отображения списков товаров требуется поддержка страниц.

Отображение содержимого главной страницы

Кроме общей информации о веб-сайте, мы хотим отображать на главной странице информацию о некоторых рекламируемых товарах.

Если пользователь просматривает страницу отдела или категории, для вывода информации используется шаблон Smarty `department`. Для главной страницы сайта мы создадим компонентный шаблон `first_page`, который будет отвечать за вывод информации о рекламируемых товарах.

Как вы, вероятно, помните, в классе `StoreFront` из файла `presentation/store_front.php` есть элемент `$mContentsCell`, содержимое которого меняется в зависимости от того, какую часть сайта просматривает пользователь. Если он просматривает страницу отдела или категории, загружается компонентный шаблон `department`, отвечающий за заполнение этого элемента. Но если посетитель просматривает главную страницу, этот элемент на данный момент ничем не заполняется.

В следующем упражнении мы напишем файл шаблона, содержащий кое-какую информацию о веб-сайте и отображающий сведения о товарах, которые должны рекламироваться на главной странице. Как вам известно, в таблице с информаци-

ей о товарах есть столбец display. Значение on_catalog в этом столбце означает, что соответствующий товар должен рекламироваться на главной странице.

Упражнение: создание компонентного шаблона first_page

1. Для начала создайте файл дизайн-шаблона Smarty. Это файл с именем first_page_contents.tpl в папке presentation/templates. Поместите в него следующий код.

```
{* first_page_contents.tpl *}
<p class="description">
  We hope you have fun developing TShirtShop, the e-commerce store from
  Beginning PHP and MySQL E-Commerce: From Novice to Professional!
</p>
<p class="description">
  We have the largest collection of t-shirts with postal stamps on Earth!
  Browse our departments and cateogories to find your favorite!
</p>
{include file="products_list.tpl"}
```

2. В файле presentation/store_front.tpl внесите изменения в строку, выделенную полужирным шрифтом в следующем фрагменте листинга.

```
<?php
class StoreFront
{
  public $mSiteUrl;
  // Определяем файл шаблона для содержимого страницы
  public $mContentsCell = 'first_page_contents.tpl';
  // Define the template file for the categories cell
  public $mCategoriesCell = 'blank.tpl';
```

   Теперь, если в строке запроса не будет параметров DepartmentId и ComponentId, файл store_front.tpl будет загружать компонентный шаблон firtst_page_contents.

3. Внесите изменения в метод init() класса ProductsList, находящийся в файле presentation/products_list.php: вставьте в этот метод строки, выделенные полужирным шрифтом в следующем листинге.

```
public function init()
{
/* Если посетитель просматривает категорию, получаем список ее товаров,
   вызывая метод уровня логики приложения GetProductsInCategory() */
  if (isset ($this->_mCategoryId))
  $this->mProducts = Catalog::GetProductsInCategory(
  $this->_mCategoryId, $this->mPage, $this->mrTotalPages);

/* Если посетитель просматривает отдел, получаем список его товаров,
   вызывая метод уровня логики приложения GetProductsOnDepartment() */
  elseif (isset ($this->_mDepartmentId))
    $this->mProducts = Catalog::GetProductsOnDepartment(
      $this->_mDepartmentId, $this->mPage, $this->mrTotalPages);

/* Если посетитель просматривает первую страницу, получаем список товаров,
   вызывая метод уровня логики приложения GetProductsOnCatalog() */
  else
    $this->mProducts = Catalog::GetProductsOnCatalog(
                        $this->mPage, $this->mrTotalPages);
```

```
/* Если список товаров разбит на несколько страниц, отображаем
   навигационные элементы управления */
if ($this->mrTotalPages > 1)
{
  // Создаем ссылку Next
  if ($this->mPage < $this->mrTotalPages)
  {
    if (isset($this->_mCategoryId))
      $this->mLinkToNextPage =
        Link::ToCategory($this->_mDepartmentId, $this->_mCategoryId,
                         $this->mPage + 1);
    elseif (isset($this->_mDepartmentId))
      $this->mLinkToNextPage =
        Link::ToDepartment($this->_mDepartmentId, $this->mPage + 1);
    else
      $this->mLinkToNextPage = Link::ToIndex($this->mPage + 1);
  }

  // Создаем ссылку Previous
  if ($this->mPage > 1)
    {
      if (isset($this->_mCategoryId))
        $this->mLinkToPreviousPage =
          Link::ToCategory($this->_mDepartmentId, $this->_mCategoryId,
                           $this->mPage - 1);
      elseif (isset($this->_mDepartmentId))
        $this->mLinkToPreviousPage =
          Link::ToDepartment($this->_mDepartmentId, $this->mPage - 1);
      else
        $this->mLinkToPreviousPage = Link::ToIndex($this->mPage - 1);
    }
  }
```

4. Откройте файл `link.php` из папки `presentation` и добавьте в класс `Link` следующий метод.

```
public static function ToIndex($page = 1)
{
  $link = '';

  if ($page > 1)
    $link .= 'index.php?Page=' . $page;

  return self::Build($link);
}
```

5. Откройте наш проект в браузере. Теперь главная страница должна выглядеть так, как на рис. 5.11.

Как это работает: компонентный шаблон `first_page_contents`

Собственно список товаров по-прежнему отображается с помощью компонентного шаблона `products_list`, который мы создали ранее в этой главе. Но теперь он загружается не из файла `department.tpl` (как при просмотре страницы отдела или категории), а из нового файла шаблона — `first_page_contents.tpl`.

Мы добавили в метод `init()` класса `ProductsList` новую функциональность для отображения сведений о рекламируемых товарах на главной странице сайта. Кроме того, мы добавили метод `ToIndex()` в класс `Link` для обеспечения поддержки многостраничных списков на главной странице.

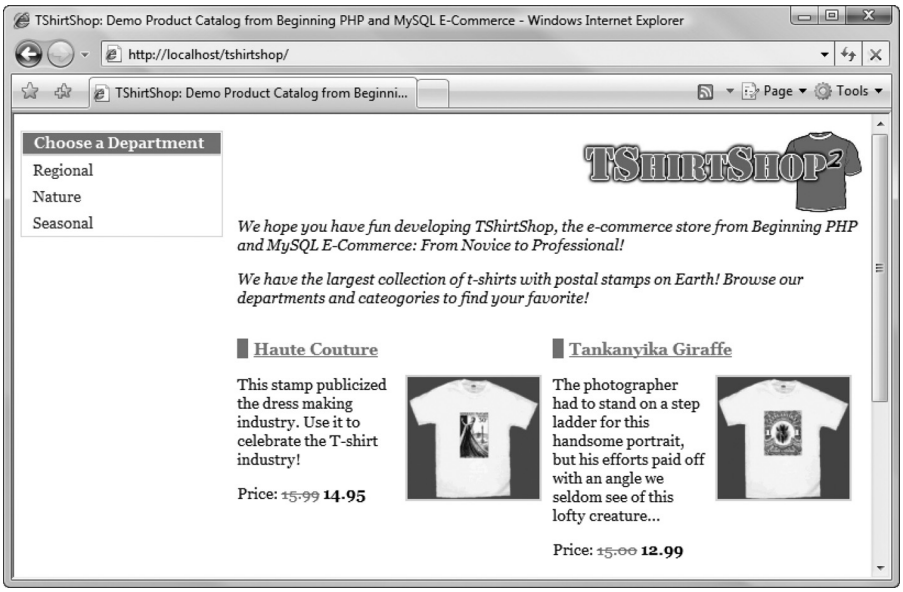


Рис. 5.11. Главная страница с рекламируемыми товарами

Отображение подробных сведений о товарах

Последний шаблон, которым мы займемся в этой главе, отвечает за отображение сведений о товарах на страницах этих товаров. Программа обращается к нему, когда посетитель щелкает на товаре в списке. На странице товара отображаются название товара, его полное описание и изображение. В последующих главах на этих страницах появятся дополнительные элементы, например рекомендации товаров и обзоры.

Кроме того, на странице товара должен присутствовать элемент, позволяющий посетителю вернуться к списку товаров. Этот механизм состоит из ссылки Continue Shopping внизу страницы, возвращающей посетителя к предыдущей открытой странице. Если посетитель открыл страницу товара, набрав ее адрес в браузере, или нашел ее с помощью поисковой системы, ссылка Continue Shopping будет перенаправлять посетителя на главную страницу сайта. Давайте посмотрим, как все это реализовать.

Упражнение: создание компонентного шаблона `product`

1. Добавьте в файл `styles/tshirtshop.css` следующие стили.

```
.product-image {
  border: 2px solid #c6e1ec;
}

ol {
  margin: 0px;
  padding: 0px 0px 0px 5px;
}
```

```
ol li {
  color: #0590C7;
  list-style-type: none;
  margin: 0px;
  padding: 5px 0px;
}
```

2. Создайте страницу, на которой будут использоваться эти стили. Создайте в папке `presentation/templates` файл `products.tpl`. Если вам не нравится вид страницы, которую сформирует приведенный ниже код, можете модифицировать его так, как сочтете нужным.

```
{load_presentation_object filename="product" assign="obj"}
<h1 class="title">{$obj->mProduct.name}</h1>
{if $obj->mProduct.image}
<img class="product-image" src="{$obj->mProduct.image}"
  alt="{$obj->mProduct.name} image" />
{/if}
{if $obj->mProduct.image_2}
<img class="product-image" src="{$obj->mProduct.image_2}"
  alt="{$obj->mProduct.name} image 2" />
{/if}
<p class="description">{$obj->mProduct.description}</p>
<p class="section">
  Price:
  {if $obj->mProduct.discounted_price != 0}
    <span class="old-price">{$obj->mProduct.price}</span>
    <span class="price">{$obj->mProduct.discounted_price}</span>
  {else}
    <span class="price">{$obj->mProduct.price}</span>
  {/if}
</p>
{if $obj->mLinkToContinueShopping}
<a href="{$obj->mLinkToContinueShopping}">Continue Shopping</a>
{/if}
<h2>Find similar products in our catalog:</h2>
<ol>
{section name=i loop=$obj->mLocations}
  <li class="navigation">
    {strip}
    <a href="{$obj->mLocations[i].link_to_department}">
      {$obj->mLocations[i].department_name}
    </a>
    {/strip}
    &raquo;
    {strip}
    <a href="{$obj->mLocations[i].link_to_category}">
      {$obj->mLocations[i].category_name}
    </a>
    {/strip}
  </li>
{/section}
</ol>
```

3. Создайте компонентный шаблон для страницы сведений о товаре, который будет отображать подробные сведения о товаре и его изображение. Этот шаблон должен находиться в файле с именем `product.php` в папке `presentation`. Поместите в него следующий код.

```
<?php
// Отвечает за отображение подробных сведений о товаре
```

```
class Product
{
  // Public-переменные для использования в шаблоне Smarty
  public $mProduct;
  public $mProductLocations;
  public $mLinkToContinueShopping;
  public $mLocations;

  // Private-переменная
  private $_mProductId;

  // Конструктор класса
  public function __construct()
  {
    // Инициализация переменной
    if (isset ($_GET['ProductId']))
      $this->_mProductId = (int)$_GET['ProductId'];
    else
      trigger_error('ProductId not set');
  }

  public function init()
  {
    // Получаем сведения о товаре из уровня логики приложения
    $this->mProduct = Catalog::GetProductDetails($this->_mProductId);

    if (isset ($_SESSION['link_to_continue_shopping']))
    {
      $continue_shopping =
        Link::QueryStringToArray($_SESSION['link_to_continue_shopping']);

      $page = 1;

      if (isset ($continue_shopping['Page']))
        $page = (int)$continue_shopping['Page'];

      if (isset ($continue_shopping['CategoryId']))
        $this->mLinkToContinueShopping =
          Link::ToCategory((int)$continue_shopping['DepartmentId'],
                           (int)$continue_shopping['CategoryId'], $page);
      elseif (isset ($continue_shopping['DepartmentId']))
        $this->mLinkToContinueShopping =
          Link::ToDepartment((int)$continue_shopping['DepartmentId'], $page);
      else
        $this->mLinkToContinueShopping = Link::ToIndex($page);
    }

    if ($this->mProduct['image'])
      $this->mProduct['image'] =
        Link::Build('product_images/' . $this->mProduct['image']);

    if ($this->mProduct['image_2'])
      $this->mProduct['image_2'] =
        Link::Build('product_images/' . $this->mProduct['image_2']);

    $this->mLocations = Catalog::GetProductLocations($this->_mProductId);

    // Генерируем ссылки на страницы отдела и категории
    for ($i = 0; $i < count($this->mLocations); $i++)
    {
```

```
      $this->mLocations[$i]['link_to_department'] =
        Link::ToDepartment($this->mLocations[$i]['department_id']);

      $this->mLocations[$i]['link_to_category'] =
        Link::ToCategory($this->mLocations[$i]['department_id'],
                         $this->mLocations[$i]['category_id']);
    }
  }
}
?>
```

4. Добавьте в класс `Link` в файле `presentation/link.php` следующий метод. Он генерирует ассоциативный массив из элементов строки запроса. Этот массив нам понадобится для генерации ссылки Continue Shopping.

```
public static function QueryStringToArray($queryString)
{
  $result = array();

  if ($queryString != '')
  {
    $elements = explode('&', $queryString);

    foreach($elements as $key => $value)
    {
      $element = explode('=', $value);
      $result[urldecode($element[0])] =
        isset($element[1]) ? urldecode($element[1]) : '';
    }
  }

  return $result;
}
```

5. В файл `presentation/products_list.php` вставьте строки кода, выделенные полужирным шрифтом в приведенном ниже листинге. Новый код будет сохранять адрес страницы каталога, открытой последней. При отображении страницы со сведениями о товаре этот адрес будет использоваться в ссылке Continue Shopping.

```
// Конструктор класса
public function __construct()
{
...
...
  if ($this->mPage < 1)
    trigger_error('Incorrect Page value');

  // Сохраняем адрес страницы, посещенной последней
  $_SESSION['link_to_continue_shopping'] = $_SERVER['QUERY_STRING'];
}
```

6. В файл `presentation/departments_list.php` вставьте строки кода, выделенные полужирным шрифтом в следующем листинге. Этот код также необходим для обеспечения работы ссылки Continue Shopping.

```
// Конструктор считывает строку запроса как параметр
public function __construct()
{
  /* Если в строке запроса есть DepartmentId, мы посещаем отдел */
  if (isset ($_GET['DepartmentId']))
```

```
    $this->mSelectedDepartment = (int)$_GET['DepartmentId'];
  elseif (isset($_GET['ProductId']) &&
          isset($_SESSION['link_to_continue_shopping']))
  {
    $continue_shopping =
      Link::QueryStringToArray($_SESSION['link_to_continue_shopping']);

    if (array_key_exists('DepartmentId', $continue_shopping))
      $this->mSelectedDepartment =
        (int)$continue_shopping['DepartmentId'];
  }
}
```

7. В файл presentation/departments_list.php вставьте строки кода, выделенные полужирным шрифтом в следующем листинге. Этот код должен обеспечивать загрузку идентификаторов отдела и категории из данных, хранимых в сеансе.

```
// Конструктор считывает параметр из строки запроса
public function __construct()
{
  if (!isset($_GET['ProductId']))
  {
    if (isset ($_GET['DepartmentId']))
      $this->mSelectedDepartment = (int)$_GET['DepartmentId'];
    else
      trigger_error('DepartmentId not set');

    if (isset ($_GET['CategoryId']))
      $this->mSelectedCategory = (int)$_GET['CategoryId'];
  }
  else
  {
    $continue_shopping =
      Link::QueryStringToArray($_SESSION['link_to_continue_shopping']);

    if (array_key_exists('DepartmentId', $continue_shopping))
      $this->mSelectedDepartment =
        (int)$continue_shopping['DepartmentId'];
    else
      trigger_error('DepartmentId not set');

    if (array_key_exists('CategoryId', $continue_shopping))
      $this->mSelectedCategory =
        (int)$continue_shopping['CategoryId'];
  }
}
```

8. Отредактируйте файл presentation/store_front.tpl так, чтобы шаблон product.tpl загружался в элементе $mContentsCell, если в строке запроса присутствует параметр ProductId. Кроме того, если страница товара открыта со страницы отдела или категории, необходимо отображать список категорий. Это маленькая деталь, но очень полезная для упрощения навигации по каталогу. Вставьте в файл строки, выделенные полужирным шрифтом в приведенном ниже листинге.

```
public function init()
{
  // Загружаем подробные сведения об отделе на страницу отдела
  if (isset ($_GET['DepartmentId']))
```

```
  {
    $this->mContentsCell = 'department.tpl';
    $this->mCategoriesCell = 'categories_list.tpl';
  }
  elseif (isset($_GET['ProductId']) &&
          isset($_SESSION['link_to_continue_shopping']) &&
          strpos($_SESSION['link_to_continue_shopping'], 'DepartmentId', 0)
          !== false)
    {
      $this->mCategoriesCell = 'categories_list.tpl';
    }

    // Загружаем сведения о товаре на страницу товара
    if (isset ($_GET['ProductId']))
      $this->mContentsCell = 'product.tpl';
  }
}
?>
```

9. Загрузите наш сайт в браузере и щелкните на изображении или названии любого товара. Должна открыться страница с подробными сведениями об этом товаре. Пример такой страницы показан на рис. 5.12.

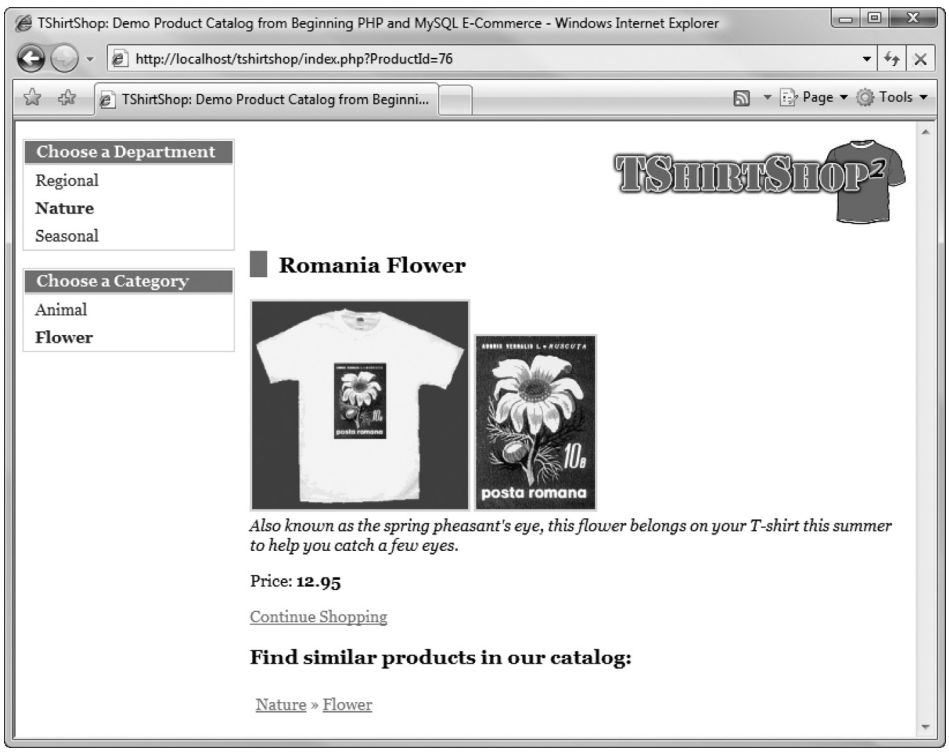


Рис. 5.12. Страница со сведениями о товаре

Как это работает: компонентный шаблон `product`

Как вы, вероятно, и ожидали, реализация страницы со сведениями о товарах была весьма прямолинейной — фактически мы в основном обновляли код, написанный раньше. Мы дополняли этот код новыми блоками, обеспечивающими работу новых элементов приложения. В дальнейшем такие модификации станут для вас проще по мере того, как вы привыкнете к структуре проекта и начнете свободно в нем ориентироваться.

В этом упражнении интересна реализация поддержки ссылки Continue Shopping с помощью данных, сохраняемых в сеансовых переменных. Кроме того, обратите внимание на механизм отображения списка категорий на странице товара. Если страница товара открыта из каталога или ее адрес напрямую набран в браузере, или ее нашла поисковая система, список категорий не отображается. Но если страница товара открыта со страницы отдела или категории, список и навигационные элементы будут отображены на странице товара.

Назначение созданного кода описано в самом упражнении, поэтому мы не будем рассматривать его еще раз. Прежде чем переходить к следующей главе, убедитесь, что вы разобрались в этом коде и четко представляете себе, как он работает.

Резюме

Поздравляем! В этой главе вы проделали большую работу. Теперь в нашем приложении есть полностью работающий каталог товаров, для реализации которого вы добавили новую функциональность на уровень данных, уровень логики приложения и уровень представления. В то же время вы получили немало теоретических сведений.

- Структуру отношений между таблицами в реляционных базах данных
- Методы извлечения данных из нескольких таблиц одновременно с помощью `JOIN`
- Методы фильтрации результатов запросов с помощью `WHERE`
- Способы отображения данных на страницах категорий и товаров
- Механизмы обеспечения навигации в каталоге
- Механизмы постраничного отображения списков товаров

Если код, который мы анализировали в этой главе, показался вам сложным, не пугайтесь — его и в самом деле вряд ли назовешь простым. Если только вы не гуру в PHP, можете гордиться собой — вы создали довольно замысловатое приложение. Что еще важнее, вы заставили это приложение работать — все эти страницы отделов, категорий, информации о товарах и много чего еще.

Глава 6 будет не менее интересней, чем эта: в ней мы займемся добавлением атрибутов к товарам.

Глава 6

Атрибуты товаров

Многие электронные магазины позволяют посетителям выбирать параметры покупаемых товаров. Например, продавая футболки, мы можем позволить покупателю выбрать нужный размер и цвет, а не ограничивать его единственным доступным вариантом. В этой главе мы займемся реализацией атрибутов товаров в TShirtShop (рис. 6.1).

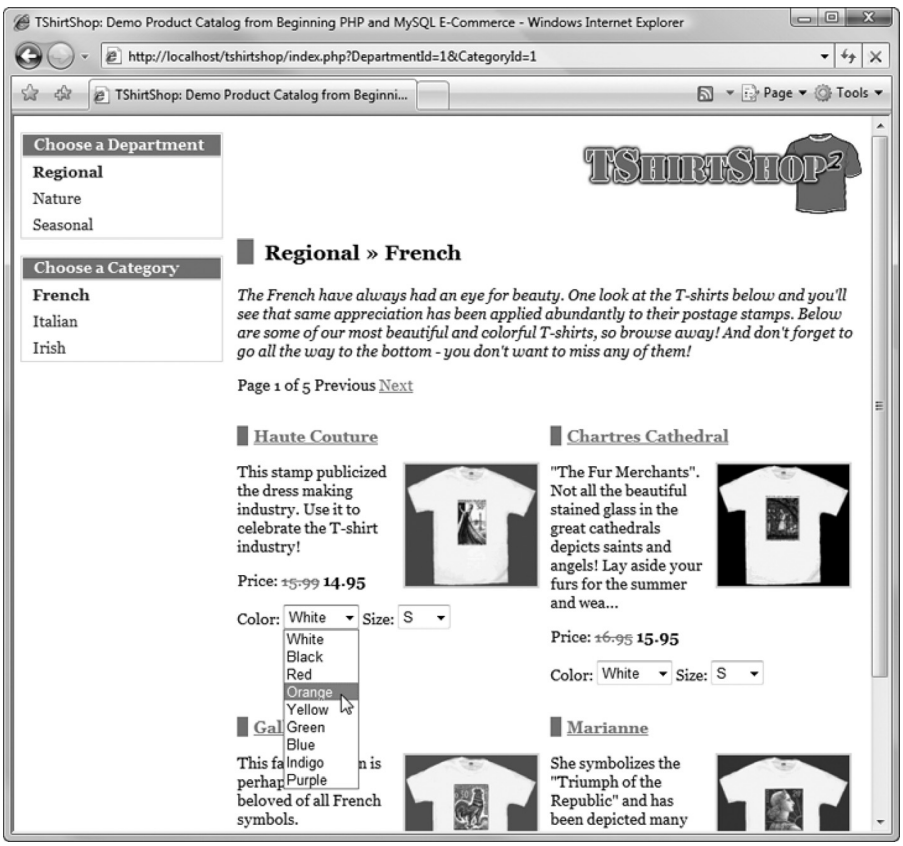


Рис. 6.1. Товары с атрибутами

Чтобы реализовать атрибуты товаров, сначала модифицируем уровень данных, в котором создадим новые таблицы данных и хранимую процедуру, а затем напишем код уровня логики приложения, вызывающий эту процедуру. И наконец мы воспользуемся созданной функциональностью, чтобы обновить шаблон Smarty для пользовательского интерфейса. В конце этой главы каталог позволит посетителям выбрать размер и цвет футболки, как показано на рис. 6.1. Поскольку данные об атрибутах хранятся в базе данных, вы можете добавлять собственные атрибуты и значения для них.

Реализация уровня данных

Компоненты уровня данных, обеспечивающие поддержку атрибутов товаров — это три таблицы (`attribute`, `attribute_value` и `product_attribute`) и хранимая процедура `catalog_get_product_attributes`.

Вот функции этих трех таблиц.

- В таблице `attribute` хранятся названия атрибутов, например `Size` и `Color`.
- Таблица `attribute_value` хранит возможные значения атрибутов для каждой группы атрибутов. Между таблицами `attribute` и `attribute_value` установлено отношение "один-ко-многим". С каждым атрибутом может быть связано несколько значений, например с атрибутом `Color` — значения `Red`, `Orange`, `Yellow` и т.д. Таблица `attribute_value` понадобится нам, чтобы отслеживать соответствие между `attribute_id` (например, идентификатор `1` для атрибута `Color`) и `attribute_value_ids` возможных значений этого атрибута (например, `Red`, `Orange`, `Yellow`). Поэтому в таблице `attribute_value` будет три столбца: столбец `attribute_id`, присутствующий в таблице `attribute` и указывающий, о каком атрибуте идет речь (например, о цвете или размере); столбец `attribute_value_id` с целочисленными значениями, однозначно указывающими на данные в самой таблице `attribute_value`; и столбец `value`, в котором будет храниться текстовое описание значения (например, `Red`, `Orange`, `Yellow`, `S` — маленький размер, `M` — средний размер и т.д.).
- Таблица `product_attribute` обеспечивает выполнение отношения "многие-ко-многим" между таблицами `product` и `attribute_value` (пары значений `product_id` и `attribute_value_id`). Таблица `attribute_value` позволит нам легко и быстро заполнить эту таблицу данными.

ЗАМЕЧАНИЕ

Система, которую мы реализуем, не позволит значениям атрибутов влиять на цену товара. Например, белая футболка будет всегда стоить столько же, сколько черная. Если вас это не устраивает и цена товара должна зависеть от его атрибутов, придется создать разные товары для разных значений атрибутов. В качестве альтернативы можно было бы модифицировать таблицу `product_attribute`, включив в нее данные об изменении цены в зависимости от значений атрибутов и внести массу изменений на уровне логики приложения и уровень представления. Мы бы не советовали вам делать это сейчас, поскольку внесенные вами изменения усложнят выполнение заданий из следующих глав книги.

Из главы 5 вы узнали, что диаграммы баз данных могут помочь разобраться в отношениях между таблицами. Графическое представление таблиц, которые мы рассматриваем, приведено на рис. 6.2. Если вы до сих пор мало работали с базами данных, потратьте пару минут на изучение этого рисунка — он поможет вам быстрее разобраться в коде, который мы напишем в этой главе. Чтобы вспомнить отно-

шения между таблицей products и другими таблицами, посмотрите на рис. 5.6 в предыдущей главе.

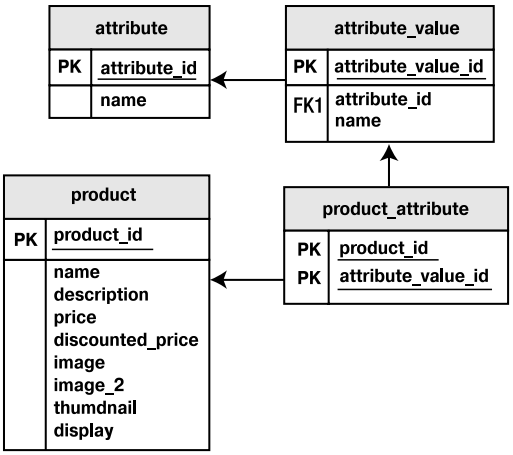


Рис. 6.2. Отношения между таблицами, необходимыми для реализации атрибутов товаров

Первое, что мы сделаем, — обновим базу данных tshirtshop, а также реализуем новую хранимую процедуру catalog_get_product_attributes в следующем упражнении.

Упражнение: создание функциональности уровня данных для поддержки атрибутов товаров

1. Откройте phpMyAdmin и выполните в нем следующий код, создающий и заполняющий данными таблицу attribute.

```
-- Создаем таблицу attribute (в ней хранятся названия
-- атрибутов, например, Size и Color)
CREATE TABLE `attribute` (
  `attribute_id` INT          NOT NULL  AUTO_INCREMENT,
  `name`         VARCHAR(100) NOT NULL, -- Например, Color, Size
   PRIMARY KEY (`attribute_id`)
) ENGINE=MyISAM;

-- Заполняем данными таблицу attribute
INSERT INTO `attribute` (`attribute_id`, `name`) VALUES
       (1, 'Size'), (2, 'Color');
```

2. Создайте и заполните данными таблицу attribute_value с помощью следующих операторов SQL.

```
-- Создаем таблицу attribute_value (в ней хранятся
-- значения вроде Yellow или XXL)
CREATE TABLE `attribute_value` (
  `attribute_value_id` INT NOT NULL AUTO_INCREMENT,
  `attribute_id` INT NOT NULL, -- Идентификатор атрибута
  `value` VARCHAR(100) NOT NULL, -- Например, Yellow/
  PRIMARY KEY (`attribute_value_id`),
  KEY `idx_attribute_value_attribute_id` (`attribute_id`)
) ENGINE=MyISAM;
```

```
-- Заполняем данными таблицу attribute_value
INSERT INTO `attribute_value` (`attribute_value_id`, `attribute_id`,
`value`)
VALUES
      (1, 1, 'S'), (2, 1, 'M'), (3, 1, 'L'), (4, 1, 'XL'), (5, 1, 'XXL'),
      (6, 2, 'White'), (7, 2, 'Black'), (8, 2, 'Red'), (9, 2, 'Orange'),
      (10, 2, 'Yellow'), (11, 2, 'Green'), (12, 2, 'Blue'),
      (13, 2, 'Indigo'), (14, 2, 'Purple');
```

3. Создайте и заполните таблицу `product_attribute` с помощью следующего кода.

```
-- Создаем таблицу product_attribute (связывающую значения
-- атрибутов с товарами)
CREATE TABLE `product_attribute` (
  `product_id`         INT NOT NULL,
  `attribute_value_id` INT NOT NULL,
  PRIMARY KEY (`product_id`, `attribute_value_id`)
) ENGINE=MyISAM;

-- Заполняем данными таблицу product_attribute
INSERT INTO `product_attribute` (`product_id`, `attribute_value_id`)
       SELECT `p`.`product_id`, `av`.`attribute_value_id`
       FROM   `product` `p`, `attribute_value` `av`;
```

4. Создайте хранимую процедуру `catalog_get_product_attributes`. Она будет выбирать из базы данных атрибуты, присвоенные товару. Чтобы создать ее, выполните следующие операторы SQL (не забудьте выбрать в качестве конечного разделителя символы `$$`, прежде чем выполнять эти операторы).

```
-- Создаем хранимую процедуру catalog_get_product_attributes
CREATE PROCEDURE catalog_get_product_attributes(IN inProductId INT)
BEGIN
  SELECT     a.name AS attribute_name,
             av.attribute_value_id, av.value AS attribute_value
  FROM       attribute_value av
  INNER JOIN attribute a
               ON av.attribute_id = a.attribute_id
  WHERE      av.attribute_value_id IN
              (SELECT attribute_value_id
                FROM  product_attribute
                WHERE product_id = inProductId)
  ORDER BY   a.name;
END$$
```

Как это работает: функциональность уровня данных для поддержки атрибутов товаров

Ниже мы рассмотрим код, заполняющий данными таблицу `product_attribute` и создающий хранимую процедуру `catalog_get_product_attributes`. Прежде чем двигаться дальше, убедитесь, что вы понимаете назначение каждого столбца в таблицах `attribute` и `attribute_value`, а также кода, заполняющего эти таблицы данными. Возможно, вам понадобится на это некоторое время. Мы не будем повторять здесь теоретические выкладки, но вы можете вернуться к предыдущим главам, если они вам понадобятся, — в главах 4 и 5 описаны основные концепции баз данных.

При заполнении данными таблицы `product_attribute` наша задача — связать все существующие значения атрибутов (с помощью столбца `attribute_value_id`) с каждым из наших товаров (через столбец `product_id`). На нашем сайте товары — это футболки, и мы хотим продавать футболки разных размеров и цветов. Код, заполняющий данными таблицу `product_attribute`, использует оператор `INSERT` для вставки записей, выбранных оператором `SELECT`:

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

В нашем случае оператор SELECT, поставляющий данные для таблицы, возвращает так называемое *декартово произведение* (cartesian product) двух множеств данных. Результат такого произведения — список всех возможных сочетаний элементов из двух множеств. Например, декартово произведение множеств {1, 2, 3} и {a, b, c} — это множество {{1,a}, {1, b}, {1, c}, {2, a}, {2, b}, {2, c}, {3, a}, {3, b}, {3, c}}.

Сгенерировав декартово произведение идентификаторов товаров и идентификаторов значений атрибутов, мы получим список пар значений product_id и attribute_value_id, который и поместим в таблицу product_attribute. В следующем фрагменте кода оператор, генерирующий декартово произведение, выделен полужирным шрифтом.

```
INSERT INTO `product_attribute` (`product_id`, `attribute_value_id`)
       SELECT `p`.`product_id`, `av`.`attribute_value_id`
       FROM   `product` `p`, `attribute_value` `av`;
```

Это, безусловно, ловкий прием для создания множества записей в таблице с минимальной затратой сил. В нашей базе данных содержатся 14 значений атрибутов и 101 товар. Декартово произведение всех значений, записанное в таблицу product_attribute, будет состоять из 1414 элементов (14 умножить на 101).

Есть несколько альтернативных методов заполнения таблицы product_attribute необходимыми данными без прямого использования декартова произведения. Среди этих методов стоит упомянуть использование UNION. Этот оператор позволяет объединить результаты, возвращаемые несколькими операторами SELECT. Например, использовав UNION для двух запросов, возвращающих по пять записей, мы получим набор из десяти записей. Разумеется, UNION позволяет объединять только результаты запросов, состоящие из одинаковых столбцов с совместимыми типами данных. Мы не будем рассматривать этот метод подробно, но если вам хочется увидеть код, использующий его, вот пример. Он не такой компактный, как вариант с декартовым произведением, но более гибок, например позволяет использовать для определенных товаров только некоторые значения атрибутов.

```
-- Заполняем значениями таблицу product_attribute
INSERT INTO `product_attribute` (`product_id`, `attribute_value_id`)
            SELECT       product_id, 1 AS attribute_value_id FROM product
            UNION SELECT product_id, 2 AS attribute_value_id FROM product
            UNION SELECT product_id, 3 AS attribute_value_id FROM product
            UNION SELECT product_id, 4 AS attribute_value_id FROM product
            UNION SELECT product_id, 5 AS attribute_value_id FROM product
            UNION SELECT product_id, 6 AS attribute_value_id FROM product
            UNION SELECT product_id, 7 AS attribute_value_id FROM product
            UNION SELECT product_id, 8 AS attribute_value_id FROM product
            UNION SELECT product_id, 9 AS attribute_value_id FROM product
            UNION SELECT product_id, 10 AS attribute_value_id FROM product
            UNION SELECT product_id, 11 AS attribute_value_id FROM product
            UNION SELECT product_id, 12 AS attribute_value_id FROM product
            UNION SELECT product_id, 13 AS attribute_value_id FROM product
            UNION SELECT product_id, 14 AS attribute_value_id FROM product;
```

И наконец давайте рассмотрим хранимую процедуру catalog_get_product_attributes. Она получает в качестве параметра идентификатор товара и возвращает список значений атрибутов этого товара. Это удобный механизм, который мы будем использовать на уровне логики приложения для получения списков значений атрибутов, из которых посетители смогут выбирать подходящие цвет и размер футболок:

```
-- Создаем хранимую процедуру catalog_get_product_attributes
CREATE PROCEDURE catalog_get_product_attributes(IN inProductId INT)
BEGIN
```

SQL-код в этой процедуре возвращает список всех значений attribute_name, attribute_value_id и attribute_value для всех атрибутов указанного товара.

```
SELECT     a.name AS attribute_name,
           av.attribute_value_id, av.value AS attribute_value
FROM       attribute_value av
INNER JOIN attribute a
             ON av.attribute_id = a.attribute_id
```

```
WHERE      av.attribute_value_id IN
            (SELECT attribute_value_id
              FROM  product_attribute
              WHERE product_id = inProductId)
ORDER BY   a.name;
```

Чтобы убедиться в том, что эта процедура работает как надо, можно выполнить ее код с помощью phpMyAdmin, заменив имена параметров соответствующими значениями. Например, вместо `inProductId` можно подставить идентификатор товара, а затем выполнить код в phpMyAdmin.

Альтернативный вариант — напрямую вызвать хранимую процедуру — не поддерживается phpMyAdmin на момент написания данной книги. Однако вы можете выполнить хранимую процедуру с помощью командной консоли MySQL. Чтобы запустить эту консоль в Windows, вам нужно выполнить следующую команду в папке `C:\xampp\mysql\bin` (при условии, что вы устанавливали XAMPP так, как указано в главе 1):

```
mysql -u root
```

Установив соединение с сервером MySQL, вам нужно будет подключиться к базе данных `tshirtshop` с помощью команды

```
USE tshirtshop;
```

После этого можно вызвать хранимую процедуру `catalog_get_product_attributes`, передав ей значение `1` в качестве параметра:

```
CALL catalog_get_product_attributes (1);
```

В результате выполнения хранимой процедуры вы получите список всех атрибутов и их значений для товара с идентификатором 1, примерно в следующем виде.

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

Реализация уровня логики приложения

Код уровня логики приложения, отвечающий за поддержку атрибутов товаров, весьма компактен — нам всего лишь нужно вызвать хранимую процедуру `catalog_get_product_attributes`. Добавьте следующий код в класс `Catalog` в файле `business/catalog.php`.

```
// Извлекаем атрибуты товаров
public static function GetProductAttributes($productId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_product_attributes(:product_id)';

  // Создаем массив параметров
```

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

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetAll($sql, $params);
}
```

Реализация уровня представления

Реализация поддержки атрибутов на уровне представления потребует создания новых элементов управления, которые позволят посетителям выбирать значения этих атрибутов. Как вы вскоре увидите, значения атрибутов не хранятся непосредственно в шаблонах. Вместо этого атрибуты (например, `Size` и `Color`) и их значения (например, `L`, `XL`, `White`, `Green` и т.д.) считываются из базы данных.

Как вы уже, вероятно, убедились, такой подход позволит менять атрибуты гораздо проще, чем подход с хранением атрибутов и их значений в коде программы. Достаточно будет поменять данные в базе данных — и изменения автоматически отобразятся на страницах сайта.

Давайте реализуем поддержку атрибутов в коде уровня представления, а затем подробно рассмотрим внесенные изменения. Чтобы вам было проще понять назначение нового кода, вспомните, как работает хранимая процедура `catalog_get_product_attributes`. Пример результатов ее выполнения вы видели немного выше.

Упражнение: реализация поддержки атрибутов товаров на уровне представления

1. Добавьте выделенный полужирным шрифтом код из следующего листинга в файл `presentation\templates\products_list.tpl`.

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

{* Генерируем список значений атрибутов *}
<p class="attributes">

{* Просматриваем список атрибутов и их значений *}
{section name=l loop=$obj->mProducts[k].attributes}

  {* Генерируем новый тег select? *}
  {if $smarty.section.l.first ||
      $obj->mProducts[k].attributes[l].attribute_name !==
      $obj->mProducts[k].attributes[l.index_prev].attribute_name}
  {$obj->mProducts[k].attributes[l].attribute_name}:
<select name="attr_{$obj->mProducts[k].attributes[l].attribute_name}">
{/if}

    {* Генерируем новый тег option *}
    <option value="{$obj->mProducts[k].attributes[l].attribute_value}">
      {$obj->mProducts[k].attributes[l].attribute_value}
    </option>
```

```
          {* Закрываем тег select? *}
          {if $smarty.section.l.last ||
              $obj->mProducts[k].attributes[l].attribute_name !==
              $obj->mProducts[k].attributes[l.index_next].attribute_name}
          </select>
          {/if}

        {/section}
        </p>
      </td>
    {if $smarty.section.k.index % 2 != 0 && !$smarty.section.k.first ||
        $smarty.section.k.last}
    </tr>
    {/if}
  {/section}
  </tbody>
</table>
{/if}
```

2. Добавьте выделенный полужирным шрифтом код из следующего листинга в метод `init()` в файле `presentation/products_list.php`.

```
    // Генерируем ссылки на страницы товаров
    for ($i = 0; $i < count($this->mProducts); $i++)
    {
      $this->mProducts[$i]['link_to_product'] =
        Link::ToProduct($this->mProducts[$i]['product_id']);

      if ($this->mProducts[$i]['thumbnail'])
        $this->mProducts[$i]['thumbnail'] =
          Link::Build('product_images/' .
          $this->mProducts[$i]['thumbnail']);

      $this->mProducts[$i]['attributes'] =
        Catalog::GetProductAttributes($this->mProducts[$i]['product_id']);
    }
  }
}
?>
```

3. Откройте файл `presentation/templates/product.tpl` и добавьте в него выделенный полужирным код из следующего листинга. Этот код генерирует список значений атрибутов для страницы со сведениями о товаре.

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

{* Генерируем список значений атрибутов *}
<p class="attributes">

{* Просматриваем список атрибутов и их значений *}
{section name=k loop=$obj->mProduct.attributes}

  {* Генерируем новый тег select? *}
```

```
  {if $smarty.section.k.first ||
      $obj->mProduct.attributes[k].attribute_name !==
      $obj->mProduct.attributes[k.index_prev].attribute_name}
    {$obj->mProduct.attributes[k].attribute_name}:
  <select name="attr_{$obj->mProduct.attributes[k].attribute_name}">
  {/if}

    {* Генерируем новый тег option *}
    <option value="{$obj->mProduct.attributes[k].attribute_value}">
      {$obj->mProduct.attributes[k].attribute_value}
    </option>

  {* Закрываем тег select? *}
  {if $smarty.section.k.last ||
      $obj->mProduct.attributes[k].attribute_name !==
      $obj->mProduct.attributes[k.index_next].attribute_name}
  </select>
  {/if}

{/section}
</p>
{if $obj->mLinkToContinueShopping}
<a href="{$obj->mLinkToContinueShopping}">Continue Shopping</a>
{/if}
```

4. Добавьте выделенный полужирным шрифтом код из следующего листинга в метод `init()` класса `Product` в файле `presentation/product.php`.

```
if ($this->mProduct['image_2'])
  $this->mProduct['image_2'] =
    Link::Build('product_images/' . $this->mProduct['image_2']);

$this->mProduct['attributes'] =
  Catalog::GetProductAttributes($this->mProduct['product_id']);

$this->mLocations = Catalog::GetProductLocations($this->_mProductId);
```

5. Добавьте в файл `styles/tshirtshop.css` следующий стиль.

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

6. Откройте в браузере страницу `http://localhost/tshirtshop/` и посмотрите, как выглядят атрибуты товаров на страницах сайта. Например, на рис. 6.1 показаны атрибуты в списке товаров, а на рис. 6.3 — атрибуты на странице отдельного товара.

Как это работает: поддержка атрибутов на уровне представления

Мы реализовали поддержку атрибутов в пользовательском интерфейсе в два шага. Сначала мы добавили в код вызов метода уровня логики приложения `Catalog::GetProductAttributes()`, чтобы получить список атрибутов заданного товара. Затем мы обновили шаблон `products_list.tpl` для отображения списка атрибутов каждого товара.

Пока что эти атрибуты не слишком полезны для посетителя — он все еще не может заказывать товары. Впрочем, этой проблемой мы вскоре займемся.



Рис. 6.3. Страница товара с атрибутами

Чтобы понять, как работает код, добавленный нами в проект в этом упражнении, давайте разберемся, как атрибуты отображаются на страницах отдельных товаров и в списке товаров. Прежде всего, посмотрим, как код Smarty анализирует входные данные и какой код он генерирует. Еще раз приведем результаты, возвращаемые хранимой процедурой `catalog_get_product_attributes` для значения ID `1`.

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

А теперь посмотрим, какой исходный HTML-код генерируется для страницы отдельного товара. Вы можете загрузить эту страницу в браузер (ее адрес — `http://localhost/tshirtshop/index.php?ProductId=1`) и

просмотреть исходный HTML-код, щелкнув правой кнопкой мыши на пустом месте страницы и выбрав из контекстного меню пункт View Source (в Internet Explorer) или View Page Source (в Firefox).

Вот что вы обнаружите. Мы удалили из приведенного ниже листинга ненужные элементы, чтобы сделать его более удобочитаемым, и добавили комментарии Smarty, которые помогут вам соотнести элементы шаблона и сгенерированный код.

```
{* Генерируем список значений атрибутов*}
<p class="attributes">

{* Просматриваем список атрибутов и их значений*}
  {* Генерируем новый тег select? *}
      Color:
  <select name="attr_Color">

    {* Генерируем новый тег option *}
    <option value="White">
      White
    </option>

{* Закрываем тег select? *}

{* Генерируем новый тег select? *}

{* Генерируем новый тег option *}

  <option value="Black">
    Black
  </option>

{* Закрываем тег select? *}

{* Генерируем новый тег select? *}

{* Генерируем новый тег option *}

  <option value="Red">
    Red
  </option>

{* Закрываем тег select? *}

{* Генерируем новый тег select? *}

...
...

{* Закрываем тег select? *}
  </select>
{* Генерируем новый тег select? *}
    Size:
  <select name="attr_Size">

{* Генерируем новый тег option *}

  <option value="S">
    S
  </option>

{* Закрываем тег select? *}

{* Генерируем новый тег select? *}
```

```
{* Генерируем новый тег option *}

  <option value="M">
    M
  </option>

...
...
  {* Закрываем тег select? *}
    </select>
</p>
```

Так как же генерируется HTML-код? Ключевым элементом, отвечающим за его генерацию, является управляющая логика в файлах `product.tpl` и `products_list.tpl`. Она просматривает список атрибутов, предоставляемый уровнем логики приложения, т.е. данные, возвращаемые хранимой процедурой `catalog_get_product_attributes`:

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

Для каждого элемента в списке атрибутов код выполняет несколько действий, чтобы определить, какой HTML-код нужно сгенерировать. Для примера возьмем следующий фрагмент кода. Это условный оператор `{if}`, возвращающий значение `true` для первого элемента и для тех элементов, имя атрибута в которых отличается от имени атрибута предыдущего элемента:

```
{* Генерируем новый тег select? *}
{if $smarty.section.k.first ||
    $obj->mProduct.attributes[k].attribute_name !==
    $obj->mProduct.attributes[k.index_prev].attribute_name}
```

Проще говоря, этот оператор возвращает `true` для каждого нового атрибута, найденного им в списке (например, `Size` или `Color`). Когда это происходит, мы генерируем новый тег `<select>`, таким образом создавая новый раскрывающийся список на странице:

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

Остальные операторы работают примерно так же. Просмотрите весь только что добавленный код, и можете поздравить себя — вы успешно добавили еще одну важную функцию в проект электронного магазина!

Резюме

В этой маленькой главе мы добавили в электронный магазин поддержку атрибутов товаров. Теперь посетители смогут выбирать размеры и цвета футболок. Кроме того, вы можете добавить новые атрибуты и их значения по своему усмотрению.

Наш электронный магазин все больше приближается к полностью рабочему состоянию. Однако он будет мало конкурентоспособным в Интернете, если его будут плохо воспринимать поисковые системы. Кроме того, если мы не будем проектировать сайт с учетом предпочтений поисковых систем, он скорее потеряет рейтинг, чем приобретет его. А поскольку переделать уже существующий сайт под эти предпочтения будет сложно, лучше заняться этим как можно раньше во время разработки. Именно этим мы и займемся в следующей главе книги.

Глава 7

Оптимизация для поисковых систем

Добавив в наш сайт поддержку атрибутов товаров, вы, вероятно, хотите сразу же заняться другими возможностями и механизмами — приемом платежей, поиском по каталогу и корзиной покупателя. Скоро мы этим займемся, будьте уверены. Но прежде нужно позаботиться об одном важном моменте: мы должны сделать наш сайт дружественным к поисковым системам. Это важно, поскольку люди будут искать наш сайт с помощью поисковых систем, и чем чаще они будут его находить, тем большими будут наши доходы.

Оптимизация для поисковых систем (Search Engine Optimization — SEO) — это совокупность приемов, позволяющих увеличить приток посетителей на сайт с помощью поисковых систем. Сейчас именно поисковые системы являются основным инструментом, с помощью которого люди ищут информацию и товары в Интернете. И, разумеется, получение хороших рейтингов для вашего сайта в поисковых системах заметно увеличит шансы на то, что покупать товары будут у вас, а не у ваших конкурентов.

В целом оптимизация для поисковых систем — обширная тема, которую мы не сможем (и не будем пытаться) рассмотреть полностью в одной небольшой главе. Мы только модифицируем сайт TShirtShop так, чтобы его архитектура была более дружелюбной к поисковым системам.

ЗАМЕЧАНИЕ

Если вас действительно интересует тема оптимизации для поисковых систем — а она должна вас интересовать, — осмелимся порекомендовать вам несколько неплохих книг на эту тему. Прежде всего, *Search Engine Optimization for Dummies* (Peter Kent; серия For Dummies, 2004) и *Search Engine Optimization: An Hour a Day* (Jennifer Grappone, Gradiva Couzin; Sybex, 2006) содержат хорошие обзоры базовых концепций этой области. Книгу Джейми Сировича и Кристиана Дари *Поисковая оптимизация сайта на PHP для профессионалов. Руководство разработчика* (пер. с англ., изд. "Диалектика", 2008) можно считать исчерпывающим руководством по SEO для языка PHP, в котором все упоминаемые здесь вопросы рассмотрены подробно. Кроме того, попробуйте почитать книги по инженерной психологии и эргономике, например *Prioritizing Web Usability* (Jakob Nielsen, Hoa Loranger; New Riders Press, 2006) или *Don't Make Me Think: A Common Sense Approach to Web Usability* (Steve Krug; New Riders Press, 2005) — они помогут вам выдержать баланс между оптимизацией сайта для поисковых систем и хорошей эргономикой.

Оптимизация в TShirtShop

Вероятно, вы удивитесь, узнав это, но в нашем небольшом проекте есть масса вещей, которые можно модифицировать, чтобы сделать его более дружественным к поисковым системам. К счастью, мы проектировали его таким образом, что отдельные компоненты можно легко менять, не затрагивая структуру сайта в целом.

В этой главе мы сделаем следующее.

- Реализуем URL, содержащие ключевые слова, с помощью переписывания URL в модуле Apache `mod_rewrite`. Это позволит открывать страницы сайта, обращаясь не к файлу `index.php` с цифрами в качестве параметров, а к адресам, понятным и людям, и поисковым системам, наподобие `http://www.example.com/mickey-mouse-t-shirt.html`. Такая модификация будет более простой, чем вам, возможно, кажется.
- Реализуем корректную обработку URL, содержащих ошибки, или URL, указывающих на несуществующие страницы. Это особенно важно, если какие-то URL существовали долгое время, и на них ссылаются веб-страницы извне нашего сайта. Таким образом, мы не потеряем рейтинг, который наш сайт мог уже набрать в поисковых системах для старых страниц. Кроме того, мы гарантируем, что независимо от варианта написания URL — `example.com` или `www.example.com` — поисковые системы все равно поймут, что эти URL относятся к нашему сайту. Такой процесс преобразования разных форм URL в стандартную форму называется канонизацией URL, и чуть позже мы подробно его рассмотрим.
- Организуем перенаправление запросов от `index.php` и `index.html` к `/`. Это важно, поскольку позволит нам избегать повторения содержимого по разным ссылкам. Как вы узнаете немного позже, такое повторение может привести к потере рейтинга в поисковых системах.
- Будем динамически генерировать заголовки страниц, отражающие содержимое этих страниц. На данный момент у всех страниц нашего сайта одинаковые заголовки, и это не лучший вариант и с точки зрения поисковых систем, и с точки зрения людей.
- Реализуем ссылки на все страницы списков с каждой их страницы (а не только на предыдущую и следующую страницы, как было раньше). Это позволит и людям, и поисковым системам быстрее просматривать длинные списки, разбитые на большое количество страниц.
- Начнем использовать коды состояния HTTP 404 (страница не найдена) и HTTP 500 (внутренняя ошибка сервера) для сообщения о проблемах сайта.

Мы еще вернемся к взаимодействию сайта с поисковыми системами, но пока что достаточно перечисленного выше. Правильная структура сайта, безусловно, облегчит нашу задачу в этой главе. В частности, в нем уже реализованы следующие элементы, важные для SEO.

- Мы корректно использовали заголовки страниц и другую разметку, и поисковые системы могут определять, какая из копий страницы важнее.
- На нашем сайте нет страниц с повторяющимся содержимым. В каталоге нет идентичных страниц или фрагментов страниц, которые могут привести к потере поисковых рейтингов.
- Страницы отделов, категорий и товаров легкодоступны.

- Мы не использовали (и не будем использовать в дальнейшем) технологии наподобие Flash и AJAX, которые генерируют непонятное поисковым системам содержимое. Позже в этой книге вы увидите пример использования AJAX, не влияющий на взаимодействие сайта с поисковыми системами.

Поддержка URL с ключевыми словами

Взгляните на два приведенных ниже URL и выберите тот, который вам нравится больше.

```
http://localhost/tshirtshop/index.php?product_id=87
http://localhost/tshirtshop/christmas-tree.html
```

Полагаю, вы предпочтете второй вариант. В нем есть ключевые слова, позволяющие понять, что будет на странице, на которую он указывает. Это важно и для посетителей-людей, и для поисковых систем, просматривающих страницы вашего сайта. URL первого типа мы будем называть *динамическими URL* (dynamic URL), а второго — *URL с ключевыми словами* (keyword-rich URL).

В общем, мы хотим, чтобы наш сайт использовал URL с ключевыми словами. Поскольку мы используем Apache и PHP, этого можно добиться с помощью модуля Apache `mod_rewrite`, который может перехватывать поступающие URL, соответствующие определенным требованиям, и переписывать их в заданной форме, понятной вашему приложению. Этот процесс так и называется — *переписывание URL* (URL rewriting).

Даже при использовании переписывания URL в конечном итоге их обработкой занимается сценарий, например `index.php` на нашем сайте. Но посетители смогут обращаться к этому сценарию с помощью более понятных URL.

На рис. 7.1 показана общая схема, по которой работает механизм переписывания URL. Как видите, модуль `mod_rewrite` перехватывает URL, поступающий от пользователя, и преобразует его в динамический URL, понятный нашему приложению. Затем выполняется PHP-сценарий, и результаты его выполнения передаются пользователю.

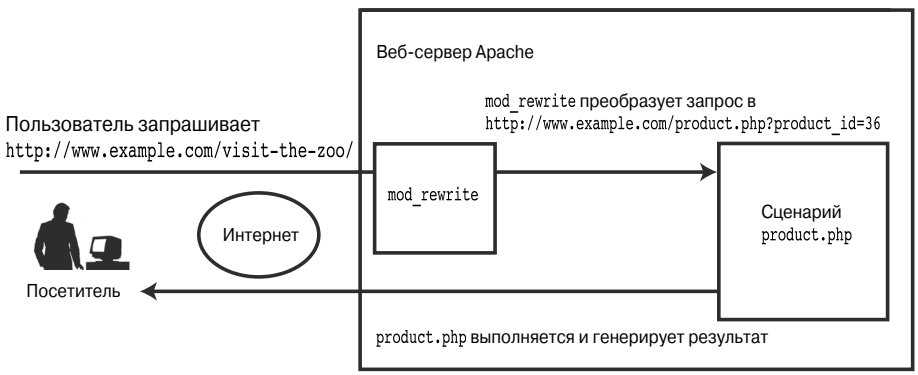


Рис. 7.1. Переписывание URL с помощью модуля Apache `mod_rewrite`

Есть несколько способов переписывания URL с ключевыми словами в динамические URL. В TShirtShop мы воспользуемся способом, одновременно эффективным и простым в реализации: спрячем идентификатор предмета в варианте URL с ключевыми словами, например `http://localhost/tshirtshop/visit-the-zoo-p36/`.

Такой способ позволяет хранить идентификатор в URL, не ухудшая их понимания для людей и поисковых систем.

В этой главе мы реализуем поддержку для вариантов URL, перечисленных в табл. 7.2. В каждой строке этой таблицы `X` обозначает номер отдела, категории или товара, а `P` — номер страницы.

Таблица 7.1. Варианты URL, поддерживаемые в TShirtShop

Тип URL	Формат URL
URL главной страницы	`http://www.example.com/`
URL главной страницы с разбивкой на страницы	`http://www.example.com/page-P/`
URL страницы отдела	`http://www.example.com/department-name-dX/`
URL страницы отдела с разбивкой на страницы	`http://www.example.com/department-name-dX/page-P/`
URL страницы категории	`http://www.example.com/dept-name-dX/cat-name-cX/`
URL страницы категории с разбивкой на страницы	`http://www.example.com/dept-name-dX/cat-name-cX/page-P/`
URL страницы товара	`http://www.example.com/product-name-pX/`

Есть еще кое-какие детали, о которых нужно позаботиться, чтобы обеспечить поддержку таких URL, но всему свое время. В приведенном ниже упражнении мы реализуем поддержку URL с ключевыми словами, а затем разберемся, как она работает.

Упражнение: реализация поддержки URL с ключевыми словами

1. Первое, что нужно сделать, — убедиться, что ваш веб-сервер позволяет использовать модуль `mod_rewrite`. Если вы работаете с веб-сервером, который вам предоставила компания, занимающаяся хостингом, этот модуль, вероятно, уже включен. Но если вы устанавливали Apache сами, вам придется включить его вручную. К счастью, это несложно сделать. Откройте конфигурационный файл Apache, `httpd.conf` и убедитесь, что в начале строки

   ```
   LoadModule rewrite_module modules/mod_rewrite.so
   ```

   нет символа # (этот символ означает, что строка закомментирована). Если такой символ есть, удалите его, сохраните изменения в файле и перезапустите Apache.

ПОДСКАЗКА

Конфигурационный файл Apache по умолчанию находится в папке `C:\xampp\apache\conf\` при установке XAMPP под управлением Windows. При установке под управлением Linux он обычно находится в папке `/opt/lampp/etc/`. Не забудьте перезапустить Apache после внесения изменений в файл `httpd.conf`, чтобы эти изменения вступили в силу. Если при запуске вы получите сообщения об ошибках, просмотрите журналы ошибок Apache, чтобы выяснить, в чем дело.

2. Создайте файл `.htaccess` в корневой папке проекта (`C:\tshirtshop`) и введите в него следующий код (как он работает, мы разберемся позже).

```
<IfModule mod_rewrite.c>
# Включаем mod_rewrite
```

```
RewriteEngine On
# Указываем папку, в которой находится приложение.
# Используем /, если приложение находится в корневой папке.
RewriteBase /tshirtshop
# Переписываем в корректный домен, чтобы избежать проблем
# с канонизацией ссылок
# RewriteCond %{HTTP_HOST} !^www\.example\.com
# RewriteRule ^(.*)$ http://www.example.com/$1 [R=301,L]
# Переписываем URL, заканчивающиеся на /index.php или /index.html в /
RewriteCond %{THE_REQUEST} ^GET\ .*/index\.(php|html?)\ HTTP
RewriteRule ^(.*)index\.(php|html?)$ $1 [R=301,L]
# Переписываем страницы категорий
RewriteRule ^.*-d([0-9]+)/.*-c([0-9]+)/page-([0-9]+)/?$ index.php?
DepartmentId=$1&CategoryId=$2&Page=$3 [L]
RewriteRule ^.*-d([0-9]+)/.*-c([0-9]+)/?$ index.php?DepartmentId=$1&
CategoryId=$2 [L]
# Переписываем страницы отделов
RewriteRule ^.*-d([0-9]+)/page-([0-9]+)/?$ index.php?DepartmentId=$1&
Page=$2 [L]
RewriteRule ^.*-d([0-9]+)/?$ index.php?DepartmentId=$1 [L]
# Переписываем подстраницы
RewriteRule ^page-([0-9]+)/?$ index.php?Page=$1 [L]
# Переписываем страницы с информацией о товарах
RewriteRule ^.*-p([0-9]+)/?$ index.php?ProductId=$1 [L]
</IfModule>
```

ПОДСКАЗКА

Если у вас нет хорошего текстового редактора, создать файл с расширением, но без имени (`.htaccess`) в Windows может оказаться непросто. Проще всего создать такой файл с помощью Notepad — наберите в нем содержимое файла, откройте окно Save As и введите в поле имени файла `".htaccess"`, включая кавычки. Кавычки заставят Notepad не добавлять используемое по умолчанию для его файлов расширение `.txt`.

3. Теперь наш сайт должен поддерживать URL с ключевыми словами в формате, описанном выше. Например, попробуйте открыть в браузере страницу `http://localhost/tshirtshop/nature-d2/`. Результат должен быть примерно таким, как на рис. 7.2.

Как это работает: поддержка URL с ключевыми словами

На данный момент вы можете протестировать работу всех вариантов URL с ключевыми словами, известных нашему веб-сайту: URL отделов и категорий, URL отдельных товаров и URL подстраниц, если какие-то страницы выводятся с разбивкой на подстраницы. Однако вы, вероятно, заметили, что ссылки, которые генерирует сам веб-сайт, остаются обычными динамическими URL. Этот недостаток мы устраним в следующем упражнении, а сейчас давайте разберемся, как URL с ключевыми словами преобразуются в динамические.

Основную роль в переписывании URL играет только что созданный вами файл `.htaccess`. Мы записали в этот конфигурационный файл Apache правила переписывания, которыми будет руководствоваться модуль `mod_rewrite`. Теоретически можно было бы воспользоваться для этого файлом `httpd.conf`, но мы выбрали `.htaccess`, поскольку многие компании, предоставляющие веб-хостинг, не позволили бы вам модифицировать `httpd.conf`. Кроме того, отредактировав файл `.htaccess`, вам не понадобится перезапускать веб-сервер, чтобы изменения вступили в силу, поскольку этот файл просматривается заново при обработке каждого запроса.

Первая строка в файле `.htaccess` задействует систему переписывания URL. Если вы не настроили `mod_rewrite` как полагается, эта строка вызовет ошибку:

```
RewriteEngine On
```



Рис. 7.2. Проверка поддержки URL с ключевыми словами

Далее мы с помощью команды `RewriteBase` задали имя папки сайта TShirtShop. Заметьте, что, если на вашем компьютере этот сайт располагается в корневой папке, вам нужно будет заменить `/tshirtshop` на символ `/`.

```
RewriteBase /tshirtshop
```

А вот теперь начинается самое интересное. Оставшаяся часть файла — это несколько правил `RewriteRule`, указывающих, какие URL должны переписываться и как это переписывание должно выполняться. Иногда `RewriteRule` предваряет команда `RewriteCond`, в которой задается условие для выполнения следующей команды `RewriteRule`.

Команда `RewriteRule` содержит как минимум два параметра. Первый параметр — это *регулярное выражение* (regular expression), описывающее структуру поступающего URL. Второй параметр указывает, в каком виде этот URL должен быть представлен.

Модуль `mod_rewrite` и регулярные выражения

Регулярные выражения — это одна из тем, которые большинство программистов либо обожают, либо ненавидят. Регулярное выражение — это текстовая строка, которая с помощью специальных правил описывает структуру другой строки. Регулярные выражения могут выполнять, например, поиск или преобразование строк в текстах большого объема. Если вам интересно, попробуйте почитать о них, к примеру, в Википедии (`http://en.wikipedia.org/wiki/Regular_expression`).

Регулярные выражения особенно полезны, когда нужно обрабатывать строки заранее известной структуры (например, фрагменты XML-документов), но эти строки трудно просматривать или обрабатывать с помощью более специализиро-

ванных механизмов. Например, с помощью регулярных выражений можно искать в тексте адреса электронной почты или даты, удалять повторяющиеся строки, выяснять, сколько раз какое-то слово или буква встречается в предложении, проверять корректность заданных IP-адресов и т.д.

В предыдущем упражнении мы использовали правила mod_rewrite с регулярными выражениями, чтобы обрабатывать URL с ключевыми словами, заменяя неподдерживаемые символы дефисами и удаляя двойные символы разделения.

Регулярные выражения поддерживаются многими языками программирования и инструментами, включая язык PHP и модуль Apache mod_rewrite, и регулярные выражения обычно можно переносить между программами, написанными на разных языках. Проще говоря, регулярное выражение из программы на PHP в большинстве случаев будет работать точно так же и в программе на C#. Чтобы выполнить какую-то операцию с помощью регулярных выражений, обычно требуются три следующих элемента.

- Исходная строка, которую нужно просмотреть или преобразовать
- Регулярное выражение, которое вы хотите применить к исходной строке
- Тип операции, которую вы хотите выполнить, — поиск подходящих подстрок или их замена чем-либо другим

Регулярные выражения используют особый синтаксис на основе обычных символов, которые воспринимаются буквально, и метасимволов, у которых есть особые свойства. Обычный символ в регулярном выражении соответствует такому же символу в обрабатываемой строке, а последовательность таких символов — такой же последовательности в обрабатываемой строке. Это похоже на простой поиск подстрок в строке. Например, подстрока `or` в строке `favorite color` встречается дважды.

Регулярное выражение может содержать метасимволы, обладающие особым значением, и именно эта возможность делает регулярные выражения такими мощными и гибкими. Например, знак вопроса (`?`) указывает, что предыдущий обычный символ является необязательным. Поэтому регулярное выражение `colou?r` позволит найти в строке текста слова `color` и `colour`.

Как уже упоминалось, регулярные выражения могут стать чрезвычайно запутанными, если начать разбираться в их тонкостях. В этом разделе вы найдете описания элементов регулярных выражений, которые мы будем использовать. Если вам нужны более подробные сведения по этой теме, предлагаем вам поискать соответствующую книгу или руководство в Интернете.

В табл. 7.2 описаны метасимволы, чаще всего используемые в регулярных выражениях. Можете пользоваться этой таблицей как справочником, анализируя правила перезаписи URL.

Таблица 7.2. Метасимволы, часто используемые в регулярных выражениях

Метасимвол	Описание
^	Обозначает начало строки. В нашем случае это начало URL. Имя домена не считается частью URL с точки зрения правил перезаписи. Можете считать символ ^ обозначающим точку, с которой в URL начинаются интересующие нас символы
.	Обозначает один произвольный символ
*	Указывает, что предыдущий символ или выражение может повторяться любое число раз — от нуля до бесконечности
+	Указывает, что предыдущий символ или выражение может повторяться один раз или более — до бесконечности

Окончание табл. 7.2

Метасимвол	Описание
?	Указывает, что предыдущий символ или выражение может встречаться нуль или один раз, т.е. является необязательным
( )	Обозначает выражение. Подстрока, соответствующая этому выражению в исходной строке, может восприниматься как переменная. Кроме того, скобки позволяют группировать их содержимое, как в математических выражениях, и к полученным выражениям могут применяться символы наподобие *, ? или +
[ ]	Обозначает класс символов. Например, [abc] обозначает любой из символов a, b или c. Для определения диапазона символов можно использовать дефис. Например, все латинские строчные буквы можно обозначить как [a-z]. Если дефис должен быть одним из символов класса, он должен располагаться непосредственно перед закрывающей скобкой. Многие метасимволы, будучи заключенными в квадратные скобки, теряют свое особое значение и считаются обычными символами
[^]	Аналогичен [ ], но с отрицанием. Например, [^a-c] обозначает любой символ, кроме a, b и c
$	Обозначает конец строки. В нашем случае это будет конец URL. Можете считать, что он привязывает предшествующие символы к концу строки, т.е. утверждает, что этими символами строка заканчивается
\	Используется для обозначения escape-последовательностей. Позволяет включать в строки метасимволы, используя их как обычные символы. Например, \. будет обозначать именно точку как символ, а не как один любой символ. Чтобы включить в выражение сам символ обратной косой черты, его нужно задать дважды, например, чтобы обозначить строку C:\Windows, ее нужно записать как C:\\Windows

Чтобы понять, как метасимволы используются на практике, давайте рассмотрим одно из правил переписывания URL в TShirtShop — то, которое переписывает адреса страниц категорий. Собственно говоря, таких правил два: одно — для страниц с разбивкой на подстраницы, другое — для страниц без разбивки. Вот правило для страниц с разбивкой на подстраницы (оно выделено полужирным шрифтом):

```
# Перенаправление для страниц категорий
RewriteRule ^.*-d([0-9]+)/.*-c([0-9]+)/page-([0-9]+)/?$
index.php?DepartmentId=$1&CategoryId=$2&Page=$3 [L]
```

Это регулярное выражение соответствует URL наподобие http://localhost/tshirtshop/regional-d1/french-c1/page-2 и выделяет из них идентификаторы отдела и категории, а также номер страницы. Проще говоря, правило ищет строки, начинающиеся с каких-то символов и заканчивающиеся на -d и номер (идентификатор отдела), за которыми следуют прямая косая черта, еще какие-то символы, -c и еще один номер (идентификатор категории), а затем — /page- и последний номер (номер страницы).

Проанализируем это выражение более подробно, используя табл. 7.2 в качестве справочника. Оно начинается с символа ^, соответствующего началу запрашиваемого URL (в URL не входит доменное имя). Метасимволы .* обозначают строку из нуля или более любых символов, поскольку метасимвол . обозначает любой символ, а метасимвол * — повторение предыдущего символа (т.е. любого) произвольное количество раз.

Следующие символы, -d([0-9]+), извлекают номер отдела. Фрагмент [0-9] обозначает любой символ от 0 до 9 (т.е. любую цифру), а + означает, что этот сим-

вол может повториться один или несколько раз. Таким образом, мы выделяем из строки любой номер, не обязательно состоящий из единственной цифры. Круглые скобки вокруг `[0-9]+` означают, что программа должна сохранить выделенную строку (это будет номер отдела) для дальнейшей обработки в переменной `$1`. Эту переменную мы используем для составления переписанного URL.

Точно таким же образом регулярное выражение выделяет из строки идентификатор категории в переменную `$2` и номер страницы в переменную `$3`. И наконец следуют символы `/?$`, означающие, что URL может заканчиваться символом косой черты, но это не обязательно.

ЗАМЕЧАНИЕ

Если вам нужно использовать в регулярном выражении метасимволы как обычные символы, ставьте перед ними символ обратной косой черты (\). Например, чтобы выделить из строки имя файла `index.php`, запишите это имя в регулярном выражении как `index\.php`.

Второй аргумент команды `RewriteRule`, `index.php?DepartmentId=$1&CategoryId=$2&Page=$3`, подставляет переменные, выделенные из предыдущего выражения, в переписываемую URL. При этом вместо имен переменных подставляются их значения, и URL передается нашему приложению.

У правила перезаписи может быть также третий аргумент, состоящий из специальных флагов. Эти флаги влияют на процесс переписывания URL. Они специфичны для команды `RewriteRule` и не относятся к регулярным выражениям. В табл. 7.3 приведены описания этих флагов. Они всегда заключаются в квадратные скобки в конце того правила, к которому должны применяться.

Таблица 7.3. Флаги `RewriteRule`

Флаг	Значение	Название
R	Redirect	Отправляет HTTP-перенаправление
F	Forbidden	Запрещает доступ по URL
G	Gone	Помечает URL как удаленный
P	Proxy	Передает URL модулю `mod_proxy`
L	Last	Предотвращает выполнение последующих правил
N	Next	Начинает выполнение правил заново с начала файла, но с использованием переписанного URL
C	Chain	Сцепляет текущее правило с последующим
T	Type	Явно указывает использование определенного типа MIME
NS	Nosubreq	Применяется, только если нет встроенного подзапроса
NC	Nocase	Проверка URL без учета регистра символов
QSA	Qsappend	Дописывает строку запроса в конец нового URL, а не заменяет ее
PT	Passthrough	Передает переписанный URL другому модулю Apache для дальнейшей обработки
S	Skip	Пропускает следующее правило
E	Env	Задает значение переменной среды

Команды RewriteRule выполняются в том порядке, в котором они записаны в конфигурационном файле. Чтобы гарантировать, что правило, которому будет соответствовать обрабатываемый URL, будет единственным выполненным, задайте в этом правиле флаг [L].

Этот флаг особенно полезен, если у нас будет много правил переписывания, поскольку после обнаружения подходящего правила Apache не будет просматривать остальные правила, и производительность приложения возрастет. В любом случае обычно после выполнения одного правила остальные выполнять не нужно.

Еще одно замечание касается файла .htaccess, а точнее — следующего фрагмента кода.

```
# Переписываем в корректный домен, чтобы избежать проблем
# с канонизацией ссылок
#RewriteCond %{HTTP_HOST} !^www\.example\.com
#RewriteRule ^(.*)$ http://www.example.com/$1 [R=301,L]
```

Как видно из файла .htaccess, команды переписывания URL можно закомментировать, указав перед ними символ #. Мы закомментировали две команды, поскольку вам нужно будет заменить в них www.example.com реальным адресом вашего сайта (пока он расположен на локальной машине, оставьте эти правила закомментированными).

RewriteCond — это команда модуля mod_rewrite, накладывающая условие на следующую за ней команду. В данном случае она проверяет, действительно ли к сайту обратились по адресу www.example.com. Если нет, она выполняет перенаправление с кодом HTTP 301. Это позволяет выполнить канонизацию URL на сайте. Если к вашему сайту будут обращаться по нескольким именам (например, www.example.com и example.com), сделайте одно из них основным доменным именем и организуйте перенаправления на него со всех остальных имен, чтобы избежать потерь рейтинга в поисковых системах из-за повторяющегося содержимого. Более подробно о перенаправлениях мы поговорим позже в этой главе.

Генерация URL с ключевыми словами

В предыдущем упражнении мы реализовали для нашего сайта поддержку URL с ключевыми словами. Однако, как вы, вероятно, заметили, наш сайт:

- поддерживает и URL с ключевыми словами, и динамические URL;
- позволяет ссылаться на свои веб-страницы извне с помощью динамических URL.

Учитывая это, сама по себе поддержка URL с ключевыми словами не дает нашему сайту никаких особых преимуществ. Поэтому нам нужно выполнить еще одну операцию, связанную с URL, — заменить в ссылках сайта динамические URL на URL с ключевыми словами.

В предыдущих главах мы уже создали класс Link, который отвечает за генерацию всех ссылок на сайте. Это значит, что теперь наша задача сводится к обновлению класса Link. Кроме того, нам понадобится создать некоторые новые элементы на уровне данных и уровне логики приложения, чтобы обеспечить поддержку новой функциональности, — нам понадобятся методы, возвращающие названия отделов, категорий и товаров по заданным числовым идентификаторам.

Упражнение: генерация URL с ключевыми словами

1. С помощью phpMyAdmin подключитесь к базе данных `tshirtshop` и выполните приведенный ниже код, создающий три хранимые процедуры. Это простые процедуры, возвращающие названия отдела, категории или товара по заданному числовому идентификатору. Не забудьте задать в качестве разделителя символы `$$`, прежде чем выполнять этот код.

```
-- Создаем хранимую процедуру catalog_get_department_name
CREATE PROCEDURE catalog_get_department_name(IN inDepartmentId INT)
BEGIN
  SELECT name FROM department WHERE department_id = inDepartmentId;
END$$

-- Создаем хранимую процедуру catalog_get_category_name
CREATE PROCEDURE catalog_get_category_name(IN inCategoryId INT)
BEGIN
  SELECT name FROM category WHERE category_id = inCategoryId;
END$$

-- Создаем хранимую процедуру catalog_get_product_name
CREATE PROCEDURE catalog_get_product_name(IN inProductId INT)
BEGIN
  SELECT name FROM product WHERE product_id = inProductId;
END$$
```

2. Теперь нужно добавить код уровня логики приложения, работающий с только что созданными хранимыми процедурами. Добавьте этот код в класс `Catalog` в файле `business/catalog.php`.

```
// Получаем название отдела
public static function GetDepartmentName($departmentId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_department_name(:department_id)';
  // Создаем массив параметров
  $params = array (':department_id' => $departmentId);
  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetOne($sql, $params);
}

// Получаем название категории
public static function GetCategoryName($categoryId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_category_name(:category_id)';
  // Создаем массив параметров
  $params = array (':category_id' => $categoryId);
  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetOne($sql, $params);
}

// Получаем название товара
public static function GetProductName($productId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_product_name(:product_id)';
  // Создаем массив параметров
  $params = array (':product_id' => $productId);
  // Выполняем запрос и возвращаем результаты
```

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

3. Откройте файл `presentation/link.php` и модифицируйте находящийся в нем код следующим образом.

```
public static function ToDepartment($departmentId, $page = 1)
{
  $link = self::CleanUrlText(Catalog::GetDepartmentName($departmentId)) .
          '-d' . $departmentId . '/';

  if ($page > 1)
    $link .= 'page-' . $page . '/';

  return self::Build($link);
}

public static function ToCategory($departmentId, $categoryId, $page = 1)
{
  $link = self::CleanUrlText(Catalog::GetDepartmentName($departmentId)) .
          '-d' . $departmentId . '/' .
          self::CleanUrlText(Catalog::GetCategoryName($categoryId)) .
          '-c' . $categoryId . '/';

  if ($page > 1)
    $link .= 'page-' . $page . '/';

  return self::Build($link);
}

public static function ToProduct($productId)
{
  $link = self::CleanUrlText(Catalog::GetProductName($productId)) .
          '-p' . $productId . '/';

  return self::Build($link);
}

public static function ToIndex($page = 1)
{
  $link = '';

  if ($page > 1)
    $link .= 'page-' . $page . '/';

  return self::Build($link);
}
```

4. Продолжайте работать с классом `Link`. Добавьте в него приведенный ниже метод `CleanUrlText()`, который только что модифицированные вами методы будут вызывать, чтобы удалить из ссылок некорректные символы.

```
// Подготавливает строку к использованию в URL
public static function CleanUrlText($string)
{
  // Удаляем все символы, кроме a-z, 0-9, дефиса,
  // знака подчеркивания и пробела
  $not_acceptable_characters_regex = '#[^-a-zA-Z0-9_ ]#';
  $string = preg_replace($not_acceptable_characters_regex, '', $string);
  // Удаляем все пробелы в начале и в конце строки
  $string = trim($string);
  // Заменяем все дефисы, знаки подчеркивания и пробелы дефисами
```

```
  $string = preg_replace('#[-_ ]+#', '-', $string);
  // Возвращаем подготовленную строку
  return strtolower($string);
}
```

5. Откройте сайт TShirtShop в браузере и обратите внимание на новые ссылки. На рис. 7.3 ссылка на товар Visit The Zoo, `http://localhost/tshirtshop/visit-the-zoo-p36/`, видна в строке состояния Internet Explorer.

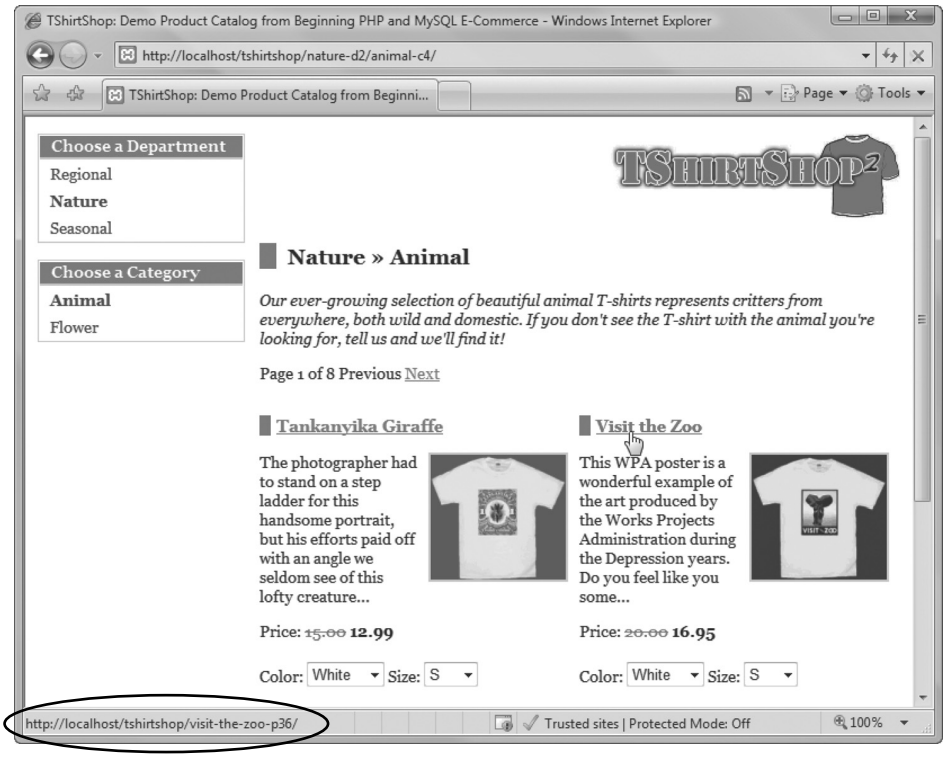


Рис. 7.3. Проверка динамически генерируемых URL с ключевыми словами

Как это работает: генерация URL с ключевыми словами

В этом упражнении мы модифицировали методы `ToIndex()`, `ToDepartment()`, `ToCategory()` и `ToProduct()` класса `Link`, чтобы генерировать URL с ключевыми словами вместо обычных динамических URL. Для поддержки этой возможности мы создали необходимую инфраструктуру (методы уровня логики приложения и хранимые процедуры в базе данных), позволяющую получать названия отделов, категорий и товаров по их числовым идентификаторам.

Кроме того, мы добавили новый метод `CleanUrlText()`, который с помощью регулярных выражений заменяет неподходящие для использования в URL символы дефисами. Например, этот метод преобразует строку `Visit the Zoo` в строку `Visit-the-Zoo`.

Потратьте некоторое время на проверку работы новых ссылок на сайте, и можно будет приступать к следующей нашей задаче в этой главе.

Коррекция URL с помощью перенаправлений

Теперь один из основных недостатков нашего сайта с точки зрения поисковых систем — то, что попасть на одни и те же страницы на нем теоретически можно с помощью разных URL. Взгляните, например, на следующие URL:

```
http://localhost/tshirtshop/nature-d2/
http://localhost/tshirtshop/TYPO-d2/
```

Поскольку наше приложение руководствуется в работе числовыми индексами в URL, а в обоих этих URL индексы равны 2, обе ссылки указывают на отдел Nature, хотя корректная ссылка на него должна иметь вид `http://localhost/tshirtshop/nature-d2/`.

Такая гибкость может привести к неприятным последствиям: если поисковые системы смогут добираться до одних и тех же страниц по разным URL, они посчитают, что на вашем сайте много страниц с одинаковым содержимым, и решат, что вы создали спамовый сайт. В таком случае потеря рейтинга будет неизбежной.

Но даже если ваш сайт не потеряет рейтинг в поисковых системах, этот рейтинг может оказаться разделенным между множеством URL, которые на самом деле ведут на одни и те же страницы. Во избежание такой ситуации лучше использовать коды состояний HTTP для перенаправления всех запросов к страницам по общим URL.

КОДЫ СОСТОЯНИЯ HTTP

Коды состояния HTTP — это стандартные коды, которые отправляются в ответ на запрос к веб-страницам и сообщают результат выполнения этого запроса. Как веб-разработчик вы, вероятно, знакомы с кодами 200 ("запрос успешно выполнен") и 404 ("запрошенный ресурс не найден").

Среди кодов состояния HTTP есть несколько кодов, специально предназначенных для организации перенаправления запросов. Чаще всего для этого используется код 301, указывающий, что запрошенный ресурс *перманентно* перемещен по новому адресу, и код 302, указывающий, что ресурс перемещен *временно*.

Когда веб-браузер (или поисковая система) отправляет запрос и получает ответ с кодом перенаправления, он продолжает работать с полученным новым адресом. В частности, веб-браузер обратится по новому адресу, чтобы загрузить страницу, и ее новый адрес появится в строке адреса.

По умолчанию для перенаправления используется код состояния HTTP 302. Это необходимо знать, поскольку при оптимизации сайта под поисковые системы обычно используется код 301 — теоретически он позволяет перенести рейтинг со старого адреса на новый.

Это значит, что если старая ссылка имела высокий рейтинг и при обращении по ней поисковая система получает в ответ код перенаправления 301, то этот рейтинг будет перенесен на адрес, на который выполнено перенаправление. На самом деле это не всегда так, но в большинстве случаев этот механизм работает.

Более подробно о перенаправлениях и кодах состояния HTTP можно прочитать, например, в книге Джейми Сировича и Кристиана Дари *Поисковая оптимизация сайта на PHP для профессионалов. Руководство разработчика* (пер. с англ., изд. "Диалектика", 2008).

Нашей целью в следующем упражнении будет создание стандартной ("корректной") версии URL для каждой страницы сайта. При загрузке страницы мы будем сравнивать известную корректную версию URL с версией, запрошенной пользователем. Если эти версии не будут совпадать, мы будем использовать перенаправление с кодом 301 на корректную версию URL.

Как уже говорилось, коррекция URL поможет в случае, если кто-то наберет URL с опечаткой, например `http://localhost/tshirtshop/natureTYPO-d2/`, или если мы поменяем название отдела, категории или товара, что приведет к изменению URL.

Упражнение: реализация коррекции URL

1. Коррекция URL и другие возможности, которые мы реализуем в этой главе, потребуют работы с заголовками HTTP. Во избежание проблем при работе с ними нам нужно внести в файл `index.php` некоторые изменения. Добавьте в этот файл строки, выделенные полужирным шрифтом в следующем листинге.

```
<?php
// Активизируем сеанс
session_start();

// Создаем выходной буфер
ob_start();

// Включаем вспомогательные файлы
require_once 'include/config.php';
require_once BUSINESS_DIR . 'error_handler.php';
```

2. В конец файла `index.php` добавьте строки, выделенные полужирным шрифтом в следующем листинге.

```
// Закрываем соединение с базой данных
DatabaseHandler::Close();
// Выводим содержимое буфера
flush();
ob_flush();
ob_end_clean();
?>
```

3. Добавьте в класс `Link` в файле `presentation/link.php` новый метод `CheckRequest()`, код которого приведен ниже.

```
// Выполняет перенаправление по корректному URL в случае необходимости
public static function CheckRequest()
{
  $proper_url = '';

  // Получаем правильный URL для страниц категорий
  if (isset ($_GET['DepartmentId']) && isset ($_GET['CategoryId']))
  {
    if (isset ($_GET['Page']))
      $proper_url = self::ToCategory($_GET['DepartmentId'],
                    $_GET['CategoryId'], $_GET['Page']);
    else
      $proper_url = self::ToCategory($_GET['DepartmentId'],
                    $_GET['CategoryId']);
  }

  // Получаем правильный URL для страниц отделов
  elseif (isset ($_GET['DepartmentId']))
  {
  if (isset ($_GET['Page']))
    $proper_url = self::ToDepartment($_GET['DepartmentId'],
                  $_GET['Page']);
    else
    $proper_url = self::ToDepartment($_GET['DepartmentId']);
```

```
    }

    // Получаем правильный URL для страниц товаров
    elseif (isset ($_GET['ProductId']))
    {
      $proper_url = self::ToProduct($_GET['ProductId']);
    }

    // Получаем правильный URL для главной страницы
    else
    {
      if (isset($_GET['Page']))
        $proper_url = self::ToIndex($_GET['Page']);
      else
        $proper_url = self::ToIndex();
    }

    /* Удаляем виртуальные локации из запрошенного URL,
       чтобы можно было сравнить пути */
    $requested_url = self::Build(str_replace(VIRTUAL_LOCATION, '',
                                             $_SERVER['REQUEST_URI']));

    // Перенаправление с кодом 301 по корректному URL при необходимости
    if ($requested_url != $proper_url)
    {
      // Очищаем буфер вывода
      ob_clean();

      // Выполняем перенаправление по коду 301
      header('HTTP/1.1 301 Moved Permanently');
      header('Location: ' . $proper_url);

      // Очищаем буфер вывода и завершаем работу
      flush();
      ob_flush();
      ob_end_clean();
      exit();
    }
  }
```

4. Откройте файл `index.php` и добавьте в него вызов только что созданного метода.

```
// Загружаем дескриптор базы данных
require_once BUSINESS_DIR . 'database_handler.php';

// Загружаем уровень логики приложения
require_once BUSINESS_DIR . 'catalog.php';

// Коррекция URL
Link::CheckRequest();

// Загружаем файл шаблонов Smarty
$application = new Application();

// Отображаем страницу
$application->display('store_front.tpl');

// Закрываем соединение с базой данных
DatabaseHandler::Close();
```

5. Попробуйте открыть в браузере страницу с адресом `http://localhost/tshirtshop/natureTYPO-d2/` и посмотрите, как ваше приложение выполняет

перенаправление на корректную ссылку — `http://localhost/tshirtshop/nature-d2/`. С помощью инструмента наподобие LiveHTTPHeaders в Firefox (`http://livehttpheaders.mozdev.org/`) можно убедиться, что действительно выполнялось перенаправление с помощью кода состояния HTTP 301 (взгляните на рис. 7.4).

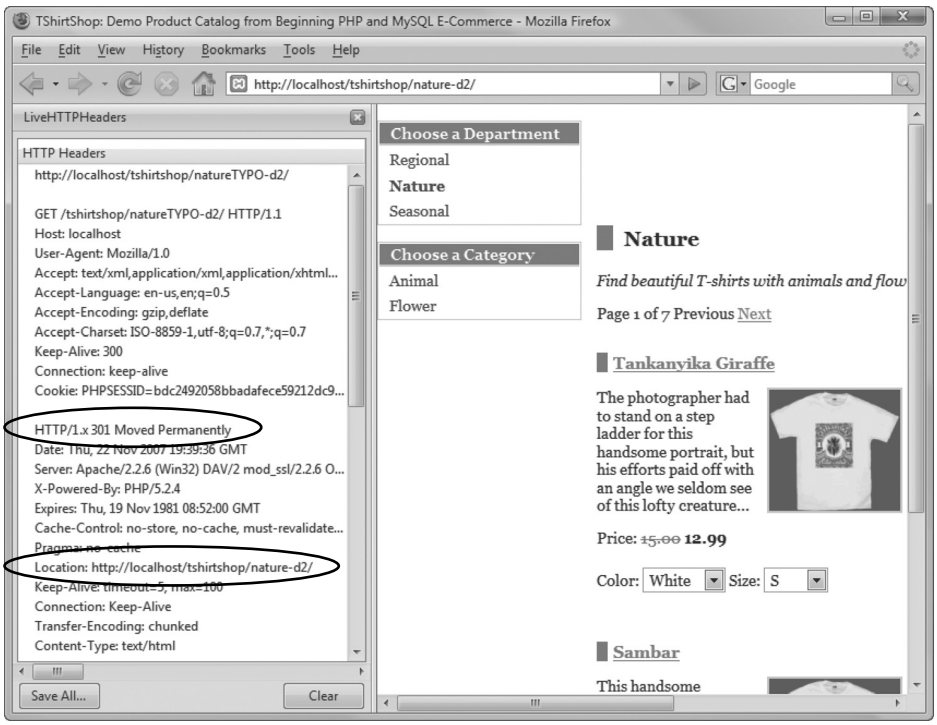


Рис. 7.4. Проверка работы перенаправлений с помощью LiveHTTPHeaders

ЗАМЕЧАНИЕ

Вместо LiveHTTPHeaders можно использовать, например, Web Development Helper and Fiddler для Internet Explorer или Firebug для Firefox.

Как это работает: использование кода состояния HTTP 301 для перенаправления

Написанный нами код выполняет все необходимое с помощью нескольких несложных операций. Метод `CheckRequest()` класса `Link` проверяет, нужно ли перенаправлять поступивший запрос по другому URL, и если нужно, то выполняет перенаправление. В PHP перенаправление осуществляется путем задания HTTP-заголовка, например, так.

```
// Перенаправление с кодом 301 по корректному URL при необходимости
if ($requested_url != $proper_url)
{
  // Очищаем буфер вывода
  ob_clean();

  // Выполняем перенаправление по коду 301
  header('HTTP/1.1 301 Moved Permanently');
```

```
    header('Location: ' . $proper_url);

    // Очищаем буфер вывода и завершаем работу
    flush();
    ob_flush();
    ob_end_clean();
    exit();
  }
}
```

Чтобы гарантировать, что метод `CheckRequest()` будет проверять все поступающие запросы, мы вызываем его в файле `index.php`.

Кроме того, мы добавили в файл `index.php` управление выводом, дабы гарантировать, что мы всегда сможем менять заголовки при необходимости — их невозможно изменить после того, как мы начали передавать данные клиенту. Более подробно функции управления выводом в PHP описаны в документации, доступной по адресу `http://php.net/outcontrol`. Кроме того, есть неплохая статья на эту тему по адресу `http://www.phpit.net/article/output-buffer-fun-php/`.

Модификация заголовков страниц

Одна из распространенных ошибок начинающих веб-дизайнеров — задание одного и того же заголовка для всех страниц веб-сайта. Это отнюдь не лучший вариант, поскольку заголовки играют важную роль в присвоении рейтингов страницам в поисковых системах. Это утверждает, например, статья, доступная по адресу `http://www.seomoz.org/article/search-ranking-factors`.

На данный момент на всех страницах сайта TShirtShop заголовок один и тот же — заданный в файле `site.conf`. Как вы увидите в следующем упражнении, задать отдельные заголовки для разных страниц сайта — задача несложная.

Упражнение: модификация заголовков страниц

1. Откройте файл `presentation/store_front.php` и добавьте в класс `StoreFront` элемент, выделенный полужирным шрифтом в следующем листинге.

```
<?php
class StoreFront
{
  public $mSiteUrl;
  // Определяем файл шаблона для содержимого страницы
  public $mContentsCell = 'first_page_contents.tpl';
  // Определяем файл шаблона для ячейки категорий
  public $mCategoriesCell = 'blank.tpl';
  // Заголовок страницы
  public $mPageTitle;
```

2. В том же классе, `StoreFront`, добавьте в конец метода `init()` код, выделенный полужирным шрифтом в следующем листинге.

```
  // Загружаем сведения о товаре на страницу товара
  if (isset ($_GET['ProductId']))
  $this->mContentsCell = 'product.tpl';

  // Загружаем заголовок страницы
  $this->mPageTitle = $this->_GetPageTitle();
}
```

3. Добавьте в класс `StoreFront` следующий метод.

```
// Возвращает заголовок страницы
private function _GetPageTitle()
{
  $page_title = 'TShirtShop: ' .
    'Demo Product Catalog from Beginning PHP and MySQL E-Commerce';

  if (isset ($_GET['DepartmentId']) && isset ($_GET['CategoryId']))
  {
    $page_title = 'TShirtShop: ' .
      Catalog::GetDepartmentName($_GET['DepartmentId']) . ' - ' .
      Catalog::GetCategoryName($_GET['CategoryId']);

    if (isset ($_GET['Page']) && ((int)$_GET['Page']) > 1)
      $page_title .= ' - Page ' . ((int)$_GET['Page']);
  }
  elseif (isset ($_GET['DepartmentId']))
  {
    $page_title = 'TShirtShop: ' .
      Catalog::GetDepartmentName($_GET['DepartmentId']);

    if (isset ($_GET['Page']) && ((int)$_GET['Page']) > 1)
      $page_title .= ' - Page ' . ((int)$_GET['Page']);
  }
  elseif (isset ($_GET['ProductId']))
  {
    $page_title = 'TShirtShop: ' .
      Catalog::GetProductName($_GET['ProductId']);
  }
  else
  {
    if (isset ($_GET['Page']) && ((int)$_GET['Page']) > 1)
      $page_title .= ' - Page ' . ((int)$_GET['Page']);
  }

  return $page_title;
}
```

4. Вставьте в файл `presentation/templates/store_front.tpl` строку, выделенную полужирным во фрагменте листинга ниже:

```
<html>
  <head>
    <title>{$obj->mPageTitle}</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <link href="{$obj->mSiteUrl}styles/tshirtshop.css" type="text/css"
      rel="stylesheet" />
  </head>
```

5. Откройте в браузере какую-нибудь страницу сайта, кроме главной, и посмотрите на новый заголовок, например такой, как на рис. 7.5 (заголовок главной страницы остался прежним).

Как это работает: создание заголовков страниц

В этом упражнении мы обновили класс `StoreFront`, используя данные, возвращаемые методами `GetDepartmentName()`, `GetCategoryName()` и `GetProductName()` класса `Catalog` для генерации заголовков страниц отделов, категорий и товаров. Кроме того, мы внесли изменения в шаблон Smarty для отображения генерируемого заголовка вместо заголовка по умолчанию. Мы не станем рассматривать весь добавленный код подробно, поскольку он довольно тривиален.

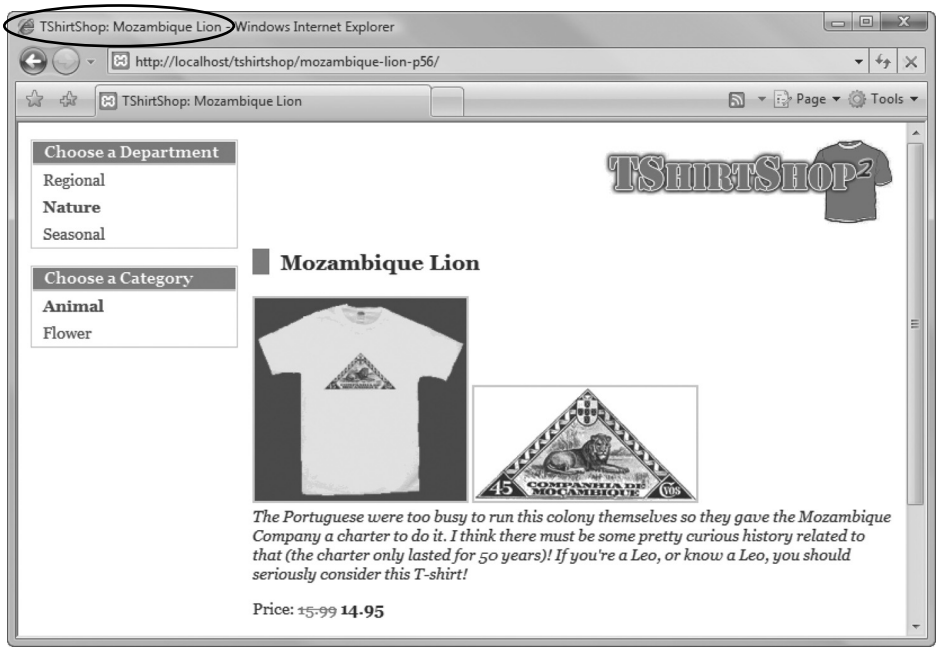


Рис. 7.5. Изменившийся заголовок страницы

Обновление разбивки списков на страницы

Поисковые системы часто считают, что страницы сайтов, на которые мало ссылок с других страниц, менее важны, чем страницы, на которые ссылаются чаще. Между тем реализованный сейчас в нашем приложении механизм навигации таков, что многие страницы погребены в недрах сайта, и чтобы добраться до них, и пользователям, и поисковым системам нужно выполнить множество переходов между страницами. Например, если список товаров разбивается на несколько страниц, то с каждой из этих страниц можно перейти только на предыдущую и последующую страницы списка. Такой подход не слишком удобен как для поисковых систем, так и для посетителей-людей.

Предположим, к примеру, что посетителю нужно добраться до четвертой страницы списка товаров в категории `Regional`. Для этого придется совершить как минимум четыре перехода между страницами.

```
Главная -> Regional -> Страница 1 -> Страница 2 -> Страница 3 -> Страница 4
```

Эту проблему мы устраним в следующем упражнении.

Упражнение: улучшение системы переходов между страницами списков

1. В классе `ProductsList` из файла `presentation/products_list.php` модифицируйте метод `init()` так, как показано в следующем листинге.

```
/* Если список товаров разбит на несколько страниц, отображаем
   навигационные элементы управления */
if ($this->mrTotalPages > 1)
```

```
{
  // Создаем ссылку Next
  if ($this->mPage < $this->mrTotalPages)
  {
    if (isset($this->_mCategoryId))
      $this->mLinkToNextPage =
        Link::ToCategory($this->_mDepartmentId, $this->_mCategoryId,
                         $this->mPage + 1);
    elseif (isset($this->_mDepartmentId))
      $this->mLinkToNextPage =
        Link::ToDepartment($this->_mDepartmentId, $this->mPage + 1);
  }

  // Создаем ссылку Previous
  if ($this->mPage > 1)
  {
    if (isset($this->_mCategoryId))
      $this->mLinkToPreviousPage =
        Link::ToCategory($this->_mDepartmentId, $this->_mCategoryId,
                         $this->mPage - 1);
    elseif (isset($this->_mDepartmentId))
      $this->mLinkToPreviousPage =
        Link::ToDepartment($this->_mDepartmentId, $this->mPage - 1);
  }

  // Создаем ссылки на страницы списка
  for ($i = 1; $i <= $this->mrTotalPages; $i++)
    if (isset($this->_mCategoryId))
      $this->mProductListPages[] =
        Link::ToCategory($this->_mDepartmentId, $this->_mCategoryId, $i);
  elseif (isset($this->_mDepartmentId))
    $this->mProductListPages[] =
      Link::ToDepartment($this->_mDepartmentId, $i);
  else
    $this->mProductListPages[] = Link::ToIndex($i);
}
```

2. Откройте файл `presentation/templates/products_list.tpl` и вставьте в него код, выделенный полужирным шрифтом в листинге ниже:

```
{* products_list.tpl *}
{load_presentation_object filename="products_list" assign="obj"}
{if count($obj->mProductListPages) > 0}
<p>

  {if $obj->mLinkToPreviousPage}
  <a href="{$obj->mLinkToPreviousPage}">Previous page</a>
  {/if}
  {section name=m loop=$obj->mProductListPages}
    {if $obj->mPage eq $smarty.section.m.index_next}
    <strong>{$smarty.section.m.index_next}</strong>
    {else}
    <a href="{$obj->mProductListPages[m]}">
            {$smarty.section.m.index_next}</a>
    {/if}
  {/section}

  {if $obj->mLinkToNextPage}
  <a href="{$obj->mLinkToNextPage}">Next page</a>
  {/if}
```

```
</p>
{/if}
{if $obj->mProducts}
<table class="product-list" border="0">
  <tbody>
  {section name=k loop=$obj->mProducts}
    {if $smarty.section.k.index % 2 == 0}
    <tr>
    {/if}
```

3. Откройте сайт TShirtShop в браузере и перейдите на страницу отдела `Regional`. На ней должны присутствовать ссылки на все страницы списка, как на рис. 7.6.

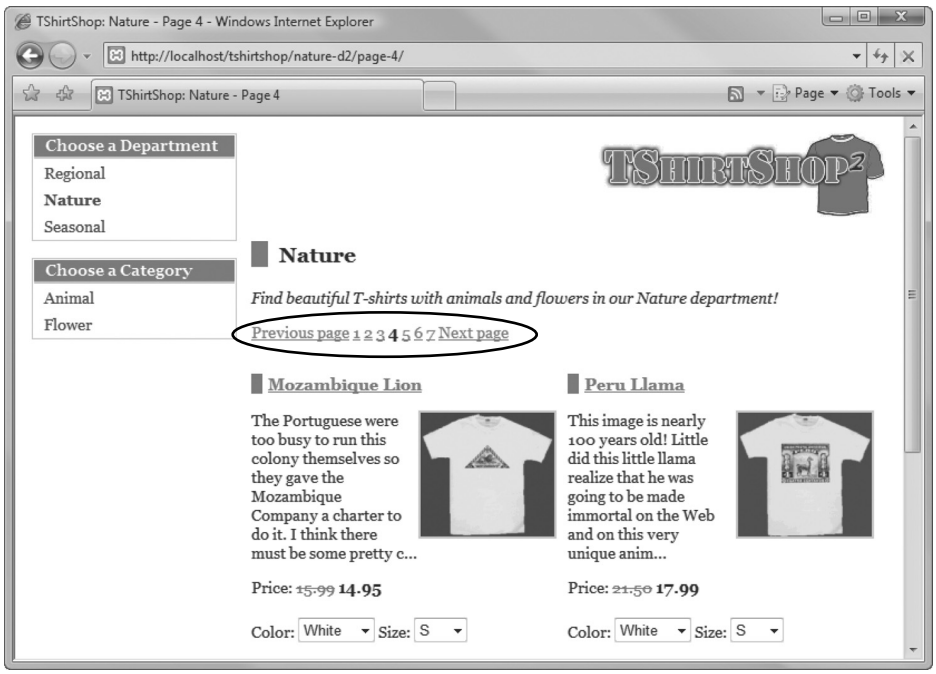


Рис. 7.6. Ссылки на страницы списка

Как это работает: ссылки на страницы списков

Теперь все страницы списков на нашем сайте легкодоступны как для посетителей, так и для поисковых систем. Посетители, безусловно, оценят возможность быстрого перехода к любой странице списка, а поисковым системам будет проще проиндексировать все страницы вашего списка.

Корректные сообщения об ошибках 404 и 500

Если при обработке пользовательского запроса происходит что-то непредвиденное, важно использовать для сообщения об этом правильные коды состояний HTTP. Вы уже видели, как мы используем код HTTP 301 для выполнения перенаправлений. В этом разделе мы обсудим использование кодов HTTP 404 и HTTP 500.

Код состояния 404 предназначен для сообщения о том, что запрошенный ресурс не найден. У браузеров и веб-серверов есть специальные шаблоны, передаваемые пользователям в таких случаях (вы наверняка их видели).

Службы веб-хостинга обычно позволяют вам указывать, какую страницу нужно передавать пользователю при ошибке 404. Это особенно полезно для нашего сайта, поскольку позволяет сообщить пользователю дополнительную информацию исходя из того, какую страницу он просматривал. Однако иногда код 404 не используется по умолчанию, и его использование приходится настраивать вручную. Если по какой-то причине ваш сайт реагирует на ошибку 404, отправляя код состояния HTTP 200 ("все нормально"), поисковые системы могут решить, что на сайте много страниц с идентичным содержимым, и рейтинг вашего сайта снизится.

Код состояния 500 означает, что при обработке запроса произошла внутренняя ошибка приложения на сервере. В следующем упражнении мы модифицируем приложение TShirtShop так, чтобы оно корректно использовало коды 404 и 500.

Упражнение: использование кода состояния HTTP 500

1. Откройте файл `business/error_handler.php` и модифицируйте содержащийся в нем метод `Handler()` так, как показано в следующем листинге.

```
  /* Выполнение не прекращается при предупреждениях,
     если IS_WARNING_FATAL равно false. Ошибки E_NOTICE и E_USER_NOTICE
     тоже не приводят к прекращению выполнения */
  if (($errNo == E_WARNING && IS_WARNING_FATAL == false) ||
      ($errNo == E_NOTICE || $errNo == E_USER_NOTICE))
  // Если ошибка не фатальная...
  {
    // Выводим сообщение, только если DEBUGGING равно true
    if (DEBUGGING == true)
      echo '<div class="error_box"><pre>' . $error_message .
'</pre></div>';
  }
  else
  // Если ошибка фатальная...
  {
    // Выводим сообщение об ошибке
    if (DEBUGGING == true)
      echo '<div class="error_box"><pre>' . $error_message .
'</pre></div>';
    else
    {
      // Очищаем буфер вывода
      ob_clean();

      // Загружаем страницу с сообщением об ошибке 500
      include '500.php';

      // Очищаем буфер вывода и останавливаем выполнение
      flush();
      ob_flush();
      ob_end_clean();
      exit();
    }

    // Останавливаем обработку запроса
    exit();
  }
}
```

2. В корневой папке нашего приложения создайте файл `500.php` и введите в него следующий код.

```
<?php
  // Задаем код состояния 500
  header('HTTP/1.0 500 Internal Server Error');

  require_once 'include/config.php';
  require_once PRESENTATION_DIR . 'link.php';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title>
      TShirtShop Application Error (500): Demo Product Catalog from
      Beginning PHP and MySQL E-Commerce
    </title>
    <link href="<?php echo Link::Build('styles/tshirtshop.css'); ?>"
      type="text/css" rel="stylesheet" />
  </head>
  <body>
    <div id="doc" class="yui-t7">
      <div id="bd">
      <div id="header" class="yui-g">
        <a href="<?php echo Link::Build(''); ?>">
          <img src="<?php echo Link::Build('images/tshirtshop.png'); ?>"
            alt="tshirtshop logo" />
        </a>
      </div>
      <div id="contents" class="yui-g">
        <h1>
          TShirtShop is experiencing technical difficulties.
        </h1>
        <p>
          Please
          <a href="<?php echo Link::Build(''); ?>">visit us</a> soon,
          or <a href="<?php echo ADMIN_ERROR_MAIL; ?>">contact us</a>.
        </p>
        <p>Thank you!</p>
        <p>The TShirtShop team.</p>
        </div>
      </div>
    </div>
  </body>
</html>
```

3. Добавьте выделенные полужирным шрифтом две строки из фрагмента приведенного ниже кода в конец файла `.htaccess`.

```
</IfModule>

# Задает страницу по умолчанию для кода состояния 500
ErrorDocument 500 /tshirtshop/500.php
```

ВНИМАНИЕ

Не забудьте задать в последней строке корректный URL для своего файла `500.php`.

4. Протестируйте файл `500.php`, искусственно создав ошибку на нашем веб-сайте. Откройте файл `include/config.php` и присвойте константе `DEBUGGING` значение `false`, чтобы отключить отладочный режим (в этом режиме сайт не будет генерировать ошибок HTTP 500).

```
// Эти значения должны быть равны true на этапе разработки
define('IS_WARNING_FATAL', true);
define('DEBUGGING', false);
```

5. Откройте файл `index.php` и добавьте в него обращение к несуществующему файлу.

```
// Проверяем URL
Link::CheckRequest();
require_once('inexistent_file.php');
```

6. Попробуйте загрузить наше приложение в браузере. Если все проделано правильно, в окне браузера откроется страница с сообщением об ошибке сервера, как на рис. 7.7.

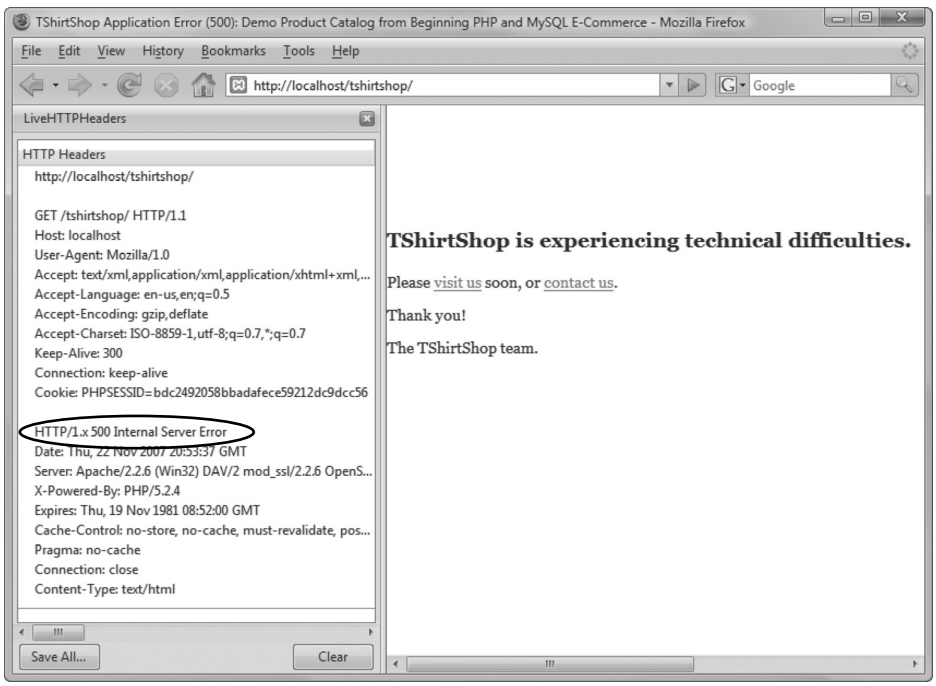


Рис. 7.7. Сообщение об ошибке сервера

Как это работает: обработка ошибок 500

Как видите, теперь наше приложение сообщает пользователю о возникших внутренних ошибках, используя для этого специальную страницу. При этом используется правильный код состояния 500, и поисковые системы смогут понять, что у веб-сайта какие-то проблемы и индексировать полученную страницу не нужно. Вместо этого в индексе будет сохранена предыдущая версия этой страницы, которая, вероятно, содержала корректные данные. Это очень важно, поскольку в противном случае весь наш сайт мог быть удален из индекса — все его страницы были бы заменены страницей с рис. 7.7.

ЗАМЕЧАНИЕ

Прежде чем приступать к следующему упражнению, не забудьте снова присвоить константе `DEBUGGING` значение `true`, чтобы снова включить режим отладки. Кроме того, удалите из файла `index.php` обращение к несуществующему файлу.

Упражнение: использование кода состояния HTTP 404

1. Откройте файл `presentation/link.php` и добавьте в содержащийся в нем метод `CheckRequest()` код, выделенный полужирным шрифтом в следующем листинге.

```
/* Удаляем виртуальные локации из запрошенного URL,
   чтобы можно было сравнить пути */
$requested_url = self::Build(str_replace(VIRTUAL_LOCATION, '',
                                          $_SERVER['REQUEST_URI']));

// Перенаправление с кодом 404, если запрошенный отдел, категория
// или товар не существует
if (strstr($proper_url, '/-'))
{
  // Очищаем буфер вывода
  ob_clean();

  // Загружаем страницу 404
  include '404.php';

  // Очищаем буфер вывода и прекращаем выполнение
  flush();
  ob_flush();
  ob_end_clean();
  exit();
}

// Перенаправление с кодом 301 по корректному URL при необходимости
if ($requested_url != $proper_url)
{
```

2. Откройте файл `presentation/products_list.php` и добавьте в функцию `init()` код, выделенный полужирным шрифтом в следующем листинге.

```
    elseif (isset($this->_mDepartmentId))
      $this->mProductListPages[] =
        Link::ToDepartment($this->_mDepartmentId, $i);
    else
      $this->mProductListPages[] = Link::ToIndex($i);
}

/* Перенаправление с кодом 404, если номер запрошенной страницы
   больше общего числа страниц списка */
if ($this->mPage > $this->mrTotalPages)
{
  // Очищаем буфер вывода
  ob_clean();

  // Загружаем страницу 404
  include '404.php';

  // Очищаем буфер вывода и прекращаем выполнение
```

```
  flush();
  ob_flush();
  ob_end_clean();
  exit();
}

// Генерируем ссылки на страницы товаров
for ($i = 0; $i < count($this->mProducts); $i++)
{
```

3. В корневой папке приложения создайте файл `404.php` и введите в него следующий код.

```
<?php
  // Задаем код состояния HTTP 404
  header('HTTP/1.0 404 Not Found');

  require_once 'include/config.php';
  require_once PRESENTATION_DIR . 'link.php';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title>
      TShirtShop Page Not Found (404): Demo Product Catalog from
      Beginning PHP and MySQL E-Commerce
    </title>
    <link href="<?php echo Link::Build('styles/tshirtshop.css'); ?>"
    type="text/css" rel="stylesheet" />
  </head>
  <body>
    <div id="doc" class="yui-t7">
      <div id="bd">
        <div id="header" class="yui-g">
          <a href="<?php echo Link::Build(''); ?>">
            <img src="<?php echo Link::Build('images/tshirtshop.png'); ?>"
              alt="tshirtshop logo" />
          </a>
        </div>
        <div id="contents" class="yui-g">
          <h1>
            The page that you have requested doesn't exist on TShirtShop.
          </h1>
          <p>
            Please visit the
            <a href="<?php echo Link::Build(''); ?>">TShirtShop catalog</a>
            if you're looking for T-shirts,
            or <a href="<?php echo ADMIN_ERROR_MAIL; ?>">email us</a>
            if you need further assistance.
          </p>
          <p>Thank you!</p>
          <p>The TShirtShop team.</p>
        </div>
      </div>
    </div>
  </body>
</html>
```

4. Добавьте выделенные полужирным шрифтом строки из фрагмента приведенного ниже кода в конец файла `.htaccess`.

```
# Задает страницу по умолчанию для кода состояния 500
ErrorDocument 500 /tshirtshop/500.php

# Задает страницу по умолчанию для кода состояния 404
ErrorDocument 404 /tshirtshop/404.php
```

ВНИМАНИЕ

Не забудьте задать в добавленной строке правильный путь к своему файлу 404.php.

5. Попробуйте открыть в браузере страницу `http://localhost/tshirtshop/seasonal-d3/page-5/`. Поскольку в отделе `Seasonal` список товаров состоит из четырех страниц, сайт должен открыть страницу для кода состояния 404, как на рис. 7.8.

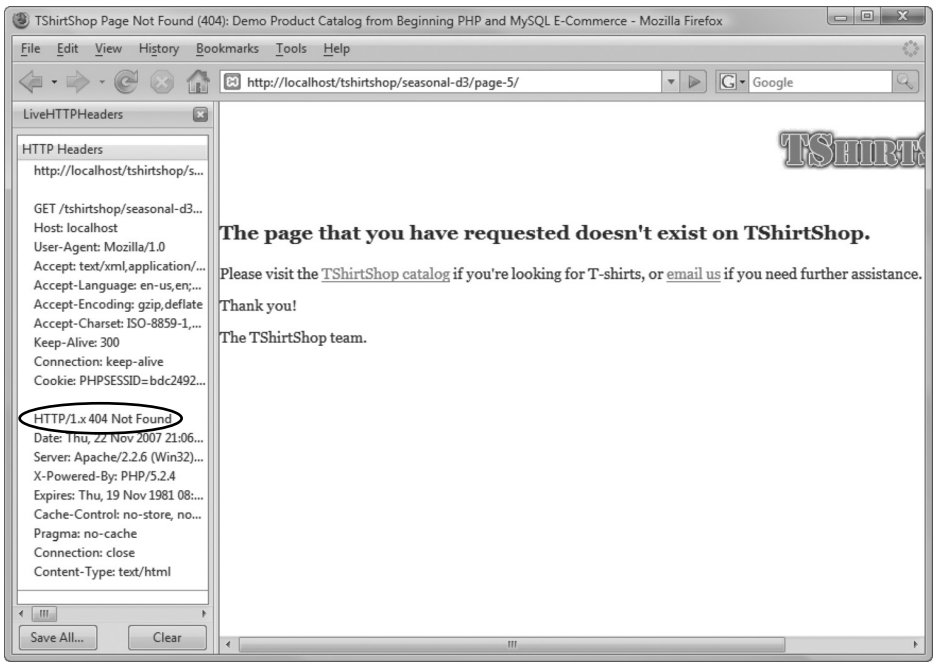


Рис. 7.8. Сообщение о том, что ресурс не найден

Как это работает: коды состояний HTTP 404 и 500

В двух последних упражнениях мы задействовали в нашем сайте коды состояний 404 и 500, модифицировав файл `.htaccess` и код на PHP. Ценность кода 404 будет, вероятно, более очевидна для вас. Если пользователь запрашивает несуществующую страницу, например страницу списка с номером, превышающим количество страниц в списке, мы отображаем страницу 404. Впрочем, использование обоих кодов состояний продемонстрировано в соответствующих упражнениях (см. рис. 7.7 и 7.8).

Резюме

В этой главе мы внесли в наше приложение малозаметные, но весьма важные изменения. Теперь сайт TShirtShop гораздо лучше приспособлен к взаимодействию с поисковыми системами. Однако, разумеется, мы сделали далеко не все, что можно сделать для повышения его рейтингов.

В последующих главах, добавляя в приложение новую функциональность, мы всегда будем стараться следовать правилам SEO, чтобы поисковые системы были нашими друзьями, а не врагами.

Следующая глава будет посвящена реализации механизма, необходимого каждому серьезному электронному магазину, — механизма поиска товаров.

Глава 8

Поиск в каталоге

"Ищете что-нибудь?" Этот вопрос вам часто задают, когда вы заходите в обычный магазин. Предложив посетителям эффективные механизмы поиска нужных товаров, вы можете заметно увеличить доход от электронного магазина. В этой главе мы добавим в наш проект механизмы поиска товаров, позволяющие посетителям легко находить то, что им требуется.

Как вы вскоре убедитесь, реализовать механизм поиска не представляет сложностей — для этого достаточно интегрировать в уже существующую архитектуру сайта несколько новых элементов. В этой главе мы

- проанализируем несколько возможных методов поиска в каталогах;
- создадим необходимые для поддержки поиска структуры данных в MySQL;
- напишем код уровня данных и уровня логики приложения, обеспечивающий работу поиска;
- с помощью компонентных шаблонов Smarty добавим на страницы сайта элементы интерфейса, используемые для поиска.

Методы поиска в каталоге

Как всегда, прежде чем начинать писать код, нужно кое-что обдумать. Добавляя новые элементы в проект, необходимо проанализировать их с точки зрения пользователя.

Визуальным представлением механизма поиска на нашем сайте будет текстовое поле, в котором пользователь сможет вводить ключевые слова. Поиск по этим ключевым словам будет выполняться в названиях и описаниях товаров. Собственно поиск может выполняться в нескольких вариантах.

- *Поиск точного совпадения* (exact-match search). Если пользователь введет строку поиска более чем из одного слова, поиск будет выполняться именно по введенной строке, без ее разбиения на слова и поиска по каждому слову в отдельности.
- *Поиск по всем словам* (all-words search). Введенная пользователем поисковая строка разбивается на отдельные слова, и выполняется поиск товаров, в названиях и/или описаниях которых встречаются все эти слова. В отличие от поиска точного совпадения, в этом варианте порядок следования слов не важен.
- *Поиск любого слова* (any-words search). В этом варианте мы ищем товары, в названиях и/или описаниях которых встречается хотя бы одно слово из поисковой строки.

Разумеется, эта простая классификация не претендует на полноту. Поисковая система на нашем сайте может быть столь же изощренной, как и используемые в Интернете, с множеством настроек и дополнительных возможностей. С другой стороны, мы можем сделать ее максимально простой, например ограничиться поиском в базе данных введенной пользователем строки.

Сайт TShirtShop будет поддерживать режимы поиска по всем словам и поиска любого слова. Мы не станем реализовывать поиск точных совпадений, поскольку от него нет особой пользы на сайтах такого рода.

Приняв это решение, мы можем оформить интерфейс поискового механизма нашего сайта так, как показано на рис. 8.1.



Рис. 8.1. Интерфейс поискового механизма на странице сайта

Кроме текстового поля и кнопки запуска поиска, нам понадобится добавить на страницу только один флажок. Этот флажок позволит посетителям выбирать между поиском всех слов и поиском любого слова.

Нам также нужно решить, как мы будем сообщать посетителю результаты поиска. Как должна выглядеть страница с этими результатами? В конце концов, на ней должен быть хотя бы список товаров, соответствующих заданным критериям.

Самый простой способ отображения такого списка — воспользоваться компонентным шаблоном `products_list`, который мы создали ранее. В таком случае страница с результатами поиска будет выглядеть так, как на рис. 8.2.

Как видно на рис. 8.2, в поле адреса отображается URL страницы с результатами поиска. Вообще говоря, это скорее дополнительное удобство для пользователя, чем функция поисковой системы, поскольку мы не позволим поисковым системам из Интернета просматривать веб-страницы с результатами поиска на нашем сайте. Так мы сможем избежать дублирования содержимого и потерь рейтинга. Однако мы не запрещаем пользователям создавать закладки на страницы с результатами поиска или редактировать их URL.

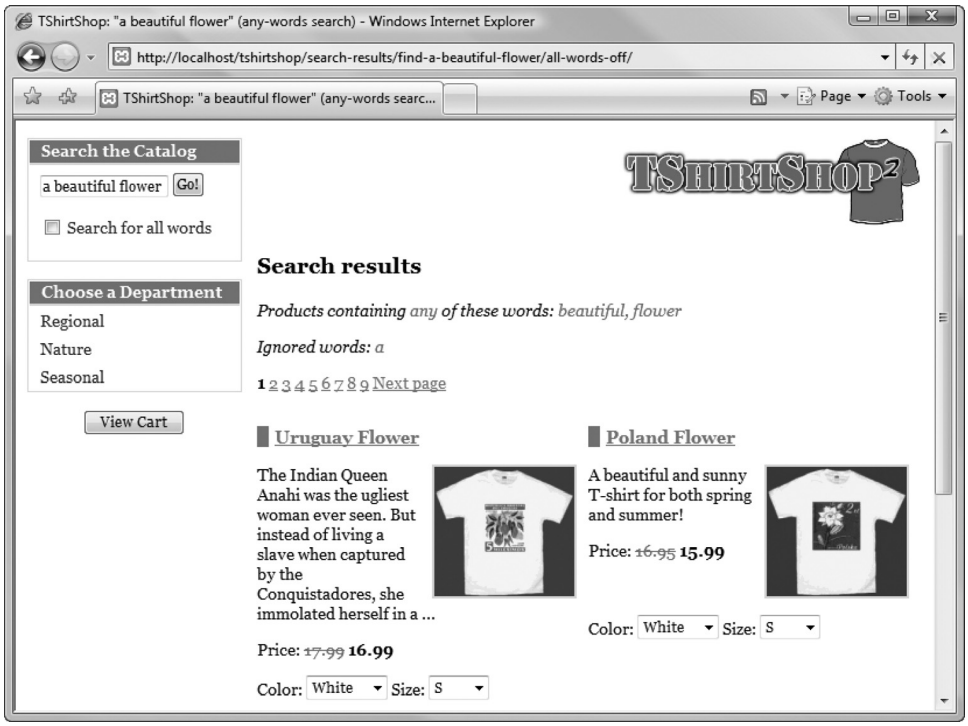


Рис. 8.2. Пример страницы с результатами поиска

Еще один момент, на который стоит обратить внимание на рис. 8.2, — разбиение списка результатов поиска на страницы. Если результатов много, то на каждой странице будет отображаться фиксированное их количество и пользователь сможет перемещаться между страницами списка с помощью навигационных ссылок.

Реализовывать новую функциональность мы начнем, как обычно, с уровня данных.

Реализация поиска в базе данных

Есть два основных способа реализации поиска в базе данных.

- С помощью ключевых слов `WHERE` и `LIKE`
- С помощью механизма полнотекстового поиска в MySQL

Давайте рассмотрим каждый из них.

Поиск с помощью ключевых слов `WHERE` и `LIKE`

Прямолинейная и часто используемая реализация механизма поиска выполняется с помощью ключевого слова `LIKE` в разделе `WHERE` оператора `SELECT`. Рассмотрим простой пример оператора. Этот оператор будет возвращать товары, в описаниях которых есть слово `flower`:

```
SELECT name FROM product WHERE description LIKE '%flower%'
```

Оператор `LIKE` сравнивает пары строк, а символ процента (`%`) обозначает любую строку из нуля или более символов. Поэтому в нашем примере строка `%flower%`

обозначает любой текст, в котором встречается слово `flower`. Этот вариант поиска нечувствителен к регистру символов.

Запрос для извлечения из базы данных сведений обо всех товарах, в названиях или описаниях которых есть слово `flower`, может выглядеть так:

```
SELECT name
FROM   product
WHERE  description LIKE '%flower%' OR name LIKE '%flower%';
```

Преимущество такого способа поиска состоит в том, что он работает с любыми таблицами в MySQL (включая InnoDB). Но у него есть и три недостатка.

- *Скорость.* При таком способе поиска каждый запрос приводит к просмотру всей таблицы товаров. Это так называемый полнотабличный поиск, и база данных не может его ускорить с помощью каких-нибудь индексов. Соответственно, производительность заметно снизится, особенно если в базе данных очень много товаров.
- *Качество результатов поиска.* Этот способ усложняет добавление дополнительных возможностей поиска, например сортировку результатов по критериям релевантности.
- *Дополнительные возможности поиска.* Этот способ не позволит посетителям задавать поиск с булевыми операторами (AND или OR), поиск по похожим словам и т.д.

Так как же реализовать более эффективный поиск? Если у нас большая база данных, а поиск нужно выполнять часто, можно ли не довести сервер до "инфаркта" постоянной перезагрузкой?

Да, можно. Для этого существуют возможности полнотекстового поиска в MySQL.

Полнотекстовый поиск в MySQL

Как говорилось выше, полнотабличный поиск с помощью ключевого слова `LIKE` очень неудобен, если операции поиска должны выполняться часто. Если вы ищете одно слово в описаниях товаров, такой поиск переберет описания всех товаров, выбирая из них подходящие. При большом количестве запросов на поиск сервер базы данных просто захлебнется.

ПОДСКАЗКА

Обычно табличные индексы, относящиеся к текстовым столбцам (например, типа `varchar`), позволяют повысить производительность поисковых операций, если идет поиск точного значения или строки, начинающейся с определенных символов. Это потому, что индекс обычно составляется из упорядоченных значений, например, отсортированных по алфавиту. Такой индекс помогает в поиске строк с известным началом, но он ничем нам не поможет, если нужно найти слово в середине строки.

К счастью, в MySQL есть так называемые `FULLTEXT`-индексы, специально предназначенные для организации поиска в текстах нужных фрагментов. Эти индексы похожи на обычные, но позволяют просматривать все содержимое текстовых строк (например, названий и описаний товаров).

`FULLTEXT`-индекс резко ускоряет операции поиска в тексте нужных слов (или наборов слов). Он позволяет выполнять поиск без просмотра таблиц целиком, как при использовании ключевого слова `LIKE`.

Полнотекстовый поиск в MySQL гораздо удобнее и быстрее, чем описанный выше метод (использующий ключевое слово `LIKE`). Вот его основные преимущества.

- Результаты поиска упорядочиваются по их релевантности.
- Короткие слова игнорируются. Слова длиной меньше четырех букв (например, `a`, `of`, `and`) по умолчанию удаляются из поискового запроса.
- Дополнительные возможности, в частности — поддержка булевых операторов. Например, поисковая строка `+beautiful+flower` означает поиск записей, в которых есть одновременно слова `beautiful` и `flower`.
- Ускорение поиска. Специальные индексы позволяют выполнять поиск гораздо быстрее, чем ключевое слово `LIKE`.

ЗАМЕЧАНИЕ

Более подробно механизм полнотекстового поиска в MySQL описан в статье, доступной по адресу `http://dev.mysql.com/doc/refman/5.1/en/fulltext-search.html`.

Как упоминалось в главе 4, основной недостаток полнотекстового поиска заключается в том, что он работает только с таблицами типа MyISAM. Альтернативный тип таблиц — InnoDB — поддерживает внешние ключи, ACID-транзакции и другие дополнительные возможности, но не поддерживает полнотекстовый поиск.

ЗАМЕЧАНИЕ

ACID — это аббревиатура, описывающая четыре основные характеристики транзакций (Atomicity, Consistency, Isolation, Durability). В этой книге рассматривать транзакции мы не будем, но вы можете сами найти хорошие статьи и книги, например *The Programmers Guide to SQL* (Apress, 2003). Главу из этой книги, посвященную транзакциям, можно бесплатно загрузить по адресу `http://cristiandarie.ro/downloads/`.

А сейчас мы подробнее рассмотрим создание `FULLTEXT`-индексов в базе данных и их использование для поиска в каталоге.

Создание структур данных, обеспечивающих поиск

В нашем проекте мы будем выполнять поиск в таблице `product`, поскольку посетителей интересуют именно товары. Прежде чем в этой таблице можно будет создать `FULLTEXT`-индексы, нужно убедиться, что она относится к типу MyISAM (если вы выполняли все инструкции из предыдущих глав, она будет относиться к этому типу). Если вы создали ее как таблицу другого типа, можете преобразовать ее в тип MyISAM, подключившись к базе данных `tshirtshop` и выполнив следующий оператор:

```
ALTER TABLE product ENGINE = MYISAM;
```

Чтобы обеспечить возможность поиска в таблице `product`, к ней нужно добавить полнотекстовый индекс для пары столбцов (`name`, `description`). Вот как это делается.

1. Запустите phpMyAdmin, откройте базу данных `tshirtshop` и перейдите на вкладку SQL.
2. На вкладке SQL введите следующий оператор, добавляющий полнотекстовый индекс с именем `idx_ft_product_name_description`:

```
-- Создание полнотекстового индекса для поиска
CREATE FULLTEXT INDEX `idx_ft_product_name_description`
ON `product` (`name`, `description`);
```

После щелчка на кнопке Go вы должны получить сообщение об успешном выполнении оператора.

Поскольку нам нужно, чтобы TShirtShop позволял посетителям искать товары с заданными словами в названиях или описаниях, мы создали полнотекстовый индекс для пары столбцов (`name`, `description`) в таблице `product`. Это не то же самое, что создать отдельные полнотекстовые индексы для каждого из столбцов.

Полнотекстовый индекс позволит нам выполнять полнотекстовый поиск по индексированным полям. Чтобы убедиться в существовании только что созданного индекса, в phpMyAdmin перейдите на вкладку Structure и щелкните на значке Structure таблицы `product`. В разделе Indexes открывшегося окна (рис. 8.3) будет виден новый индекс типа `FULLTEXT` для столбцов `name` и `description`.

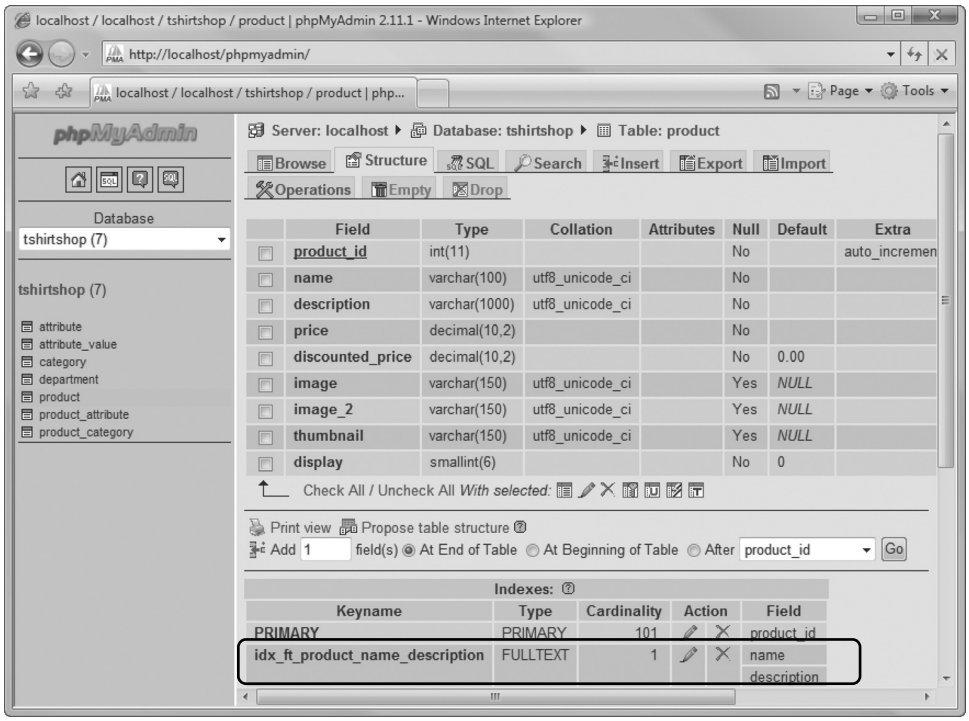


Рис. 8.3. `FULLTEXT`-индекс в окне phpMyAdmin

ПОДСКАЗКА

Как видите, phpMyAdmin явно сообщает, что новый `FULLTEXT`-индекс относится к двум столбцам сразу.

Организация поиска любого из слов в MySQL

Стандартный синтаксис SQL для выполнения полнотекстового поиска выглядит так:

```
SELECT <список столбцов>
FROM <таблица>
WHERE MATCH <столбец или список столбцов> AGAINST <критерии поиска>
```

ПОДСКАЗКА

Официальную документацию по полнотекстовому поиску в MySQL можно найти по адресу `http://dev.mysql.com/doc/refman/5.1/en/fulltext-search.html`.

Столбцы, в которых вы хотите выполнить полнотекстовый поиск, должны быть снабжены полнотекстовыми индексами. Если вы используете группу столбцов, должен существовать индекс для этой группы столбцов, например мы создали индекс `idx_ft_product_name_description` для пары столбцов (`name`, `description`).

Как использовать этот индекс для поиска любого из слов в названиях и описаниях товаров? Предположим, что вы хотите искать слова `beautiful` и/или `flower` в столбцах `name` и `description`. Это можно сделать с помощью следующего оператора SQL:

```
SELECT name, description FROM product
WHERE MATCH (name, description) AGAINST ("beautiful flower");
```

Выполнение этого оператора вернет 33 записи, если база данных заполнена тестовыми данными с сайта книги.

При выполнении таких операций поиска обычно требуется отсортировать результаты в порядке убывания их релевантности. Это можно сделать, воспользовавшись разделом `ORDER BY` и задав правило `MATCH` в качестве аргумента. Опция `DESC` задает упорядочение по убыванию, чтобы наиболее релевантный результат был первым в списке.

```
SELECT name, description FROM product
WHERE MATCH (name, description) AGAINST ("beautiful flower")
ORDER BY MATCH (name, description) AGAINST ("beautiful flower") DESC
```

Этот запрос также вернет 33 записи, но они будут расположены в другом порядке. Первые несколько записей показаны на рис. 8.4. Таблица, представленная на этом рисунке, сгенерирована с помощью phpMyAdmin; чтобы получить похожую, выполните запрос, а затем щелкните на ссылке Print view (with full texts) в нижней части страницы phpMyAdmin.

name	description
Uruguay Flower	The Indian Queen Anahi was the ugliest woman ever seen. But instead of living a slave when captured by the Conquistadores, she immolated herself in a fire and was reborn the most beautiful of flowers: the ceibo, national flower of Uruguay. Of course, you won't need to burn to wear this T-shirt, but you may cause some pretty hot glances to be thrown your way!
Poland Flower	A beautiful and sunny T-shirt for both spring and summer!
Costa Rica Flower	This national flower of Costa Rica is one of our most beloved flower T-shirts (you can see one on Jill, above). You will surely stand out in this T-shirt!
Afghan Flower	This beautiful image was issued to celebrate National Teachers Day. Perhaps you know a teacher who would love this T-shirt?
Albania Flower	Well, these crab apples started out as flowers, so that's close enough for us! They still make for a uniquely beautiful T-shirt.
Romania Flower	Also known as the spring pheasant's eye, this flower belongs on your T-shirt this summer to help you catch a few eyes.
Israel Flower	This plant is native to the rocky and sandy regions of the western United States, so when you come across one, it really stands out. And so will you when you put on this beautiful T-shirt!
Bulgarian Flower	For your interest (and to impress your friends), this beautiful stamp was issued to honor the George Dimitrov state printing works. You'll need to know this when you wear the T-shirt.
Congo Flower	The Congo is not at a loss for beautiful flowers, and we've picked a few of them for your T-shirts.
Austria Flower	Have you ever had nasturtiums on your salad? Try it--they're almost as good as having them on your T-shirt!
Ghana Flower	This is one of the first gingers to bloom in the spring--just like you when you wear this T-shirt!

Рис. 8.4. Пример результатов поиска

При сортировке по убыванию релевантности, например, записи, в которых встречаются одновременно слова `beautiful` и `flower`, окажутся выше в списке, чем записи, в которых встречается только одно из этих слов.

Тонкая настройка полнотекстового поиска в MySQL: минимальная длина слов

По умолчанию слова из менее чем четырех символов не индексируются (и, соответственно, не учитываются в поиске), но при желании вы можете это изменить. Минимальная длина слов, учитываемых в FULLTEXT-индексах, задается серверной переменной ft_min_word_len.

Например, если вы хотите, чтобы в индексах учитывались слова из трех символов, задайте значение переменной ft_min_word_len в файле конфигурации сервера MySQL:

```
[mysqld]
  ft_min_word_len=3
```

Конфигурационный файл, в котором хранится эта переменная, — это обычно /opt/lampp/etc/my.cnf под управлением UNIX и C:\xampp\mysql\bin\my.cnf или C:\Windows\php.ini под управлением Windows. Подробные инструкции по изменению настроек полнотекстового поиска можно найти в статье, доступной по адресу http://dev.mysql.com/doc/refman/5.0/en/fulltext-fine-tuning.html.

Изменив значение переменной ft_min_word_len, вы должны перезапустить сервер MySQL, чтобы изменение вступило в силу. Перезапустив сервер, вы можете запросить у него значение переменной, чтобы убедиться, что изменение прошло успешно. Для этого выполните оператор

```
SHOW VARIABLES LIKE 'ft_%';
```

После изменения значения переменной ft_min_word_len вам придется перестроить FULLTEXT-индексы. Это можно сделать, либо удалив индекс и создав его заново, либо выполнив оператор

```
REPAIR TABLE product QUICK;
```

Заметьте, что выполнять оператор REPAIR нужно только для таблиц, для которых существуют FULLTEXT-индексы. Если по какой-то причине вы хотите вместо этого создать индекс заново (мы все же рекомендуем применять оператор REPAIR), используйте такие операторы.

```
ALTER TABLE product
  DROP INDEX idx_ft_product_name_description;

CREATE FULLTEXT INDEX idx_ft_product_name_description
  ON product (name, description);
```

Организация поиска всех слов в MySQL

Как вы уже видели, поиск по любому из слов возвращает все товары, в названиях или описаниях которых встречается *хотя бы одно* из заданных слов (в нашем примере — beautiful *или* flower). В результатах поиска всех слов должны содержаться только товары, в названиях или описаниях которых встречаются *все* заданные слова (в нашем примере — beautiful *и* flower). Для организации такого поиска нам понадобится булев режим механизма полнотекстового поиска, позволяющий использовать операторы AND/OR в критериях поиска.

Новый запрос будет выглядеть следующим образом.

```
SELECT   name, description FROM product
WHERE    MATCH (name, description) AGAINST ("+beautiful +flower"
           IN BOOLEAN MODE)
ORDER BY MATCH (name, description) AGAINST ("+beautiful +flower"
           IN BOOLEAN MODE)
DESC;
```

Сортировка в порядке убывания не обязательна, но она позволяет упорядочить результаты поиска по релевантности. Знак + обозначает требуемые слова, поэтому его нужно ставить перед каждым словом при поиске всех слов. В отличие от поиска по любому из слов этот запрос вернет всего семь записей, как показано на рис. 8.5.

name	description
Afghan Flower	This beautiful image was issued to celebrate National Teachers Day. Perhaps you know a teacher who would love this T-shirt?
Albania Flower	Well, these crab apples started out as flowers, so that's close enough for us! They still make for a uniquely beautiful T-shirt.
Bulgarian Flower	For your interest (and to impress your friends), this beautiful stamp was issued to honor the George Dimitrov state printing works. You'll need to know this when you wear the T-shirt.
Congo Flower	The Congo is not at a loss for beautiful flowers, and we've picked a few of them for your T-shirts.
Israel Flower	This plant is native to the rocky and sandy regions of the western United States, so when you come across one, it really stands out. And so will you when you put on this beautiful T-shirt!
Poland Flower	A beautiful and sunny T-shirt for both spring and summer!
Uruguay Flower	The Indian Queen Anahi was the ugliest woman ever seen. But instead of living a slave when captured by the Conquistadores, she immolated herself in a fire and was reborn the most beautiful of flowers: the ceibo, national flower of Uruguay. Of course, you won't need to burn to wear this T-shirt, but you may cause some pretty hot glances to be thrown your way!

Рис. 8.5. Результаты поиска всех слов

Тонкая настройка полнотекстового поиска в MySQL: расширенные запросы

Возможность расширения запроса при полнотекстовом поиске появилась в MySQL 4.1.1. Она позволяет находить не только слова, заданные в поисковой строке, но и родственные им.

Если выполняется поиск с расширенным запросом, то на самом деле MySQL выполняет две операции поиска. Сначала он находит слова, наиболее релевантные относительно тех, которые есть в поисковой строке. Затем найденные слова добавляются к поисковой строке, и выполняется поиск в базе данных.

Этот метод позволяет получить при поиске большее количество результатов, но повышает вероятность попадания в эти результаты "шума" — излишних записей. Официальная документация по этой теме доступна по адресу `http://dev.mysql.com/doc/refman/5.1/en/fulltext-query-expansion.html`.

Чтобы включить расширение запроса, нужно добавить в раздел критериев поиска ключевые слова `WITH QUERY EXPANSION` так, как показано в следующем примере.

```
SELECT   name, description FROM product
WHERE    MATCH (name, description)
           AGAINST ("flower" WITH QUERY EXPANSION)
ORDER BY MATCH (name, description)
           AGAINST ("flower" WITH QUERY EXPANSION) DESC;
```

Поиск в нашей базе данных слова `flower` с расширением запроса выдает 75 результатов. Поиск без расширения запроса выдает только 15. Код уровня данных, который мы создадим в следующем упражнении, использовать расширение запросов не будет, но при желании вы можете легко его включить.

Написание хранимых процедур для поиска

Теперь мы можем приступить к созданию хранимых процедур, которые будут выполнять поиск товаров. Если вы хотите более подробно изучить возможности полнотекстового поиска в MySQL, обратитесь к официальной документации по адресу `http://dev.mysql.com/doc/refman/5.1/en/fulltext-search.html`.

Упражнение: написание кода для поиска в базе данных

1. С помощью phpMyAdmin выполните приведенный ниже код, чтобы создать хранимую процедуру `catalog_count_search_result` в базе данных `tshirtshop`. Не забудьте задать разделитель `$$`, прежде чем выполнять этот код.

```
-- Создаем хранимую процедуру catalog_count_search_result
CREATE PROCEDURE catalog_count_search_result(
  IN inSearchString TEXT, IN inAllWords VARCHAR(3))
BEGIN
```

```
  IF inAllWords = "on" THEN
    PREPARE statement FROM
      "SELECT count(*)
        FROM product
        WHERE MATCH (name, description) AGAINST (? IN BOOLEAN MODE)";
  ELSE
    PREPARE statement FROM
      "SELECT count(*)
        FROM product
        WHERE MATCH (name, description) AGAINST (?)";
  END IF;

  SET @p1 = inSearchString;

  EXECUTE statement USING @p1;
END$$
```

2. Выполните следующий фрагмент кода, чтобы создать хранимую процедуру `catalog_search`.

```
-- Создаем хранимую процедуру catalog_search
CREATE PROCEDURE catalog_search(
  IN inSearchString TEXT, IN inAllWords VARCHAR(3),
  IN inShortProductDescriptionLength INT,
  IN inProductsPerPage INT, IN inStartItem INT)
BEGIN
  IF inAllWords = "on" THEN
    PREPARE statement FROM
      "SELECT product_id, name,
                IF(LENGTH(description) <= ?,
                   description,
                   CONCAT(LEFT(description, ?),
                               '...')) AS description,
                price, discounted_price, thumbnail
       FROM     product
       WHERE    MATCH (name, description)
                AGAINST (? IN BOOLEAN MODE)
       ORDER BY MATCH (name, description)
                AGAINST (? IN BOOLEAN MODE) DESC
       LIMIT    ?, ?";
  ELSE
    PREPARE statement FROM
      "SELECT   product_id, name,
                IF(LENGTH(description) <= ?,
                   description,
                   CONCAT(LEFT(description, ?),
                          '...')) AS description,
                price, discounted_price, thumbnail
       FROM     product
       WHERE    MATCH (name, description) AGAINST (?)
       ORDER BY MATCH (name, description) AGAINST (?) DESC
       LIMIT    ?, ?";
  END IF;

  SET @p1 = inShortProductDescriptionLength;
  SET @p2 = inSearchString;
  SET @p3 = inStartItem;
  SET @p4 = inProductsPerPage;

  EXECUTE statement USING @p1, @p1, @p2, @p2, @p3, @p4;
END$$
```

Как это работает: функциональность поиска в каталоге

В этом упражнении мы добавили в базу данных функциональность, поддерживающую функции поиска на уровне данных. Эта функциональность состоит из двух хранимых процедур.

- `catalog_count_search_result`. Эта хранимая процедура подсчитывает количество результатов поиска. Это необходимо для того, чтобы уровень представления мог определить, на сколько страниц разбивать список результатов при выводе.
- `catalog_search`. Эта хранимая процедура выполняет собственно поиск подходящих товаров в базе данных.

Хотя, на первый взгляд, эти хранимые процедуры выглядят устрашающе, прежде всего из-за их размера, они просто выполняют полнотекстовый поиск, который мы только что обсуждали. Кроме того, процедура `catalog_search` реализует методику разбиения результатов на страницы, которая используется и в других процедурах, возвращающих списки товаров.

Реализация кода уровня логики приложения

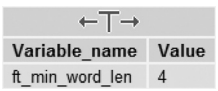
На уровне логики приложения вся функциональность поиска в каталоге сосредоточена в единственном методе `Search()`, вызывающем две созданные нами хранимые процедуры для поиска в базе данных. Давайте сначала создадим его, а затем подробно рассмотрим его код.

Упражнение: реализация кода уровня логики приложения

1. Возможность полнотекстового поиска в MySQL игнорирует слова из поисковой строки, длина которых меньше заданной. Было бы неплохо сообщить посетителю, какие слова использовались при поиске, а какие были проигнорированы. Чтобы сделать это, сначала нужно узнать, какова минимальная длина слов, используемых в поиске в нашей базе данных. Эта длина указана в переменной среды MySQL `ft_min_word_len`. С помощью phpMyAdmin выполните в базе данных `tshirtshop` следующий оператор SQL:

```
SHOW VARIABLES LIKE 'ft_min_word_len';
```

   Если вы не изменяли значение этой переменной, оно должно быть равно 4 (рис. 8.6).



Variable_name	Value
ft_min_word_len	4

Рис. 8.6. Отображение значения переменной `ft_min_word_len` в phpMyAdmin

2. Сохраните значение этой переменной в конфигурационном файле, чтобы оно было доступно для кода нашего приложения. Откройте файл `include/config.php` и добавьте в него следующий код:

```
/* Минимальная длина слов, используемых в поиске; эта константа должна
быть равна значению переменной MySQL ft_min_word_len */
define('FT_MIN_WORD_LEN', 4);
```

3. Откройте файл `business/catalog.php` и добавьте в него код метода `Search()`.

```
// Поиск в каталоге
public static function Search($searchString, $allWords,
```

```
                                $pageNo, &$rHowManyPages)
{
  //Результат поиска будет массивом следующей структуры
  $search_result = array ('accepted_words' => array (),
                          'ignored_words' => array (),
                          'products' => array ());

  // Возвращаем void, если строка поиска пустая
  if (empty ($searchString))
    return $search_result;

  // Символы-разделители
  $delimiters = ',.; ';

  /* При первом вызове strtok мы передаем ей всю строку поиска
     и список разделителей. Она возвращает первое слово строки. */
  $word = strtok($searchString, $delimiters);
  // Просматриваем строку до конца, слово за словом

  while ($word)
  {
    // Короткие слова добавляются в список ignored_words из $search_result
    if (strlen($word) < FT_MIN_WORD_LEN)
      $search_result['ignored_words'][] = $word;
    else
      $search_result['accepted_words'][] = $word;

    // Получаем следующее слово из строки поиска
    $word = strtok($delimiters);
  }

  // Если подходящих слов нет, возвращаем $search_result
  if (count($search_result['accepted_words']) == 0)
    return $search_result;

  // Составляем $search_string из подходящих слов
  $search_string = '';

  // Если $allWords в значении 'on', добавляем символы ' +' к каждому слову
  if (strcmp($allWords, "on") == 0)
    $search_string = implode(" +", $search_result['accepted_words']);
  else
    $search_string = implode(" ", $search_result['accepted_words']);

  // Подсчитываем количество результатов поиска
  $sql = 'CALL catalog_count_search_result(:search_string, :all_words)';
  $params = array(':search_string' => $search_string,
                  ':all_words' => $allWords);

  // Вычисляем количество страниц, необходимое для отображения товаров
  $rHowManyPages = Catalog::HowManyPages($sql, $params);
  // Определяем номер первого товара
  $start_item = ($pageNo - 1) * PRODUCTS_PER_PAGE;

  // Извлекаем список подходящих товаров
  $sql = 'CALL catalog_search(:search_string, :all_words,
                              :short_product_description_length,
                              :products_per_page, :start_item)';

  // Создаем массив параметров
  $params = array (':search_string' => $search_string,
```

```
                    ':all_words' => $allWords,
                    ':short_product_description_length' =>
                      SHORT_PRODUCT_DESCRIPTION_LENGTH,
                    ':products_per_page' => PRODUCTS_PER_PAGE,
                    ':start_item' => $start_item);

  // Выполняем запрос
  $search_result['products'] = DatabaseHandler::GetAll($sql, $params);

  // Возвращаем результаты
  return $search_result;
}
```

Как это работает: метод `Search()` уровня логики приложения

Метод `Search()` удаляет из строки поиска слова, длина которых меньше минимальной, заданной в константе `FT_MIN_WORD_LEN`. Значение этой константы должно быть равно значению серверной переменной MySQL `ft_min_word_len`.

Нам нужно знать значение этой переменной, чтобы мы могли сообщить пользователю, какие слова из введенной им строки поиска действительно применялись при поиске, а какие были отброшены. Сначала мы выясняем, какие слова отбрасываются, сравнивая их длину со значением `FT_MIN_WORD_LEN`. В результате мы делим все слова на две группы: принятые слова и отброшенные — и помещаем их в ассоциативный массив, в котором мы будем хранить также результаты поиска. В конце метода мы вернем этот массив в качестве результата.

Метод `Search()` будет вызываться из уровня представления со следующими параметрами (заметьте, что все они, кроме первого, аналогичны параметрам хранимых процедур уровня данных).

- `$searchString` содержит введенную пользователем поисковую строку.
- `$allWords` указывает, нужно ли выполнять поиск по всем словам.
- `$pageNo` указывает, какая страница списка найденных товаров запрашивается.
- `$rHowManyPages` указывает общее количество страниц в списке найденных товаров.

Метод начинается с объявления ассоциативного массива `$search_result`, в котором будут храниться результаты поиска товаров.

```
// Поиск в каталоге
public static function Search($searchString, $allWords,
                              $pageNo, &$rHowManyPages)
{
  //Результат поиска будет массивом следующей структуры
  $search_result = array ('accepted_words' => array (),
                          'ignored_words' => array (),
                          'products' => array ());
```

Как видите, массив содержит три других массива, назначение которых понятно по их именам. В массиве `accepted_words` хранятся слова, которые будут использоваться при поиске, а в `ignored_words` — отбрасываемые слова. Уровень представления будет использовать эти массивы, чтобы сообщить пользователю, какие слова использовались при поиске, а какие — нет. Массив `products` предназначен для хранения списка товаров, найденных в результате поиска.

После создания массива метод `Search()` проверяет, не пуста ли строка поиска. Если строка пуста, возвращается пустой массив `$search_result`, указывающий, что поиск не дал результатов.

```
// Возвращаем void, если строка поиска пустая
if (empty ($searchString))
  return $search_result;
```

Убедившись, что строка поиска не пустая, мы начинаем фильтровать слова из этой строки. Если слово короче, чем задано в `FT_MIN_WORD_LEN`, оно отправляется в массив `ignored_words` и не используется при выполнении поиска. Подходящие слова отправляются в массив `accepted_words`. Поскольку MySQL не сообщает нам, какие слова используются при поиске, а какие игнорируются, нам приходится самим заниматься фильт-

рацией поисковой строки в коде уровня логики приложения. Разбивая строку на слова, мы считаем, что символами-разделителями могут быть пробел, запятая, точка и точка с запятой; все остальные символы считаются частями слов. Ниже приведен фрагмент кода, отвечающий за разбиение строки на слова и запись слов в массивы `ignored_words` и `accepted_words`.

```
// Символы-разделители
$delimiters = ',.; ';

/* При первом вызове strtok мы передаем ей всю строку поиска
   и список разделителей. Она возвращает первое слово строки. */
$word = strtok($searchString, $delimiters);
// Просматриваем строку до конца, слово за словом

while ($word)
{
  // Короткие слова добавляются в список ignored_words из $search_result
  if (strlen($word) < FT_MIN_WORD_LEN)
    $search_result['ignored_words'][] = $word;
  else
    $search_result['accepted_words'][] = $word;

  // Получаем следующее слово из строки поиска
  $word = strtok($delimiters);
}
```

Завершив обработку строки, мы еще раз проверяем, есть ли у нас слова, по которым можно выполнять поиск. Если они есть, то они должны находиться в массиве `accepted_words`, и мы проверяем, не пустой ли этот массив. Если он пустой, нам нечего искать, и мы возвращаем пустой массив `$search_result`.

```
// Если подходящих слов нет, возвращаем $search_result
if (count($search_result['accepted_words']) == 0)
  return $search_result;
```

Теперь мы знаем, что у нас есть хотя бы одно подходящее для поиска слово, и можем подготовить строку поиска для передачи базе данных. Как вы, вероятно, помните, при поиске по всем словам используются булевы операторы, и каждое искомое слово должно предваряться символом `+`. Например, для поиска по всем словам из строки `beautiful flower` мы должны отправить базе данных строку `+beautiful +flower`. Для поиска по любому из слов мы просто отправляем строку `beautiful flower`. Приведенный ниже фрагмент кода создает переменную `$search_string` и присваивает ей значение — строку поиска, которую нужно передать MySQL.

```
// Составляем $search_string из подходящих слов
$search_string = '';

// Если $allWords в значении 'on', добавляем символы ' +' к каждому слову
if (strcmp($allWords, "on") == 0)
  $search_string = implode(" +", $search_result['accepted_words']);
else
  $search_string = implode(" ", $search_result['accepted_words']);
```

Оставшийся код в методе `Search()` запрашивает страницы товаров, и мы уже его рассматривали, когда занимались каталогом товаров. Метод возвращает результаты работы в массиве `$search_result` — в нем содержатся найденные товары и списки использованных при поиске и проигнорированных слов. Это все данные, которые нужно сообщить пользователю после выполнения поиска.

```
  // Выполняем запрос
  $search_result['products'] = DatabaseHandler::GetAll($sql, $params);

  // Возвращаем результаты
  return $search_result;
}
```

Реализация уровня представления

Механизм поиска в каталоге будет использовать два отдельных элемента интерфейса, которые нам нужно создать.

- Компонентный шаблон `search_box`, который должен позволять посетителю задавать поисковую строку (см. рис. 8.1)
- Компонентный шаблон `search_results`, отображающий результаты поиска (см. рис. 8.2)

Созданию каждого из этих элементов посвящено отдельное упражнение.

Создание поля ввода

Выполните следующее упражнение, чтобы создать компонентный шаблон `search_box` и интегрировать его в приложение TShirtShop.

Упражнение: создание компонентного шаблона `search_box`

1. Создайте в папке `presentation/templates` файл `search_box.tpl` и поместите в него следующий код.

```
{* search_box.tpl *}
{load_presentation_object filename="search_box" assign="obj"}
{* Начало поля поиска *}
<div class="box">
  <p class="box-title">Search the Catalog</p>
  <form class="search_form" method="post" action="{$obj->mLinkToSearch}">
    <p>
      <input maxlength="100" id="search_string" name="search_string"
        value="{$obj->mSearchString}" size="19" />
      <input type="submit" value="Go!" /><br />
    </p>
    <p>
      <input type="checkbox" id="all_words" name="all_words"
        {if $obj->mAllWords == "on"} checked="checked" {/if}/>
      Search for all words
    </p>
  </form>
</div>
{* Конец поля поиска *}
```

2. Создайте в файле `search_box.php` в папке `presentation` объект представления `SearchBox` со следующим кодом.

```
<?php
// Управляет полем ввода
class SearchBox
{
  // Public-переменные для шаблона Smarty
  public $mSearchString = '';
  public $mAllWords = 'off';
  public $mLinkToSearch;

  // Конструктор класса
  public function __construct()
  {
    $this->mLinkToSearch = Link::ToSearch();
```

```
    if (isset ($_GET['Search']))
    {
      $this->mSearchString = trim($_POST['search_string']);
      $this->mAllWords = isset ($_POST['all_words']) ?
                         $_POST['all_words'] : 'off';

      // Очищаем буфер вывода
      ob_clean();

      // Перенаправление с кодом 302
      header('HTTP/1.1 302 Found');
      header('Location: ' .
              Link::ToSearchResults($this->mSearchString, $this->mAllWords));

      // Очищаем буфер вывода и останавливаем выполнение
      flush();
      ob_flush();
      ob_end_clean();
      exit();
    }
    elseif (isset ($_GET['SearchResults']))
    {
      $this->mSearchString = trim(str_replace('-', ' ',
        $_GET['SearchString']));
      $this->mAllWords = isset ($_GET['AllWords']) ?
        $_GET['AllWords'] : 'off';
    }

    if (isset ($_GET['ProductId']) &&
          isset ($_SESSION['link_to_continue_shopping']))
    {
      $continue_shopping =
        Link::QueryStringToArray($_SESSION['link_to_continue_shopping']);

      if (isset ($continue_shopping['SearchResults']))
      {
        $this->mSearchString =
          trim(str_replace('-', ' ', $continue_shopping['SearchString']));
        $this->mAllWords = $continue_shopping['AllWords'];
      }
    }
  }
}
?>
```

3. Откройте файл `presentation/link.php` и добавьте в класс `Link` следующих два метода.

```
// Создание ссылки на страницу поиска
public static function ToSearch()
{
  return self::Build('index.php?Search');
}

// Создание ссылки на страницу с результатами поиска
public static function ToSearchResults($searchString, $allWords,
                                       $page = 1)
{
  $link = 'search-results/find';

  if (empty($searchString))
    $link .= '/';
```

```
  else
    $link .= '-' . self::CleanUrlText($searchString) . '/';

  $link .= 'all-words-' . $allWords . '/';

  if ($page > 1)
    $link .= 'page-' . $page . '/';

  return self::Build($link);
}
```

4. В том же файле модифицируйте код метода `CheckRequest()`, вставив в него фрагмент кода, выделенный полужирным шрифтом в приведенном ниже листинге. Этот метод мы создали в главе 7, чтобы перенаправлять поступающие запросы по корректным адресам, но он не должен перенаправлять поисковые запросы — у них нет "корректных" адресов.

```
// Выполняет перенаправление по корректному URL в случае необходимости
public static function CheckRequest()
{
  $proper_url = '';

  if (isset ($_GET['Search']) || isset($_GET['SearchResults']))
  {
    return ;
  }
  // Получаем правильный URL для страниц категорий
  elseif (isset ($_GET['DepartmentId']) && isset ($_GET['CategoryId']))
  {
    if (isset ($_GET['Page']))
```

5. Добавьте в файл `presentation/templates/store_front.tpl` стили, используемые в шаблоне `search_box`.

```
div.yui-b div form {
  padding: 5px 10px;
}

input, select, textarea {
  font-family: "georgia";
  font-size: 85%;
}
```

6. Внесите изменения в файл `presentation/templates/store_front.tpl`, чтобы он загружал шаблон `search_box`.

```
...
      <div class="yui-b">
        {include file="search_box.tpl"}
        {include file="departments_list.tpl"}
        {include file=$obj->mCategoriesCell}
      </div>
...
```

7. Откройте проект в браузере, и вы увидите поле поиска на том же месте, что и на рис. 8.1.

Как это работает: компонентный шаблон `search_box`

К этому моменту поле поиска должно быть полностью работоспособным. Однако прежде чем вы сможете просмотреть результаты поиска, нужно создать страницу для их отображения. Мы займемся этим в следующем упражнении, а сейчас давайте разберемся, как работает шаблон `search_box`.

В файле search_box.tpl содержится HTML-разметка поля поиска — ничего сложного в ней нет. С другой стороны, объект представления, поддерживающий этот шаблон, определенно, заслуживает подробного рассмотрения. В классе SearchBox в файле search_box.php есть две функции.

- Когда пользователь вводит строку поиска и передает ее на сервер, поле поиска через перенаправление по коду HTTP 302 передает запрос по URL, на который можно создать закладку, например http://localhost/tshirtshop/search-results/find-beautiful-flower/all-words-off/. Если такой механизм не реализовать, результаты поиска будут отображаться на той странице, которую пользователь просматривал в момент ввода поисковой строки.
- Если в момент начала поиска в браузере была загружена страница результатов поиска или страница с информацией о товаре, открытая со страницы результатов поиска, в поле поиска вводятся ключевые слова, использованные при поиске.

Адрес, по которому выполняется перенаправление при поиске, определяется методом Link::ToSearchResults(). Обратите внимание на то, что используется перенаправление по коду HTTP 302. Этот код указывает на временное перемещение страницы, хотя в нашем случае он ничего особенного не значит, поскольку поисковые системы из Интернета все равно не будут отсылать формы на наш сервер. Это перенаправление выполняется не с целью оптимизации для поисковых систем, а чтобы предоставить пользователям возможность создавать закладки на результаты поиска. Кроме того, это позволяет другим сайтам ссылаться на результаты поиска на нашем сайте.

```
if (isset ($_GET['Search']))
{
  $this->mSearchString = trim($_POST['search_string']);
  $this->mAllWords = isset ($_POST['all_words']) ?
                     $_POST['all_words'] : 'off';

  // Очищаем буфер вывода
  ob_clean();

  // Перенаправление с кодом 302
  header('HTTP/1.1 302 Found');
  header('Location: ' .
          Link::ToSearchResults($this->mSearchString, $this->mAllWords));

  // Очищаем буфер вывода и останавливаем выполнение
  flush();
  ob_flush();
  ob_end_clean();
  exit();
}
```

Разумеется, при реализации страницы с результатами поиска в следующем упражнении нам нужно будет знать, что означают ссылки на результаты поиска. Например, ссылка для поиска любого из слов beautiful flower выглядит так: http://localhost/tshirtshop/search-results/find-beautiful-flower/all-words-off/. А в ссылке для поиска всех слов последний фрагмент изменится на all-words-on. Если результатов поиска много и они разбиты на несколько страниц, то за этим фрагментом следует номер страницы, например /page-2/.

Если открыта страница с результатами поиска, мы получаем строку поиска и значение параметра AllWords из строки запроса. Их считывает и отображает шаблон search_box.tpl.

```
  elseif (isset ($_GET['SearchResults']))
  {
    $this->mSearchString = trim(str_replace('-', ' ',
      $_GET['SearchString']));
    $this->mAllWords = isset ($_GET['AllWords']) ?
      $_GET['AllWords'] : 'off';
  }
```

Кроме того, мы проверяем, является ли открытая страница страницей с информацией о товаре, открытой со страницы результатов поиска. Если да, в поле поиска помещаются ключевые слова, по которым выполнялся поиск.

```
if (isset ($_GET['ProductId']) &&
    isset ($_SESSION['link_to_continue_shopping']))
{
  $continue_shopping =
    Link::QueryStringToArray($_SESSION['link_to_continue_shopping']);

  if (isset ($continue_shopping['SearchResults']))
  {
    $this->mSearchString =
      trim(str_replace('-', ' ', $continue_shopping['SearchString']));
    $this->mAllWords = $continue_shopping['AllWords'];
  }
}
```

Отображение результатов поиска

В следующем упражнении мы создадим компонентный шаблон, отвечающий за вывод результатов поиска. Чтобы упростить свою задачу, в качестве основы мы используем компонентный шаблон `product_list`, который уже применяется для отображения списков товаров на главной странице, страницах отделов и категорий. Разумеется, если вы захотите отображать результаты поиска в особом формате, вам придется создать для этого отдельный шаблон.

Нам понадобится модифицировать управляющую логику шаблона в файле `products_list.php`, чтобы он мог определять, когда он используется для списков товаров, а когда — для результатов поиска. Давайте создадим шаблон `search_results` и обновим управляющую логику шаблона `products_list`.

Упражнение: создание компонентного шаблона `search_results`

1. Создайте в папке `presentation/templates` файл `search_results.tpl` и поместите в него следующий код.

```
{* search_results.tpl *}
<h1>Search results</h1>
{include file="products_list.tpl"}
```

2. Внесите изменения в файл `presentation/products_list.php`, добавив следующие строки в конструктор класса `ProductsList` — метод `__construct()`.

```
// Получаем поисковую строку и параметр AllWords из строки запроса
if (isset ($_GET['SearchResults']))
{
  $this->mSearchString = trim(str_replace('-', ' ',
$_GET['SearchString']));
  $this->mAllWords = isset ($_GET['AllWords']) ? $_GET['AllWords'] : 'off';
}
```

3. Добавьте элементы `$mSearchDescription`, `$mAllWords` и `$mSearchString` в класс `ProductsList` в том же файле.

```
<?php
class ProductsList
{
  // Public-переменные, доступные из шаблона Smarty
  public $mPage = 1;
  public $mrTotalPages;
  public $mLinkToNextPage;
```

```
  public $mLinkToPreviousPage;
  public $mProductListPages = array();
  public $mProducts;
  public $mSearchDescription;
  public $mAllWords = 'off';
  public $mSearchString;

  // Private-переменные
  private $_mDepartmentId;
  private $_mCategoryId;
```

4. Модифицируйте метод `init()` класса `ProductsList` следующим образом.

```
public function init()
{
  /* Если выполнялся поиск, получаем список товаров, вызывая
     метод уровня логики приложения Search() */
  if (isset ($this->mSearchString))
  {
    // Получаем результаты поиска
    $search_results = Catalog::Search($this->mSearchString,
                                      $this->mAllWords,
                                      $this->mPage,
                                      $this->mrTotalPages);
    // Получаем список товаров
    $this->mProducts = $search_results['products'];
    // Составляем заголовок для списка товаров
    if (count($search_results['accepted_words']) > 0)
      $this->mSearchDescription =
        '<p class="description">Products containing <font class="words">'
        . ($this->mAllWords == 'on' ? 'all' : 'any') . '</font>'
        . ' of these words: <font class="words">'
        . implode(', ', $search_results['accepted_words']) .
        '</font></p>';
    if (count($search_results['ignored_words']) > 0)
      $this->mSearchDescription .=
        '<p class="description">Ignored words: <font class="words">'
        . implode(', ', $search_results['ignored_words']) .
        '</font></p>';
    if (!(count($search_results['products']) > 0))
      $this->mSearchDescription .=
        '<p class="description">Your search generated no results.</p>';
  }
  /* Если посетитель просматривает категорию, получаем список ее товаров,
     вызывая метод уровня логики приложения GetProductsInCategory() */
  elseif (isset ($this->_mCategoryId))
    $this->mProducts = Catalog::GetProductsInCategory(
    $this->_mCategoryId, $this->mPage, $this->mrTotalPages);
...
```

5. Продолжайте обновлять метод `init()`, вставляя в его код фрагменты, выделенные полужирным шрифтом в следующем листинге.

```
  /* Если список товаров разбит на несколько страниц, отображаем
     навигационные элементы управления */
  if ($this->mrTotalPages > 1)
  {
    // Создаем ссылку Next
    if ($this->mPage < $this->mrTotalPages)
    {
     if (isset($_GET['SearchResults']))
```

```
        $this->mLinkToNextPage =
          Link::ToSearchResults($this->mSearchString, $this->mAllWords,
                                $this->mPage + 1);
      elseif (isset($this->_mCategoryId))
        $this->mLinkToNextPage =
          Link::ToCategory($this->_mDepartmentId, $this->_mCategoryId,
                           $this->mPage + 1);
      elseif (isset($this->_mDepartmentId))
        $this->mLinkToNextPage =
          Link::ToDepartment($this->_mDepartmentId, $this->mPage + 1);
    }

    // Создаем ссылку Previous
    if ($this->mPage > 1)
      {
        if (isset($_GET['SearchResults']))
          $this->mLinkToPreviousPage =
            Link::ToSearchResults($this->mSearchString, $this->mAllWords,
                                  $this->mPage - 1);
        if (isset($this->_mCategoryId))
          $this->mLinkToPreviousPage =
            Link::ToCategory($this->_mDepartmentId, $this->_mCategoryId,
                             $this->mPage - 1);
        elseif (isset($this->_mDepartmentId))
          $this->mLinkToPreviousPage =
            Link::ToDepartment($this->_mDepartmentId, $this->mPage - 1);
      }
    }

    // Генерируем ссылки на страницы товаров
    for ($i = 1; $i <= $this->mrTotalPages; $i++)
      if (isset($_GET['SearchResults']))
        $this->mProductListPages[] =
          Link::ToSearchResults($this->mSearchString, $this->mAllWords, $i);
      elseif (isset($this->_mCategoryId))
        $this->mProductListPages[] =
          Link::ToCategory($this->_mDepartmentId, $this->_mCategoryId, $i);
      elseif (isset($this->_mDepartmentId))
        $this->mProductListPages[] =
          Link::ToDepartment($this->_mDepartmentId, $i);
      else
        $this->mProductListPages[] = Link::ToIndex($i);
    }
...
/* Перенаправление с кодом 404, если номер запрошенной страницы
   больше общего числа страниц списка */
if ($this->mPage > $this->mrTotalPages && !empty($this->mrTotalPages))
{
  // Очищаем буфер вывода
  ob_clean();
...
```

6. Добавьте в начало файла presentation/templates/products_list.tpl две строки, выделенные полужирным шрифтом во фрагменте листинга ниже:

```
{* products_list.tpl *}
{load_presentation_object filename="products_list" assign="obj"}
{if $obj->mSearchDescription != ""}
  <p class="description">{$obj->mSearchDescription}</p>
{/if}
```

7. Откройте файл .htaccess и добавьте в него следующие правила RewriteRule, отвечающие за перенаправления для страниц с результатами поиска.

```
# Переписываем страницы отделов
RewriteRule ^.*-d([0-9]+)/page-([0-9]+)/?$ index.php?DepartmentId=$1&
Page=$2 [L]
RewriteRule ^.*-d([0-9]+)/?$ index.php?DepartmentId=$1 [L]

# Переписываем страницы с результатами поиска
RewriteRule ^search-results/find-(.*)/all-words-(on|off)/page-([0-9]+)/?$
index.php?SearchResults&SearchString=$1&AllWords=$2&Page=$3 [L]
RewriteRule ^search-results/find-?(.*)/all-words-(on|off)/?$
index.php?SearchResults&SearchString=$1&AllWords=$2&Page=1 [L]

# Переписываем подстраницы
RewriteRule ^page-([0-9]+)/?$ index.php?Page=$1 [L]
```

8. Откройте файл presentation/store_front.php и вставьте в код метода init() фрагмент, выделенный полужирным шрифтом в следующем листинге. Этот фрагмент будет загружать компонентный шаблон search_results при выполнении поиска.

```
...
// Загружаем сведения о товаре на страницу товара
if (isset ($_GET['ProductId']))
  $this->mContentsCell = 'product.tpl';

// Загружаем страницу с результатами поиска, если выполнялся поиск
elseif (isset ($_GET['SearchResults']))
  $this->mContentsCell = 'search_results.tpl';

// Загружаем заголовок страницы
$this->mPageTitle = $this->_GetPageTitle();
...
```

9. В том же файле, store_front.php, добавьте выделенные полужирным шрифтом строки из следующего листинга в метод _GetPageTitle(), чтобы на странице с результатами поиска отображался требуемый заголовок.

```
...
    elseif (isset ($_GET['ProductId']))
    {
      $page_title = 'TShirtShop: ' .
        Catalog::GetProductName($_GET['ProductId']);
    }
    elseif (isset ($_GET['SearchResults']))
    {
      $page_title = 'TShirtShop: "';

      // Отображаем строку поиска
      $page_title .= trim(str_replace
        ('-', ' ', $_GET['SearchString'])) . '" (';

      // Отображаем "all-words search" или "any-words search"
      $all_words = isset ($_GET['AllWords']) ? $_GET['AllWords'] : 'off';

      $page_title .= (($all_words == 'on') ? 'all' : 'any') .
                     '-words search';

      // Отображаем номер страницы
      if (isset ($_GET['Page']) && ((int)$_GET['Page']) > 1)
```

```
        $page_title .= ', page ' . ((int)$_GET['Page']);

      $page_title .= ')';
    }
    else
    {
      if (isset ($_GET['Page']) && ((int)$_GET['Page']) > 1)
        $page_title .= ' - Page ' . ((int)$_GET['Page']);
    }
    return $page_title;
  }
...
```

10. Откройте файл `presentation/product.php` и добавьте в метод `init()` класса `Product` код, выделенный полужирным шрифтом в приведенном ниже листинге. Этот код позволит продолжать навигацию по сайту после перехода на страницу товара со страницы результатов поиска.

```
    elseif (isset ($continue_shopping['DepartmentId']))
      $this->mLinkToContinueShopping =
        Link::ToDepartment((int)$continue_shopping['DepartmentId'], $page);
    elseif (isset ($continue_shopping['SearchResults']))
      $this->mLinkToContinueShopping =
        Link::ToSearchResults(
          trim(str_replace('-', ' ', $continue_shopping['SearchString'])),
          $continue_shopping['AllWords'], $page);
  else
    $this->mLinkToContinueShopping = Link::ToIndex($page);
}
```

11. Добавьте новый стиль в файл `styles/tshirtshop.css`.

```
.words {
color: #ff0000;
}
```

12. Откройте наш проект в браузере, попробуйте выполнить поиск для строки `christmas` и перейдите на вторую страницу результатов поиска. Она должна выглядеть примерно так, как на рис. 8.7.

Как это работает: поиск в каталоге товаров

Поздравляем, теперь наш каталог товаров полностью поддерживает поиск! Нам пришлось написать довольно солидный объем кода, но сам этот код был не слишком сложным.

Список товаров отображается с помощью шаблона `products_list`, который мы создали в главе 5, но теперь этот шаблон распознает элемент Search в строке запроса и может вызывать метод `Search()` уровня логики приложения для получения списка товаров.

Метод `Search()` из уровня логики приложения возвращает массив, в котором содержатся список найденных товаров, список слов, использованных при поиске, и список проигнорированных слов (с длиной меньше минимально допустимой). Все эти сведения сообщаются пользователю.

Вероятно, самая необычная часть этого упражнения — добавление нового правила в файл `.htaccess`. Как вы знаете, когда пользователь отправляет на сервер новый поисковый запрос, этот запрос переписывается в виде URL, содержащего ключевые слова. На такой URL можно создать закладку. Правила переписывания преобразуют URL с ключевыми словами в динамический URL.

```
# Переписываем страницы с результатами поиска
RewriteRule ^search-results/find-(.*)/all-words-(on|off)/page-([0-9]+)/?$
index.php?SearchResults&SearchString=$1&AllWords=$2&Page=$3 [L]
RewriteRule ^search-results/find-?(.*)/all-words-(on|off)/?$
index.php?SearchResults&SearchString=$1&AllWords=$2&Page=1 [L]
```



Рис. 8.7. Пример результатов поиска в TShirtShop

Первое правило работает с URL, содержащими номера страниц, а второе — с URL, в которых номеров страниц нет.

Резюме

В этой главе мы снабдили наш сайт возможностями поиска товаров в каталоге с помощью механизма полнотекстового поиска MySQL. Возможность поиска хорошо интегрируется в текущую структуру веб-сайта и взаимодействует с механизмом разбиения списков на страницы, созданным в главе 4. Самой интересной частью главы было описание механизма полнотекстового поиска в MySQL. Кроме того, в этой главе впервые уровень логики приложения получил функциональность, не сводящуюся к передаче данных между уровнем данных и уровнем представления.

В главе 9 мы займемся продажей товаров с помощью PayPal.

Глава 9

Прием платежей через PayPal

Ну что ж, каркас нашего сайта постепенно обрастает необходимой функциональностью, и мы можем подумать о том, чтобы заработать с его помощью деньги. Для этого понадобится система приема платежей от посетителей. Большие компании обычно разрабатывают такую систему самостоятельно, но для маленьких магазинов удобнее воспользоваться услугами какой-нибудь сторонней службы и не обрабатывать самостоятельно конфиденциальные данные о номерах кредитных карточек, счетов и т.д.

Существует целый ряд компаний, предоставляющих малому бизнесу услуги по работе с кредитными карточками и банковскими транзакциями. Они могут выступать в роли посредников между электронными магазинами и посетителями. Многие из таких компаний появились относительно недавно, а данные о банковских счетах — одна из самых конфиденциальных категорий информации. Кроме того, поиск в Интернете покажет вам, что у любой из этих компаний есть и довольные, и недовольные клиенты. Поэтому мы не станем рекомендовать какую-то конкретную компанию.

Вместо этого мы приведем список компаний, предлагающих поддержку приема платежей, и продемонстрируем некоторые их возможности на примере PayPal. Вы узнаете, как интегрировать PayPal в проект TShirtShop на первых двух стадиях разработки. Из этой главы вы узнаете, как

- создать учетную запись PayPal Website Payments Standard;
- интегрировать PayPal в TShirtShop на первой стадии разработки, когда нам достаточно поддержки корзины покупателя и механизма поддержки платежей;
- интегрировать PayPal в TShirtShop на второй стадии разработки, когда мы создадим собственный вариант корзины покупателя.

ЗАМЕЧАНИЕ

Эта глава представляет собой не руководство по работе с PayPal, а только краткий обзор. За подробной информацией об услугах, предоставляемых PayPal, обратитесь на веб-сайт этой компании (`http://www.paypal.com`). Если вы решите пользоваться услугами другой платежной компании, изучите ее веб-сайт.

Выбор платежной компании

Существует не меньше полусотни компаний, предоставляющих услуги по приему платежей через Интернет. Они весьма разнообразны, и предоставляемые ими услуги также сильно различаются. Одни позволяют только передавать деньги от одного человека другому, и каждый платеж нужно подтверждать лично; другие позволяют полностью интегрировать их механизмы в ваш веб-сайт. Одни работают по всему миру, а другие ограничиваются отдельной страной.

Приведенный ниже список ни в коем случае не претендует на полноту. Вы можете найти множество платежных компаний, отсутствующих в нем, задав в Google поисковую строку `Internet payment service providers`.

- *2Checkout*: `http://www.2checkout.com`
- *AnyPay*: `http://www.anypay.com`
- *CCNow*: `http://www.ccnow.com`
- *Electronic Transfer*: `http://www.electronictransfer.com`
- *Moneybookers*: `http://www.moneybookers.com`
- *MultiCards*: `http://www.multicards.com`
- *Pay By Web*: `http://www.paybyweb.com`
- *Paymate*: `https://www.paymate.com.au`
- *PayPal*: `http://www.paypal.com`
- *PaySystems*: `http://www.paysystems.com`
- *ProPay*: `http://www.propay.com`
- *QuickPayPro*: `http://www.quickpaypro.com`
- *WorldPay*: `http://www.worldpay.com`

Помимо популярности, в пользу PayPal говорит и удобство ее интеграции в наш веб-сайт на первых двух стадиях разработки. PayPal доступна во многих странах. Чтобы узнать, есть ли в этом списке ваша страна, зайдите на ее веб-сайт по адресу `http://www.paypal.com`.

На первой стадии разработки (сейчас мы находимся именно на этой стадии) у нас есть только каталог товаров с возможностью поиска. На этой стадии PayPal позволит добавить на сайт поддержку корзины покупателя и приема платежей с помощью всего лишь нескольких строк HTML. На второй стадии, создав собственный механизм корзины покупателя, мы сможем воспользоваться услугой PayPal Single Item Purchase. Она позволяет покупателям сразу оплачивать отдельные товары, не используя корзину покупателя. Этой возможностью мы займемся в главе 14.

Краткий обзор услуг, предоставляемых PayPal, можно найти в разделе Merchant Services сайта `http://www.paypal.com`. Кроме того, в этом разделе есть несколько полезных ссылок на сторонние ресурсы.

Начинаем работу с PayPal

Пожалуй, наилучшее описание PayPal приведено на ее собственном веб-сайте: “PayPal — это основанная на учетных записях система, позволяющая любому человеку, имеющему адрес электронной почты, безопасно отправлять и получать электронные платежи с помощью кредитной карты или банковского счета”.

Вместо того чтобы платить непосредственно продавцу, посетитель платит PayPal с помощью кредитной карты или банковского счета. Позже продавец может с помощью своей учетной записи в PayPal получить деньги, отправленные ему покупателями. На момент написания этой книги создание учетной записи в PayPal является бесплатным, и покупатель ничего не платит компании за пользование ее услугами. За получение денег с продавца взимается небольшой процент. Более подробная информация доступна по адресу `http://www.paypal.com/cgi-bin/webscr?cmd=_display-fees-outside`.

Ссылки и ресурсы PayPal

Если вам нужно больше информации, чем приведено в этой короткой главе, обратитесь к следующим ресурсам.

- *The Website Payments Standard Integration Guide.* Руководство по интеграции в сайты средств приема платежей. Оно доступно по адресу `https://www.paypal.com/en_US/pdf/PP_WebsitePaymentsStandard_IntegrationGuide.pdf`.
- *The PayPal Developer Network.* Официальный ресурс разработчиков, работающих с PayPal, доступный по адресу `https://www.paypal.com/pdn`.
- *PayPalDev*. Согласно этому сайту не зависит от владельцев PayPal. На этом сайте (`http://www.paypaldev.org/`) можно найти ссылки на различные ресурсы, связанные с PayPal.

Упражнение: создание учетной записи PayPal

1. Откройте в браузере сайт PayPal — `http://www.paypal.com`.
2. Щелкните на ссылке Sign Up.
3. PayPal поддерживает три типа учетных записей: Personal, Premier и Business. Чтобы получать электронные платежи, вам понадобится учетная запись типа Premier или Business. Выберите свою страну из раскрывающегося списка и щелкните на кнопке Get Started.
4. Введите всю требуемую информацию, и вы получите на свой почтовый адрес письмо с просьбой повторно посетить сайт PayPal, чтобы подтвердить введенные данные.

Как это работает: учетные записи PayPal

Когда вы создадите учетную запись PayPal, заданный при ее создании почтовый адрес будет вашим идентификатором — PayPal ID.

Службы PayPal предоставляют разнообразные возможности (поскольку сайт компании хорошо документирован, мы не будем описывать их здесь). Не забывайте, что компания PayPal существует за счет таких клиентов, как вы и ее служба поддержки будет рада ответить на любые возникшие у вас вопросы.

А теперь давайте посмотрим, как можно использовать созданную учетную запись на нашем веб-сайте.

Интеграция корзины покупателя и приема платежей в сайт TShirtShop

Первая наша задача — интегрировать в наш сайт функциональность корзины покупателя и приема платежей PayPal. В дальнейшем мы создадим собственную корзину покупателя и будем использовать только предоставляемый PayPal механизм приема платежей.

Чтобы принимать платежи, нужно добавить на сайт два важных элемента: кнопки Add to Cart для добавления товара в корзину (по одной для каждого товара) и кнопку View Cart для просмотра корзины (по одной на страницу). PayPal позволяет легко реализовать оба данных элемента.

Функциональность этих кнопок обеспечивается безопасными ссылками на веб-сайт PayPal, которые могут быть реализованы в виде форм или специально составленных URL. Давайте рассмотрим оба варианта. Ниже приведен пример использования форм, реализующих кнопку Add To Cart для товара Mistletoe T-Shirt с ценой $17.99.

```
<form target="paypal" action="https://www.paypal.com/cgi-bin/webscr"
 method="post">
  <input type="hidden" name="cmd" value="_cart" />
  <input type="hidden" name="business" value="youremail@example.com" />
  <input type="hidden" name="item_name" value="Mistletoe T-Shirt" />
  <input type="hidden" name="on0" value="Size" />
  <select name="os0">
    <option value="S">S</option>
    <option value="M">M</option>
    <option value="L">L</option>
    <option value="XL">XL</option>
    <option value="XXL">XXL</option>
  </select>
  <input type="hidden" name="on1" value="Color" />
  <select name="os1">
    <option value=" White">White</option>
    <option value=" Black">Black</option>
    <option value="Red">Red</option>
    <option value="Orange">Orange</option>
    <option value="Yellow">Yellow</option>
    <option value="Green">Green</option>
    <option value="Blue">Blue</option>
    <option value="Indigo">Indigo</option>
    <option value="Purple">Purple</option>
  </select>
  <input type="hidden" name="amount" value="17.99" />
  <input type="hidden" name="currency_code" value="USD" />
  <input type="hidden" name="add" value="1" />
  <input type="hidden" name="shopping_url"
   value="http://www.example.com/mistletoe-t-shirt-p84/" />
  <input type="hidden" name="return" value="http://www.example.com" />
  <input type="hidden" name="cancel_return" value="http://www.example.com" />
  <input type="submit" name="submit" value="Add to Cart" />
</form>
```

Загрузив такую форму в браузер, вы получите страницу наподобие показанной на рис. 9.1.

ПОДСКАЗКА

Хотя мы и не будем их использовать, PayPal предоставляет генераторы кнопок на основе данных, которые вы предоставляете (например, названия и цены товаров). Эти генераторы создают блок HTML-кода, похожий на приведенный выше.

Да, это все, что требуется для добавления на сайт кнопки Add to Cart. Поля формы, за исключением `on0`, `os0`, `on1` и `os1`, имеют предопределенные значения, и их назначение понятно из названий. Элементы `on0`, `os0`, `on1` и `os1` позволяют определять дополнительные атрибуты товаров; в нашем случае мы определяем с их помощью цвет и размер футболки. Обратите внимание на то, что PayPal поддерживает максимум два атрибута — больше вы добавить не сможете.

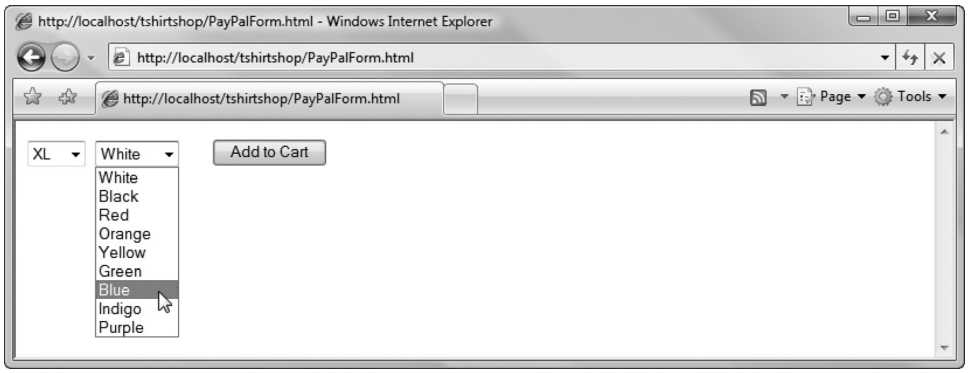


Рис. 9.1. Форма PayPal с кнопкой Add To Cart

Самый важный параметр в этой форме — `business`. В нем нужно точно задать адрес электронной почты, на который зарегистрирована учетная запись PayPal (эта учетная запись будет получать деньги). За более подробной информацией по этой теме обратитесь к руководству на сайте PayPal *Website Payments Standard Integration Guide*. О том, как преодолеть ограничение на два атрибута товаров, мы поговорим немного позже.

Можно генерировать URL для кнопки Add to Cart и без использования форм. Эта возможность полезна, если кнопка Add to Cart уже находится внутри формы, поскольку HTML не поддерживает вложенных форм. При использовании URL вместо форм все требуемые значения атрибутов задаются как параметры в строке запроса, например, так.

```
https://www.paypal.com/cgi-
bin/webscr?cmd=_cart&business=yourmail@example.com&item_name=Mistletoe+
T-Shirt&on0=Size&os0=S&on1=Color&os1=White&amount=17.99&currency_code=USD&
add=1&cancel_return=http://www.example.com&shopping_url=http://www.
example.com/mistletoe-t-shirt-p84/&return=http://www.example.com
```

ЗАМЕЧАНИЕ

При передаче информации на сервер с помощью параметров URL нужно правильно кодировать значения атрибутов. Например, чтобы передать строку `Mistletoe T-Shirt` в качестве параметра URL, нужно закодировать символ пробела, использование которого в URL запрещено. Этот символ можно кодировать как `%20` или `+`. В PHP есть специальная функция `urlencode()`, с помощью которой можно кодировать строки для включения в URL.

Открытие этой URL в браузере приведет к появлению страницы с корзиной покупателя PayPal, примерно такой, как на рис. 9.2.

ПРЕДУПРЕЖДЕНИЕ

При получении платежа нужно тщательно проверять соответствие полученной суммы общей цене заказанных товаров. Поскольку все данные о содержимом корзины передаются на сервер с помощью форм или URL, при желании злоумышленник может легко добавить в нее неучтенный товар или изменить цену учтенного. Для этого ему достаточно составить фальшивую ссылку Add To Cart и открыть ее в браузере. Более подробно эта проблема описана в статье, доступной по адресу `http://www.alphabetware.com/pptamper.asp`.

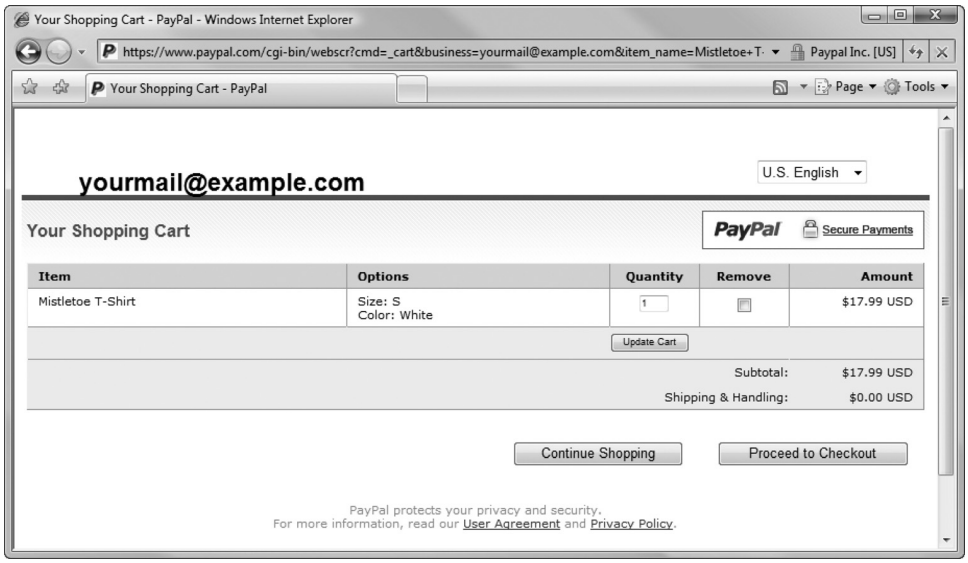


Рис. 9.2. Корзина покупателя PayPal

При использовании корзин покупателя наподобие предоставляемой PayPal необходимо контролировать длину имен товаров и атрибутов, хотя и маловероятно, что вы превысите допустимый предел длины URL для браузеров. По адресу `http://www.boutell.com/newfaq/misc/urllength.html` доступна статья с интересным анализом предельной длины URL для разных браузеров.

В TShirtShop мы будем использовать формы для создания кнопки View Cart и URL для кнопок Add to Cart. Примеры таких кнопок показаны на рис. 9.3.

Преодолеть накладываемое PayPal ограничение в два атрибута на товар можно, упаковывая значения всех атрибутов товара в одно значение атрибута PayPal. Значение атрибута PayPal будет представлять собой текст, содержащий всю информацию об атрибутах, и эта информация будет отображаться в корзине покупателя. Пример использования такого подхода показан на рис. 9.4.

Поскольку мы не можем добавлять товарам атрибуты в форме PayPal, наша система будет работать следующим образом.

1. Кнопки Add To Cart передают форму нашему приложению со ссылками наподобие `http://localhost/tshirtshop/add-product-65/`.
2. URL переписывается модулем `mod_rewrite` в формат наподобие `index.php?AddProduct=65`.
3. Файл `index.php` генерирует атрибут PayPal, в значении которого собраны значения всех атрибутов, выбранные пользователем для товара. Это значение может выглядеть примерно как `Size/Color: S/White`, но оно будет автоматически корректироваться в зависимости от требуемого количества атрибутов.
4. Сгенерированное значение атрибута будет использовано при генерации URL Add to Cart.
5. С помощью кода HTTP 302 выполняется перенаправление по сгенерированному URL.



Рис. 9.3. Страница TShirtShop с кнопками Add to Cart и View Cart

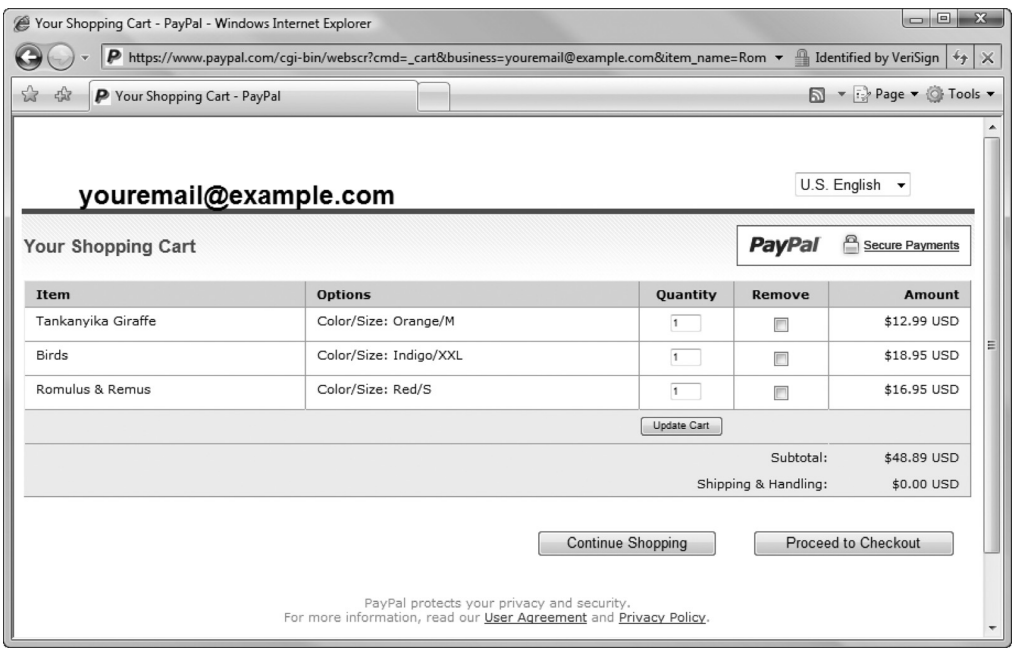


Рис. 9.4. Корзина покупателя PayPal с товарами из TShirtShop

Подводя итоги сказанному выше, после щелчка на кнопке Add To Cart веб-браузер пользователя получит перенаправление с кодом HTTP 302 на ссылку Add to Cart в PayPal. Эта ссылка будет вести на страницу корзины покупателя, как показано на рис. 9.4.

Рассмотрев все необходимые операции, мы можем приступить к написанию кода.

Упражнение: интеграция в приложение корзины покупателя PayPal и приема платежей

1. Откройте файл `config.php` из папки `include` и добавьте в него следующие объявления констант.

```
// Конфигурация PayPal
define('PAYPAL_URL', 'https://www.paypal.com/cgi-bin/webscr');
define('PAYPAL_EMAIL', 'youremail@example.com');
define('PAYPAL_CURRENCY_CODE', 'USD');
define('PAYPAL_RETURN_URL', 'http://www.example.com');
define('PAYPAL_CANCEL_RETURN_URL', 'http://www.example.com');
```

ПРЕДУПРЕЖДЕНИЕ

Не забудьте указать вместо `youremail@example.com` адрес электронной почты, на который вы зарегистрировали учетную запись PayPal. Если вы этого не сделаете, деньги не будут поступать на ваш счет. Кроме того, замените `http://www.example.com` реальным адресом сайта своего магазина.

2. Откройте файл `presentation/products_list.php` и добавьте в метод `init()` код, выделенный полужирным шрифтом в следующем листинге.

```
  // Генерируем ссылки на страницы товаров
  for ($i = 0; $i < count($this->mProducts); $i++)
  {
    $this->mProducts[$i]['link_to_product'] =
      Link::ToProduct($this->mProducts[$i]['product_id']);

    if ($this->mProducts[$i]['thumbnail'])
      $this->mProducts[$i]['thumbnail'] =
        Link::Build('product_images/' . $this->mProducts[$i]['thumbnail']);

    // Генерируем ссылку Add to Cart
    $this->mProducts[$i]['link_to_add_product'] =
      Link::ToAddProduct($this->mProducts[$i]['product_id']);

    $this->mProducts[$i]['attributes'] =
      Catalog::GetProductAttributes($this->mProducts[$i]['product_id']);
  }
}
```

3. Откройте файл `products_list.tpl` из папки `presentation/templates` и вставьте в него код, выделенный полужирным шрифтом в листинге ниже:

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

```
  </p>

  {* Форма Add to Cart *}
  <form class="add-product-form" target="_self" method="post"
    action="{$obj->mProducts[k].link_to_add_product}">

  {* Генерируем список значений атрибутов *}
  <p class="attributes">

  {* Просматриваем список атрибутов и их значений *}
  {section name=l loop=$obj->mProducts[k].attributes}

    {* Генерируем новый тег select? *}
    {if $smarty.section.l.first ||
        $obj->mProducts[k].attributes[l].attribute_name !==
        $obj->mProducts[k].attributes[l.index_prev].attribute_name}
      {$obj->mProducts[k].attributes[l].attribute_name}:
      <select name="attr_{$obj->mProducts[k].attributes[l].attribute_name}">
      {/if}

        {* Генерируем новый тег option *}
        <option value="{$obj->mProducts[k].attributes[l].attribute_value}">
          {$obj->mProducts[k].attributes[l].attribute_value}
        </option>

      {* Закрываем тег select? *}
      {if $smarty.section.l.last ||
          $obj->mProducts[k].attributes[l].attribute_name !==
          $obj->mProducts[k].attributes[l.index_next].attribute_name}
      </select>
      {/if}

    {/section}
    </p>

    {* Добавляем кнопку для отправки формы *}
    <p>
      <input type="submit" name="submit" value="Add to Cart" />
    </p>
    </form>

  </td>
{if $smarty.section.k.index % 2 != 0 && !$smarty.section.k.first ||
    $smarty.section.k.last}
</tr>
{/if}
```

4. В файле `presentation/link.php` модифицируйте начало метода `CheckRequest()` так, как показано ниже.

```
// Выполняет перенаправление по корректному URL в случае необходимости
public static function CheckRequest()
{
  $proper_url = '';

if (isset ($_GET['Search']) || isset($_GET['SearchResults']) ||
    isset ($_GET['AddProduct']))
{
  return ;
}
```

5. В том же файле `link.php` добавьте в конец класса `Link` следующий метод.

```
// Создает ссылку Add to Cart
public static function ToAddProduct($productId)
{
  return self::Build('index.php?AddProduct=' . $productId);
}
```

6. Теперь откройте файл `presentation/product.php` и добавьте в метод `init()` класса `Product` код, выделенный полужирным шрифтом в следующем листинге.

```
$this->mProduct['attributes'] =
  Catalog::GetProductAttributes($this->mProduct['product_id']);

$this->mLocations = Catalog::GetProductLocations($this->_mProductId);

// Генерируем ссылку Add to Cart
$this->mProduct['link_to_add_product'] =
  Link::ToAddProduct($this->_mProductId);

// Генерируем ссылки на страницы отдела и категории
for ($i = 0; $i < count($this->mLocations); $i++)
{
  $this->mLocations[$i]['link_to_department'] =
    Link::ToDepartment($this->mLocations[$i]['department_id']);
```

7. В файл `product.tpl` в папке `presentation/templates` добавьте код, который будет выводить кнопку Add to Cart на страницу товара:

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

  {* Форма Add to Cart *}
  <form class="add-product-form" target="_self" method="post"
    action="{$obj->mProducts[k].link_to_add_product}">

  {* Генерируем список значений атрибутов *}
  <p class="attributes">

  {* Просматриваем список атрибутов и их значений *}
  {section name=l loop=$obj->mProducts[k].attributes}

  {* Генерируем новый тег select? *}
  {if $smarty.section.l.first ||
      $obj->mProducts[k].attributes[l].attribute_name !==
      $obj->mProducts[k].attributes[l.index_prev].attribute_name}
    {$obj->mProducts[k].attributes[l].attribute_name}:
    <select name="attr_{$obj->mProducts[k].attributes[l].attribute_name}">
  {/if}

    {* Генерируем новый тег option *}
    <option value="{$obj->mProducts[k].attributes[l].attribute_value}">
      {$obj->mProducts[k].attributes[l].attribute_value}
    </option>

  {* Закрываем тег select? *}
```

```
  {if $smarty.section.l.last ||
      $obj->mProducts[k].attributes[l].attribute_name !==
      $obj->mProducts[k].attributes[l.index_next].attribute_name}
  </select>
  {/if}

{/section}
</p>

{* Добавляем кнопку для отправки формы *}
<p>
  <input type="submit" name="submit" value="Add to Cart" />
</p>
</form>

{if $obj->mLinkToContinueShopping}
<a href="{$obj->mLinkToContinueShopping}">Continue Shopping</a>
{/if}
```

8. В класс `StoreFront` в файле `presentation/store_front.php` добавьте код, выделенный полужирным шрифтом в листинге ниже:

```
// Заголовок страницы
public $mPageTitle;
// Ссылка PayPal для возврата в магазин
public $mPayPalContinueShoppingLink;

// Конструктор класса
public function __construct()
{
  $this->mLinkToIndex = Link::ToIndex();
}

public function init()
{
  // Создаем ссылку для возврата в магазин в корзине покупателя PayPal
  if (!isset ($_GET['AddProduct']))
  {
    /* Сохраняем текущий запрос, необходимый для работы ссылки в paypal,
       возвращающей в магазин */
    $_SESSION['paypal_continue_shopping'] =
      Link::Build(str_replace(VIRTUAL_LOCATION, '',
                              $_SERVER['REQUEST_URI']));
    $this->mPayPalContinueShoppingLink =
      $_SESSION['paypal_continue_shopping'];
  }
  // Если нажата кнопка Add to Cart, подготавливаем переменные PayPal
  else
  {
    // Очищаем буфер вывода
    ob_clean();

    $product_id = 0;

    // Получаем идентификатор товара, который добавляется в корзину
    if (isset ($_GET['AddProduct']))
      $product_id = (int)$_GET['AddProduct'];
    else
      trigger_error('AddProduct not set');

    $selected_attribute_groups = array ();
```

```
    $selected_attribute_values = array ();

    // Получаем атрибуты выбранного товара, если таковые есть
    foreach ($_POST as $key => $value)
    {
      // Если в массиве POST есть поля, начинающиеся с "attr_"
      if (substr($key, 0, 5) == 'attr_')
      {
        // Получаем имя и значение выбранного атрибута
        $selected_attribute_groups[] = substr($key, strlen('attr_'));
        $selected_attribute_values[] = $_POST[$key];
      }
    }

  // Получаем сведения о товаре
  $product = Catalog::GetProductDetails($product_id);

  // Генерируем URL для PayPal, чтобы добавить товар в корзину
  $paypal_url = PAYPAL_URL . '?cmd=_cart&business=' . PAYPAL_EMAIL .
                '&item_name=' . rawurlencode($product['name']);

    if (count($selected_attribute_groups) > 0)
      $paypal_url .= '&on0=' . implode('/', $selected_attribute_groups) .
                     '&os0=' . implode('/', $selected_attribute_values);

    $paypal_url .=
      '&amount=' . ($product['discounted_price'] == 0 ?
                    $product['price'] : $product['discounted_price']) .
      '&currency_code=' . PAYPAL_CURRENCY_CODE . '&add=1' .
      '&shopping_url=' .
        rawurlencode($_SESSION['paypal_continue_shopping']) .
      '&return=' . rawurlencode(PAYPAL_RETURN_URL) .
      '&cancel_return=' . rawurlencode(PAYPAL_CANCEL_RETURN_URL);

    // Выполняем перенаправление на страницу корзины PayPal
    header('HTTP/1.1 302 Found');
    header('Location: ' . $paypal_url);

    // Очищаем выходной буфер и завершаем выполнение
    flush();
    ob_flush();
    ob_end_clean();
    exit();
  }

  // Загружаем подробные сведения об отделе на страницу отдела
  if (isset ($_GET['DepartmentId']))
  {
```

9. Откройте файл `store_front.tpl` из папки `presentation/templates` и добавьте в него код для кнопки View Cart:

```
<div class="yui-b">
  {include file="search_box.tpl"}
  {include file="departments_list.tpl"}
  {include file=$obj->mCategoriesCell}
  <div class="view-cart">
    <form target="_self" method="post"
     action="{$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>
</div>
```

10. Добавьте в файл `styles/tshirtshop.css` следующие стили.

```
div.yui-b div form.add-product-form
{
  margin: 0;
  padding: 0;
}

.add-product-form p
{
  margin: 0;
  padding: 0 0 10px 0;
}

.view-cart
{
  padding: 10px;
  text-align: center;
}
```

11. Загрузите сайт TShirtShop в браузере и попробуйте щелкнуть на кнопке Add to Cart или View Cart. Должна открыться страница корзины покупателя PayPal, примерно такая, как на рис. 9.4. Поэкспериментируйте с корзиной, если хотите.

Как это работает: интеграция с PayPal

Мы написали солидный фрагмент кода, но он превращает всех посетителей нашего сайта в потенциальных покупателей! Теперь они могут делать покупки в нашем магазине с помощью корзины покупателя PayPal.

Чтобы добиться такого результата, мы сначала добавили кнопки Add to Cart в списки товаров. Для этого мы модифицировали файл `presentation/products_list.php` так, чтобы он добавлял URL Add to Cart к каждому товару:

```
$this->mProducts[$i]['link_to_add_product'] =
  Link::ToAddProduct($this->mProducts[$i]['product_id']);
```

После этого мы внесли изменения в файл шаблона, добавив в него кнопки:

```
    <form class="add-product-form" target="_self" method="post"
     action="{$obj->mProducts[k].link_to_add_product}">
...
    <p>
      <input type="submit" name="submit" value="Add to Cart" />
    </p>
    </form>
```

Такие же изменения мы внесли в файлы `product.php` и `product.tpl`, чтобы реализовать кнопки Add to Cart на страницах каждого товара. Чтобы увидеть, что произойдет после щелчка на такой кнопке, мы используем подключаемый модуль LiveHTTPHeaders для Firefox. На рис. 9.5 выделены URL, генерируемые после щелчка на кнопке, когда выполняется перенаправление с кодом HTTP 302.

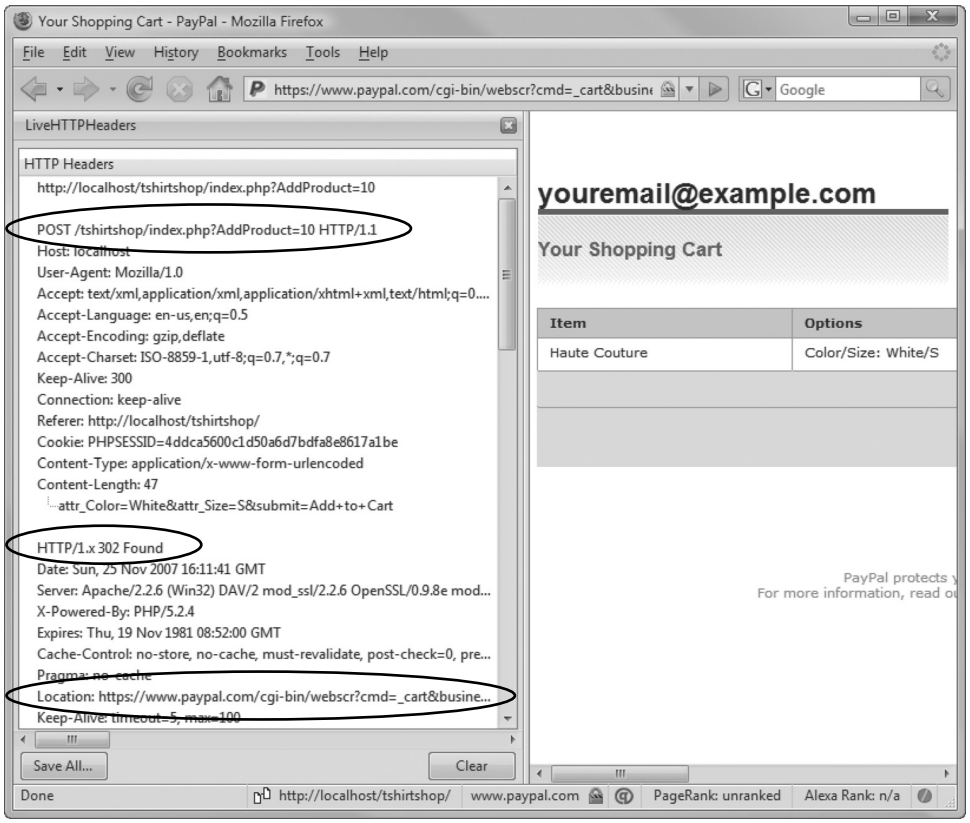


Рис. 9.5. Работа корзины покупателя PayPal

После того как покупатель выполнит оплату, на почтовый адрес, зарегистрированный в PayPal, будет отправлено соответствующее уведомление. Такое же уведомление будет отправлено и покупателю. В учетной записи PayPal отразится изменение баланса на счету, и вы сможете увидеть платеж в истории транзакций.

После получения подтверждения платежа от PayPal вы можете отправлять покупателю купленные им товары. Если вы решите использовать PayPal на своем веб-сайте, потратьте некоторое время на изучение всех ее возможностей. Например, вы можете научить PayPal автоматически учитывать стоимость доставки и налоги.

Использование механизма покупки отдельных товаров в PayPal

Покупка отдельных товаров (Single Item Purchase) в PayPal — это механизм, позволяющий посетителям магазина оплачивать и покупать отдельные товары в обход корзины покупателя. Сама корзина покупателя PayPal станет ненужной нам в главе 12, когда мы создадим собственную корзину.

В главе 14 мы создадим в корзине покупателя кнопку Place Order, которая сохраняет заказ в базе данных и открывает страницу оплаты PayPal. Чтобы открыть страницу оплаты (в обход корзины покупателя PayPal), нужно выполнить перенаправление по ссылке примерно такого вида.

```
https://www.paypal.com/cgi-bin/webscr?cmd=_xclick&business=youremail
@example.com&item_name=Order#123&item_number=123&amount=123&currency_code=
USD&return=www.example.com&cancel_return=www.example.com
```

За более подробной информацией об этом механизме обращайтесь к руководству *Website Payments Standard Integration Guide* на сайте PayPal.

ПОДСКАЗКА

На третьем этапе разработки сайта (в главе 17) мы создадим собственную систему обработки платежей, способную работать с кредитными картами.

Реализуя поддержку покупки отдельных товаров в главе 14, мы будем использовать код, генерирующий URL для перехода на страницу оплаты PayPal. Этот код будет выглядеть примерно так.

```
// Вычисляем общую сумму для корзины
$this->mTotalAmount = ShoppingCart::GetTotalAmount();

// Если нажата кнопка Place Order...
if(isset ($_POST['place_order']))
{
  // Создаем заказ и получаем его ID
  $order_id = ShoppingCart::CreateOrder();

  // Здесь будет находиться ссылка на PayPal
  $redirect =
    PAYPAL_URL . '&item_name=TShirtShop Order ' . urlencode('#') .
$order_id .
    '&item_number=' . $order_id .
    '&amount=' . $this->mTotalAmount .
    '&currency_code=' . PAYPAL_CURRENCY_CODE .
    '&return=' . PAYPAL_RETURN_URL .
    '&cancel_return=' . PAYPAL_CANCEL_RETURN_URL;

  // Заголовок страницы
  header('Location: ' . $redirect);

  exit();
}
```

Работу этого кода мы рассмотрим в главе 14.

Резюме

В этой главе мы интегрировали механизмы PayPal в наш сайт. Это простой и удобный вариант приема платежей от покупателей, которым пользуются многие небольшие предприятия, не желающие самостоятельно работать с кредитными карточками и банковскими счетами покупателей.

Сначала мы кратко рассмотрели основные возможности PayPal, затем создали учетную запись и написали код в приложении, обеспечивающий работу с этой учетной записью. Мы реализовали работу с корзиной покупателя и прием платежей.

В следующей главе мы займемся механизмами администрирования нашего сайта.

Глава 10

Администрирование каталога: отделы и категории

В предыдущих главах наше приложение работало только с информацией, изначально присутствовавшей в базе данных. Возможно, вы ввели эту информацию в базу данных собственноручно или загрузили с сайта книги сценарии, которые заполнили ее данными. Вероятно, вы понимаете, что для реального сайта оба этих способа малопригодны, — в нашем приложении должны быть механизмы добавления в базу данных новых товаров и редактирования данных об уже существующих товарах. Поэтому необходимо создать интерфейс администратора. Хотя посетители сайта никогда его не увидят, он необходим для успешной работы веб-сайта.

В этой и следующей главах мы будем заниматься именно созданием интерфейса администратора. Создав его, мы завершим первую стадию разработки нашего приложения. Поскольку интерфейс администратора можно реализовать множеством способов, его лучше обсудить с заказчиком проекта перед тем, как начинать писать код.

В нашем случае мы создадим панель управления, которая позволит управлять отделами, категориями, товарами и атрибутами товаров на сайте. В этой главе мы займемся только отделами и категориями, оставив все остальное для главы 11. Точнее говоря, в этой главе мы создадим механизмы для

- добавления и удаления отделов,
- изменения информации о существующих отделах (их названий и описаний),
- просмотра списков категорий каждого отдела,
- добавления и удаления категорий,
- изменения информации о существующих категориях (их названий и описаний).

Чтобы предотвратить несанкционированный доступ к интерфейсу администратора, мы также

- создадим форму для ввода идентификатора пользователя и пароля и разрешим доступ к интерфейсу администратора только пользователям с заданными идентификаторами и паролями;
- выясним, как обеспечивать шифрование данных при передаче по сети с помощью SSL.

Предварительное обсуждение интерфейса администратора

На первый взгляд, длинный список необходимых действий может показаться устрашающим. Однако на самом деле все довольно просто — большую часть теории мы уже рассмотрели в предыдущих главах, хотя и из этой вы узнаете немало нового.

Первый шаг к созданию механизма администрирования — создание страницы входа на сайт, примерно такой, как на рис. 10.1.

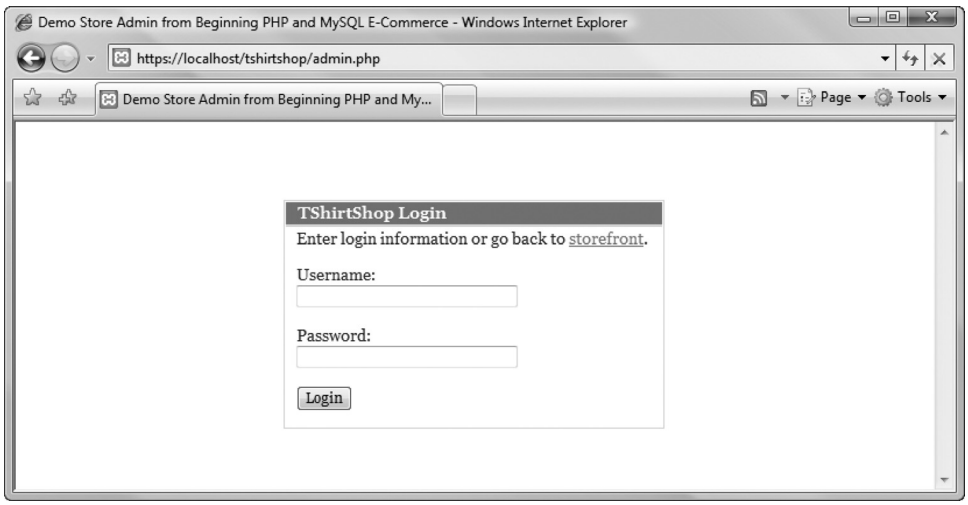


Рис. 10.1. Страница входа на сайт TShirtShop

Далее нам нужно будет создать административную часть сайта (часто ее называют *панелью управления* (control panel)). Главная страница этой части будет содержаться в файле `admin.php`, с которым будет связан шаблон (`store_admin.php`). Кроме того, понадобятся шаблон главного меню (`admin_menu.tpl`), используемого для навигации в административной части, шаблон, отвечающий за аутентификацию (`admin_login`), и два компонентных шаблона для администрирования каталога (`admin_departments` и `admin_categories`).

Войдя на сайт, администратор увидит список отделов, сгенерированный шаблоном Smarty `admin_departments`, загружаемым с главной страницы администрирования, `admin.php`. Этот список показан на рис. 10.2. В нем администратор может

- редактировать название или описание любого отдела, щелкнув на кнопке Edit,
- просматривать категории, относящиеся к отделу, щелкнув на кнопке Edit Categories,
- полностью удалить отдел из базы данных, щелкнув на кнопке Delete (это сработает, только если в отделе нет ни одной категории).

После щелчка на кнопке Edit соответствующая строка таблицы переходит в режим редактирования, как показано на рис. 10.3. Кроме того, вместо кнопки Edit появляются кнопки Update и Cancel. Щелкнув на кнопке Update, можно сохранить измененные данные в базе данных, а щелкнув на кнопке Cancel — просто возвратить таблицу в ее исходное состояние.

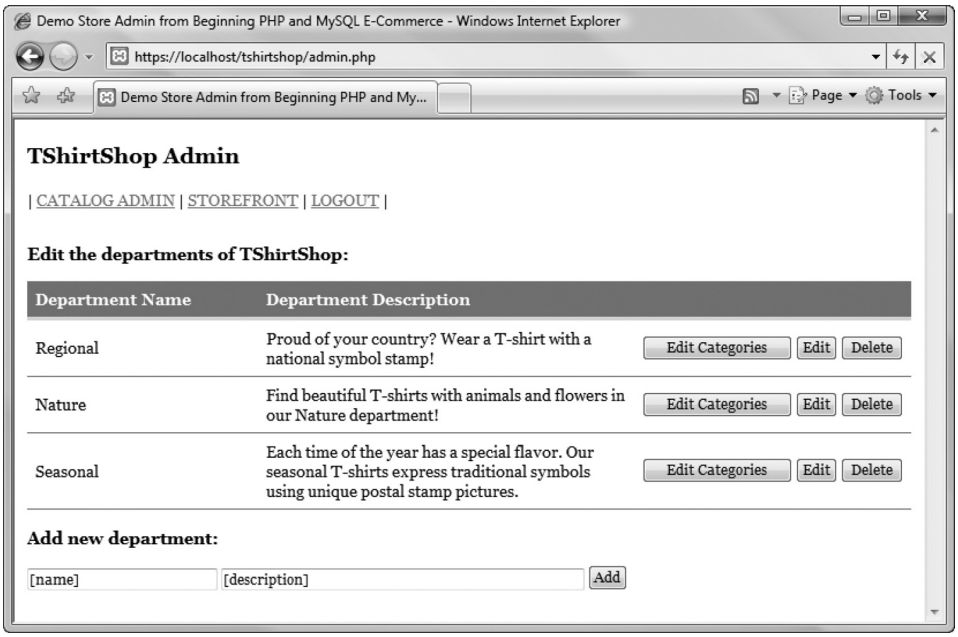


Рис. 10.2. Страница управления отделами сайта TShirtShop

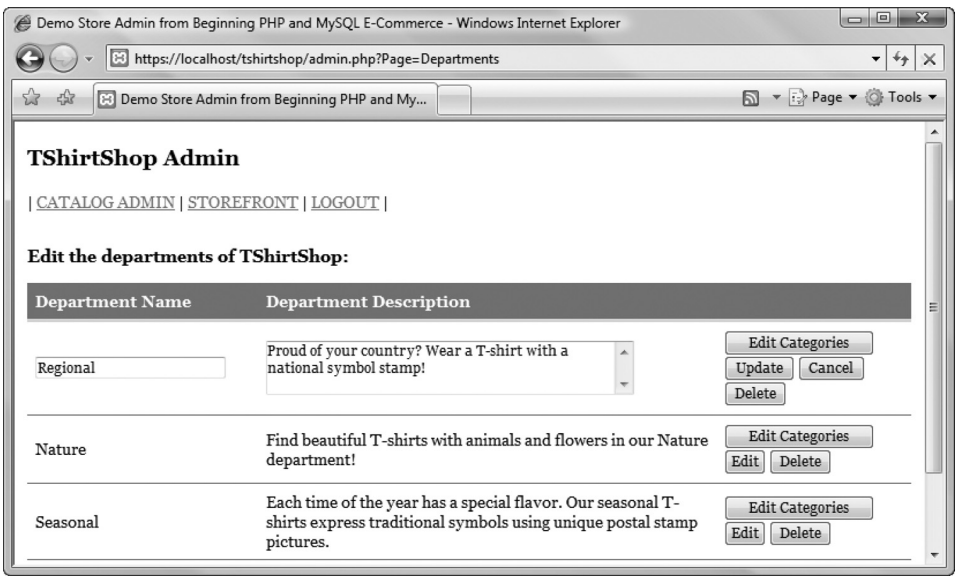


Рис. 10.3. Редактирование информации об отделе

Чтобы добавить в базу данных новый отдел, администратор должен ввести его название и описание в полях в нижней части страницы и щелкнуть на кнопке Add.

После щелчка на кнопке Edit Categories страница `admin.php` загружается заново с дополнительным параметром в строке запроса, `DepartmentId`. Этот параметр приказывает `admin.php` загрузить шаблон Smarty `admin_categories`, который позволяет редактировать информацию о категориях выбранного отдела (рис. 10.4).

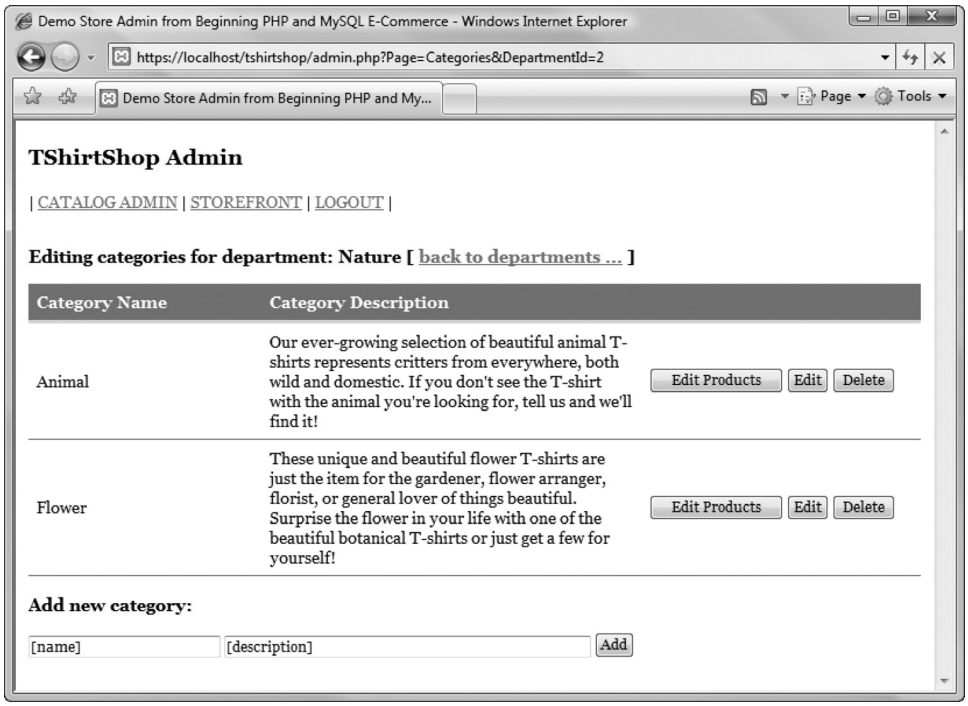


Рис. 10.4. Страница редактирования категорий в TShirtShop

Эта страница функционирует аналогично странице для работы с отделами. Кроме того, на ней есть ссылка для возврата на страницу администрирования о тделов.

Перемещением между страницами администрирования отделов, категорий и товаров управляют параметры в строке запроса. Как видно на рис. 10.4, при выборе отдела его идентификатор указывается в строке запроса. Такой же подход мы использовали при создании страницы `index.php` — там мы определяли, какой компонентный шаблон нужно загружать, анализируя параметры из строки запр оса.

Часть сайта, предназначенная для администрирования каталога, состоит из страницы `admin.php`, нескольких вспомогательных файлов PHP и шаблонов Smarty. Мы создадим все эти компоненты по очереди. Сначала для каждого компонента мы создадим уровень представления, затем напишем код уровня логики приложения и наконец напишем методы уровня данных.

В последующих главах мы будем расширять административный раздел сайта. В главе 11 мы добавим возможности администрирования товаров, а в дальнейшем — механизмы управления заказами и корзинами покупателей, а также возможность работы с данными пользователей — адресами, номерами телефонов, данными о кредитных картах и т.д.

Создание страницы администрирования каталога

Прежде чем создавать страницы администрирования, нужно позаботиться о механизмах, которые ограничат доступ к этим страницам. Вряд ли вам понравится, если каждый зашедший на сайт сможет менять данные в каталоге.

Безопасность веб-сайтов — это обширная тема, и чем ценнее данные, которые нужно защитить, тем более изощренной должна быть защита. Хотя мы и не располагаем ресурсами для создания защищенной среды уровня крупных компаний, нам все равно нужно защищать от несанкционированного доступа свои данные и данные наших посетителей.

Механизмы безопасности, которые мы реализуем, будут основываться на трех основных концепциях.

- *Аутентификация.* Это процесс однозначного опознавания пользователей. Обычно для этого применяются имена пользователей и пароли, и мы будем использовать именно этот метод аутентификации на нашем сайте.
- *Авторизация.* Это процесс предоставления пользователю доступа только к тем ресурсам, которыми он имеет право пользоваться. Например, одни администраторы могут только редактировать данные в каталоге, а другие — только работать с данными пользователей. На нашем маленьком сайте все администраторы смогут работать с любыми данными, но на крупных сайтах обычно используется более четкое разграничение обязанностей — и для удобства, и для безопасности.
- *Защита каналов связи.* Разумеется, все усилия по аутентификации и авторизации будут напрасными, если хакер сможет перехватить конфиденциальные данные, просто прослушивая трафик нашего сайта и отбирая из него все интересующие его данные. Кроме трафика, интерес для хакеров могут представлять журналы операций, в которых указываются имена пользователей и пароли. Для защиты от таких угроз мы будем использовать протокол HTTPS, шифрующий передаваемые по сети данные и обеспечивающий приемлемый уровень безопасности.

Использование защищенных соединений

Протокол HTTP не является защищенным, и даже если вы защитите конфиденциальные данные на своем сайте паролями (или другими методами аутентификации), передаваемые по сети данные все равно могут быть перехвачены и украдены. Во избежание этого мы применим соединения SSL (Secure Socket Layer — уровень защищенных соединений) в протоколе HTTPS.

Чтобы сервер мог принимать поступающие запросы на соединения HTTPS, он должен обладать *сертификатом безопасности* (security certificate). Эти сертификаты практически представляют собой пары из открытого и закрытого ключей, используемые в асимметричных алгоритмах шифрования. Вы можете сгенерировать такой сертификат самостоятельно, но сертификаты, не выданные крупными компаниями, специализирующимися на безопасности (например, VeriSign или Thawte), обычно не пользуются доверием в Интернете.

Сертификаты, в которых отсутствует цифровая подпись известных компаний, заставят браузеры клиентов сомневаться в вашей надежности. Когда пользователь пытается открыть защищенную страницу, сертификат которой выдан непонятно кем, браузер выдает предупреждающее сообщение. Возможно, это не страшно, если ваши работники будут посещать внутренний сайт вашей компании, но у посетителей, увидевших такое сообщение на странице оплаты заказа, сразу возникнут сомнения.

Если вы установили XAMPP так, как описано в главе 1, значит, ваш сервер Apache уже обладает сертификатом безопасности. В тестовых целях можно загрузить поддерживающую SSL версию Apache по адресу `http://www.devside.net/web/server/free/download`.

Для рабочего сайта вам понадобится купить сертификат через компанию, предоставляющую веб-хостинг, или, если вы используете собственный сервер, заказать его самостоятельно у известной и уважаемой компании, предоставляющей такие услуги.

- VeriSign (`http://www.verisign.com/`)
- Thawte (`http://www.thawte.com/`)
- InstantSSL (`http://www.instantssl.com/`)

В веб-браузерах есть встроенные базовые сертификаты от таких компаний, и они могут проверять цифровые подписи сертификатов SSL, которые вы используете. При успешной проверке предупреждения появляться не будут, и использующие SSL соединения будут применяться практически незаметно для пользователя. Например, при загрузке защищенной страницы в Opera рядом со строкой адреса появляется маленький значок в виде замка. Щелкнув на этом значке, можно открыть окно со сведениями об используемом сертификате (пример такого окна показан на рис. 10.5).

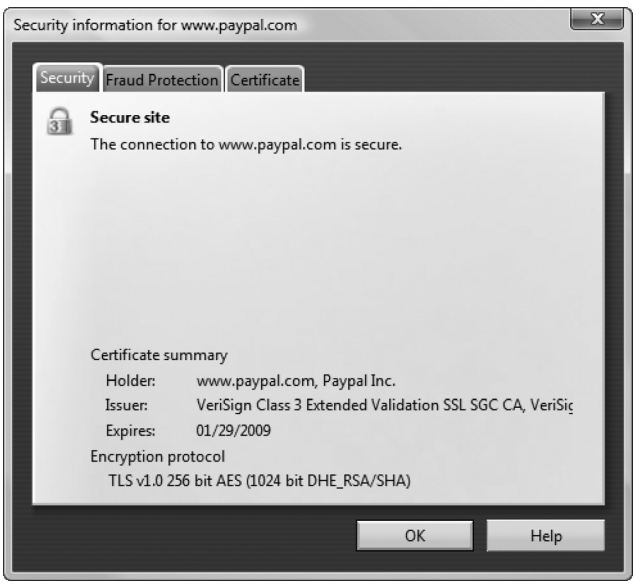


Рис. 10.5. Проверка сертификата веб-сайта в Opera

Разумеется, сертификат, выданный локальным компьютером в XAMPP, не будет принят браузерами — они не узнают его цифровую подпись. В таком случае браузеры будут выдавать пользователю предупреждения. Пример такого предупреждения в Safari показан на рис. 10.6.

Если вы щелкнете на кнопке Show Certificate, то увидите, что сертификат выдан локальным компьютером для Apache Friends (Apache Friends — это создатели пакета XAMPP).

Разные браузеры по-разному реагируют на подозрительные сертификаты. Например, в Internet Explorer предупреждение еще строже, чем в Safari (рис. 10.7).



Рис. 10.6. Safari не нравятся сертификаты, выданные непонятно кем

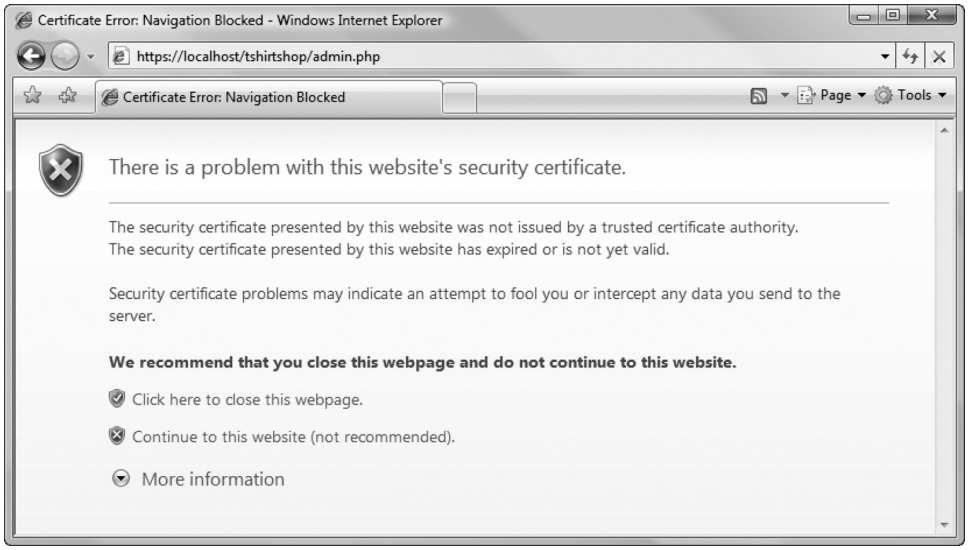


Рис. 10.7. Internet Explorer такие сертификаты тоже не нравятся

Конфигурирование TShirtShop для SSL

Если вы решите использовать SSL, вам понадобится установить сертификат. Как это сделать, рассказано ниже. При использовании SSL рекомендуется предоставлять доступ к любым страницам с важной информацией только через SSL, т.е. автоматически выполнять перенаправление со ссылок `http://` на ссылки `https://`.

Однако, если вы хотите отложить использование SSL и на первое время сосредоточиться на создании административной части сайта, можете это сделать. Чтобы сделать сайт конфигурируемым, добавим константу с именем `USE_SSL` в файл `include/config.php`. Если значение этой константы — `yes`, защищенные страницы можно будет загружать только через HTTPS; в противном случае они будут доступны через HTTP.

Получение сертификата SSL

Получение сертификата — задача относительно простая. Мы опишем шаги, необходимые для получения сертификата от VeriSign, но процесс выглядит примерно так же и для других провайдеров. Подробные инструкции доступны на сайте Verisign

(http://www.verisign.com/). Кроме того, вы можете получить от Verisign бесплатный пробный сертификат, который действителен в течение определенного срока. Вот основные действия.

1. Зарегистрируйтесь для получения пробного сертификата на веб-сайте Verisign.
2. Сгенерируйте запрос на сертификат, Certificate Signing Request (CSR), на своем веб-сайте. Для этого вам придется указать кое-какую личную информацию, прежде всего — имя вашего веб-сайта. Чтобы сгенерировать запрос, понадобится установить модуль SSL на ваш веб-сервер. Как это сделать, описано в учебном курсе по адресу http://sitepoint.com/article/securing-apache-2-server-ssl.
3. Скопируйте содержимое сгенерированного CSR в систему запросов VeriSign.
4. Через небольшой промежуток времени вы получите сертификат на сайте VeriSign. Этот сертификат нужно скопировать и установить на вашем веб-сайте.

Если у вас возникнут какие-то затруднения, обратитесь к подробным инструкциям на сайте VeriSign.

Принудительное использование соединений SSL

Установив сертификат, вы можете организовать доступ к любой странице вашего веб-сайта через SSL, просто заменив http:// в ее адресе на https:// (при условии, что ваш брандмауэр позволяет устанавливать соединения SSL, по умолчанию использующие порт 443).

Разумеется, разрешать доступ только с помощью SSL ко всем страницам сайта бессмысленно. Выбирая, какие страницы делать доступными только через HTTPS, учитывайте следующие моменты.

- Поисковые системы не индексируют страницы, доступные через HTTPS.
- Для передачи страниц через HTTPS задействуются ресурсы сервера — он должен шифровать эти страницы перед отправкой.

Как видите, есть серьезные причины защищать с помощью SSL только важные части сайта. В этой главе мы обеспечим доступ только с применением SSL к странице входа в административную часть. В дальнейшем, когда мы сами будем обрабатывать платежи покупателей, мы используем SSL для защиты страниц регистрации покупателей, страниц входа и страниц приема платежей.

Если вы хотите, чтобы доступ к административному сценарию (admin.php) был возможен только через HTTPS, добавьте в начало файла presentation/store_admin.php следующий код (мы сделаем это в упражнении, не добавляйте этот код в файл сейчас).

```
// Конструктор класса
public function __construct()
{
  $this->mSiteUrl = Link::Build('', 'https');
  // Разрешаем доступ к странице только через HTTPS, если USE_SSL
  // установлен в yes
  if (USE_SSL == 'yes' && getenv('HTTPS') != 'on')
  {
    header ('Location: https://' . getenv('SERVER_NAME') .
                                   getenv('REQUEST_URI'));

    exit();
  }
}
```

Заметьте, что этот код не заставляет использовать HTTPS, если константа `USE_SSL`, определенная в файле `include/config.php`, не установлена в значение `yes`. Это может быть полезно, если вам нужно отлаживать сайт, а у вас нет доступа к серверу, на котором можно использовать SSL.

Аутентификация администраторов

Поскольку мы не хотим, чтобы инструменты администрирования каталога были доступны всем пользователям, мы должны создать механизмы аутентификации и авторизации, контролирующие доступ к этим инструментам. Пользователи, которые пожелают воспользоваться инструментами администрирования, должны будут пройти аутентификацию. После того как сайт убедится, что у пользователя есть право доступа к этим инструментам, мы можем разрешить загрузку административного интерфейса в браузере этого пользователя. На данном этапе у нас будет всего две категории пользователей — анонимные (обычные посетители сайта) и администраторы, у которых есть доступ к административному интерфейсу. Позже мы разрешим обычным пользователям создавать личные учетные записи на нашем сайте, но мы еще не добрались до этого этапа.

В TShirtShop мы будем использовать так называемую HTTP-аутентификацию, которая позволяет контролировать процесс входа на сайт с помощью HTML-формы. После того как клиент прошел аутентификацию, мы сохраняем cookie-файл на его компьютере и используем этот файл для аутентификации при последующих посещениях сайта. Если cookie-файл не найден, клиент должен пройти аутентификацию заново.

ЗАМЕЧАНИЕ

Мы подразумеваем, что администратор будет обращаться к страницам, требующим аутентификации, с машины, на которой разрешено использование cookie-файлов.

Имена пользователей и соответствующие им пароли можно хранить разными способами. Например, из главы 16 вы узнаете, как можно хранить в базе данных хешированные (зашифрованные) пароли пользователей.

ПОДСКАЗКА

Хеширование — это часто используемый прием при хранении паролей. Хеш-значение пароля вычисляется как результат определенных математических операций над байтами этого пароля. Когда пользователь пытается пройти аутентификацию и вводит пароль, этот пароль хешируется и результат хеширования сравнивается с хранящимся в базе данных. Если значения совпадают, значит, пароль введен верно. Такой подход позволяет не хранить на сервере незашифрованные пароли. Важное свойство алгоритмов хеширования — теоретически невозможно вычислить исходный пароль по хеш-значению. Что касается практики, во многих распространенных алгоритмах хеширования (MD5, SHA-0 и SHA-1) недавно были найдены слабые места, позволяющие относительно легко подбирать пароли по их хеш-значениям.

Более простой метод (но мало подходящий для реальных приложений) — хранить сочетания имен пользователей и паролей в файле PHP. Практически единственное его преимущество — скорость. Разумеется, в файле PHP имена пользователей и пароли также можно шифровать, но делать это не так удобно, как в базе данных.

В следующем упражнении мы реализуем хранение паролей в виде открытого текста, но в реальных коммерческих приложениях так поступать не следует. К хешированию мы еще вернемся в главе 16.

Упражнение: создание каркаса страницы администрирования

1. Модифицируйте файл presentation/templates/first_page_contents.tpl, добавив в него ссылку на страницу администрирования. В общем-то, это необязательно — данная ссылка понадобится только нам для упрощения тестирования. Если вы не хотите ее добавлять, пропустите первые два шага этого упражнения.

```
{* first_page_contents.tpl *}
{load_presentation_object filename="first_page_contents" assign="obj"}
<p class="description">
  We hope you have fun developing TShirtShop, the e-commerce store from
  Beginning PHP and MySQL E-Commerce: From Novice to Professional!
</p>
<p class="description">
  We have the largest collection of t-shirts with postal stamps on Earth!
  Browse our departments and categories to find your favorite!
</p>
<p>Access the <a href="{$obj->mLinkToAdmin}">admin page</a>.</p>
{include file="products_list.tpl"}
```

2. Создайте файл first_page_contents.php в папке presentation и добавьте в него следующий код (он необходим для добавления на главную страницу ссылки, ведущей на страницу администрирования):

```
<?php
class FirstPageContents
{
  public $mLinkToAdmin;

  public function __construct()
  {
    $this->mLinkToAdmin = Link::ToAdmin();
  }
}
?>
```

3. Создайте в корневой папке сайта (в папке tshirtshop) файл admin.php и вставьте в него приведенный ниже код. Как вы, возможно, заметите, этот код весьма похож на содержащийся в файле index.php, однако здесь мы не проверяем входную ссылку с помощью Link::CheckRequest().

```
<?php
// Запускаем сеанс
session_start();

// Создаем буфер вывода
ob_start();

// Включаем вспомогательные файлы
require_once 'include/config.php';
require_once BUSINESS_DIR . 'error_handler.php';

// Задаем обработчик ошибок
ErrorHandler::SetHandler();

// Загружаем шаблон страницы приложения
require_once PRESENTATION_DIR . 'application.php';
require_once PRESENTATION_DIR . 'link.php';

// Загружаем дескриптор базы данных
require_once BUSINESS_DIR . 'database_handler.php';
```

```
// Загружаем код уровня логики приложения
require_once BUSINESS_DIR . 'catalog.php';

// Загружаем файл шаблона Smarty
$application = new Application();

// Отображаем страницу
$application->display('store_admin.tpl');

// Закрываем соединение с базой данных
DatabaseHandler::Close();

// Выводим содержимое буфера
flush();
ob_flush();
ob_end_clean();
?>
```

4. Создайте в папке `presentation/templates` файл шаблона `store_admin.tpl`, который загружается в только что созданном нами файле `admin.php`. Поместите в созданный файл такой код.

```
{load_presentation_object filename="store_admin" assign="obj"}
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title>Demo Store Admin from Beginning PHP and MySQL E-Commerce</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <link href="{$obj->mSiteUrl}styles/tshirtshop.css" type="text/css"
     rel="stylesheet" />
  </head>
  <body>
    <div id="doc" class="yui-t7">
      <div id="bd">
        <div class="yui-g">
          {include file=$obj->mMenuCell}
        </div>
        <div class="yui-g">
          {include file=$obj->mContentsCell}
        </div>
      </div>
    </div>
  </body>
</html>
```

5. Создайте в папке `presentation` файл `store_admin.php` и поместите в него такой код.

```
<?php
class StoreAdmin
{
  public $mSiteUrl;
  // Определяем файл шаблона для меню
  public $mMenuCell = 'blank.tpl';
  // Определяем файл шаблона для содержимого страницы
  public $mContentsCell = 'blank.tpl';

  // Конструктор класса
  public function __construct()
  {
    $this->mSiteUrl = Link::Build('', 'https');
```

```
    // Разрешаем доступ к этой странице только через HTTPS,
    // если USE_SSL установлена в значение yes
    if (USE_SSL == 'yes' && getenv('HTTPS') != 'on')
    {
      header ('Location: https://' . getenv('SERVER_NAME') .
              getenv('REQUEST_URI'));

      exit();
    }
  }

  public function init()
  {
    // Если аутентификация не выполнялась, загружаем шаблон admin_login
    if (!(isset ($_SESSION['admin_logged'])) ||
        $_SESSION['admin_logged'] != true)
      $this->mContentsCell = 'admin_login.tpl';
    else
    {
      // Если аутентификация проведена, загружаем
      // меню страницы администрирования
      $this->mMenuCell = 'admin_menu.tpl';

      // При выходе...
      if (isset ($_GET['Page']) && ($_GET['Page'] == 'Logout'))
      {
        unset($_SESSION['admin_logged']);
        header('Location: ' . Link::ToAdmin());

        exit();
      }
    }
  }
}
?>
```

6. Добавьте константу `USE_SSL` и информацию об администраторах в конец файла `include/config.php`. Если вы предпочитаете пока что не использовать SSL, задайте `USE_SSL` значение `no`. Как видите, по умолчанию учетная запись администратора называется `tshirtshopadmin`, с таким же паролем. Разумеется, в реальном сайте лучше использовать идентификаторы и пароли, которые будет труднее отгадать злоумышленникам.

```
// Если эта константа не установлена в no, доступ к
// страницам администрирования
возможен только с помощью SSL
define('USE_SSL', 'yes');

// Идентификатор и пароль администратора
define('ADMIN_USERNAME', 'tshirtshopadmin');
define('ADMIN_PASSWORD', 'tshirtshopadmin');
```

ЗАМЕЧАНИЕ

Выше уже упоминалось, что в главе 16 мы подробнее рассмотрим хеширование и методы работы с хешированными паролями, хранящимися в базе данных. Чтобы использовать хеширование сейчас, в файле `config.php` нужно хранить хеш-код пароля, а не сам пароль. Вычислить хеш-код можно, применив к паролю функцию `sha1()` — она вычисляет этот код по алгоритму SHA1. Если это звучит невразумительно, не волнуйтесь — в главе 16 мы рассмотрим данную тему гораздо подробнее.

7. Создайте компонентный шаблон `admin_login`, отображающий поля для ввода идентификатора пользователя и пароля. Создайте файл `presentation/templates/admin_login.tpl` и поместите в него следующий код.

```
{* admin_login.tpl *}
{load_presentation_object filename="admin_login" assign="obj"}
<div class="login">
  <p class="login-title">TShirtShop Login</p>
  <form method="post" action="{$obj->mLinkToAdmin}">
    <p>
      Enter login information or go back to
      <a href="{$obj->mLinkToIndex}">storefront</a>.
    </p>
{if $obj->mLoginMessage neq ""}
    <p class="error">{$obj->mLoginMessage}</p>
{/if}
    <p>
      <label for="username">Username:</label>
      <input type="text" name="username" size="35" value="{$obj->mUsername}" />
    </p>
    <p>
      <label for="password">Password:</label>
      <input type="password" name="password" size="35" value="" />
    </p>
    <p>
      <input type="submit" name="submit" value="Login" />
    </p>
  </form>
</div>
```

8. Создайте в папке `presentation` файл `admin_login.php` и введите в него следующий код.

```
<?php
// Класс, отвечающий за аутентификацию администраторов
class AdminLogin
{
  // Public-переменные, доступные в шаблонах Smarty
  public $mUsername;
  public $mLoginMessage = '';
  public $mLinkToAdmin;
  public $mLinkToIndex;

  // Конструктор класса
  public function __construct()
  {
  // Проверяем правильность ввода идентификатора пользователя и пароля
  if (isset ($_POST['submit']))
  {
    if ($_POST['username'] == ADMIN_USERNAME
        && $_POST['password'] == ADMIN_PASSWORD)
    {
      $_SESSION['admin_logged'] = true;

      header('Location: ' . Link::ToAdmin());
      exit();
    }
    else
    {
      $this->mLoginMessage = 'Login failed. Please try again:';
    }
```

```
    $this->mLinkToAdmin = Link::ToAdmin();
    $this->mLinkToIndex = Link::ToIndex();
  }
}
?>
```

9. Создайте в папке `presentation/templates` файл `admin_menu.tpl` и поместите в него следующий код.

```
{* admin_menu.tpl *}
{load_presentation_object filename="admin_menu" assign="obj"}
<h1>TShirtShop Admin</h1>
<p> |
  <a href="{$obj->mLinkToStoreAdmin}">CATALOG ADMIN</a> |
  <a href="{$obj->mLinkToStoreFront}">STOREFRONT</a> |
  <a href="{$obj->mLinkToLogout}">LOGOUT</a> |
</p>
```

10. Создайте в папке `presentation` файл `admin_menu.php` и введите в него такой код.

```
<?php
class AdminMenu
{
  public $mLinkToStoreAdmin;
  public $mLinkToStoreFront;
  public $mLinkToLogout;

  public function __construct()
  {
    $this->mLinkToStoreAdmin = Link::ToAdmin();
    $this->mLinkToStoreFront = Link::ToIndex();
    $this->mLinkToLogout = Link::ToLogout();
  }
}
?>
```

11. Откройте файл `presentation/link.php` и модифицируйте метод `Build()` в этом файле так, как показано в следующем листинге. Это добавит в данный метод поддержку ссылок HTTPS.

```
public static function Build($link, $type = 'http')
{
  $base = (($type == 'http' || USE_SSL == 'no') ? 'http://' : 'https://') .
          getenv('SERVER_NAME');
  // Если константа HTTP_SERVER_PORT определена, и значение отличается
  // от используемого по умолчанию...
  if (defined('HTTP_SERVER_PORT') && HTTP_SERVER_PORT != '80' &&
      strpos($base, 'https') === false)
  {
    // Добавляем номер порта
    $base .= ':' . HTTP_SERVER_PORT;
  }
  $link = $base . VIRTUAL_LOCATION . $link;
  // Escape-символы для html
  return htmlspecialchars($link, ENT_QUOTES);
}
```

12. В том же файле добавьте в конец класса `Link` два метода.

```
// Создает ссылку на страницу администрирования
public static function ToAdmin($params = '')
{
  $link = 'admin.php';
```

```
  if ($params != '')
    $link .= '?' . $params;

  return self::Build($link, 'https');
}

// Создает ссылку для выхода
public static function ToLogout()
{
  return self::ToAdmin('Page=Logout');
}
```

13. Добавьте в файл `styles/tshirtshop.css` следующие стили.

```
.login {
  color: #333333;
  display: block;
  border: 1px solid #c6e1ec;
  margin: 50px auto;
  width: 325px;
}

.login form p {
  padding: 0 10px;
}

label {
  display: block;
}

.error {
  color: #ff0000;
  font-size: 85%;
  font-weight: bold;
}

.login-title {
  background: #0590c7;
  border-bottom: 2px solid #c6e1ec;
  color: #ffffff;
  display: block;
  font-size: 93%;
  font-weight: bold;
  margin: 1px;
  padding: 2px 10px;
}

.no-items-found {
  color: #ff0000;
}

table.tss-table {
  width: 100%;
}

table.tss-table th {
  background: #0590c7;;
  border: none;
  border-bottom: 3px solid #c6e1ec;
  color: #ffffff;
  text-align: left;
}

table.tss-table td {
```

```
  border: none;
  border-bottom: 1px solid #0590c7;
}
```

14. Откройте сайт TShirtShop в своем любимом браузере. На загрузившейся странице приветствия должна присутствовать ссылка на страницу администрирования. Щелкните на этой ссылке, и откроется HTML-форма для ввода идентификатора пользователя и пароля. На рис. 10.8 показано сообщение, которое вы должны получить, введя неправильный пароль. При вводе правильных идентификатора и пароля (`tshirtshopadmin`) должна открыться страница администрирования каталога, на которой пока что есть только основное меню. Скоро мы добавим на эту страницу остальные необходимые элементы.

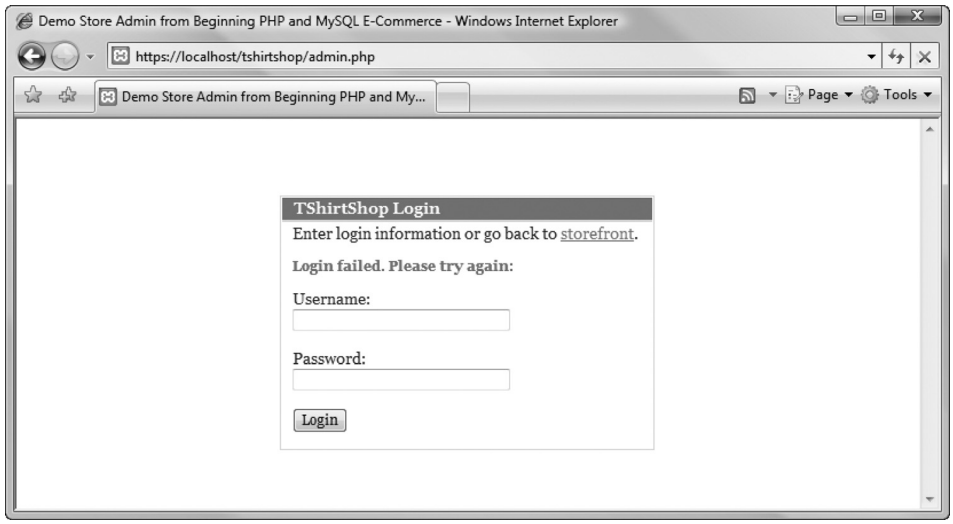


Рис. 10.8. Страница входа на административную часть сайта

Как это работает: аутентификация администраторов

В этом упражнении мы создали следующие компоненты.

- `admin.php`. Это точка входа в административную часть сайта. Мы продолжим добавлять функциональность в этот файл в следующих упражнениях этой главы. Просмотрев его код, вы, вероятно, заметили, что он ничего не выводит в окне браузера — он только подготавливает среду и загружает компонентный шаблон Smarty `store_admin`.
- `store_admin`. Компонентный шаблон (состоящий из файлов `store_admin.php` и `store_admin.tpl`), загружаемый файлом `admin.php` для генерации каркаса страницы администрирования каталога. Создавая различные инструменты администрирования, мы будем добавлять новые шаблоны. Например, позже в этой главе мы создадим инструменты для работы с отделами и категориями.
- `admin_login`. Компонентный шаблон, отвечающий за выполнение аутентификации и авторизации администраторов. Он отображает форму для ввода идентификатора пользователя и пароля и выполняет аутентификацию.

Из всех созданных в этом упражнении файлов наиболее интересен, вероятно, `store_admin.php`. Сначала код в этом файле проверяет, прошел ли посетитель аутентификацию как администратор (проверяется значение сеансовой переменной `admin_logged`). Если нет (`admin_logged` установлено равным `false`), загружается компонентный шаблон `admin_login`.

```
public function init()
{
  // Если аутентификация не проводилась, загружаем шаблон admin_login
  if (!(isset ($_SESSION['admin_logged'])) ||
      $_SESSION['admin_logged'] != true)
    $this->mContentsCell = 'admin_login.tpl';
```

Механизм аутентификации в объекте `AdminLogin` сохраняет текущее состояние в сеансовой переменной `admin_logged`. В функции `__construct()` мы проверяем, совпадают ли введенные пользователем идентификатор и пароль с теми, которые указаны в файле `config.php` в константах `ADMIN_USERNAME` и `ADMIN_PASSWORD`; если совпадают, устанавливаем `admin_logged` равным значению `true` и выполняем перенаправление на `admin.php`.

```
// Проверяем правильность ввода идентификатора пользователя и пароля
if (isset ($_POST['submit']))
{
  if ($_POST['username'] == ADMIN_USERNAME
      && $_POST['password'] == ADMIN_PASSWORD)
  {
    $_SESSION['admin_logged'] = true;

    header('Location: ' . Link::ToAdmin());
    exit();
  }
  else
  {
    $this->mLoginMessage = 'Login failed. Please try again:';
  }
```

Ссылка для выхода из административной части сайта в файле `admin_menu.tpl` просто устанавливает переменную `admin_logged` равной значению `false` и выполняет перенаправление на `admin.php`. В результате при новой попытке зайти в административную часть пользователь опять должен будет пройти аутентификацию.

```
// При выходе...
if (isset ($_GET['Page']) && ($_GET['Page'] == 'Logout'))
{
  unset($_SESSION['admin_logged']);
  header('Location: ' . Link::ToAdmin());

  exit();
}
```

Администрирование отделов

Раздел администрирования отделов должен позволять администраторам добавлять и удалять отделы, а также редактировать информацию о существующих отделах. Чтобы реализовать эту функциональность, необходимо добавить новый код в уровень представления, уровень логики приложения и уровень данных.

Одно из основных свойств трехуровневых приложений (в том числе нашего) заключается в том, что в конечном итоге уровень логики приложения и уровень данных существуют только для обеспечения работы уровня представления. Определив, какой внешний вид и функциональность будет иметь сайт, мы практически определяем, что должны уметь уровень данных и уровень логики приложения (другими словами, какая функциональность нужна для обеспечения работы пользовательского интерфейса).

Составив функциональные и технические спецификации, вы будете точно знать, что к какому уровню относить. При этом порядок написания кода будет не слишком важен. Однако, за исключением очень больших проектов, требующих тщательного планирования и проектирования, такая гибкость редко достижима на практике.

В этой книге мы почти всегда начинаем реализовывать новую функциональность с уровня данных. Однако сейчас мы начнем работать с уровня представления, поскольку вы уже хорошо знаете архитектуру приложения и можете предвидеть, как будет реализована новая функциональность на уровне логики приложения и уровне данных. Такие знания необходимы, поскольку из уровня представления мы будем вызывать методы уровня логики приложения, новый код которого еще не написан. Если бы вы не знали, какую функциональность создавать на уровне логики приложения и уровне данных, мы не смогли бы начать работу с уровня представления.

Реализация уровня представления

Еще раз посмотрите на шаблон `admin_departments` в работе (на рис. 10.9 показан этот шаблон, работающий в окне Internet Explorer для Windows).

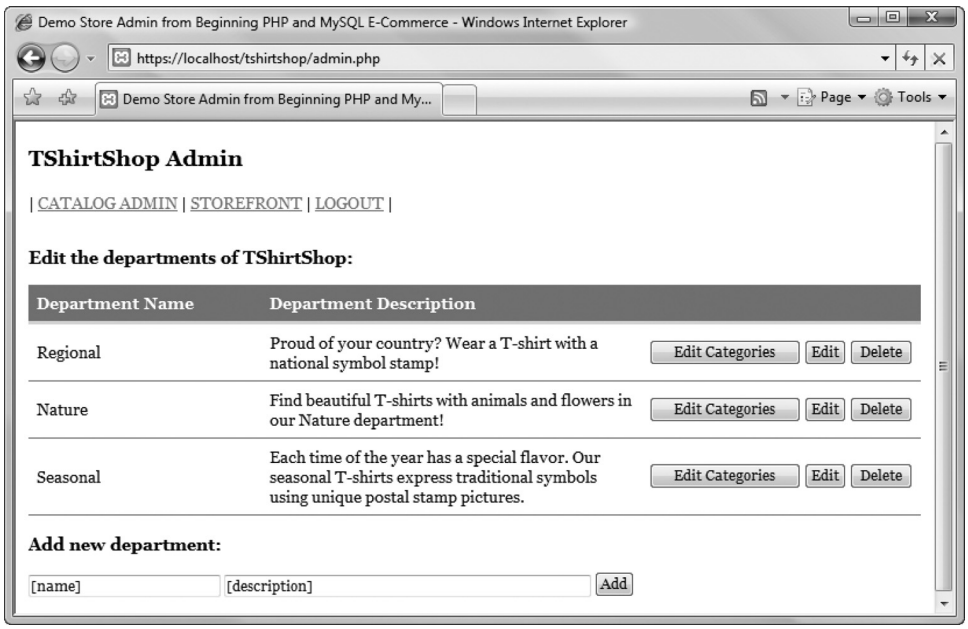


Рис. 10.9. Компонентный шаблон `admin_departments` в работе

Этот компонентный шаблон генерирует список, заполненный информацией об отделах, и инструменты для редактирования этой информации (два текстовых поля и кнопку добавления нового отдела в список).

После щелчка на кнопке Edit в строке какого-нибудь из отделов в списке имя и описание этого отдела становятся редактируемыми, и вместо этой кнопки появляются кнопки Update и Cancel, как показано на рис. 10.3.

Упражнение: создание компонентного шаблона admin_departments

1. Создайте в папке presentation/templates файл admin_departments.tpl и поместите в него следующий код.

```
{* admin_departments.tpl *}
{load_presentation_object filename="admin_departments" assign="obj"}
<form method="post"
 action="{$obj->mLinkToDepartmentsAdmin}">
  <h3>Edit the departments of TShirtShop:</h3>
{if $obj->mErrorMessage}<p class="error">{$obj->mErrorMessage}</p>{/if}
{if $obj->mDepartmentsCount eq 0}
  <p class="no-items-found">There are no departments in your database!</p>
{else}
  <table class="tss-table">
    <tr>
      <th width="200">Department Name</th>
      <th>Department Description</th>
      <th width="240"> </th>
    </tr>
  {section name=i loop=$obj->mDepartments}
    {if $obj->mEditItem == $obj->mDepartments[i].department_id}
    <tr>
      <td>
        <input type="text" name="name"
         value="{$obj->mDepartments[i].name}" size="30" />
      </td>
      <td>
      {strip}
        <textarea name="description" rows="3" cols="60">
          {$obj->mDepartments[i].description}
        </textarea>
      {/strip}
      </td>
      <td>
        <input type="submit"
         name="submit_edit_cat_{$obj->mDepartments[i].department_id}"
         value="Edit Categories" />
        <input type="submit"
         name="submit_update_dept_{$obj->mDepartments[i].department_id}"
         value="Update" />
        <input type="submit" name="cancel" value="Cancel" />
        <input type="submit"
         name="submit_delete_dept_{$obj->mDepartments[i].department_id}"
         value="Delete" />
      </td>
    </tr>
    {else}
    <tr>
      <td>{$obj->mDepartments[i].name}</td>
      <td>{$obj->mDepartments[i].description}</td>
      <td>
        <input type="submit"
         name="submit_edit_cat_{$obj->mDepartments[i].department_id}"
         value="Edit Categories" />
        <input type="submit"
         name="submit_edit_dept_{$obj->mDepartments[i].department_id}"
         value="Edit" />
        <input type="submit"
         name="submit_delete_dept_{$obj->mDepartments[i].department_id}"
```

```
          value="Delete" />
      </td>
    </tr>
    {/if}
  {/section}
  </table>
{/if}
  <h3>Add new department:</h3>
  <p>
    <input type="text" name="department_name" value="[name]" size="30" />
    <input type="text" name="department_description" value="[description]"
     size="60" />
    <input type="submit" name="submit_add_dept_0" value="Add" />
  </p>
</form>
```

2. Создайте в папке `presentation` файл `admin_departments.php` и вставьте в него такой код.

```
<?php
// Класс, обеспечивающий функциональность администрирования отделов
class AdminDepartments
{
  // Public-переменные, доступные в шаблоне Smarty
  public $mDepartmentsCount;
  public $mDepartments;
  public $mErrorMessage;
  public $mEditItem;
  public $mLinkToDepartmentsAdmin;

  // Private-переменные
  private $_mAction;
  private $_mActionedDepartmentId;

  // Конструктор класса
  public function __construct()
  {
    // Просматриваем список переданных переменных
    foreach ($_POST as $key => $value)
      // При щелчке на кнопке отправки...
      if (substr($key, 0, 6) == 'submit')
      {
        /* Получаем позицию последнего символа '_' из имени кнопки,
           например, strrpos('submit_edit_dept_1', '_') вернет значение 17 */
        $last_underscore = strrpos($key, '_');
        /* Получаем область действия из имени кнопки
           (например, 'edit_dep' из 'submit_edit_dept_1') */
        $this->_mAction = substr($key, strlen('submit_'),
                                 $last_underscore - strlen('submit_'));
        /* Получаем идентификатор отдела, к которому относится нажатая кнопка
           (цифру в конце имени кнопки),
           например, '1' из 'submit_edit_dept_1' */
        $this->_mActionedDepartmentId = substr($key, $last_underscore + 1);

        break;
      }

    $this->mLinkToDepartmentsAdmin = Link::ToDepartmentsAdmin();
  }

  public function init()
  {
```

```
// При добавлении нового отдела...
if ($this->_mAction == 'add_dept')
{
  $department_name = $_POST['department_name'];
  $department_description = $_POST['department_description'];

  if ($department_name == null)
    $this->mErrorMessage = 'Department name required';

  if ($this->mErrorMessage == null)
  {
    Catalog::AddDepartment($department_name, $department_description);

    header('Location: ' . $this->mLinkToDepartmentsAdmin);
  }
}

// При редактировании существующего отдела...
if ($this->_mAction == 'edit_dept')
  $this->mEditItem = $this->_mActionedDepartmentId;

// При обновлении отдела...
if ($this->_mAction == 'update_dept')
{
  $department_name = $_POST['name'];
  $department_description = $_POST['description'];

  if ($department_name == null)
    $this->mErrorMessage = 'Department name required';

  if ($this->mErrorMessage == null)
  {
    Catalog::UpdateDepartment($this->_mActionedDepartmentId,
                              $department_name, $department_description);

    header('Location: ' . $this->mLinkToDepartmentsAdmin);
  }
}

// При удалении отдела...
if ($this->_mAction == 'delete_dept')
{
  $status = Catalog::DeleteDepartment($this->_mActionedDepartmentId);

  if ($status < 0)
    $this->mErrorMessage = 'Department not empty';
  else
    header('Location: ' . $this->mLinkToDepartmentsAdmin);
}

// При редактировании категорий отдела...
if ($this->_mAction == 'edit_cat')
{
  header('Location: ' .
         htmlspecialchars_decode(
           Link::ToDepartmentCategoriesAdmin(
             $this->_mActionedDepartmentId)));

  exit();
}
```

```
    // Загружаем список отделов
    $this->mDepartments = Catalog::GetDepartmentsWithDescriptions();
    $this->mDepartmentsCount = count($this->mDepartments);
  }
}
?>
```

3. Откройте файл `presentation/link.php` и добавьте в конец класса `Link` следующий метод.

```
// Создание ссылки на страницу администрирования отделов
public static function ToDepartmentsAdmin()
{
  return self::ToAdmin('Page=Departments');
}
```

4. Модифицируйте метод `init()` класса `StoreAdmin` в файле `presentation/store_admin.php`, чтобы обеспечить загрузку только что созданного компонентного шаблона `admin_departments`.

```
      // При выходе...
      if (isset ($_GET['Page']) && ($_GET['Page'] == 'Logout'))
      {
        unset($_SESSION['admin_logged']);
        header('Location: ' . Link::ToAdmin());

        exit();
      }

      // Если значение Page не задано явно, подразумеваем страницу Departments
      $admin_page = isset ($_GET['Page']) ? $_GET['Page'] : 'Departments';

      // Выбираем, какую страницу администрирования загружать...
      if ($admin_page == 'Departments')
        $this->mContentsCell = 'admin_departments.tpl';
    }
  }
}
?>
```

Как это работает: компонентный шаблон `admin_departments`

В этом упражнении мы написали длинный-предлинный фрагмент кода, но все равно еще не можем ничего протестировать! Это, увы, характерный недостаток подхода, при котором сначала создается интерфейс. Давайте просто посмотрим, что должен делать шаблон `admin_departments.tpl`.

Вот схема конструкции `{section}`, используемой для генерации страниц таблицы `department`.

```
{section name=i loop=$obj->mDepartments}
  {if $obj->mEditItem == $obj->mDepartments[i].department_id}
    <!--
      Здесь находится форма, в которой администратор может редактировать
      название отдела и его описание, с кнопками Update/Cancel,
      Edit Categories и  Delete.
    //-->
  {else}
    <!--
      Здесь находится форма, отображающая названиe и описание отдела,
      а также кнопки Edit, Edit Categories и Delete.
    //-->
  {/if}
{/section}
```

По умолчанию название и описание отдела недоступны для редактирования, но после щелчка на кнопке Edit в строке отдела элементу `$obj->mEditItem` присваивается номер `department_id` этого отдела, и логика представления Smarty генерирует позволяющие редактирование текстовые поля вместо надписей. Это позволяет администратору редактировать название и описание отдела (в режиме редактирования вместо кнопки Edit отображаются кнопки Update и Cancel, как показано на приведенных выше рисунках).

Объект представления из шаблона `admin_departments` — `AdminDepartments` — содержит всю управляющую логику, которая заставляет работать страницу администрирования отделов. При загрузке он проверяет, на какой кнопке был выполнен щелчок для загрузки, и выполняет соответствующие действия. Имена кнопок сформированы так, чтобы их можно было распознавать в коде приложения.

Все имена кнопок начинаются с `submit` и заканчиваются номером-идентификатором отдела. Между `submit` и номером находится код типа кнопки, указывающий, какую операцию она должна выполнять. Вот возможные коды типов.

- `add_dept` — для кнопок Add
- `edit_dept` — для кнопок Edit
- `update_dept` — для кнопок Update
- `delete_dept` — для кнопок Delete
- `edit_cat` — для кнопок Edit Categories

Объект распознает, на какой кнопке был выполнен щелчок, и знает, что нужно делать после просмотра списка переданных переменных и анализа имени кнопки. Например, кнопка с именем `submit_edit_dept_1` приказывает объекту представления перейти в режим редактирования для отдела со значением `department_id`, равным `1`.

Заметьте, что после щелчка на кнопке Add не важно, какой номер отдела указан: номер нового отдела автоматически генерирует база данных (поскольку столбец `department_id` снабжен атрибутом `AUTO_INCREMENT`).

В зависимости от того, на какой кнопке был выполнен щелчок, вызывается один из методов уровня логики приложения. Сейчас мы ими займемся.

Реализация уровня логики приложения

Класс `AdminDepartment` вызывает четыре метода уровня логики приложения. Сейчас мы создадим эти методы.

- `GetDepartmentsWithDescriptions()` возвращает список отделов, который будет отображен на странице администрирования.
- `AddDepartment()` добавляет новый отдел с заданными названием и описанием. Идентификатор нового отдела генерируется автоматически базой данных (столбец `department_id` снабжен атрибутом `AUTO_INCREMENT`).
- `UpdateDepartment()` обновляет информацию об отделе. Его параметры — идентификатор требуемого отдела, его новое название и описание.
- `DeleteDepartment()` удаляет из базы данных отдел с заданным идентификатором.

Упражнение: реализация уровня логики приложения

Пора реализовать эти четыре метода. Добавьте следующий код в класс `Catalog` в файле `business/catalog.php`.

```
// Извлекает из базы данных названия и описания всех отделов
public static function GetDepartmentsWithDescriptions()
{
```

```
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_departments()';

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetAll($sql);
}

// Добавляет отдел
public static function AddDepartment($departmentName, $departmentDescription)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_add_department(:department_name,
                                      :department_description)';

  // Создаем массив параметров
  $params = array (':department_name' => $departmentName,
                   ':department_description' => $departmentDescription);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

// Обновляет сведения об отделе
public static function UpdateDepartment($departmentId, $departmentName,/
                                         $departmentDescription)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_update_department(:department_id, :department_name,
                                         :department_description)';

  // Создаем массив параметров
  $params = array (':department_id' => $departmentId,
                   ':department_name' => $departmentName,
                   ':department_description' => $departmentDescription);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

// Удаляет отдел
public static function DeleteDepartment($departmentId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_delete_department(:department_id)';

  // Создаем массив параметров
  $params = array (':department_id' => $departmentId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetOne($sql, $params);
}
```

Реализация уровня данных

В уровень данных нужно добавить четыре хранимые процедуры, соответствующие методам уровня логики приложения, созданным нами в предыдущем упражнении.

Упражнение: добавление хранимых процедур уровня данных в базу данных

1. С помощью phpMyAdmin создайте четыре хранимые процедуры так, как описано в последующих шагах данного упражнения. Перед тем как выполнять код из этих шагов, не забудьте задать `$$` в качестве символа-разделителя.
2. Выполните приведенный ниже фрагмент кода, который создаст хранимую процедуру `catalog_get_departments` в базе данных `tshirtshop`. Это самая простая из хранимых процедур, которые мы создадим в этом упражнении. Она возвращает список, содержащий идентификаторы отделов, их названия и описания. Она похожа на хранимую процедуру `catalog_get_departments_list`, которая заполняет список отделов в основной части сайта, но в отличие от нее возвращает еще и описания отделов.

```
-- Создаем хранимую процедуру catalog_get_departments
CREATE PROCEDURE catalog_get_departments()
BEGIN
  SELECT   department_id, name, description
  FROM     department
  ORDER BY department_id;
END$$
```

3. Выполните следующий фрагмент кода, который создаст хранимую процедуру `catalog_add_department`. Эта хранимая процедура будет добавлять в базу данных новый отдел.

```
-- Создаем хранимую процедуру catalog_add_department
CREATE PROCEDURE catalog_add_department(
  IN inName VARCHAR(100), IN inDescription VARCHAR(1000))
BEGIN
  INSERT INTO department (name, description)
         VALUES (inName, inDescription);
END$$
```

4. Выполните следующий фрагмент кода, который создаст хранимую процедуру `catalog_update_department`. Эта хранимая процедура будет обновлять информацию о существующем отделе в базе данных.

```
-- Создаем хранимую процедуру catalog_update_department
CREATE PROCEDURE catalog_update_department(IN inDepartmentId INT,
  IN inName VARCHAR(100), IN inDescription VARCHAR(1000))
BEGIN
  UPDATE department
  SET    name = inName, description = inDescription
  WHERE  department_id = inDepartmentId;
END$$
```

5. Выполните следующий фрагмент кода, который создаст хранимую процедуру `catalog_delete_department`. Эта хранимая процедура будет удалять из базы данных существующий отдел, но только если к этому отделу не принадлежат никакие категории.

```
-- Создаем хранимую процедуру catalog_delete_department
CREATE PROCEDURE catalog_delete_department(IN inDepartmentId INT)
BEGIN
  DECLARE categoryRowsCount INT;

  SELECT count(*)
  FROM   category
  WHERE  department_id = inDepartmentId
  INTO   categoryRowsCount;
```

```
  IF categoryRowsCount = 0 THEN
    DELETE FROM department WHERE department_id = inDepartmentId;

    SELECT 1;
  ELSE
    SELECT -1;
  END IF;
END$$
```

6. Откройте `admin.php` в браузере и полюбуйтесь результатами своего труда. Теперь должны работать механизмы добавления, удаления и редактирования отделов, как показано на рис. 10.2, 10.3 и 10.9.

Администрирование категорий

Поскольку администрирование категорий основано на тех же концепциях и механизмах, что и администрирование отделов, мы просто перечислим шаги, которые необходимо выполнить для создания этой части сайта.

Выполнив следующее упражнение, последнее в этой главе, вы сможете вносить изменения в сведения о категориях в базе данных. Для этого достаточно будет на странице администрирования отделов щелкнуть на кнопке Edit Categories, а на открывшейся странице категорий — на кнопке Edit (рис. 10.10).

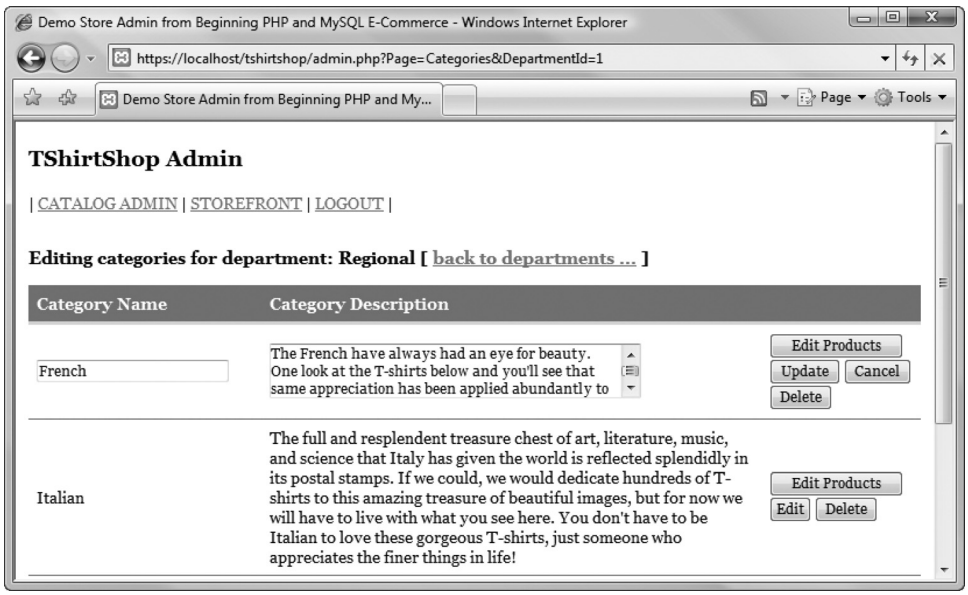


Рис. 10.10. Редактирование категории

Упражнение: администрирование категорий

1. Создайте в папке `presentation/templates` файл шаблона `admin_categories.tpl` и добавьте в него следующий код.

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

```
<form method="post"
 action="{$obj->mLinkToDepartmentCategoriesAdmin}">
  <h3>
    Editing categories for department: {$obj->mDepartmentName} [
    <a href="{$obj->mLinkToDepartmentsAdmin}">back to departments ...</a>
]
  </h3>
{if $obj->mErrorMessage}<p class="error">{$obj->mErrorMessage}</p>{/if}
{if $obj->mCategoriesCount eq 0}
  <p class="no-items-found">There are no categories in this
department!</p>
{else}
  <table class="tss-table">
    <tr>
      <th width="200">Category Name</th>
      <th>Category Description</th>
      <th width="240"> </th>
    </tr>
  {section name=i loop=$obj->mCategories}
    {if $obj->mEditItem == $obj->mCategories[i].category_id}
    <tr>
      <td>
        <input type="text" name="name"
         value="{$obj->mCategories[i].name}" size="30" />
      </td>
      <td>
        {strip}
        <textarea name="description" rows="3" cols="60">
          {$obj->mCategories[i].description}
        </textarea>
        {/strip}
      </td>
      <td>
        <input type="submit"
         name="submit_edit_prod_{$obj->mCategories[i].category_id}"
         value="Edit Products" />
        <input type="submit"
         name="submit_update_cat_{$obj->mCategories[i].category_id}"
         value="Update" />
        <input type="submit" name="cancel" value="Cancel" />
        <input type="submit"
         name="submit_delete_cat_{$obj->mCategories[i].category_id}"
         value="Delete" />
      </td>
    </tr>
    {else}
    <tr>
      <td>{$obj->mCategories[i].name}</td>
      <td>{$obj->mCategories[i].description}</td>
      <td>
        <input type="submit"
         name="submit_edit_prod_{$obj->mCategories[i].category_id}"
         value="Edit Products" />
        <input type="submit"
         name="submit_edit_cat_{$obj->mCategories[i].category_id}"
         value="Edit" />
        <input type="submit"
         name="submit_delete_cat_{$obj->mCategories[i].category_id}"
         value="Delete" />
      </td>
    </tr>
```

```
    {/if}
  {/section}
  </table>
{/if}
  <h3>Add new category:</h3>
  <input type="text" name="category_name" value="[name]" size="30" />
  <input type="text" name="category_description" value="[description]"
    size="60" >
  <input type="submit" name="submit_add_cat_0" value="Add" />
</form>
```

2. Создайте в папке `presentation` файл `admin_categories.php` и вставьте в него следующий код.

```
<?php
// Класс, обеспечивающий администрирование категорий
class AdminCategories
{
  // Public-переменные, доступные в шаблоне Smarty
  public $mCategoriesCount;
  public $mCategories;
  public $mErrorMessage;
  public $mEditItem;
  public $mDepartmentId;
  public $mDepartmentName;
  public $mLinkToDepartmentsAdmin;
  public $mLinkToDepartmentCategoriesAdmin;

  // Private-переменные
  private $_mAction;
  private $_mActionedCategoryId;

  // Конструктор класса
  public function __construct()
  {
    if (isset ($_GET['DepartmentId']))
      $this->mDepartmentId = (int)$_GET['DepartmentId'];
    else
      trigger_error('DepartmentId not set');

    $department_details = Catalog::GetDepartmentDetails(
    $this->mDepartmentId);
    $this->mDepartmentName = $department_details['name'];

    foreach ($_POST as $key => $value)
      // При щелчке на кнопке...
      if (substr($key, 0, 6) == 'submit')
      {
        /* Получаем позицию последнего символа '_' из имени нажатой
           кнопки, например e.g strrpos('submit_edit_cat_1', '_') is 16 */
        $last_underscore = strrpos($key, '_');
        /* Узнаем тип нажатой кнопки
           (например, 'edit_cat' из 'submit_edit_cat_1') */
        $this->_mAction = substr($key, strlen('submit_'),
                                 $last_underscore - strlen('submit_'));
        /* Получаем идентификатор категории, для которой нажата кнопка
           (номер в конце имени нажатой кнопки),
           например, '1' из 'submit_edit_cat_1' */
        $this->_mActionedCategoryId = (int)substr($key, $last_underscore + 1);

        break;
```

```
    }
  $this->mLinkToDepartmentsAdmin = Link::ToDepartmentsAdmin();

  $this->mLinkToDepartmentCategoriesAdmin =
    Link::ToDepartmentCategoriesAdmin($this->mDepartmentId);
}

public function init()
{
  // При добавлении новой категории...
  if ($this->_mAction == 'add_cat')
  {
    $category_name = $_POST['category_name'];
    $category_description = $_POST['category_description'];

    if ($category_name == null)
      $this->mErrorMessage = 'Category name is empty';

    if ($this->mErrorMessage == null)
    {
      Catalog::AddCategory($this->mDepartmentId, $category_name,
                           $category_description);

      header('Location: ' .
             htmlspecialchars_decode(
               $this->mLinkToDepartmentCategoriesAdmin));
    }
  }

  // При редактировании существующей категории...
  if ($this->_mAction == 'edit_cat')
  {
    $this->mEditItem = $this->_mActionedCategoryId;
  }

  // При обновлении категории...
  if ($this->_mAction == 'update_cat')
  {
    $category_name = $_POST['name'];
    $category_description = $_POST['description'];

    if ($category_name == null)
      $this->mErrorMessage = 'Category name is empty';

    if ($this->mErrorMessage == null)
    {
      Catalog::UpdateCategory($this->_mActionedCategoryId, $category_name,
                              $category_description);
      header('Location: ' .
             htmlspecialchars_decode(
               $this->mLinkToDepartmentCategoriesAdmin));
    }
  }

  // При удалении категории...
  if ($this->_mAction == 'delete_cat')
  {
    $status = Catalog::DeleteCategory($this->_mActionedCategoryId);

    if ($status < 0)
      $this->mErrorMessage = 'Category not empty';
```

```
      else
        header('Location: ' .
               htmlspecialchars_decode(
                 $this->mLinkToDepartmentCategoriesAdmin));
    }

    // При редактировании товаров из категории ...
    if ($this->_mAction == 'edit_prod')
    {
      header('Location: ' .
             htmlspecialchars_decode(
               Link::ToCategoryProductsAdmin($this->mDepartmentId,
                                             $this->_mActionedCategoryId)));
      exit();
    }

    // Загружаем список категорий
    $this->mCategories =
      Catalog::GetDepartmentCategories($this->mDepartmentId);
    $this->mCategoriesCount = count($this->mCategories);
  }
}
?>
```

3. Откройте файл `presentation/link.php` и добавьте в конец класса `Link` следующий метод.

```
// Создает ссылку на страницу администрирования категорий
public static function ToDepartmentCategoriesAdmin($departmentId)
{
  $link = 'Page=Categories&DepartmentId=' . $departmentId;

  return self::ToAdmin($link);
}
```

4. Откройте файл `business/catalog.php` и добавьте в класс `Catalog` новые методы:

```
// Возвращает список категорий отдела
public static function GetDepartmentCategories($departmentId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_department_categories(:department_id)';

  // Создаем массив параметров
  $params = array (':department_id' => $departmentId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetAll($sql, $params);
}

// Добавляет новую категорию
public static function AddCategory($departmentId, $categoryName,
                                   $categoryDescription)
{
  // Составляем SQL-запрос
$sql = 'CALL catalog_add_category(:department_id, :category_name,
                                  :category_description)';
  // Создаем массив параметров
  $params = array (':department_id' => $departmentId,
                   ':category_name' => $categoryName,
                   ':category_description' => $categoryDescription);
```

```
  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

// Обновляет категорию
public static function UpdateCategory($categoryId, $categoryName,
                                       $categoryDescription)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_update_category(:category_id, :category_name,
                                       :category_description)';

  // Создаем массив параметров
  $params = array (':category_id' => $categoryId,
                   ':category_name' => $categoryName,
                   ':category_description' => $categoryDescription);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

// Удаляет категорию
public static function DeleteCategory($categoryId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_delete_category(:category_id)';

  // Создаем массив параметров
  $params = array (':category_id' => $categoryId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetOne($sql, $params);
}
```

5. Модифицируйте метод `init()` класса `StoreAdmin` в файле `presentation/store_admin.php` для загрузки нового компонентного шаблона.

```
// Выбираем, какую страницу администрирования загружать...
if ($admin_page == 'Departments')
  $this->mContentsCell = 'admin_departments.tpl';
elseif ($admin_page == 'Categories')
  $this->mContentsCell = 'admin_categories.tpl';
```

6. С помощью phpMyAdmin выполните приведенный ниже код, который создаст в базе данных tshirtshop четыре хранимые процедуры. Не забудьте задать символы `$$` в качестве разделителя.

```
-- Создание хранимой процедуры catalog_get_department_categories
CREATE PROCEDURE catalog_get_department_categories(IN inDepartmentId INT)
BEGIN
  SELECT   category_id, name, description
  FROM     category
  WHERE    department_id = inDepartmentId
  ORDER BY category_id;
END$$

-- Создание хранимой процедуры catalog_add_category
CREATE PROCEDURE catalog_add_category(IN inDepartmentId INT,
  IN inName VARCHAR(100), IN inDescription VARCHAR(1000))
BEGIN
INSERT INTO category (department_id, name, description)
       VALUES (inDepartmentId, inName, inDescription);
END$$
```

```
-- Создание хранимой процедуры catalog_update_category
CREATE PROCEDURE catalog_update_category(IN inCategoryId INT,
  IN inName VARCHAR(100), IN inDescription VARCHAR(1000))
BEGIN
    UPDATE category
    SET    name = inName, description = inDescription
    WHERE  category_id = inCategoryId;
END$$

-- Создание хранимой процедуры catalog_delete_category
CREATE PROCEDURE catalog_delete_category(IN inCategoryId INT)
BEGIN
  DECLARE productCategoryRowsCount INT;

  SELECT     count(*)
  FROM       product p
  INNER JOIN product_category pc
  ON         p.product_id = pc.product_id
  WHERE      pc.category_id = inCategoryId
  INTO       productCategoryRowsCount;

  IF productCategoryRowsCount = 0 THEN
    DELETE FROM category WHERE category_id = inCategoryId;

    SELECT 1;
  ELSE
    SELECT -1;
  END IF;
END$$
```

7. Откройте файл `admin.php` в браузере, выберите отдел и попробуйте щелкнуть на кнопке Edit Categories для этого отдела. Должен загрузиться компонентный шаблон `admin_categories`, и страница должна выглядеть примерно так, как на рис. 10.4. При редактировании страница должна принять вид, показанный на рис. 10.10.

Как это работает: администрирование категорий

Код для администрирования категорий очень похож на код для работы с отделами. Если кратко, мы решили следующие задачи.

- Создали компонентный шаблон `admin_categories`, состоящий из двух частей: собственно шаблона (файл `admin_categories.tpl`) и связанного с ним объекта представления (файл `admin_categories.php`). Эти два файла образуют уровень представления.
- Обновили шаблон `store_admin` (точнее, его объект представления), организовав загрузку шаблона `admin_categories` при выборе категории. Как уже говорилось в начале главы, `store_admin` — это компонентный шаблон, загружаемый в `admin.php` и определяющий, какой шаблон администрирования нужно загружать.
- Создали методы уровня логики приложения, поддерживающие всю функциональность администрирования каталога: добавление, удаление и редактирование категорий.
- Создали в базе данных хранимые процедуры, вызываемые методами уровня логики приложения при работе с категориями.

Потратьте немного времени на изучение только что добавленного кода и убедитесь, что понимаете, как он работает, прежде чем переходить к следующей главе (в ней мы займемся механизмами администрирования товаров и атрибутов).

Резюме

Код, добавленный нами в приложение в этой главе, образует сразу несколько компонентных шаблонов и поддерживающие их работу методы и хранимые процедуры. Вы узнали, как реализовать простейший механизм аутентификации, разрешающий только администраторам использовать инструменты для изменения информации в базе данных. Кроме того, теперь наш сайт снабжен механизмами администрирования отделов и категорий. В главе 11 мы закончим создание инструментов администрирования, добавив в проект инструменты для работы с товарами и атрибутами.

Глава 11

Администрирование каталога: товары и атрибуты

Теперь у администраторов сайта есть возможность редактировать информацию об отделах и категориях. В этой главе мы создадим механизмы для выполнения таких же операций с товарами и атрибутами. Точнее говоря, механизмы, которые мы создадим, позволят

- управлять атрибутами товаров, явно задавать списки значений атрибутов, добавлять новые атрибуты и удалять их, а также присваивать атрибуты товарам;
- просматривать списки товаров в выбранных категориях;
- редактировать сведения о товарах, в том числе их названия, описания, цены (также можно будет указывать, является ли товар акционным);
- добавлять в категории новые товары (товар может принадлежать одновременно к нескольким категориям) или перемещать товары между категориями;
- удалять товары из категорий;
- удалять товары из каталога;
- переходить на страницы администрирования отделов, категорий или товаров прямо из каталога.

Нам придется написать солидный объем кода, но этот код не будет содержать почти ничего принципиально нового. Если хотите, можете заранее просмотреть рисунки этой главы, чтобы увидеть, что у нас получится в результате.

Ну что ж, приступим.

Администрирование атрибутов товаров

Первое, что мы создадим в этой главе, — страница управления атрибутами товаров. Она будет доступна по новой ссылке PRODUCT ATTRIBUTES ADMIN в меню администрирования. Щелчок на этой ссылке приведет к загрузке в браузере страницы, представленной на рис. 11.1.

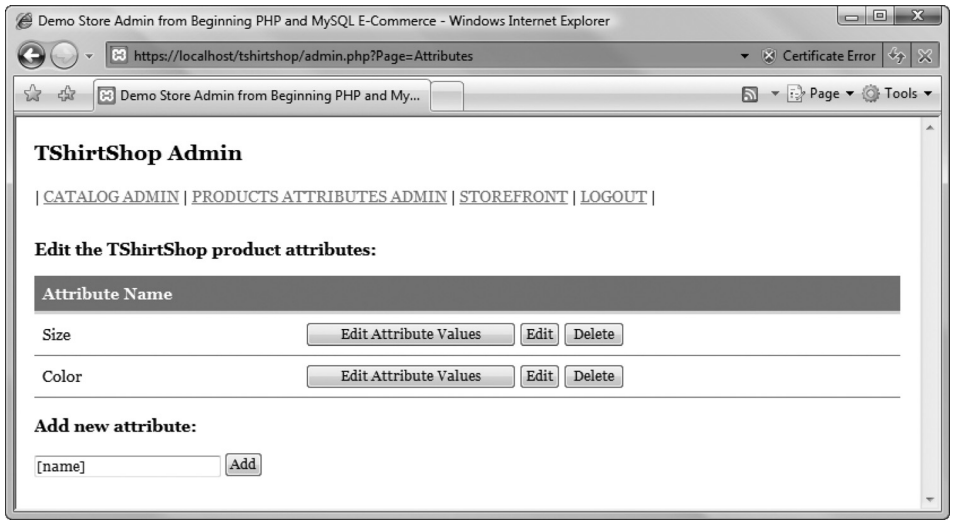


Рис. 11.1. Страница редактирования атрибутов товаров

Как вы знаете из главы 6, в которой мы добавили поддержку атрибутов товаров, у каждого атрибута есть список допустимых значений. Например, у атрибута `Color` допустимыми могут быть значения `Green`, `Pink`, `Black` и т.д. Если товар поддерживает атрибут, мы должны перечислить значения этого атрибута, одно из которых покупатель сможет выбрать, покупая товар. Щелчок на кнопке Edit Attribute Values в строке атрибута открывает страницу, на которой можно задавать допустимые значения атрибута. Эта страница показана на рис. 11.2.

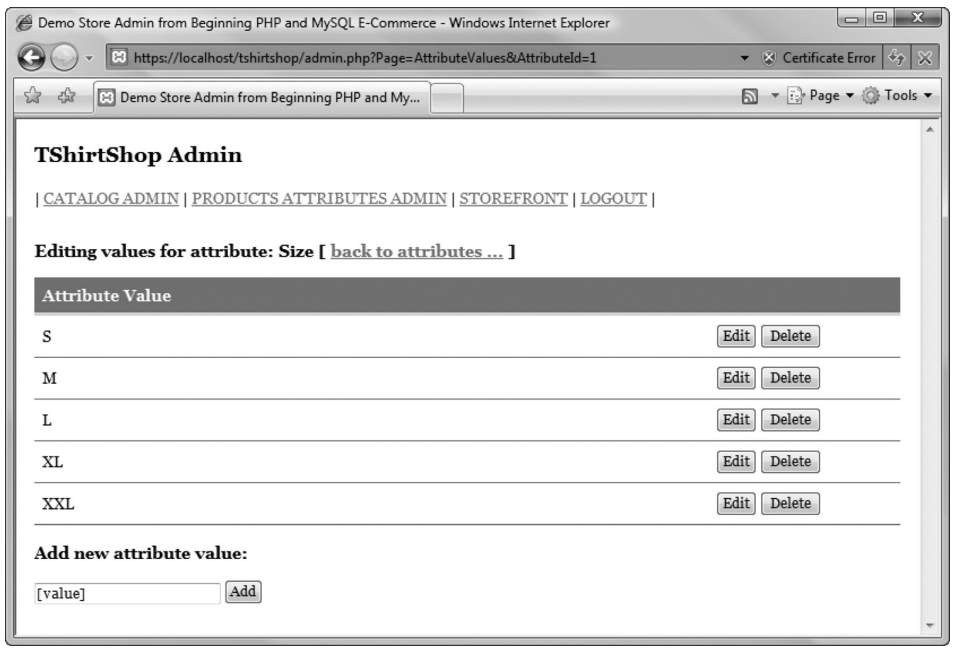


Рис. 11.2. Страница редактирования атрибутов товаров

Давайте реализуем эти две страницы в упражнении, а затем рассмотрим детали реализации.

Упражнение: реализация шаблона admin_products

1. Создайте в папке presentation/templates файл admin_attributes.tpl и добавьте в него следующий код.

```
{* admin_attributes.tpl *}
{load_presentation_object filename="admin_attributes" assign="obj"}
<form method="post"
 action="{$obj->mLinkToAttributesAdmin}">
  <h3>Edit the TShirtShop product attributes:</h3>
{if $obj->mErrorMessage}<p class="error">{$obj->mErrorMessage}</p>{/if}
{if $obj->mAttributesCount eq 0}
  <p class="no-items-found">
    There are no products attributes in your database!
  </p>
{else}
  <table class="tss-table">
    <tr>
      <th>Attribute Name</th>
      <th width="240"> </th>
    </tr>
  {section name=i loop=$obj->mAttributes}
    {if $obj->mEditItem == $obj->mAttributes[i].attribute_id}
    <tr>
    <td>
      <input type="text" name="name"
       value="{$obj->mAttributes[i].name}" size="30" />
    </td>
    <td>
      <input type="submit"
       name="submit_edit_attr_val_{$obj->mAttributes[i].attribute_id}"
       value="Edit Attribute Values" />
      <input type="submit"
       name="submit_update_attr_{$obj->mAttributes[i].attribute_id}"
       value="Update" />
      <input type="submit" name="cancel" value="Cancel" />
      <input type="submit"
       name="submit_delete_attr_{$obj->mAttributes[i].attribute_id}"
       value="Delete" />
      </td>
    </tr>
    {else}
    <tr>
      <td>{$obj->mAttributes[i].name}</td>
      <td>
        <input type="submit"
         name="submit_edit_val_{$obj->mAttributes[i].attribute_id}"
         value="Edit Attribute Values" />
        <input type="submit"
         name="submit_edit_attr_{$obj->mAttributes[i].attribute_id}"
         value="Edit" />
        <input type="submit"
         name="submit_delete_attr_{$obj->mAttributes[i].attribute_id}"
         value="Delete" />
      </td>
    </tr>
    {/if}
```

```
  {/section}
  </table>
{/if}
  <h3>Add new attribute:</h3>
  <p>
    <input type="text" name="attribute_name" value="[name]" size="30" />
    <input type="submit" name="submit_add_attr_0" value="Add" />
  </p>
</form>
```

2. Создайте в папке `presentation` файл объекта представления с именем `admin_attributes.php` и добавьте в него следующий код.

```
<?php
// Класс, обеспечивающий администрирование атрибутов
class AdminAttributes
{
  // Public-переменные, доступные в шаблоне Smarty
  public $mAttributesCount;
  public $mAttributes;
  public $mErrorMessage;
  public $mEditItem;
  public $mLinkToAttributesAdmin;

  // Private-переменные
  private $_mAction;
  private $_mActionedAttributeId;

  // Конструктор класса
  public function __construct()
  {
    // Просматриваем список переданных переменных
    foreach ($_POST as $key => $value)
      // Если нажата кнопка отправки...
      if (substr($key, 0, 6) == 'submit')
      {
        /* Получаем позицию последнего символа '_' из имени кнопки,
           например, strrpos('submit_edit_attr_1', '_') вернет значение 17 */
        $last_underscore = strrpos($key, '_');
        /* Получаем область действия кнопки
           (например, 'edit_attr' из 'submit_edit_attr_1') */
        $this->_mAction = substr($key, strlen('submit_'),
                                 $last_underscore - strlen('submit_'));

        /* Получаем идентификатор атрибута, к которому относится кнопка
           (номер в конце имени этой кнопки), например,
           '1' из 'submit_edit_attr_1' */
        $this->_mActionedAttributeId = substr($key, $last_underscore + 1);

        break;
      }

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

  public function init()
  {
    // При добавлении нового атрибута...
    if ($this->_mAction == 'add_attr')
    {
      $attribute_name = $_POST['attribute_name'];

      if ($attribute_name == null)
```

```
        $this->mErrorMessage = 'Attribute name required';

      if ($this->mErrorMessage == null)
      {
        Catalog::AddAttribute($attribute_name);
        header('Location: ' . $this->mLinkToAttributesAdmin);
      }
    }

    // При редактировании существующего атрибута...
    if ($this->_mAction == 'edit_attr')
      $this->mEditItem = $this->_mActionedAttributeId;

    // При обновлении атрибута...
    if ($this->_mAction == 'update_attr')
    {
      $attribute_name = $_POST['name'];
      if ($attribute_name == null)
        $this->mErrorMessage = 'Attribute name required';

      if ($this->mErrorMessage == null)
      {
        Catalog::UpdateAttribute($this->_mActionedAttributeId,
          $attribute_name);

        header('Location: ' . $this->mLinkToAttributesAdmin);
      }
    }

    // При удалении атрибута...
    if ($this->_mAction == 'delete_attr')
    {
      $status = Catalog::DeleteAttribute($this->_mActionedAttributeId);

      if ($status < 0)
        $this->mErrorMessage =
          'Attribute has one or more values and cannot be deleted';
      else
        header('Location: ' . $this->mLinkToAttributesAdmin);
    }

    // При редактировании значений атрибута...
    if ($this->_mAction == 'edit_val')
    {
      header('Location: ' .
             htmlspecialchars_decode(
               Link::ToAttributeValuesAdmin(
                 $this->_mActionedAttributeId)));

      exit();
    }

    // Загружаем список атрибутов
    $this->mAttributes = Catalog::GetAttributes();
    $this->mAttributesCount = count($this->mAttributes);
  }
}
?>
```

3. Создайте в папке `presentation/templates` файл шаблона с именем `admin_attribute_values.tpl` и поместите в него следующий код.

```
{* admin_attribute_values.tpl *}
{load_presentation_object filename="admin_attribute_values" assign="obj"}
<form method="post"
 action="{$obj->mLinkToAttributeValuesAdmin}">
  <h3>
    Editing values for attribute: {$obj->mAttributeName} [
    <a href="{$obj->mLinkToAttributesAdmin}">back to attributes ...</a> ]
  </h3>
{if $obj->mErrorMessage}<p class="error">{$obj->mErrorMessage}</p>{/if}
{if $obj->mAttributeValuesCount eq 0}
  <p class="no-items-found">There are no values for this attribute!</p>
{else}
  <table class="tss-table">
    <tr>
      <th>Attribute Value</th>
      <th width="170"> </th>
    </tr>
  {section name=i loop=$obj->mAttributeValues}
    {if $obj->mEditItem == $obj->mAttributeValues[i].attribute_value_id}
    <tr>
      <td>
        <input type="text" name="value"
         value="{$obj->mAttributeValues[i].value}" size="30" />
        </td>
        <td>
          <input type="submit"
           name="submit_update_val_{$obj->mAttributeValues[i].
           attribute_value_id}" value="Update" />
          <input type="submit" name="cancel" value="Cancel" />
          <input type="submit"
           name="submit_delete_val_{$obj->mAttributeValues[i].
           attribute_value_id}" value="Delete" />
        </td>
      </tr>
      {else}
      <tr>
        <td>{$obj->mAttributeValues[i].value}</td>
        <td>
          <input type="submit"
           name="submit_edit_val_{$obj->mAttributeValues[i].
           attribute_value_id}" value="Edit" />
          <input type="submit"
           name="submit_delete_val_{$obj->mAttributeValues[i].
           attribute_value_id}" value="Delete" />
        </td>
      </tr>
      {/if}
    {/section}
  </table>
{/if}
  <h3>Add new attribute value:</h3>
  <input type="text" name="attribute_value" value="[value]" size="30" />
  <input type="submit" name="submit_add_val_0" value="Add" />
</form>
```

4. Создайте в папке `presentation` файл объекта представления с именем `admin_attribute_values.php` и добавьте в него следующий код.

```
<?php
// Класс, обеспечивающий администрирование значений атрибутов
class AdminAttributeValues
{
```

```
  // Public-переменные, доступные в шаблоне Smarty
  public $mAttributeValuesCount;
  public $mAttributeValues;
  public $mErrorMessage;
  public $mEditItem;
  public $mAttributeId;
  public $mAttributeName;
  public $mLinkToAttributeAdmin;
  public $mLinkToAttributeValuesAdmin;

  // Private-переменные
  private $_mAction;
  private $_mActionedAttributeValueId;

  // Конструктор класса
  public function __construct()
  {
    if (isset ($_GET['AttributeId']))
      $this->mAttributeId = (int)$_GET['AttributeId'];
    else
      trigger_error('AttributeId not set');

    $attribute_details = Catalog::GetAttributeDetails
    ($this->mAttributeId);
    $this->mAttributeName = $attribute_details['name'];
    foreach ($_POST as $key => $value)
      // Если нажата кнопка отправки...
      if (substr($key, 0, 6) == 'submit')
      {
        /* Получаем позицию последнего символа '_' из имени кнопки,
           например, strrpos('submit_edit_val_1', '_') вернет значение 16
*/
        $last_underscore = strrpos($key, '_');

        /* Получаем область действия кнопки
           (например, 'edit_cat' from 'submit_edit_val_1') */
        $this->_mAction = substr($key, strlen('submit_'),
          $last_underscore - strlen('submit_'));

        /* Получаем идентификатор значения атрибута, к которому
           относилась нажатая кнопка (номер в конце имени кнопки),
           например, '1' из 'submit_edit_val_1' */
        $this->_mActionedAttributeValueId =
          (int)substr($key, $last_underscore + 1);

        break;
      }

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

    $this->mLinkToAttributeValuesAdmin =
      Link::ToAttributeValuesAdmin($this->mAttributeId);
  }

  public function init()
  {
    // При добавлении нового значения атрибута...
    if ($this->_mAction == 'add_val')
    {
      $attribute_value = $_POST['attribute_value'];
```

```
      if ($attribute_value == null)
        $this->mErrorMessage = 'Attribute value is empty';

      if ($this->mErrorMessage == null)
      {
        Catalog::AddAttributeValue($this->mAttributeId, $attribute_value);

        header('Location: ' .
               htmlspecialchars_decode(
                 $this->mLinkToAttributeValuesAdmin));
      }
    }

    // При редактировании существующего значения атрибута...
    if ($this->_mAction == 'edit_val')
    {
      $this->mEditItem = $this->_mActionedAttributeValueId;
    }

    // При обновлении значения атрибута...
    if ($this->_mAction == 'update_val')
    {
      $attribute_value = $_POST['value'];

      if ($attribute_value == null)
        $this->mErrorMessage = 'Attribute value is empty';
      if ($this->mErrorMessage == null)
      {
        Catalog::UpdateAttributeValue(
          $this->_mActionedAttributeValueId, $attribute_value);

        header('Location: ' .
               htmlspecialchars_decode(
                 $this->mLinkToAttributeValuesAdmin));
      }
    }

    // При удалении значения атрибута...
    if ($this->_mAction == 'delete_val')
    {
      $status =
        Catalog::DeleteAttributeValue($this->_mActionedAttributeValueId);

      if ($status < 0)
        $this->mErrorMessage = 'Cannot delete this attribute value. ' .
                               'One or more products are using it!';
      else
        header('Location: ' .
               htmlspecialchars_decode(
                 $this->mLinkToAttributeValuesAdmin));
    }

    // Загружаем список значений атрибута
    $this->mAttributeValues =
      Catalog::GetAttributeValues($this->mAttributeId);
    $this->mAttributeValuesCount = count($this->mAttributeValues);
  }
}
?>
```

5. Откройте файл `presentation/templates/admin_menu.tpl` и добавьте в него новую строку, которая добавляет ссылку на страницу администрирования атрибутов.

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

6. Откройте файл `presentation/admin_menu.php` и добавьте в него код, выделенный полужирным шрифтом в следующем листинге.

```
<?php
class AdminMenu
{
  public $mLinkToStoreAdmin;
  public $mLinkToAttributesAdmin;
  public $mLinkToStoreFront;
  public $mLinkToLogout;

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

7. Откройте файл `presentation/link.php` и добавьте в конец класса `Link` два следующих метода.

```
// Создаем ссылку на страницу администрирования атрибутов
public static function ToAttributesAdmin()
{
  return self::ToAdmin('Page=Attributes');
}

// Создает ссылку на страницу администрирования значений атрибутов
public static function ToAttributeValuesAdmin($attributeId)
{
  $link = 'Page=AttributeValues&AttributeId=' . $attributeId;

  return self::ToAdmin($link);
}
```

8. Откройте файл `business/catalog.php` и добавьте методы из следующего листинга в класс `Catalog`. Эти методы используются для администрирования атрибутов.

```
// Возвращает все атрибуты
public static function GetAttributes()
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_attributes()';

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetAll($sql);
}
// Добавляет атрибут
```

```
public static function AddAttribute($attributeName)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_add_attribute(:attribute_name)';

  // Создаем массив параметров
  $params = array (':attribute_name' => $attributeName);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

// Обновляет имя атрибута
public static function UpdateAttribute($attributeId, $attributeName)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_update_attribute(:attribute_id, :attribute_name)';

  // Создаем массив параметров
  $params = array (':attribute_id' => $attributeId,
                   ':attribute_name' => $attributeName);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

// Удаляет атрибут
public static function DeleteAttribute($attributeId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_delete_attribute(:attribute_id)';

  // Создаем массив параметров
  $params = array (':attribute_id' => $attributeId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetOne($sql, $params);
}

// Возвращает сведения о выбранном атрибуте
public static function GetAttributeDetails($attributeId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_attribute_details(:attribute_id)';

  // Создаем массив параметров
  $params = array (':attribute_id' => $attributeId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetRow($sql, $params);
}

// Возвращает значения атрибута
public static function GetAttributeValues($attributeId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_attribute_values(:attribute_id)';

  // Создаем массив параметров
  $params = array (':attribute_id' => $attributeId);
```

```
  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetAll($sql, $params);
}

// Добавляет значение атрибута
public static function AddAttributeValue($attributeId, $attributeValue)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_add_attribute_value(:attribute_id, :value)';

  // Создаем массив параметров
  $params = array (':attribute_id' => $attributeId,
  ':value' => $attributeValue);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

// Обновляет значение атрибута
public static function UpdateAttributeValue(
                          $attributeValueId, $attributeValue)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_update_attribute_value(
                :attribute_value_id, :value)';
  // Создаем массив параметров
  $params = array (':attribute_value_id' => $attributeValueId,
                   ':value' => $attributeValue);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

// Удаляет значение атрибута
public static function DeleteAttributeValue($attributeValueId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_delete_attribute_value(:attribute_value_id)';

  // Создаем массив параметров
  $params = array (':attribute_value_id' => $attributeValueId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetOne($sql, $params);
}
```

9. Модифицируйте метод `init()` класса `StoreAdmin` в файле `presentation/store_admin.php`, добавив в него загрузку новых компонентных шаблонов.

```
// Выбираем, какую страницу администрирования загружать...
if ($admin_page == 'Departments')
  $this->mContentsCell = 'admin_departments.tpl';
elseif ($admin_page == 'Categories')
  $this->mContentsCell = 'admin_categories.tpl';
elseif ($admin_page == 'Attributes')
  $this->mContentsCell = 'admin_attributes.tpl';
elseif ($admin_page == 'AttributeValues')
  $this->mContentsCell = 'admin_attribute_values.tpl';
```

10. С помощью phpMyAdmin выполните следующие операторы SQL, чтобы добавить в уровень данных новые хранимые процедуры. Не забудьте задать разделитель

$$ (если вы хотите просмотреть список всех хранимых процедур в базе данных tshirtshop, можете использовать команду SHOW PROCEDURE STATUS).

```
-- Создаем хранимую процедуру catalog_get_attributes
CREATE PROCEDURE catalog_get_attributes()
BEGIN
  SELECT attribute_id, name FROM attribute ORDER BY attribute_id;
END$$

-- Создаем хранимую процедуру catalog_add_attribute
CREATE PROCEDURE catalog_add_attribute(IN inName VARCHAR(100))
BEGIN
  INSERT INTO attribute (name) VALUES (inName);
END$$

-- Создаем хранимую процедуру catalog_update_attribute
CREATE PROCEDURE catalog_update_attribute(
  IN inAttributeId INT, IN inName VARCHAR(100))
BEGIN
  UPDATE attribute SET name = inName WHERE attribute_id = inAttributeId;
END$$

-- Создаем хранимую процедуру catalog_delete_attribute
CREATE PROCEDURE catalog_delete_attribute(IN inAttributeId INT)
BEGIN
  DECLARE attributeRowsCount INT;

  SELECT count(*)
  FROM   attribute_value
  WHERE  attribute_id = inAttributeId
  INTO   attributeRowsCount;

  IF attributeRowsCount = 0 THEN
    DELETE FROM attribute WHERE attribute_id = inAttributeId;

    SELECT 1;
  ELSE
    SELECT -1;
  END IF;
END$$

-- Создаем хранимую процедуру catalog_get_attribute_details
CREATE PROCEDURE catalog_get_attribute_details(IN inAttributeId INT)
BEGIN
  SELECT attribute_id, name
  FROM   attribute
  WHERE  attribute_id = inAttributeId;
END$$

-- Создаем хранимую процедуру catalog_get_attribute_values
CREATE PROCEDURE catalog_get_attribute_values(IN inAttributeId INT)
BEGIN
  SELECT   attribute_value_id, value
  FROM     attribute_value
  WHERE    attribute_id = inAttributeId
  ORDER BY attribute_id;
END$$

-- Создаем хранимую процедуру catalog_add_attribute_value
CREATE PROCEDURE catalog_add_attribute_value(
  IN inAttributeId INT, IN inValue VARCHAR(100))
```

```
BEGIN
  INSERT INTO attribute_value (attribute_id, value)
         VALUES (inAttributeId, inValue);
END$$

-- Создаем хранимую процедуру catalog_update_attribute_value
CREATE PROCEDURE catalog_update_attribute_value(
  IN inAttributeValueId INT, IN inValue VARCHAR(100))
BEGIN
  UPDATE attribute_value
  SET    value = inValue
  WHERE  attribute_value_id = inAttributeValueId;
END$$

-- Создаем хранимую процедуру catalog_delete_attribute_value
CREATE PROCEDURE catalog_delete_attribute_value(IN inAttributeValueId INT)
BEGIN
  DECLARE productAttributeRowsCount INT;

  SELECT     count(*)
  FROM       product p
  INNER JOIN product_attribute pa
               ON p.product_id = pa.product_id
  WHERE      pa.attribute_value_id = inAttributeValueId
  INTO       productAttributeRowsCount;

  IF productAttributeRowsCount = 0 THEN
    DELETE FROM attribute_value WHERE attribute_value_id =
inAttributeValueId;

    SELECT 1;
  ELSE
    SELECT -1;
  END IF;
END$$
```

11. Загрузите страницу администрирования TShirtShop и убедитесь, что все инструменты администрирования (показанные на рис. 11.1 и 11.2) работают правильно.

Как это работает: администрирование атрибутов

Признаем, что написание кода администрирования атрибутов — занятие не самое увлекательное. Ничего особо сложного в этом коде нет, но при его наборе можно сделать кучу мелких ошибок, которые впоследствии придется долго искать. Однако написать этот код необходимо — в каждом серьезном сайте электронного магазина должен быть раздел, позволяющий администрировать содержимое этого сайта.

Вот основные задачи, которые мы решили в этом упражнении.

- Создали компонентный шаблон `admin_attributes`, состоящий из файлов `admin_attributes.tpl` и `admin_attributes.php`. Этот шаблон реализует страницу, показанную на рис. 11.1.
- Создали компонентный шаблон `admin_attribute_values`, состоящий из файлов `admin_attribute_values.tpl` и `admin_attribute_values.php`. Этот шаблон реализует страницу, показанную на рис. 11.2.
- Добавили ссылки на новые элементы в шаблон `admin_menu` (содержащий ссылки на все административные страницы сайта) и поместили в класс `Link` новые методы для генерации ссылок.
- Написали код уровня логики приложения и хранимые процедуры для базы данных, обеспечивающие добавление, удаление и редактирование атрибутов и их значений.

Администрирование товаров

Если существует такая категория, как нескучные задачи администрирования, то администрирование товаров наверняка относится к ней. Во-первых, у товаров есть изображения (а рисунки всегда интереснее текста), а во-вторых, у товаров гораздо больше свойств, с которыми наше приложение должно работать.

Поскольку администрирование товаров намного сложнее, чем администрирование отделов или категорий, мы создадим две отдельные страницы для работы с товарами. Первая страница будет называться Product Admin — на ней будет отображаться список товаров, относящихся к выбранной категории. Эта страница будет загружаться после щелчка на кнопке Edit Products в списке категорий. Как она будет выглядеть, показано на рис. 11.3.

Вторая страница, которой мы займемся чуть позже, будет называться Product Details Admin. Она будет загружаться после щелчков на кнопках Edit, показанных на рис. 11.3. Но пока что давайте сосредоточимся на первой странице.

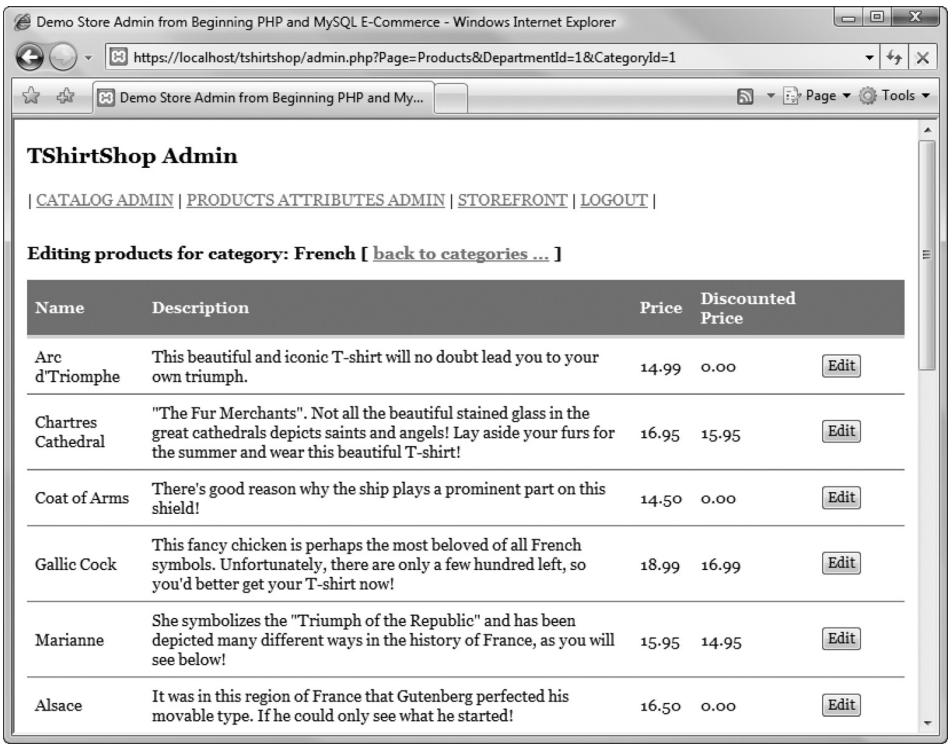


Рис. 11.3. Редактирование товаров

Упражнение: администрирование данных товарах

1. Создайте в папке `presentation/templates` файл шаблона `admin_products.tpl`. Поместите в него следующий код.

```
{* admin_products.tpl *}
{load_presentation_object filename="admin_products" assign="obj"}
<form method="post"
```

```
 action="{$obj->mLinkToCategoryProductsAdmin}">
  <h3>
      Editing products for category: {$obj->mCategoryName} [
      <a href="{$obj->mLinkToDepartmentCategoriesAdmin}">
       back to categories ...</a> ]
    </h3>
  {if $obj->mErrorMessage}<p class="error">{$obj->mErrorMessage}</p>{/if}
  {if $obj->mProductsCount eq 0}
    <p class="no-items-found">There are no products in this category!</p>
  {else}
  <table class="tss-table">
    <tr>
      <th>Name</th>
      <th>Description</th>
      <th>Price</th>
      <th>Discounted Price</th>
      <th width="80"> </th>
    </tr>
  {section name=i loop=$obj->mProducts}
    <tr>
      <td>{$obj->mProducts[i].name}</td>
      <td>{$obj->mProducts[i].description}</td>
      <td>{$obj->mProducts[i].price}</td>
      <td>{$obj->mProducts[i].discounted_price}</td>
      <td>
        <input type="submit"
         name="submit_edit_prod_{$obj->mProducts[i].product_id}"
         value="Edit" />
      </td>
    </tr>
  {/section}
  </table>
{/if}
  <h3>Add new product:</h3>
  <input type="text" name="product_name" value="[name]" size="30" />
  <input type="text" name="product_description" value="[description]"
   size="60" />
  <input type="text" name="product_price" value="[price]" size="10" />
  <input type="submit" name="submit_add_prod_0" value="Add" />
</form>
```

2. Создайте в папке `presentation` файл объекта представления с именем `admin_products.php` и поместите в него следующий код.

```
<?php
// Класс, отвечающий за администрирование товаров категории
class AdminProducts
{
  // Public-переменные, доступные в шаблоне Smarty
  public $mProductsCount;
  public $mProducts;
  public $mErrorMessage;
  public $mDepartmentId;
  public $mCategoryId;
  public $mCategoryName;
  public $mLinkToDepartmentCategoriesAdmin;
  public $mLinkToCategoryProductsAdmin;

  // Private-переменные
  private $_mAction;
  private $_mActionedProductId;
```

```
// Конструктор класса
public function __construct()
{
  if (isset ($_GET['DepartmentId']))
    $this->mDepartmentId = (int)$_GET['DepartmentId'];
  else
    trigger_error('DepartmentId not set');

  if (isset ($_GET['CategoryId']))
    $this->mCategoryId = (int)$_GET['CategoryId'];
  else
    trigger_error('CategoryId not set');

  $category_details = Catalog::GetCategoryDetails($this->mCategoryId);
  $this->mCategoryName = $category_details['name'];
  foreach ($_POST as $key => $value)
    // Если нажата кнопка отправки...
    if (substr($key, 0, 6) == 'submit')
    {
      /* Получаем позицию последнего символа '_' из имени кнопки,
         например, strrpos('submit_edit_prod_1', '_') вернет значение 17 */
      $last_underscore = strrpos($key, '_');
      /* Получаем область действия кнопки
         (например, 'edit_prod' из 'submit_edit_prod_1') */
      $this->_mAction = substr($key, strlen('submit_'),
      $last_underscore - strlen('submit_'));
      /* Получаем идентификатор атрибута, к которому относится кнопка
         (номер в конце имени этой кнопки), например,
         '1' из 'submit_edit_prod_1' */
      $this->_mActionedProductId = (int)substr($key, $last_underscore + 1);
      break;
    }
  $this->mLinkToDepartmentCategoriesAdmin =
    Link::ToDepartmentCategoriesAdmin($this->mDepartmentId);

  $this->mLinkToCategoryProductsAdmin =
    Link::ToCategoryProductsAdmin($this->mDepartmentId,
                                  $this->mCategoryId);
}

public function init()
{
  // При добавлении нового товара...
  if ($this->_mAction == 'add_prod')
  {
    $product_name = $_POST['product_name'];
    $product_description = $_POST['product_description'];
    $product_price = $_POST['product_price'];

    if ($product_name == null)
      $this->mErrorMessage = 'Product name is empty';

    if ($product_description == null)
      $this->mErrorMessage = 'Product description is empty';

    if ($product_price == null || !is_numeric($product_price))
      $this->mErrorMessage = 'Product price must be a number!';

    if ($this->mErrorMessage == null)
    {
      Catalog::AddProductToCategory($this->mCategoryId, $product_name,
```

```
          $product_description, $product_price);

        header('Location: ' .
               htmlspecialchars_decode(
                 $this->mLinkToCategoryProductsAdmin));
      }
    }

    // Если мы хотим просмотреть сведения о товаре
    if ($this->_mAction == 'edit_prod')
    {
      header('Location: ' .
             htmlspecialchars_decode(
             Link::ToProductAdmin($this->mDepartmentId,
                                  $this->mCategoryId,
                                  $this->_mActionedProductId)));
      exit();
    }
    $this->mProducts = Catalog::GetCategoryProducts($this->mCategoryId);
    $this->mProductsCount = count($this->mProducts);
  }
}
?>
```

3. Откройте файл `presentation/link.php` и добавьте в конец класса `Link` следующий метод.

```
// Создает ссылку на страницу администрирования товаров
public static function ToCategoryProductsAdmin($departmentId, $categoryId)
{
  $link = 'Page=Products&DepartmentId=' . $departmentId .
          '&CategoryId=' . $categoryId;
  return self::ToAdmin($link);
}
```

4. Добавьте в метод `init()` класса `StoreAdmin` в файле `store_admin.php` загрузку компонентного шаблона `admin_products`:

```
// Выбираем, какую страницу администрирования загружать...
if ($admin_page == 'Departments')
  $this->mContentsCell = 'admin_departments.tpl';
elseif ($admin_page == 'Categories')
  $this->mContentsCell = 'admin_categories.tpl';
elseif ($admin_page == 'Attributes')
  $this->mContentsCell = 'admin_attributes.tpl';
elseif ($admin_page == 'AttributeValues')
  $this->mContentsCell = 'admin_attribute_values.tpl';
elseif ($admin_page == 'Products')
  $this->mContentsCell = 'admin_products.tpl';
```

5. Добавьте приведенный ниже код в класс `Catalog` в файле `business/catalog.php`.

```
// Получаем товары заданной категории
public static function GetCategoryProducts($categoryId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_category_products(:category_id)';

  // Создаем массив параметров
  $params = array (':category_id' => $categoryId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetAll($sql, $params);
```

```
}

// Создаем товар и зачисляем его в категорию
public static function AddProductToCategory($categoryId, $productName,
                        $productDescription, $productPrice)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_add_product_to_category(:category_id, :product_name,
                :product_description, :product_price)';

  // Создаем массив параметров
  $params = array (':category_id' => $categoryId,
                   ':product_name' => $productName,
                   ':product_description' => $productDescription,
                   ':product_price' => $productPrice);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}
```

6. С помощью phpMyAdmin выполните приведенные ниже операторы SQL, чтобы создать необходимые хранимые процедуры. Не забудьте задать символы `$$` в качестве разделителя.

```
-- Создает хранимую процедуру catalog_get_category_products
CREATE PROCEDURE catalog_get_category_products(IN inCategoryId INT)
BEGIN
  SELECT     p.product_id, p.name, p.description, p.price,
             p.discounted_price
  FROM       product p
  INNER JOIN product_category pc
               ON p.product_id = pc.product_id
  WHERE      pc.category_id = inCategoryId
  ORDER BY   p.product_id;
END$$

-- Создает хранимую процедуру catalog_add_product_to_category
CREATE PROCEDURE catalog_add_product_to_category(IN inCategoryId INT,
  IN inName VARCHAR(100), IN inDescription VARCHAR(1000),
  IN inPrice DECIMAL(10, 2))
BEGIN
  DECLARE productLastInsertId INT;

  INSERT INTO product (name, description, price)
         VALUES (inName, inDescription, inPrice);

  SELECT LAST_INSERT_ID() INTO productLastInsertId;

  INSERT INTO product_category (product_id, category_id)
         VALUES (productLastInsertId, inCategoryId);
END$$
```

7. Загрузите раздел администрирования в браузере, перейдите на страницу категорий и щелкните на кнопке Edit Products для какой-нибудь из категорий в списке. Должна загрузиться страница, похожая на изображенную на рис. 11.3.

Как это работает: администрирование товаров

Итак, теперь мы можем просматривать на странице администрирования товары по категориям. Пока что мы не можем редактировать описания товаров, но возможность их просматривать — уже заметный шаг вперед. Кроме того, вы можете добавлять в категории новые товары с помощью текстовых полей под списком товаров.

В этом упражнении мы реализовали следующие элементы.

- Базовую функциональность администрирования товаров в шаблоне Smarty `admin_products` (этот шаблон состоит из файлов `admin_products.tpl` и `admin_products.php`)
- Ссылку на новую страницу из шаблона `store_admin`
- Работающие с товарами служебные методы уровня логики приложения и хранимые процедуры в базе данных

Администрирование описаний товаров

Список товаров, который мы организовали в предыдущем упражнении, безусловно, полезен, но у него нет нескольких важных возможностей. Последний компонентный шаблон, которым мы займемся в этой главе, `admin_product_details`, активизируется после щелчков на кнопках Edit в списке товаров. Этот шаблон позволит нам делать следующее.

- Просматривать изображение товара
- Удалять товар из категории
- Удалять товар из базы данных безвозвратно
- Добавлять товар в дополнительную категорию
- Переносить товар в другую категорию
- Изменять цену товара и цену со скидкой
- Изменять название и/или описание товара

Процедура удаления товара не так проста, как может показаться на первый взгляд. Можно исключить товар из категории, удалив соответствующую запись из таблицы `product_category`, или удалить запись о товаре из таблицы `product`. Поскольку мы обращаемся к товарам в каталоге через категории, нам нужно следить, чтобы в базе данных не было товаров, не числящихся ни в одной категории.

На странице администрирования сведений о товарах будет две кнопки удаления: Remove from category (для исключения товара из категории) и Remove from catalog (для полного удаления товара из каталога, включая записи в таблицах `product` и `product_category`). Если товар относится к нескольким категориям, для него будет доступна только кнопка Remove from category, а если к одной — только кнопка Remove from catalog. Это позволит нам и реализовать механизмы удаления товаров, и гарантировать отсутствие в базе данных "сирот" — товаров, не относящихся ни к одной категории.

Как будет выглядеть страница администрирования сведений о товарах, показано на рис. 11.4 и 11.5. Мы создадим эту страницу в следующих упражнениях.

Сведения о товарах: реализация уровня представления

Администрированием сведений о товарах занимается весьма солидный фрагмент кода, поэтому мы будем создавать его в нескольких упражнениях. Для начала займемся уровнем представления.

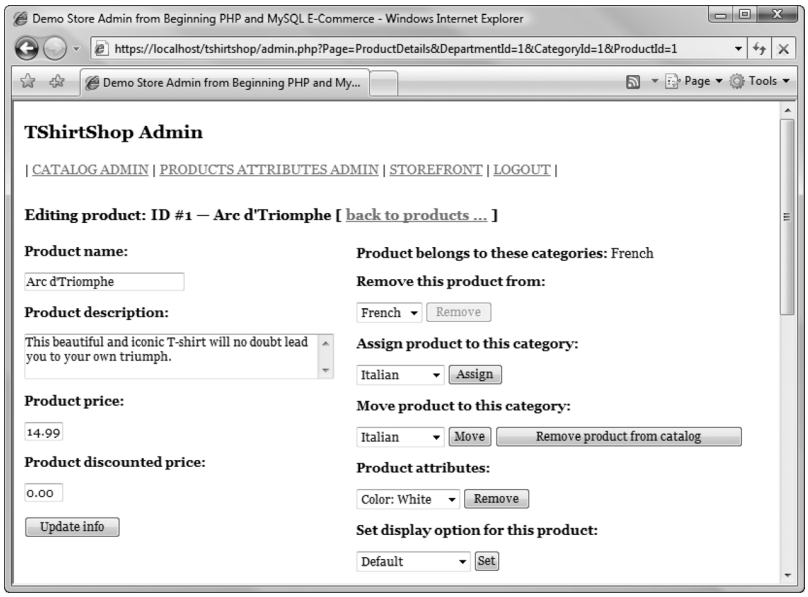


Рис. 11.4. Страница администрирования сведений о товарах

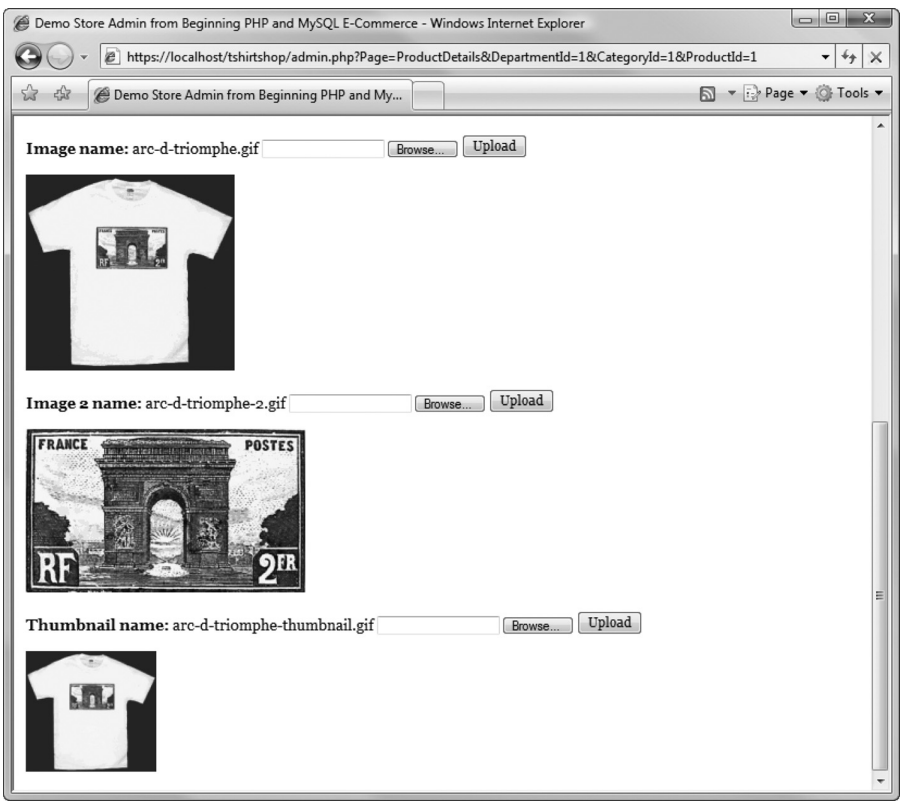


Рис. 11.5. Просмотр и редактирование изображений товаров

Упражнение: реализация уровня представления для администрирования сведений о товарах

1. Откройте файл `tshirtshop.css` в папке `styles` и добавьте в него следующие определения стилей.

```
.borderless-table {
  width: 100%;
}

.borderless-table td {
  border: none;
  padding-left: 0;
}

.bold-text {
  font-size: 93%;
  font-weight: bold;
}
```

2. В папке `presentation/templates` создайте файл `admin_product_details.tpl` и поместите в него следующий код.

```
{* admin_product_details.tpl *}
{load_presentation_object filename="admin_product_details" assign="obj"}
<form enctype="multipart/form-data" method="post"
 action="{$obj->mLinkToProductDetailsAdmin}">
  <h3>
    Editing product: ID #{$obj->mProduct.product_id} &mdash;
    {$obj->mProduct.name} [
    <a href="{$obj->mLinkToCategoryProductsAdmin}">
      back to products ...</a> ]
  </h3>
  {if $obj->mErrorMessage}<p class="error">{$obj->mErrorMessage}</p>{/if}
  <table class="borderless-table">
    <tbody>
      <tr>
        <td valign="top">
          <p class="bold-text">
            Product name:
          </p>
          <p>
            <input type="text" name="name"
             value="{$obj->mProduct.name}" size="30" />
          </p>
          <p class="bold-text">
            Product description:
          </p>
          <p>
            {strip}
            <textarea name="description" rows="3" cols="60">
              {$obj->mProduct.description}
            </textarea>
            {/strip}
        </p>
        <p class="bold-text">
          Product price:
        </p>
        <p>
          <input type="text" name="price"
           value="{$obj->mProduct.price}" size="5" />
```

```
    </p>
    <p class="bold-text">
      Product discounted price:
    </p>
    <p>
      <input type="text" name="discounted_price"
       value="{$obj->mProduct.discounted_price}" size="5" />
    </p>
    <p>
      <input type="submit" name="UpdateProductInfo"
       value="Update info" />
    </p>
  </td>
  <td valign="top">
    <p>
    <font class="bold-text">Product belongs to these categories:</font>
      {$obj->mProductCategoriesString}
    </p>
    <p class="bold-text">
      Remove this product from:
    </p>
    <p>
      {html_options name="TargetCategoryIdRemove"
       options=$obj->mRemoveFromCategories}
      <input type="submit" name="RemoveFromCategory" value="Remove"
      {if $obj->mRemoveFromCategoryButtonDisabled}
       disabled="disabled" {/if}/>
    </p>
    <p class="bold-text">
      Assign product to this category:
    </p>
    <p>
      {html_options name="TargetCategoryIdAssign"
       options=$obj->mAssignOrMoveTo}
      <input type="submit" name="Assign" value="Assign" />
    </p>
    <p class="bold-text">
      Move product to this category:
    </p>
    <p>
      {html_options name="TargetCategoryIdMove"
       options=$obj->mAssignOrMoveTo}
      <input type="submit" name="Move" value="Move" />
      <input type="submit" name="RemoveFromCatalog"
       value="Remove product from catalog"
      {if !$obj->mRemoveFromCategoryButtonDisabled}
        disabled="disabled" {/if}/>
      </p>
      {if $obj->mProductAttributes}
      <p class="bold-text">
        Product attributes:
      </p>
      <p>
        {html_options name="TargetAttributeValueIdRemove"
         options=$obj->mProductAttributes}
        <input type="submit" name="RemoveAttributeValue"
         value="Remove" />
      </p>
      {/if}
      {if $obj->mCatalogAttributes}
      <p class="bold-text">
```

```
            Assign attribute to product:
          </p>
          <p>
            {html_options name="TargetAttributeValueIdAssign"
             options=$obj->mCatalogAttributes}
            <input type="submit" name="AssignAttributeValue"
             value="Assign" />
          </p>
          {/if}
          <p class="bold-text">
            Set display option for this product:
          </p>
          <p>
            {html_options name="ProductDisplay"
             options=$obj->mProductDisplayOptions
             selected=$obj->mProduct.display}
            <input type="submit" name="SetProductDisplayOption" value="Set" />
          </p>
        </td>
      </tr>
    </tbody>
  </table>
  <p>
    <font class="bold-text">Image name:</font> {$obj->mProduct.image}
    <input name="ImageUpload" type="file" value="Upload" />
    <input type="submit" name="Upload" value="Upload" />
  </p>
  {if $obj->mProduct.image}
  <p>
    <img src="product_images/{$obj->mProduct.image}"
     border="0" alt="{$obj->mProduct.name} image" />
  </p>
  {/if}
  <p>
    <font class="bold-text">Image 2 name:</font> {$obj->mProduct.image_2}
    <input name="Image2Upload" type="file" value="Upload" />
    <input type="submit" name="Upload" value="Upload" />
  </p>
  {if $obj->mProduct.image_2}
  <p>
    <img src="product_images/{$obj->mProduct.image_2}"
     border="0" alt="{$obj->mProduct.name} image 2" />
  </p>
  {/if}
  <p>
    <font class="bold-text">Thumbnail name:</font> {
     $obj->mProduct.thumbnail}
    <input name="ThumbnailUpload" type="file" value="Upload" />
    <input type="submit" name="Upload" value="Upload" />
  </p>
  {if $obj->mProduct.thumbnail}
  <p>
    <img src="product_images/{$obj->mProduct.thumbnail}"
     border="0" alt="{$obj->mProduct.name} thumbnail" />
  </p>
  {/if}
</form>
```

3. Откройте файл `business/catalog.php` и добавьте элемент `$mProductDisplayOptions` в класс `Catalog`, так, как показано ниже:

```
<?php
// Класс уровня логики приложения для считывания информации
// о каталоге товаров
class Catalog
{
  // Определяет места отображения товара
  public static $mProductDisplayOptions = array ('Default', // 0
                                                 'On Catalog', // 1
                                                 'On Department', // 2
                                                 'On Both'); // 3

  // Получаем список отделов
  public static function GetDepartments()
  {
```

4. Создайте в папке `presentation` файл `admin_product_details.php` и поместите в него следующий код.

```
<?php
// Класс, отвечающий за администрирование сведений о товарах
class AdminProductDetails
{
  // Public-элементы
  public $mProduct;
  public $mErrorMessage;
  public $mProductCategoriesString;
  public $mProductDisplayOptions;
  public $mProductAttributes;
  public $mCatalogAttributes;
  public $mAssignOrMoveTo;
  public $mRemoveFromCategories;
  public $mRemoveFromCategoryButtonDisabled = false;
  public $mLinkToCategoryProductsAdmin;
  public $mLinkToProductDetailsAdmin;

  // Private-элементы
  private $_mProductId;
  private $_mCategoryId;
  private $_mDepartmentId;

  // Конструктор класса
  public function __construct()
  {
    // В строке запроса должен присутствовать DepartmentId
    if (!isset ($_GET['DepartmentId']))
      trigger_error('DepartmentId not set');
    else
      $this->_mDepartmentId = (int)$_GET['DepartmentId'];

    // В строке запроса должен присутствовать CategoryId
    if (!isset ($_GET['CategoryId']))
      trigger_error('CategoryId not set');
    else
      $this->_mCategoryId = (int)$_GET['CategoryId'];

    // В строке запроса должен присутствовать ProductId
    if (!isset ($_GET['ProductId']))
      trigger_error('ProductId not set');
    else
      $this->_mProductId = (int)$_GET['ProductId'];
```

```
    $this->mProductDisplayOptions = Catalog::$mProductDisplayOptions;

    $this->mLinkToCategoryProductsAdmin =
      Link::ToCategoryProductsAdmin($this->_mDepartmentId,
                                    $this->_mCategoryId);

    $this->mLinkToProductDetailsAdmin =
      Link::ToProductAdmin($this->_mDepartmentId,
                           $this->_mCategoryId,
                           $this->_mProductId);
  }

  public function init()
  {
    // При загрузке изображения товара...
    if (isset ($_POST['Upload']))
    {
      /* Проверяем, есть ли доступ на запись в папку product_images */
      if (!is_writeable(SITE_ROOT . '/product_images/'))
      {
        echo "Can't write to the product_images folder";
        exit();
      }

    // Если код ошибки - 0, файл успешно загружен
    if ($_FILES['ImageUpload']['error'] == 0)
    {
      /* Используем функцию PHP move_uploaded_file для перемещения
         загруженного файла из временной папки в папку product_images */
      move_uploaded_file($_FILES['ImageUpload']['tmp_name'],
                         SITE_ROOT . '/product_images/' .
                         $_FILES['ImageUpload']['name']);

      // Обновляем информацию о товаре в базе данных
      Catalog::SetImage($this->_mProductId,
                        $_FILES['ImageUpload']['name']);
    }
    // Если код ошибки - 0, файл успешно загружен
    if ($_FILES['Image2Upload']['error'] == 0)
    {
      /* Используем функцию PHP move_uploaded_file для перемещения
         загруженного файла из временной папки в папку product_images */
      move_uploaded_file($_FILES['Image2Upload']['tmp_name'],
                         SITE_ROOT . '/product_images/' .
                         $_FILES['Image2Upload']['name']);

      // Обновляем информацию о товаре в базе данных
      Catalog::SetImage2($this->_mProductId,
                         $_FILES['Image2Upload']['name']);
    }

    // Если код ошибки - 0, файл успешно загружен
    if ($_FILES['ThumbnailUpload']['error'] == 0)
    {
      // Перемещаем загруженный файл в папку product_images
      move_uploaded_file($_FILES['ThumbnailUpload']['tmp_name'],
                         SITE_ROOT . '/product_images/' .
                         $_FILES['ThumbnailUpload']['name']);

      // Обновляем информацию о товаре в базе данных
      Catalog::SetThumbnail($this->_mProductId,
```

```
                                $_FILES['ThumbnailUpload']['name']);
    }
  }

  // При обновлении информации о товаре...
  if (isset ($_POST['UpdateProductInfo']))
  {
    $product_name = $_POST['name'];
    $product_description = $_POST['description'];
    $product_price = $_POST['price'];
    $product_discounted_price = $_POST['discounted_price'];

    if ($product_name == null)
      $this->mErrorMessage = 'Product name is empty';

    if ($product_description == null)
      $this->mErrorMessage = 'Product description is empty';

    if ($product_price == null || !is_numeric($product_price))
      $this->mErrorMessage = 'Product price must be a number!';

    if ($product_discounted_price == null ||
        !is_numeric($product_discounted_price))
      $this->mErrorMessage = 'Product discounted price must be a number!';

    if ($this->mErrorMessage == null)
      Catalog::UpdateProduct($this->_mProductId, $product_name,
        $product_description, $product_price, $product_discounted_price);
  }

  // При удалении товара из категории...
  if (isset ($_POST['RemoveFromCategory']))
  {
    $target_category_id = $_POST['TargetCategoryIdRemove'];
    $still_exists = Catalog::RemoveProductFromCategory(
                      $this->_mProductId, $target_category_id);

    if ($still_exists == 0)
    {
      header('Location: ' .
             htmlspecialchars_decode(
               $this->mLinkToCategoryProductsAdmin));

      exit();
    }
  }

  // При задании режима отображения товара...
  if (isset ($_POST['SetProductDisplayOption']))
  {
    $product_display = $_POST['ProductDisplay'];
    Catalog::SetProductDisplayOption($this->_mProductId,
$product_display);
  }

  // При удалении товара из каталога...
  if (isset ($_POST['RemoveFromCatalog']))
  {
    Catalog::DeleteProduct($this->_mProductId);
    header('Location: ' .
           htmlspecialchars_decode(
```

```
            $this->mLinkToCategoryProductsAdmin));

  exit();
}

// При зачислении товара в другую категорию...
if (isset ($_POST['Assign']))
{
  $target_category_id = $_POST['TargetCategoryIdAssign'];
  Catalog::AssignProductToCategory($this->_mProductId,
                                   $target_category_id);
}

// При переносе товара в другую категорию...
if (isset ($_POST['Move']))
{
  $target_category_id = $_POST['TargetCategoryIdMove'];
  Catalog::MoveProductToCategory($this->_mProductId,
    $this->_mCategoryId, $target_category_id);

  header('Location: ' .
         htmlspecialchars_decode(
           Link::ToProductAdmin($this->_mDepartmentId,
                                $target_category_id,
                                $this->_mProductId)));
  exit();
}

// При присвоении товару значения атрибута...
if (isset ($_POST['AssignAttributeValue']))
{
  $target_attribute_value_id = $_POST['TargetAttributeValueIdAssign'];
  Catalog::AssignAttributeValueToProduct($this->_mProductId,
                                         $target_attribute_value_id);
}

// При удалении значения атрибута из товара...
if (isset ($_POST['RemoveAttributeValue']))
{
  $target_attribute_value_id = $_POST['TargetAttributeValueIdRemove'];
  Catalog::RemoveProductAttributeValue($this->_mProductId,
                                       $target_attribute_value_id);
}

// Получаем информацию о товаре
$this->mProduct = Catalog::GetProductInfo($this->_mProductId);
$product_categories = Catalog::GetCategoriesForProduct(
$this->_mProductId);

$product_attributes =
  Catalog::GetProductAttributes($this->_mProductId);

for ($i = 0; $i < count($product_attributes); $i++)
  $this->mProductAttributes[
    $product_attributes[$i]['attribute_value_id']] =
    $product_attributes[$i]['attribute_name'] . ': ' .
    $product_attributes[$i]['attribute_value'];

$catalog_attributes =
  Catalog::GetAttributesNotAssignedToProduct($this->_mProductId);
```

```
    for ($i = 0; $i < count($catalog_attributes); $i++)
      $this->mCatalogAttributes[
        $catalog_attributes[$i]['attribute_value_id']] =
        $catalog_attributes[$i]['attribute_name'] . ': ' .
        $catalog_attributes[$i]['attribute_value'];

    if (count($product_categories) == 1)
      $this->mRemoveFromCategoryButtonDisabled = true;

    // Отображаем категории, к которым принадлежит товар
    for ($i = 0; $i < count($product_categories); $i++)
      $temp1[$product_categories[$i]['category_id']] =
        $product_categories[$i]['name'];

    $this->mRemoveFromCategories = $temp1;
    $this->mProductCategoriesString = implode(', ', $temp1);
    $all_categories = Catalog::GetCategories();

    for ($i = 0; $i < count($all_categories); $i++)
      $temp2[$all_categories[$i]['category_id']] =
        $all_categories[$i]['name'];

    $this->mAssignOrMoveTo = array_diff($temp2, $temp1);
  }
}
?>
```

5. Откройте файл `presentation/link.php` и добавьте в конец класса `Link` следующий метод.

```
// Создает ссылку на страницу администрирования информации о товарах
public static function ToProductAdmin($departmentId, $categoryId,
$productId)
{
  $link = 'Page=ProductDetails&DepartmentId=' . $departmentId .
          '&CategoryId=' . $categoryId . '&ProductId=' . $productId;
  return self::ToAdmin($link);
}
```

6. Модифицируйте метод `init()` класса `StoreAdmin` в файле `store_admin.php` для загрузки шаблона `admin_product_details`:

```
// Выбираем, какую страницу администрирования загружать...
if ($admin_page == 'Departments')
  $this->mContentsCell = 'admin_departments.tpl';
elseif ($admin_page == 'Categories')
  $this->mContentsCell = 'admin_categories.tpl';
elseif ($admin_page == 'Attributes')
  $this->mContentsCell = 'admin_attributes.tpl';
elseif ($admin_page == 'AttributeValues')
  $this->mContentsCell = 'admin_attribute_values.tpl';
elseif ($admin_page == 'Products')
  $this->mContentsCell = 'admin_products.tpl';
elseif ($admin_page == 'ProductDetails')
  $this->mContentsCell = 'admin_product_details.tpl';
```

Как это работает: создание шаблона `admin_product`

Хотя созданный нами шаблон пока что неработоспособен, в нем есть несколько элементов, которые стоит обсудить прямо сейчас. В файле `admin_product_details.tpl` содержится единственная форма с атрибутом `enctype="multipart/form-data"`. Этот атрибут необходим для загрузки рисунков и работает в сочетании с кодом HTML, разрешающим загрузку.

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

Обработка щелчков на кнопках загрузки рисунков обеспечивается методом `init()` класса `AdminProductDetails` (в файле `presentation/admin_product_details.php`).

```
    // При загрузке изображения товара...
    if (isset ($_POST['Upload']))
    {
      /* Проверяем, есть ли доступ на запись в папку product_images */
      if (!is_writeable(SITE_ROOT . '/product_images/'))
      {
        echo "Can't write to the product_images folder";
        exit();
      }

    // Если код ошибки - 0, файл успешно загружен
    if ($_FILES['ImageUpload']['error'] == 0)
    {
      /* Используем функцию PHP move_uploaded_file для перемещения
         загруженного файла из временной папки в папку product_images */
      move_uploaded_file($_FILES['ImageUpload']['tmp_name'],
                         SITE_ROOT . '/product_images/' .
                         $_FILES['ImageUpload']['name']);

      // Обновляем информацию о товаре в базе данных
      Catalog::SetImage($this->_mProductId,
                        $_FILES['ImageUpload']['name']);
    }
    // Если код ошибки - 0, файл успешно загружен
    if ($_FILES['Image2Upload']['error'] == 0)
    {
      /* Используем функцию PHP move_uploaded_file для перемещения
         загруженного файла из временной папки в папку product_images */
      move_uploaded_file($_FILES['Image2Upload']['tmp_name'],
                         SITE_ROOT . '/product_images/' .
                         $_FILES['Image2Upload']['name']);

      // Обновляем информацию о товаре в базе данных
      Catalog::SetImage2($this->_mProductId,
                         $_FILES['Image2Upload']['name']);
    }

    // Если код ошибки - 0, файл успешно загружен
    if ($_FILES['ThumbnailUpload']['error'] == 0)
    {
      // Перемещаем загруженный файл в папку product_images
      move_uploaded_file($_FILES['ThumbnailUpload']['tmp_name'],
                         SITE_ROOT . '/product_images/' .
                         $_FILES['ThumbnailUpload']['name']);

      // Обновляем информацию о товаре в базе данных
      Catalog::SetThumbnail($this->_mProductId,
                            $_FILES['ThumbnailUpload']['name']);
    }
  }
```

Глобальная переменная `$_FILES` — это двумерный массив, в котором хранится информация о загруженном файле (или файлах). Если переменная `$_FILES['ImageUpload']['error']` имеет значение 0, значит, основное изображение товара успешно загрузилось, и с ним можно работать. В переменной `$_FILES['ImageUpload']`

['tmp_name'] хранится временное имя загруженного на сервер файла, а в переменной $_FILES ['ImageUpload']['name'] — имя этого файла в том виде, в каком оно было указано при загрузке.

ЗАМЕЧАНИЕ

Полное описание переменной $_FILES доступно по адресу http://www.php.net/manual/en/features.file-upload.php.

Для перемещения загруженного на сервер файла из временной папки в папку product_images используется функция PHP move_uploaded_file().

```
/* Используем функцию PHP move_uploaded_file для перемещения
   загруженного файла из временной папки в папку product_images */
move_uploaded_file($_FILES['Image2Upload']['tmp_name'],
                   SITE_ROOT . '/product_images/' .
                   $_FILES['Image2Upload']['name']);
```

После загрузки файла изображения товара имя этого файла нужно сохранить в базе данных (в противном случае сама загрузка файла на сервер будет бессмысленной).

```
// Обновляем информацию о товаре в базе данных
Catalog::SetThumbnail($this->_mProductId,
                      $_FILES['ThumbnailUpload']['name']);
```

Как видите, загрузка файлов на сервер с помощью PHP сложностей не представляет.

Сведения о товарах: реализация уровня логики приложения

Чтобы обеспечить поддержку администрирования сведений о товарах на уровне логики приложения, необходимо реализовать следующие методы.

- UpdateProduct — для обновления сведений о товаре (названия, описания, цены и цены со скидкой)
- DeleteProduct — для полного удаления товара из каталога
- RemoveProductFromCategory — для удаления товара из категории (этот метод будет вызываться после щелчка на кнопке Remove from Category)
- GetCategories — для получения полного списка категорий из каталога
- GetProductInfo — для получения сведений о товаре
- GetCategoriesForProduct — для получения списка категорий, к которым относится заданный товар
- SetProductDisplayOption — для задания режима отображения товара
- AssignProductToCategory — для зачисления товара в категорию
- MoveProductToCategory — для перемещения товара в другую категорию
- GetAttributesNotAssignedToProduct — для получения из таблицы attribute_values всех значений атрибутов, не назначенных ни одному товару
- AssignAttributeValueToProduct — для присвоения значения атрибута товару
- RemoveProductAttributeValue — для удаления связи между товаром и значением атрибута из таблицы product_attribute
- SetImage1 — для изменения имени файла с изображением для заданного товара
- SetImage2 — для изменения имени второго файла с изображением для заданного товара

- `SetThumbnail` — для изменения имени файла с уменьшенным изображением для заданного товара

Упражнение: реализация методов уровня логики приложения

Поскольку функциональность более понятна на уровне данных, чем на уровне логики приложения, мы обсудим ее при создании хранимых процедур для базы данных. А сейчас просто добавьте в класс `Catalog` в файле `business/catalog.php` следующий код.

```
// Обновляет сведения о товаре
public static function UpdateProduct($productId, $productName,
                $productDescription, $productPrice,
                $productDiscountedPrice)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_update_product(:product_id, :product_name,
                :product_description, :product_price,
                :product_discounted_price)';

  // Создаем массив параметров
  $params = array (':product_id' => $productId,
                   ':product_name' => $productName,
                   ':product_description' => $productDescription,
                   ':product_price' => $productPrice,
                   ':product_discounted_price' => $productDiscountedPrice);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

// Удаляет товар из каталога
public static function DeleteProduct($productId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_delete_product(:product_id)';

  // Создаем массив параметров
  $params = array (':product_id' => $productId);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

// Удаляет товар из категории
public static function RemoveProductFromCategory($productId, $categoryId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_remove_product_from_category(
                :product_id, :category_id)';

  // Создаем массив параметров
  $params = array (':product_id' => $productId,
                   ':category_id' => $categoryId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetOne($sql, $params);
}
```

```
// Возвращает список категорий
public static function GetCategories()
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_categories()';

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetAll($sql);
}

// Возвращает сведения о товаре
public static function GetProductInfo($productId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_product_info(:product_id)';

  // Создаем массив параметров
  $params = array (':product_id' => $productId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetRow($sql, $params);
}

// Возвращает список категорий, к которым относится товар
public static function GetCategoriesForProduct($productId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_categories_for_product(:product_id)';

  // Создаем массив параметров
  $params = array (':product_id' => $productId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetAll($sql, $params);
}

// Включает товар в категорию
public static function SetProductDisplayOption($productId, $display)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_set_product_display_option(
           :product_id, :display)';

  // Создаем массив параметров
  $params = array (':product_id' => $productId,
                   ':display' => $display);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

// Включает товар в категорию
public static function AssignProductToCategory($productId, $categoryId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_assign_product_to_category(
            :product_id, :category_id)';

  // Создаем массив параметров
  $params = array (':product_id' => $productId,
                   ':category_id' => $categoryId);
```

```
  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

// Перемещает товар из одной категории в другую
public static function MoveProductToCategory($productId, $sourceCategoryId,
                        $targetCategoryId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_move_product_to_category(:product_id,
            :source_category_id, :target_category_id)';

  // Создаем массив параметров
  $params = array (':product_id' => $productId,
                   ':source_category_id' => $sourceCategoryId,
                   ':target_category_id' => $targetCategoryId);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

// Возвращает атрибуты, не присвоенные никаким товарам
public static function GetAttributesNotAssignedToProduct($productId)
{
  // Составляем SQL-запрос
  $sql = 'CALL
catalog_get_attributes_not_assigned_to_product(:product_id)';

  // Создаем массив параметров
  $params = array (':product_id' => $productId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetAll($sql, $params);
}

// Присваивает значение атрибута указанному товару
public static function AssignAttributeValueToProduct($productId,
                        $attributeValueId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_assign_attribute_value_to_product(
    :product_id, :attribute_value_id)';

  // Создаем массив параметров
  $params = array (':product_id' => $productId,
                   ':attribute_value_id' => $attributeValueId);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

// Удаляет значение атрибута для товара
public static function RemoveProductAttributeValue($productId,
                        $attributeValueId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_remove_product_attribute_value(
            :product_id, :attribute_value_id)';

  // Создаем массив параметров
  $params = array (':product_id' => $productId,
```

```
                    ':attribute_value_id' => $attributeValueId);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

// Изменяет имя файла изображения товара в базе данных
public static function SetImage($productId, $imageName)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_set_image(:product_id, :image_name)';

  // Создаем массив параметров
  $params = array (':product_id' => $productId, ':image_name' =>
$imageName);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

// Изменяет имя файла второго изображения товара в базе данных
public static function SetImage2($productId, $imageName)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_set_image_2(:product_id, :image_name)';

  // Создаем массив параметров
  $params = array (':product_id' => $productId, ':image_name' =>
$imageName);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

// Изменяет имя файла уменьшенного изображения товара в базе данных
public static function SetThumbnail($productId, $thumbnailName)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_set_thumbnail(:product_id, :thumbnail_name)';

  // Создаем массив параметров
  $params = array (':product_id' => $productId,
                   ':thumbnail_name' => $thumbnailName);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}
```

Сведения о товарах: реализация уровня данных

На уровне данных нам нужно создать хранимые процедуры, используемые только что созданными методами уровня логики приложения.

Упражнение: добавление хранимых процедур

1. С помощью phpMyAdmin выполните операторы SQL, приведенные в последующих шагах этого упражнения. Не забудьте задать символы-разделители `$$` перед выполнением этих операторов.
2. Первая хранимая процедура, которую мы создадим в этом упражнении, — `catalog_update_product`. Она обновляет сведения о товарах, используя данные из входных параметров `inProductId`, `inName`, `inDescription`, `inPrice` и `inDiscountedPrice`.

```
-- Создаем хранимую процедуру catalog_update_product
CREATE PROCEDURE catalog_update_product(IN inProductId INT,
  IN inName VARCHAR(100), IN inDescription VARCHAR(1000),
  IN inPrice DECIMAL(10, 2), IN inDiscountedPrice DECIMAL(10, 2))
BEGIN
  UPDATE product
  SET    name = inName, description = inDescription, price = inPrice,
         discounted_price = inDiscountedPrice
  WHERE  product_id = inProductId;
END$$
```

3. Следующая хранимая процедура — `catalog_delete_product`. Она полностью удаляет товар из базы данных `tshirtshop`, включая записи в таблицах `product_attribute`, `product_category` и `product`.

```
-- Создаем хранимую процедуру catalog_delete_product
CREATE PROCEDURE catalog_delete_product(IN inProductId INT)
BEGIN
  DELETE FROM product_attribute WHERE product_id = inProductId;
  DELETE FROM product_category WHERE product_id = inProductId;
  DELETE FROM product WHERE product_id = inProductId;
END$$
```

4. Создайте хранимую процедуру `catalog_remove_product_from_category`. Эта процедура сначала проверяет, к какому количеству категорий принадлежит указанный товар. Если он принадлежит более чем к одной категории, процедура удаляет его из заданной категории (идентификатор этой категории передается процедуре через параметр). Если товар принадлежит к одной категории, он полностью удаляется из базы данных.

```
-- Создаем хранимую процедуру catalog_remove_product_from_category
CREATE PROCEDURE catalog_remove_product_from_category(
IN inProductId INT, IN inCategoryId INT)
BEGIN
  DECLARE productCategoryRowsCount INT;

  SELECT count(*)
  FROM   product_category
  WHERE  product_id = inProductId
  INTO   productCategoryRowsCount;

  IF productCategoryRowsCount = 1 THEN
    CALL catalog_delete_product(inProductId);

    SELECT 0;
  ELSE
    DELETE FROM product_category
    WHERE category_id = inCategoryId AND product_id = inProductId;
```

```
    SELECT 1;
  END IF;
END$$
```

5. Следующая хранимая процедура — `catalog_get_categories`. Она возвращает список всех категорий в каталоге.

```
-- Создаем хранимую процедуру catalog_get_categories
CREATE PROCEDURE catalog_get_categories()
BEGIN
  SELECT   category_id, name, description
  FROM     category
  ORDER BY category_id;
END$$
```

6. Следующая хранимая процедура — `catalog_get_product_info`. Она считывает из базы данных название товара, его описание, цену, цену со скидкой, изображения и заданный режим отображения. Товар выбирается по идентификатору, переданному процедуре через параметр.

```
-- Создаем хранимую процедуру catalog_get_product_info
CREATE PROCEDURE catalog_get_product_info(IN inProductId INT)
BEGIN
  SELECT product_id, name, description, price, discounted_price,
         image, image_2, thumbnail, display
  FROM   product
  WHERE  product_id = inProductId;
END$$
```

7. Следующая хранимая процедура — `catalog_get_categories_for_product`. Она возвращает список всех категорий, к которым принадлежит указанный товар. В списке указываются идентификаторы и названия категорий.

```
-- Создаем хранимую процедуру catalog_get_categories_for_product
CREATE PROCEDURE catalog_get_categories_for_product(IN inProductId INT)
BEGIN
  SELECT   c.category_id, c.department_id, c.name
  FROM     category c
  JOIN     product_category pc
             ON c.category_id = pc.category_id
  WHERE    pc.product_id = inProductId
  ORDER BY category_id;
END$$
```

8. Следующая хранимая процедура — `catalog_set_product_display_option`. Она задает режим отображения товара на сайте.

```
-- Создаем хранимую процедуру catalog_set_product_display_option
CREATE PROCEDURE catalog_set_product_display_option(
  IN inProductId INT, IN inDisplay SMALLINT)
BEGIN
  UPDATE product SET display = inDisplay WHERE product_id = inProductId;
END$$
```

9. Следующая хранимая процедура — `catalog_assign_product_to_category`. Она зачисляет товар в категорию, добавляя пару значений (`product_id`, `category_id`) в таблицу `product_category`.

```
-- Создаем хранимую процедуру catalog_assign_product_to_category
CREATE PROCEDURE catalog_assign_product_to_category(
IN inProductId INT, IN inCategoryId INT)
BEGIN
  INSERT INTO product_category (product_id, category_id)
```

```
        VALUES (inProductId, inCategoryId);
END$$
```

10. Следующая хранимая процедура — `catalog_move_product_to_category`. Она перемещает товар из одной категории в другую.

```
-- Создаем хранимую процедуру catalog_move_product_to_category
CREATE PROCEDURE catalog_move_product_to_category(IN inProductId INT,
  IN inSourceCategoryId INT, IN inTargetCategoryId INT)
BEGIN
    UPDATE product_category
    SET    category_id = inTargetCategoryId
    WHERE  product_id = inProductId
           AND category_id = inSourceCategoryId;
END$$
```

11. Следующая хранимая процедура — `catalog_get_attributes_not_assigned_to_product`. Она возвращает список всех значений атрибутов, не используемых ни в одном товаре каталога (такие атрибуты не упоминаются в таблице `product_attribute`).

```
-- Создаем хранимую процедуру catalog_get_attributes_not_assigned_to_product
CREATE PROCEDURE catalog_get_attributes_not_assigned_to_product(
  IN inProductId INT)
BEGIN
  SELECT     a.name AS attribute_name,
             av.attribute_value_id, av.value AS attribute_value
  FROM       attribute_value av
  INNER JOIN attribute a
               ON av.attribute_id = a.attribute_id
  WHERE      av.attribute_value_id NOT IN
             (SELECT attribute_value_id
               FROM  product_attribute
               WHERE product_id = inProductId)
  ORDER BY   attribute_name, av.attribute_value_id;
END$$
```

12. Следующая хранимая процедура — `catalog_assign_attribute_to_product`. Она присваивает значение атрибута товару, добавляя соответствующую запись в таблицу `product_attribute`.

```
-- Создаем хранимую процедуру catalog_assign_attribute_value_to_product
CREATE PROCEDURE catalog_assign_attribute_value_to_product(
  IN inProductId INT, IN inAttributeValueId INT)
BEGIN
  INSERT INTO product_attribute (product_id, attribute_value_id)
         VALUES (inProductId, inAttributeValueId);
END$$
```

13. Следующая хранимая процедура — `catalog_remove_product_attribute`. Она удаляет значение атрибута, назначенное товару, удалив соответствующую строку из таблицы `product_attribute`.

```
-- Создаем хранимую процедуру catalog_remove_product_attribute_value
CREATE PROCEDURE catalog_remove_product_attribute_value(
  IN inProductId INT, IN inAttributeValueId INT)
BEGIN
  DELETE FROM  product_attribute
         WHERE product_id = inProductId AND
               attribute_value_id = inAttributeValueId;
END$$
```

14. Последние три хранимые процедуры, которые нужно создать, задают файлы, в которых хранятся изображения товаров. Это процедуры `catalog_set_image`, `catalog_set_image2` и `catalog_set_thumbnail`, которые вызываются, когда мы загружаем на сервер новые файлы с изображениями.

```
-- Создаем хранимую процедуру catalog_set_image
CREATE PROCEDURE catalog_set_image(
  IN inProductId INT, IN inImage VARCHAR(150))
BEGIN
  UPDATE product SET image = inImage WHERE product_id = inProductId;
END$$

-- Создаем хранимую процедуру catalog_set_image_2
CREATE PROCEDURE catalog_set_image_2(
  IN inProductId INT, IN inImage VARCHAR(150))
BEGIN
  UPDATE product SET image_2 = inImage WHERE product_id = inProductId;
END$$

-- Создаем хранимую процедуру catalog_set_thumbnail
CREATE PROCEDURE catalog_set_thumbnail(
  IN inProductId INT, IN inThumbnail VARCHAR(150))
BEGIN
  UPDATE product
  SET thumbnail = inThumbnail
  WHERE product_id = inProductId;
END$$
```

15. Загрузите страницу администрирования сведений о товарах в браузере и проверьте, все ли работает как полагается. На такую проверку уйдет немало времени — мы добавили в наш проект солидное количество новых возможностей. Как должна выглядеть страница администрирования сведений о товарах, показано на рис. 11.4 и 11.5.

Как это работает: администрирование сведений о товарах

Создавать механизмы администрирования — всегда скучное занятие, но мы уже почти закончили. Теперь администраторы сайта смогут изменять изображения товаров, редактировать их описания, задавать цены, удалять товары из отдельных категорий или из каталога в целом и управлять атрибутами товаров.

На данный момент наш сайт снабжен всеми необходимыми инструментами администрирования каталога. Но лучше будет добавить еще один механизм, который облегчит работу администраторов: ссылки для администрирования на страницах сайта.

Создание ссылок для администрирования

Созданные нами страницы администрирования позволяют управлять отделами каталога, категориями, товарами и атрибутами. В последней части этой главы мы создадим механизм, который немного облегчит работу администраторов. Этот механизм состоит из кнопок редактирования подобных показанным на рис. 11.6 кнопкам Edit Department Details и Edit Product Details. Эти кнопки отображаются, только если посетитель сайта прошел аутентификацию как администратор, и позволяют ему перейти прямо на соответствующие страницы администрирования.

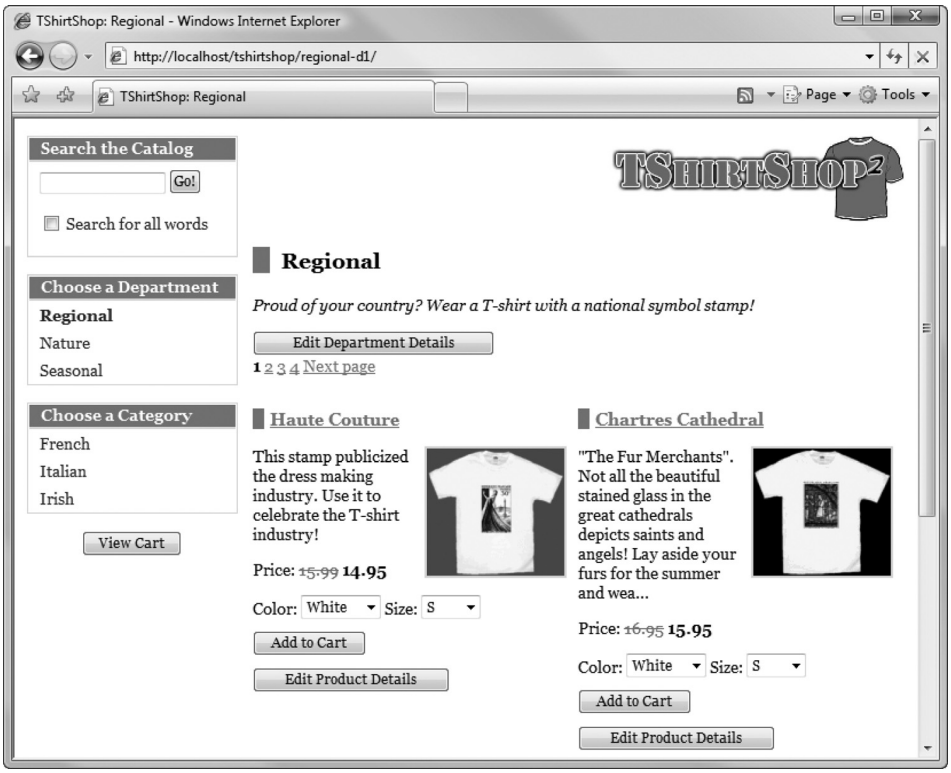


Рис. 11.6. Кнопки редактирования на странице сайта

Упражнение: реализация ссылок для администрирования

1. Откройте файл `tshirtshop.css` в папке `styles` и добавьте в него следующие определения стилей.

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

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

2. Откройте файл `presentation/templates/department.tpl` и внесите в него изменения, показанные ниже.

```
{* department.tpl *}
{load_presentation_object filename="department" assign="obj"}
<h1>{$obj->mName}</h1>
<p class="description">{$obj->mDescription}</p>
{if $obj->mShowEditButton}
<form action="{$obj->mEditActionTarget}" method="post" class="edit-form">
```

```
  <input type="submit" name="submit_{$obj->mEditAction}"
    value="{$obj->mEditButtonCaption}" />
</form>
{/if}
{include file="products_list.tpl"}
```

3. Добавьте выделенные полужирным шрифтом элементы из следующего листинга в класс `Department` в файле `presentation/department.php`.

```
<?php
// Занимается извлечением сведений об отделе
class Department
{
  // Public-переменные для шаблонов Smarty
  public $mName;
  public $mDescription;
  public $mEditActionTarget;
  public $mEditAction;
  public $mEditButtonCaption;
  public $mShowEditButton;
```

4. В том же файле, `presentation/department.php`, добавьте в конструктор класса `Department` фрагмент кода, выделенный полужирным шрифтом в следующем листинге.

```
  /* Если CategoryId есть в строке запроса, мы сохраняем его значение
    (преобразуя его в integer для защиты от некорректных значений) */
  if (isset ($_GET['CategoryId']))
    $this->_mCategoryId = (int)$_GET['CategoryId'];

  // Отображаем кнопку редактирования, если посетитель - администратор
  if (!(isset ($_SESSION['admin_logged'])) ||
      $_SESSION['admin_logged'] != true)
    $this->mShowEditButton = false;
  else
    $this->mShowEditButton = true;
}
```

5. В том же файле найдите метод `init()` класса `Department` и добавьте в конец этого метода код, выделенный полужирным шрифтом в следующем листинге.

```
public function init()
{
  // Если посещаем отдел...
  $department_details =
    Catalog::GetDepartmentDetails($this->_mDepartmentId);

  $this->mName = $department_details['name'];
  $this->mDescription = $department_details['description'];

  // Если посещаем категорию...
  if (isset ($this->_mCategoryId))
  {
    $category_details =
      Catalog::GetCategoryDetails($this->_mCategoryId);
    $this->mName = $this->mName . ' &raquo; ' .
      $category_details['name'];
    $this->mDescription = $category_details['description'];

    $this->mEditActionTarget =
      Link::ToDepartmentCategoriesAdmin($this->_mDepartmentId);
    $this->mEditAction = 'edit_cat_' . $this->_mCategoryId;
```

```
    $this->mEditButtonCaption = 'Edit Category Details';
  }
  else
  {
    $this->mEditActionTarget = Link::ToDepartmentsAdmin();
    $this->mEditAction = 'edit_dept_' . $this->_mDepartmentId;
    $this->mEditButtonCaption = 'Edit Department Details';
  }
}
```

6. Добавьте в файл `presentation/templates/department.tpl` код, выделенный полужирным шрифтом в листинге ниже:

```
{* Добавляем кнопку для отправки формы *}
<p>
  <input type="submit" name="submit" value="Add to Cart" />
</p>
</form>

{* Отображаем кнопку редактирования для администраторов *}
{if $obj->mShowEditButton}
<form action="{$obj->mEditActionTarget}" target="_self"
 method="post" class="edit-form">
  <p>
    <input type="submit" name="submit_edit" value="Edit Product Details" />
  </p>
</form>
{/if}

{if $obj->mLinkToContinueShopping}
<a href="{$obj->mLinkToContinueShopping}">Continue Shopping</a>
{/if}
```

7. Добавьте новые элементы в класс `Product` в файле `presentation/product.php`.

```
<?php
// Отвечает за отображение подробных сведений о товаре
class Product
{
  // Public-переменные для использования в шаблоне Smarty
  public $mProduct;
  public $mProductLocations;
  public $mLinkToContinueShopping;
  public $mLocations;
  public $mEditActionTarget;
  public $mShowEditButton;
```

8. Вставьте код, выделенный полужирным шрифтом в следующем листинге, в конец конструктора класса `Product` в файле `presentation/product.php`. Этот код обеспечивает отображение кнопки редактирования, если посетитель — администратор.

```
  else
    trigger_error('ProductId not set');

  // Отображаем кнопку редактирования для администраторов
  if (!(isset ($_SESSION['admin_logged'])) ||
      $_SESSION['admin_logged'] != true)
    $this->mShowEditButton = false;
  else
    $this->mShowEditButton = true;
}
```

9. В том же файле добавьте код, выделенный полужирным шрифтом в следующем листинге, в конец метода `init()` в классе `Product`. Этот код будет создавать ссылку на страницу администрирования товара.

```
    // Генерируем ссылки на страницы отдела и категории
    for ($i = 0; $i < count($this->mLocations); $i++)
    {
      $this->mLocations[$i]['link_to_department'] =
        Link::ToDepartment($this->mLocations[$i]['department_id']);

      $this->mLocations[$i]['link_to_category'] =
        Link::ToCategory($this->mLocations[$i]['department_id'],
                         $this->mLocations[$i]['category_id']);
    }

    // Подготавливаем кнопку редактирования
    $this->mEditActionTarget =
      Link::Build(str_replace(VIRTUAL_LOCATION, '', getenv('REQUEST_URI')));

    if (isset ($_SESSION['admin_logged']) &&
        $_SESSION['admin_logged'] == true &&
        isset ($_POST['submit_edit']))
    {
      $product_locations = $this->mLocations;

      if (count($product_locations) > 0)
      {
        $department_id = $product_locations[0]['department_id'];
        $category_id = $product_locations[0]['category_id'];

        header('Location: ' .
               htmlspecialchars_decode(
               Link::ToProductAdmin($department_id,
                                    $category_id,
                                    $this->_mProductId)));
      }
    }
  }
}
?>
```

10. Добавьте код, выделенный полужирным шрифтом в следующем листинге, в файл `presentation/templates/products_list.tpl`. Этот код добавляет кнопки редактирования в списки товаров.

```
    {* Добавляем кнопку для отправки формы *}
    <p>
      <input type="submit" name="submit" value="Add to Cart" />
    </p>
    </form>

  {* Отображаем кнопку редактирования для администраторов Jn *}
  {if $obj->mShowEditButton}
    <form action="{$obj->mEditActionTarget}" target="_self"
     method="post" class="edit-form">
      <input type="hidden" name="product_id"
       value="{$obj->mProducts[k].product_id}" />
      <input type="submit" name="submit" value="Edit Product Details" />
    </form>
  {/if}

  </td>
```

```
  {if $smarty.section.k.index % 2 != 0 && !$smarty.section.k.first ||
      $smarty.section.k.last}
    </tr>
  {/if}
{/section}
```

11. Откройте файл presentation/products_list.php и добавьте в класс ProductsList два новых элемента.

```
public $mAllWords = 'off';
public $mSearchString;
public $mEditActionTarget;
public $mShowEditButton;

// Private-элементы
private $_mDepartmentId;
private $_mCategoryId;
```

12. Добавьте код, выделенный полужирным шрифтом в следующем листинге, в конец метода __construct() класса ProductsList.

```
  $_SESSION['link_to_continue_shopping'] = $_SERVER['QUERY_STRING'];

  // Отображаем кнопку редактирования для администраторов
  if (!(isset ($_SESSION['admin_logged'])) ||
      $_SESSION['admin_logged'] != true)
    $this->mShowEditButton = false;
  else
    $this->mShowEditButton = true;
}
```

13. В том же файле presentation/products_list.php добавьте в начало метода init() класса ProductsList код, выделенный полужирным шрифтом в листинге ниже:

```
public function init()
{
  // Подготавливаем кнопку редактирования
  $this->mEditActionTarget =
    Link::Build(str_replace(VIRTUAL_LOCATION, '', getenv('REQUEST_URI')));

  if (isset ($_SESSION['admin_logged']) &&
      $_SESSION['admin_logged'] == true &&
      isset ($_POST['product_id']))
  {
    if (isset ($this->_mDepartmentId) && isset ($this->_mCategoryId))
      header('Location: ' .
             htmlspecialchars_decode(
             Link::ToProductAdmin($this->_mDepartmentId,
                                  $this->_mCategoryId,
                                  (int)$_POST['product_id'])));
    else
    {
      $product_locations =
        Catalog::GetProductLocations((int)$_POST['product_id']);

      if (count($product_locations) > 0)
      {
        $department_id = $product_locations[0]['department_id'];
        $category_id = $product_locations[0]['category_id'];
```

```
        header('Location: ' .
               htmlspecialchars_decode(
               Link::ToProductAdmin($department_id,
                                    $category_id,
                                    (int)$_POST['product_id'])));
      }
    }
  }

  /* Если выполнялся поиск, получаем список товаров, вызывая
     метод уровня логики приложения Search() */
  if (isset ($this->mSearchString))
  {
    // Получаем результаты поиска
    $search_results = Catalog::Search($this->mSearchString,
                                      $this->mAllWords,
                                      $this->mPage,
                                      $this->mrTotalPages);
```

14. Откройте файл `presentation/store_front.php` и добавьте в начало метода `init()` класса `StoreFront` две новые строки:

```
public function init()
{
  $_SESSION['link_to_store_front'] =
    Link::Build(str_replace(VIRTUAL_LOCATION, '', getenv('REQUEST_URI')));

  // Создаем ссылку для возврата в магазин в корзине покупателя PayPal
  if (!isset ($_GET['AddProduct']))
  {
```

15. Откройте файл `presentation/admin_menu.php` и придайте конструктору класса `AdminMenu` следующий вид.

```
public function __construct()
{
  $this->mLinkToStoreAdmin = Link::ToAdmin();
  $this->mLinkToAttributesAdmin = Link::ToAttributesAdmin();

  if (isset ($_SESSION['link_to_store_front']))
    $this->mLinkToStoreFront = $_SESSION['link_to_store_front'];
  else
    $this->mLinkToStoreFront = Link::ToIndex();

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

16. Попробуйте перемещаться между страницами сайта TShirtShop в браузере, пройдя аутентификацию в качестве администратора, и вы увидите кнопки редактирования, подобные показанным на рис. 11.6 и 11.7.

Как это работает: кнопки редактирования

В этом упражнении мы создали кнопки открытия страниц редактирования. Они отображаются после аутентификации посетителя в качестве администратора. Это позволит администратору, который заметит, например, опечатку в описании товара, быстро открыть страницу редактирования и исправить эту опечатку.

Вероятно, реализация этой возможности была не самой захватывающей частью нашей работы, но результаты вашего труда наверняка оценят те, кому придется обслуживать сайт.

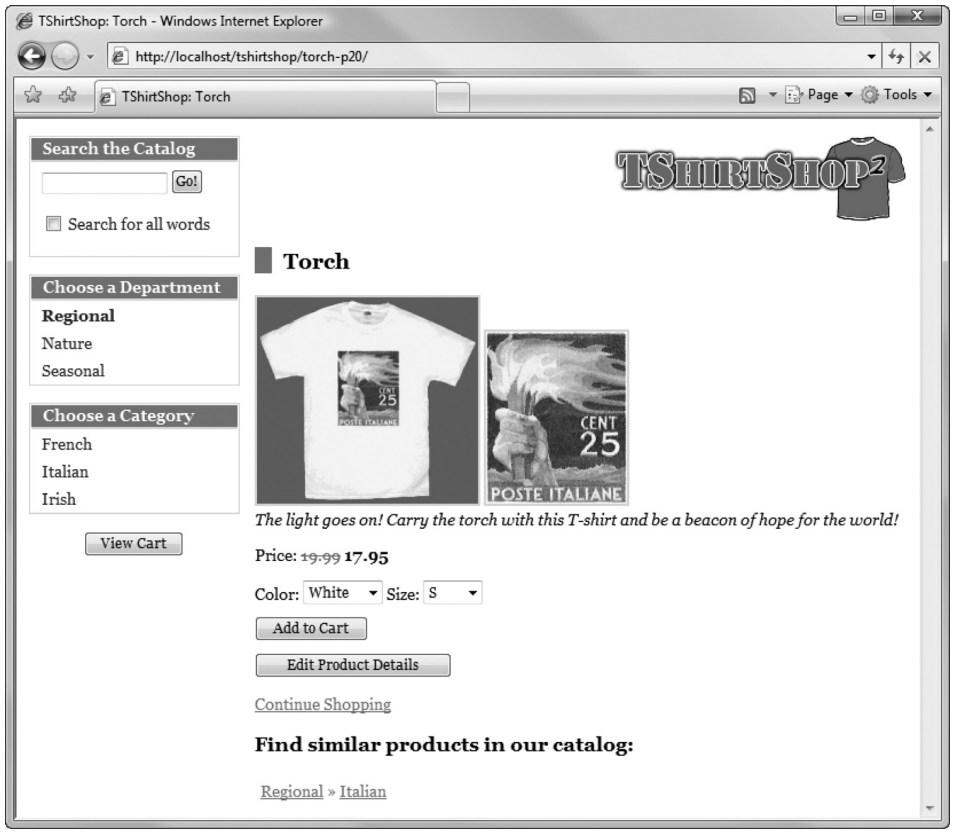


Рис. 11.7. Кнопка редактирования на странице сайта

Резюме

В этой главе мы завершили создание механизмов администрирования сайта, включая механизмы редактирования товаров и их атрибутов, зачисления товаров в категории, загрузки изображений и т.д. Кроме того, мы добавили на страницы TShirtShop кнопки, позволяющие администраторам быстро открывать страницы администрирования для обновления информации в каталоге.

На этом первый этап разработки нашего сайта закончен, и мы можем заняться более интересными делами. В главе 12 мы создадим для сайта собственную корзину покупателя, заменив ту, которой нас снабдил PayPal.

Часть II

Второй этап разработки

Глава 12

Создание оригинальной корзины покупателя

Мы перешли ко второй стадии разработки! На этой стадии мы займемся улучшением нашего сайта и добавлением в него новых элементов, которые сделают работу с ним более удобной.

Итак, что именно мы можем улучшить? Ответ на этот вопрос несложно найти, посетив несколько популярных сайтов электронных магазинов. Они ведут статистику посещений, запоминают предпочтения конкретных посетителей, сообщают о новых товарах и акциях и делают еще массу вещей, чтобы соблазнить посетителя купить что-нибудь, прежде чем уйти с сайта.

На первой стадии разработки мы воспользовались услугами сторонней компании (PayPal) для организации корзины покупателя. Это позволило нам избежать возни с сохранением данных о корзинах покупателей в базе данных. Но это привело к заметному пробелу в возможностях сайта — мы не можем, например, показать посетителю список самых покупаемых товаров, поскольку наш сайт вообще не отслеживает, сколько каких товаров было продано. Сохранение информации о продажах в базе данных — одна из наших приоритетных задач, поскольку большинство механизмов, которые мы реализуем в дальнейшем, будут полагаться на эту информацию.

К концу этой главы у нас будет полностью рабочая корзина покупателя, но посетитель еще не сможет заказывать товары, находящиеся в ней. Функциональность заказа мы добавим на сайт в главе 14, реализовав собственную систему приема платежей, интегрированную с корзиной покупателя. Из этой главы вы узнаете, как

- проанализировать содержимое корзины покупателя;
- создать структуру в базе данных, которая позволит хранить данные о содержимом корзин конкретных покупателей;
- реализовать компоненты корзины покупателя на уровне представления, уровне логики приложения и уровне данных;
- обновить кнопки PayPal Add to Cart, созданные в главе 9, для работы с новой корзиной покупателя;
- создать поле, отображающее сумму заказа и содержимое корзины, которое будет напоминать посетителям, что они заказали;

- реализовать страницу администрирования корзины покупателя, которая позволит администраторам удалять из базы данных корзины, не обновлявшиеся длительное время.

Проектирование корзины покупателя

В этой главе мы создадим корзину покупателя, которая будет сохранять данные о заказах в базе данных `tshirtshop`. Это даст нам массу возможностей, которых мы были лишены при использовании корзины PayPal. Корзиной PayPal мы практически не могли управлять, и ее содержимое нельзя было сохранять в базе данных для дальнейшей обработки и анализа. Когда посетитель будет щелкать на кнопке Add to Cart в нашей корзине покупателя, добавляемый в корзину товар будет запоминаться в нашей базе данных, а не в базе данных PayPal, недоступной для нас. После щелчка на кнопке Add To Cart будет открываться страница, подобная показанной на рис. 12.1.

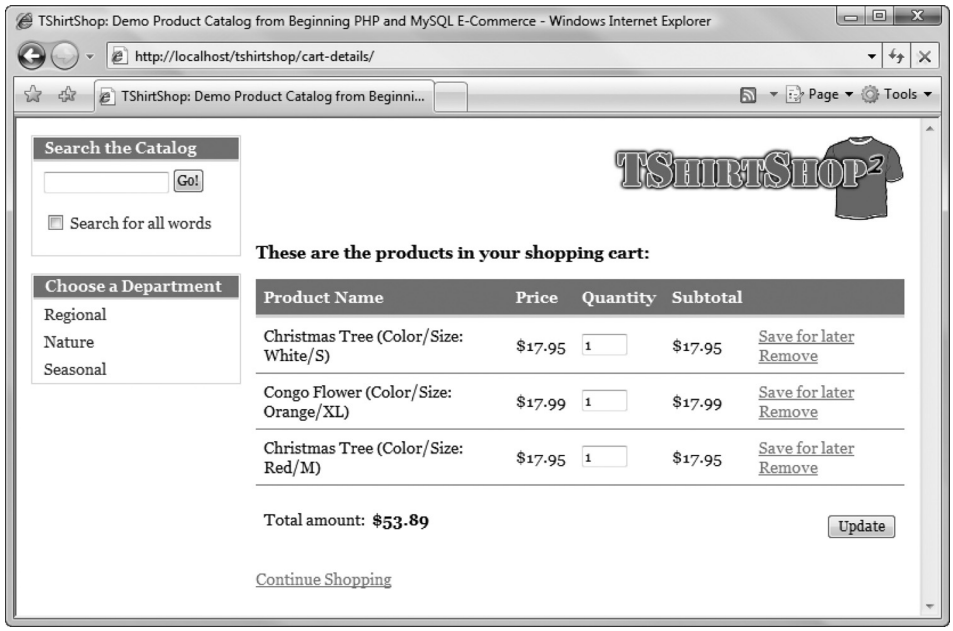


Рис. 12.1. Новая корзина покупателя на сайте TShirtShop

Кроме того, в новой корзине будет возможность сохранять заказы, чтобы позже посетители могли к ним вернуться. Можно будет сохранять только часть товаров из корзины, оплачивая остальные немедленно. Если товар откладывается для оплаты в будущем, он переносится в отдельный список и не включается в счет, выставляемый к оплате (рис. 12.2).

На всех страницах сайта, кроме страницы корзины, посетитель сможет видеть содержимое корзины в левой части страницы, как на рис. 12.3.

Прежде чем приступить к написанию кода, давайте подумаем, что и в какой последовательности нужно сделать.

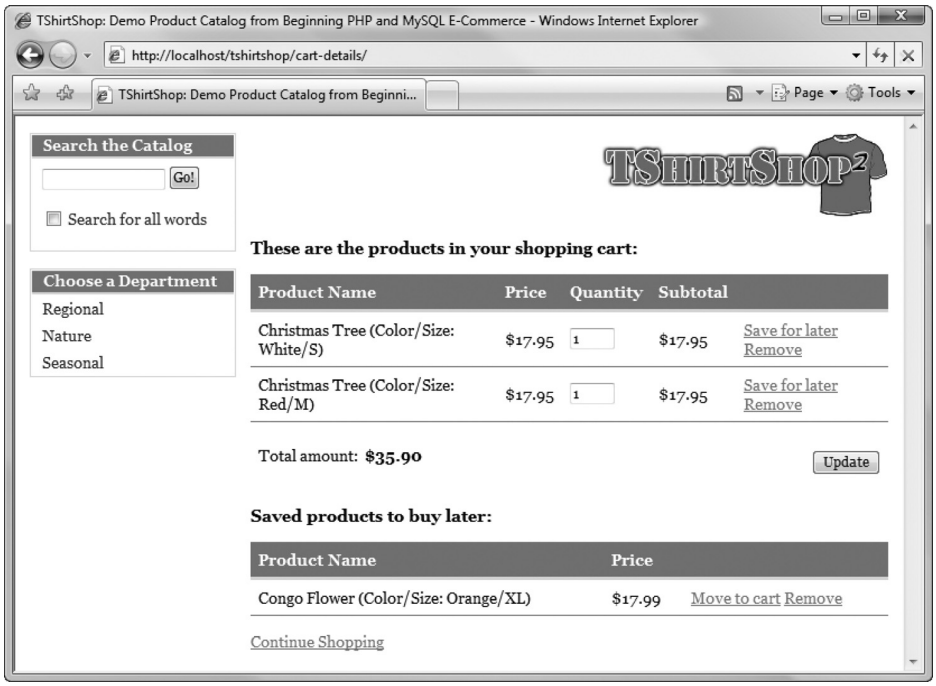


Рис. 12.2. Товары в корзине, отложенные для оплаты в будущем

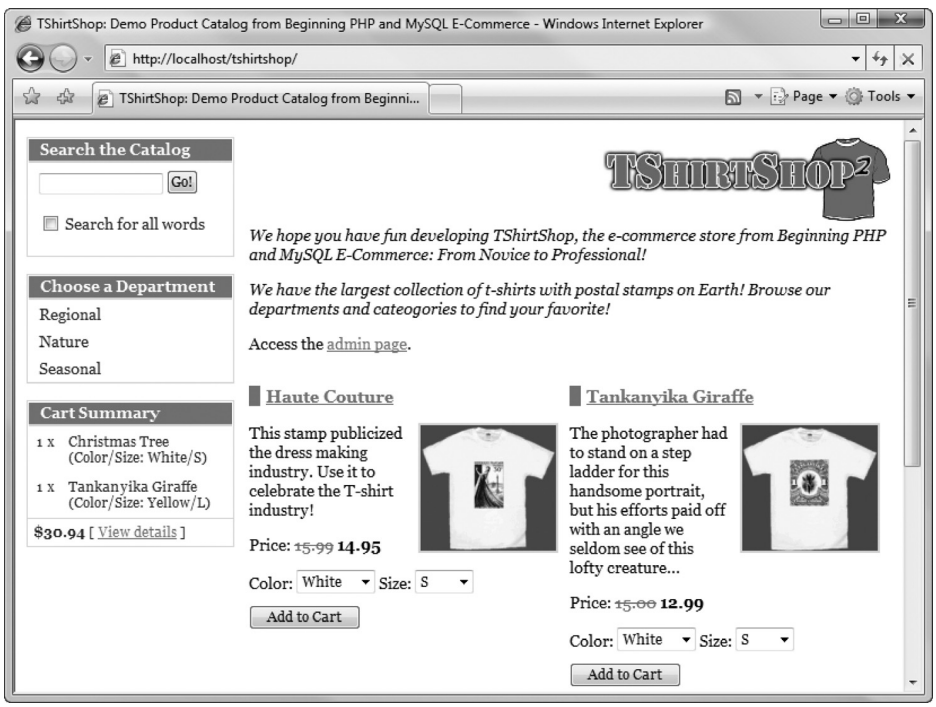


Рис. 12.3. Содержимое корзины на странице сайта

Заметим, что на данный момент на сайте нет никаких механизмов персонализации пользователей. Пока что это приемлемо — нам не нужно знать, кто именно покупает каждый товар; достаточно знать, чего и сколько продано. Когда позже мы добавим на сайт механизмы персонализации пользователей, наша задача будет несложной: если пользователь пройдет аутентификацию, временная (анонимная) корзина покупателя будет ассоциирована с учетной записью этого пользователя. Поскольку мы работаем с временными корзинами покупателя, даже после создания системы учетных записей пользователей посетителям не обязательно будет проходить аутентификацию раньше, чем это необходимо.

Для отслеживания корзин покупателей мы будем использовать cookie-файлы. Когда пользователь щелкает на кнопке Add to Cart, сервер сначала проверяет, существует ли cookie-файл сайта на компьютере пользователя. Если существует, выбранный товар добавляется в имеющуюся корзину. Если нет, сервер генерирует уникальный идентификатор, сохраняет его в cookie-файле на компьютере пользователя и добавляет товар в только что созданную новую корзину покупателя.

Сохранение информации о корзинах покупателей

Всю информацию о корзинах покупателей мы будем хранить в базе данных в единственной таблице — `shopping_cart`. Следующее упражнение посвящено созданию этой таблицы.

Упражнение: создание таблицы `shopping_cart`

1. Загрузите phpMyAdmin, выберите базу данных `tshirtshop` и откройте страницу нового SQL-запроса.
2. Выполните следующий код, который создаст таблицу `shopping_cart` в базе данных tshirtshop.

```
-- Создаем таблицу shopping_cart
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`)
);
```

Как это работает: таблица `shopping_cart`

Давайте рассмотрим поочередно каждый столбец таблицы `shopping_cart`.

- `item_id` — это первичный ключ таблицы. Значения в этом столбце однозначно идентифицируют каждую запись в таблице.
- `cart_id` — значение типа `CHAR`, однозначно идентифицирующее корзину покупателя (корзине могут соответствовать несколько записей в таблице, по одной записи на товар).
- `product_id` — значение, соответствующее идентификатору товара в базе данных.
- `attributes` — символьное поле переменной длины, в котором хранятся атрибуты, выбранные пользователем для товара при заказе. Например, это поле может иметь значение `Color/Size:White/XL`.

- `quantity` — количество экземпляров товара в корзине.
- `buy_now` — булево значение, по умолчанию — `true`, обеспечивающее поддержку откладывания товаров для оплаты в будущем. Когда пользователь добирается до этапа оплаты заказа, в заказ включаются только товары, для которых `buy_now` установлена равной `true`; все остальные товары остаются в корзине для оплаты в будущем.
- `added_on` — поле даты добавления товара в корзину. Это значение используется для вычисления возраста корзины, чтобы можно было удалять из базы данных старые корзины.

Значение в первом столбце, `item_id`, вычисляется при каждом добавлении в таблицу новой записи. Ничего особенного — вы уже встречались со столбцами с автоинкрементом.

Значение в столбце `cart_id` — это идентификатор корзины покупателя. Оно вычисляется кодом уровня логики приложения при создании новой корзины и однозначно идентифицирует конкретную корзину.

Важно понимать, что в каждой корзине покупателя может находиться несколько товаров, а у каждого товара есть множество конфигураций атрибутов, и покупатель вполне может заказать несколько одинаковых товаров, но с разными конфигурациями атрибутов. Например, можно заказать черную футболку среднего размера и белую — большого. Поэтому единственное сочетание полей, гарантирующее уникальность предметов в корзине, — сочетание (`cart_id`, `product_id`, `attributes`), которое можно было бы использовать как первичный ключ.

Однако MySQL накладывает определенные ограничения на длину значений в столбцах первичного ключа, и попытка создать такой ключ приведет к сообщению об ошибке `#1071 - Specified key was too long; max key length is 999 bytes`. Как говорится в этом сообщении, суммарная длина столбцов первичного ключа не может быть больше 999 байт, а в нашем случае их суммарная длина больше 3000 байт (каждый символ в кодировке UTF-8 занимает 3 байт). Это значит, что нам пришлось бы ограничить длину столбца `attributes` 269 символами, чтобы создать первичный ключ из этих трех столбцов. Возможно, сейчас такой длины нам хватит, но всегда следует думать о будущем. Поэтому в качестве первичного ключа мы используем специальный столбец — `item_id`. Такой подход нарушает правила третьей нормальной формы, т.е. наша база данных перестает соответствовать правилам проектирования баз данных. Но она все равно работает.

Реализация уровня данных

Теперь мы создадим хранимые процедуры, выполняющие все необходимые операции с корзиной покупателя. Вот список необходимых хранимых процедур.

- `shopping_cart_add_product` — для добавления товара в корзину
- `shopping_cart_update` — для обновления количества товаров в корзине и подсчета стоимости
- `shopping_cart_remove_product` — для удаления товара из корзины
- `shopping_cart_get_products` — для получения списка товаров из указанной корзины (эта хранимая процедура вызывается, когда нужно показать пользователю содержимое его корзины)
- `shopping_cart_get_saved_products` — для получения списка товаров, отложенных для оплаты в будущем (эта хранимая процедура также вызывается для показа пользователю содержимого корзины)
- `shopping_cart_get_total_amount` — для подсчета общей стоимости товаров в корзине без учета налогов и цены доставки
- `shopping_cart_save_product_for_later` — для откладывания товара из корзины для оплаты в будущем
- `shopping_cart_move_product_to_cart` — для перемещения товара из списка отложенных в корзину

А теперь создадим каждую из этих хранимых процедур.

Упражнение: создание хранимых процедур

1. С помощью phpMyAdmin создайте хранимые процедуры, перечисленные выше. Не забывайте задавать символы-разделители `$$` перед выполнением кода из каждого шага упражнения.
2. Создайте хранимую процедуру `shopping_cart_add_product`, выполнив следующий оператор SQL.

```
-- Создаем хранимую процедуру shopping_cart_add_product
CREATE PROCEDURE shopping_cart_add_product(IN inCartId CHAR(32),
  IN inProductId INT, IN inAttributes VARCHAR(1000))
BEGIN
  DECLARE productQuantity INT;

  -- Получаем количество единиц заданного товара в корзине
  SELECT quantity
  FROM   shopping_cart
  WHERE  cart_id = inCartId
         AND product_id = inProductId
         AND attributes = inAttributes
  INTO   productQuantity;

  -- Создаем новую запись в корзине, или увеличиваем количество
  -- товаров в уже существующей записи
  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$$
```

   Эта хранимая процедура вызывается после щелчка на кнопке Add to Cart для любого товара в каталоге. Если такой товар уже есть в корзине, его количество увеличивается на единицу; в противном случае в корзину добавляется новая разновидность товаров (создается новая запись `shopping_cart`).

   Процедура получает три параметра: `inCartId`, `inProductId` и `inAttributes`. Сначала она определяет, есть ли такой товар в корзине, просматривая атрибуты `cart_id`, `product_id` и `attributes`. Если такое сочетание этих атрибутов есть в корзине, значит, заданный товар помещен в корзину и нужно просто увеличить его количество на единицу. В противном случае хранимая процедура создает новую запись в таблице `shopping_cart` со значением `1` в столбце `quantity`. Для заполнения столбца `added_on` используется функция MySQL `NOW()`.

3. Выполните следующий оператор SQL, создающий хранимую процедуру `shopping_cart_update`.

```
-- Создаем хранимую процедуру shopping_cart_update_product
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$$
```

Эта хранимая процедура обновляет количество единиц одного товара в корзине. Она вызывается по одному разу для каждого товара в корзине, количество которого нужно обновить, после щелчка на кнопке Update на странице корзины покупателя.

Данная процедура использует два входных параметра: `inItemId` и `inQuantity`. Если значение `inQuantity` — нуль или меньше, эта процедура удаляет соответствующий товар из корзины. В противном случае она обновляет количество единиц товара и обновляет значение столбца `added_on`, чтобы зафиксировать время последнего обновления корзины. Это время проверяется на странице администрирования, чтобы определить, какие корзины давно не обновлялись и подлежат удалению.

4. Выполните следующий оператор SQL, создающий хранимую процедуру `shopping_cart_remove_product`.

```
-- Создаем хранимую процедуру shopping_cart_remove_product
CREATE PROCEDURE shopping_cart_remove_product(IN inItemId INT)
BEGIN
  DELETE FROM shopping_cart WHERE item_id = inItemId;
END$$
```

Эта хранимая процедура используется для удаления из корзины выбранного товара, когда пользователь щелкает на кнопке Remove в строке этого товара на странице корзины покупателя.

5. Выполните следующий оператор SQL, создающий хранимую процедуру `shopping_cart_get_products`.

```
-- Создаем хранимую процедуру shopping_cart_get_products
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$$
```

Эта процедура возвращает товары в корзине по значению `inCartId`. Поскольку в таблице `shopping_cart` хранятся значения `product_id` всех упоминаемых в ней товаров, для получения нужной информации мы соединяем таблицы `shopping_cart` и `product`.

Заметьте, что у некоторых товаров есть цены со скидками. Если у товара есть цена со скидкой (т.е. эта цена не равна нулю), для вычислений должна использоваться именно эта цена. В противном случае должна использоваться обычная цена. Следующее выражение возвращает значение `discount_price`, если оно не равно нулю; в противном случае оно возвращает значение `price`:

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

ЗАМЕЧАНИЕ

Поскольку это первая наша встреча с операторами COALESCE и NULLIF, сейчас я объясню, что они делают. COALESCE может принимать сколько угодно параметров и возвращает первый из них, не равный NULL. Оператор NULLIF принимает два параметра и возвращает значение NULL, если эти параметры равны; в противном случае он возвращает первый параметр. В нашем случае мы используем NULLIF, чтобы проверить, равен ли нулю параметр p.discount_price; если равен, NULLIF возвращает значение NULL, и COALESCE вернет цену без скидки; если не равен, все выражение вернет значение price.

6. Выполните следующий оператор SQL, создающий хранимую процедуру shopping_cart_get_saved_products.

```
-- Создаем хранимую процедуру shopping_cart_get_saved_products
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$$
```

Эта хранимая процедура возвращает товары из корзины, отложенные для оплаты в будущем.

7. Выполните следующий оператор SQL, создающий хранимую процедуру shopping_cart_get_total_amount.

```
-- Создаем хранимую процедуру shopping_cart_get_total_amount
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$$
```

Эта хранимая процедура возвращает общую стоимость всех товаров в корзине покупателя. Она вызывается для отображения этой стоимости на странице корзины. Если корзина пуста, процедура вернет значение нулю.

8. Выполните следующий оператор SQL, создающий хранимую процедуру shopping_cart_save_product_for_later.

```
-- Создаем хранимую процедуру shopping_cart_save_product_for_later
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$$
```

Эта хранимая процедура помечает товар как отложенный для оплаты в будущем, записывая в столбец buy_now значение false. После этого данный товар не будет учитываться при подсчете стоимости товаров в корзине во время оплаты.

9. Выполните следующий оператор SQL, создающий хранимую процедуру shopping_cart_move_product_to_cart.

```
-- Создаем хранимую процедуру shopping_cart_move_product_to_cart
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$$
```

Эта хранимая процедура делает товар, ранее отложенный для оплаты в будущем, снова подлежащим немедленной оплате.

Реализация уровня логики приложения

На уровне логики приложения поддержку корзины покупателя обеспечит класс `ShoppingCart`, код которого будет находиться в файле `shopping_cart.php`. В этом классе будут такие методы.

- `SetCartId()` — для генерации нового идентификатора корзины и его сохранения в cookie-файле и в сеансе
- `GetCartId()` — для получения идентификатора корзины
- `AddProduct()` — для добавления нового товара в корзину
- `Update()` — для изменения количества единиц товара в корзине
- `RemoveProduct()` — для удаления товара из корзины
- `GetCartProduct()` — для получения списка товаров, находящихся в корзине
- `GetTotalAmount()` — для подсчета общей стоимости товаров в корзине без учета налогов и цены доставки
- `SaveProductForLater()` — для откладывания товара в корзине для оплаты в будущем
- `MoveProductToCart()` — для возврата в корзину товара, ранее отложенного для оплаты в будущем

Давайте создадим этот класс.

Упражнение: реализация управляющей логики для корзины покупателя

1. Сначала добавьте две новые строки в конец файла `include/config.php`. Это объявления констант, которые позволят коду отличать товары, подлежащие оплате сейчас, от товаров, отложенных для оплаты в будущем.

```
// Типы товаров в корзине покупателя
define('GET_CART_PRODUCTS', 1);
define('GET_CART_SAVED_PRODUCTS', 2);
```

2. Создайте в папке `business` файл `shopping_cart.php`. Поместите в него следующий код (как он работает, мы рассмотрим немного позже).

```
<?php
// Класс уровня логики приложения, отвечающий за корзину покупателя
class ShoppingCart
{
  // Хранит идентификатор корзины
  private static $_mCartId;

  // Private-конструктор, не позволяющий напрямую создавать объекты класса
```

```
  private function __construct()
  {
  }

  /* Этот метод будет вызываться GetCartId(), чтобы убедиться, что
     в сеансе пользователя есть идентификатор корзины, если в
     $_mCartID не задано значение*/
  public static function SetCartId()
  {
    // Если идентификатор корзины еще не задан...
    if (self::$_mCartId == '')
    {
      // Если в сеансе есть идентификатор корзины, получаем его оттуда
      if (isset ($_SESSION['cart_id']))
      {
        self::$_mCartId = $_SESSION['cart_id'];
      }
      // Если нет, проверяем, не сохранен ли идентификатор в cookie_файле
      elseif (isset ($_COOKIE['cart_id']))
      {
        // Сохраняем идентификатор из cookie-файла
        self::$_mCartId = $_COOKIE['cart_id'];
        $_SESSION['cart_id'] = self::$_mCartId;

        // Регенерируем cookie-файл, чтобы он был
        // действителен 7 дней(604800 секунд)
        setcookie('cart_id', self::$_mCartId, time() + 604800);
      }
      else
      {
        /* Генерируем идентификатор корзины и сохраняем его в элементе класса
           $_mCartId, сеансе и cookie-файле (при последующих вызовах $_mCartId
           будет получать значение из сеанса) */
        self::$_mCartId = md5(uniqid(rand(), true));

        // Сохраняем идентификатор корзины в сеансе
        $_SESSION['cart_id'] = self::$_mCartId;
        // Cookie-файл будет действителен в течение 7 дней (604800 секунд)
        setcookie('cart_id', self::$_mCartId, time() + 604800);
      }
    }
  }

  // Возвращает идентификатор текущей корзины покупателя
  public static function GetCartId()
  {
    // Проверяем, есть ли идентификатор у корзины
    if (!isset (self::$_mCartId))
      self::SetCartId();

    return self::$_mCartId;
  }

  // Добавляет товар в корзину покупателя
  public static function AddProduct($productId, $attributes)
  {
    // Составляем SQL-запрос
    $sql = 'CALL shopping_cart_add_product(
              :cart_id, :product_id, :attributes)';

    // Создаем массив параметров
```

```
    $params = array (':cart_id' => self::GetCartId(),
                     ':product_id' => $productId,
                     ':attributes' => $attributes);

    // Выполняем запрос
    DatabaseHandler::Execute($sql, $params);
  }

  // Обновляет количества товаров в корзине покупателя
  public static function Update($itemId, $quantity)
  {
    // Составляем SQL-запрос
    $sql = 'CALL shopping_cart_update(:item_id, :quantity)';

    // Создаем массив параметров
    $params = array (':item_id' => $itemId,
                     ':quantity' => $quantity);

    // Выполняем запрос
    DatabaseHandler::Execute($sql, $params);
  }

  // Удаляет товар из корзины
  public static function RemoveProduct($itemId)
  {
    // Составляем SQL-запрос
    $sql = 'CALL shopping_cart_remove_product(:item_id)';

    // Создаем массив параметров
    $params = array (':item_id' => $itemId);

    // Выполняем запрос
    DatabaseHandler::Execute($sql, $params);
  }

  // Получает список товаров в корзине
  public static function GetCartProducts($cartProductsType)
  {
    $sql = '';
    // При получении списка товаров для немедленной оплаты...
    if ($cartProductsType == GET_CART_PRODUCTS)
    {
      // Составляем SQL-запрос
      $sql = 'CALL shopping_cart_get_products(:cart_id)';
    }
    // При получении списка товаров, отложенных для оплаты в будущем...
    elseif ($cartProductsType == GET_CART_SAVED_PRODUCTS)
    {
      // Составляем SQL-запрос
      $sql = 'CALL shopping_cart_get_saved_products(:cart_id)';
    }
    else
      trigger_error($cartProductsType. ' value unknown', E_USER_ERROR);

    // Создаем массив параметров
    $params = array (':cart_id' => self::GetCartId());

    // Выполняем запрос и возвращаем результаты
    return DatabaseHandler::GetAll($sql, $params);
  }
```

```
  /* Получает общую стоимость товаров в корзине (кроме товаров,
     отложенных для оплаты в будущем) */
  public static function GetTotalAmount()
  {
    // Составляем SQL-запрос
    $sql = 'CALL shopping_cart_get_total_amount(:cart_id)';

    // Создаем массив параметров
    $params = array (':cart_id' => self::GetCartId());

    // Выполняем запрос и возвращаем результаты
    return DatabaseHandler::GetOne($sql, $params);
  }

  // Переносит товар в список отложенных для оплаты в будущем
  public static function SaveProductForLater($itemId)
  {
    // Составляем SQL-запрос
    $sql = 'CALL shopping_cart_save_product_for_later(:item_id)';

    // Создаем массив параметров
    $params = array (':item_id' => $itemId);

    // Выполняем запрос
    DatabaseHandler::Execute($sql, $params);
  }

  // Возвращает товар из списка отложенных в список
  // подлежащих немедленной оплате
  public static function MoveProductToCart($itemId)
  {
    // Составляем SQL-запрос
    $sql = 'CALL shopping_cart_move_product_to_cart(:item_id)';

    // Создаем массив параметров
    $params = array (':item_id' => $itemId);

    // Выполняем запрос
    DatabaseHandler::Execute($sql, $params);
  }
}
?>
```

3. Создайте в файле `index.php` ссылку на только что созданный файл.

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

Как это работает: поддержка корзины покупателя на уровне логики приложения

Когда пользователь добавляет товар в корзину или запрашивает еще какую-нибудь операцию с корзиной, мы генерируем идентификатор корзины, если его еще нет. Этим занимается метод `SetCartId()`, помещающий сгенерированный идентификатор в элемент `$_mCartId` класса `ShoppingCart`. Кроме того, этот идентификатор сохраняется в сеансе пользователя и в cookie-файле на компьютере пользователя.

Сначала метод `SetCartId()` проверяет, задано ли значение элемента `$_mCartId`. Если задано, то значение идентификатора незачем считывать из внешних источников.

```
public static function SetCartId()
{
  // Если идентификатор корзины еще не задан...
  if (self::$_mCartId == '')
  {
```

Если значение элемента `$_mCartId` не задано, мы ищем его в сеансе пользователя.

```
// Если в сеансе есть идентификатор корзины, получаем его оттуда
if (isset ($_SESSION['cart_id']))
{
  self::$_mCartId = $_SESSION['cart_id'];
}
```

Если и в сеансе пользователя идентификатора корзины нет, проверяем, не сохранен ли он в cookie-файле. Если сохранен, записываем значение из cookie-файла и в сеанс пользователя, и в `$_mCartId` и регенерируем cookie-файл, чтобы обновить его срок действия.

```
// Если нет, проверяем, не сохранен ли идентификатор в cookie_файле
elseif (isset ($_COOKIE['cart_id']))
{
  // Сохраняем идентификатор из cookie-файла
  self::$_mCartId = $_COOKIE['cart_id'];
  $_SESSION['cart_id'] = self::$_mCartId;

  // Регенерируем cookie-файл, чтобы он был
  // действителен 7 дней(604800 секунд)
  setcookie('cart_id', self::$_mCartId, time() + 604800);
}
```

Наконец, если и в cookie-файле идентификатора корзины нет, генерируется новый идентификатор. Сгенерированный идентификатор сохраняется в элементе `$_mCartId`, в сеансе пользователя и в cookie-файле.

```
    else
    {
      /* Генерируем идентификатор корзины и сохраняем его в элементе класса
         $_mCartId, сеансе и cookie-файле (при последующих вызовах $_mCartId
         будет получать значение из сеанса) */
      self::$_mCartId = md5(uniqid(rand(), true));

      // Сохраняем идентификатор корзины в сеансе
      $_SESSION['cart_id'] = self::$_mCartId;
      // Cookie-файл будет действителен в течение 7 дней (604800 секунд)
      setcookie('cart_id', self::$_mCartId, time() + 604800);
    }
  }
}
```

Для генерации нового идентификатора используются три функции: `md5()`, `uniqid()` и `rand()`. Вызов `md5(uniqid(rand(), true))` генерирует труднопредсказуемое 32-байтовое значение, которое и будет идентификатором корзины покупателя.

ЗАМЕЧАНИЕ

Если вам интересно узнать немного больше подробностей о генерации идентификатора, вот эти подробности. Функция `md5()` генерирует хеш-значение своего параметра по алгоритму MD5. Хеш-значение имеет длину 32 байт. Функция `uniqid()` генерирует уникальный идентификатор, основанный на текущем времени в микросекундах; ее первый параметр — префикс, добавляемый к генерируемому значению. В нашем случае первый параметр — значение, возвращаемое функцией `rand()`, генерирующей случайное число в диапазоне от 0 до `RAND_MAX`, платформенно-зависимой величины. Если второй параметр `uniqid()` равен `true`, эта функция добавляет дополнительную энтропию для генератора случайных чисел, еще больше усложняя предсказание результата.

Проще говоря, этот вызов трех функций позволяет получать случайную последовательность из 32 символов, которая и будет идентификатором корзины покупателя.

Метод `SetCartId()` вызывается только из метода `GetCartId()`, возвращающего идентификатор корзины. Сначала метод `GetCartId()` проверяет, задано ли значение элемента `$_mCartId`, и, если не задано, вызывает метод `SetCartId()` перед тем, как возвратить это значение.

```
// Возвращает идентификатор текущей корзины покупателя
public static function GetCartId()
{
  // Проверяем, есть ли идентификатор у корзины
  if (!isset (self::$_mCartId))
    self::SetCartId();

  return self::$_mCartId;
}
```

Значение идентификатора сохраняется в столбце `cart_id` таблицы `shopping_cart` — этот столбец имеет тип `CHAR(32)`, выбранный специально для хранения значений, возвращаемых функцией `md5()`.

Кроме того, кое-что интересное есть в методе `GetCartProducts()`. Этот метод возвращает список товаров, находящихся в корзине покупателя. Он получает в качестве аргумента значение `$cartProductsType`, указывающее, нужно ли вернуть список товаров, подлежащих немедленной оплате, или список товаров, отложенных для оплаты в будущем.

Если `$cartProductsType` равно константе `GET_CART_PRODUCTS`, метод `GetCartProducts()` вернет товары, подлежащие немедленной оплате. Если `$cartProductsType` равно константе `GET_CART_SAVED_PRODUCTS`, метод вернет список товаров, отложенных для оплаты в будущем. Если значение `$cartProductsType` не равно ни одной из этих констант, `GetCartProducts()` сгенерирует ошибку.

Все остальные методы уровня логики приложения, созданные нами в этом упражнении, просто вызывают соответствующие хранимые процедуры для выполнения различных операций с корзиной покупателя.

Реализация уровня представления

А теперь займемся пользовательским интерфейсом корзины покупателя. Обновив этот интерфейс, мы получим кнопки Add to Cart для каждого товара и ссылку View Cart на панели просмотра корзины в левой части страниц. Если корзина покупателя пуста, ссылка отображаться не будет, как показано на рис. 12.4.

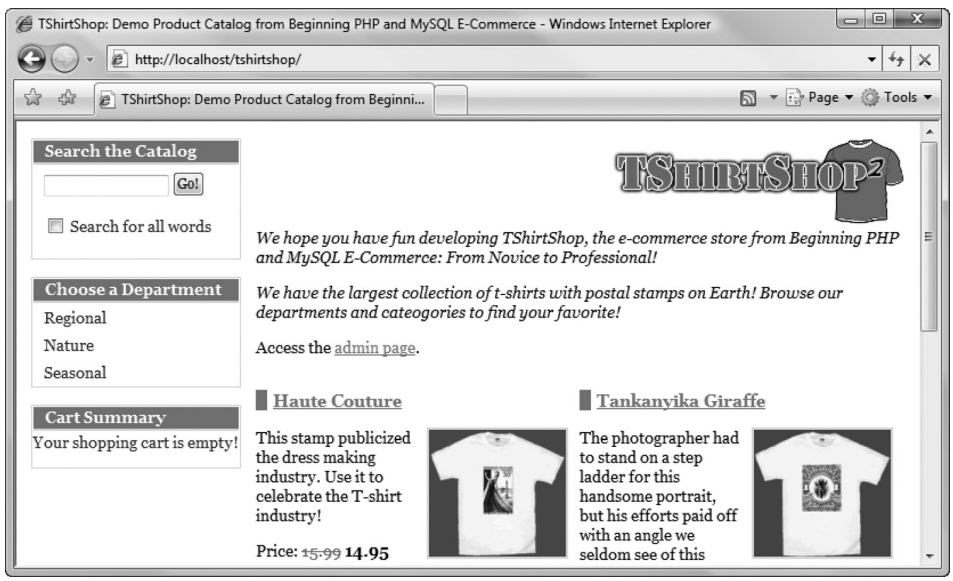


Рис. 12.4. Ссылки View Cart нет, если корзина покупателя пуста

Если в главе 9 вы интегрировали в сайт механизмы PayPal так, как описано в этой книге, то эти кнопки уже должны присутствовать на страницах, и сейчас мы просто изменим их функции.

После щелчка на кнопке View Cart загружается компонентный шаблон для отображения содержимого корзины. Результаты работы этого шаблона видны на рис. 12.1. Механизм загрузки компонентного шаблона содержимого корзины ничем не отличается от такового для других шаблонов в `store_front.tpl`. После щелчка на кнопке Add to Cart файл `index.php` загружается заново с дополнительным параметром (`CartAction`) в строке запроса, например, таким:

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

После щелчка на ссылке View Cart параметру `CartAction` в строке запроса значение не присваивается.

С корзиной покупателя можно выполнять пять основных операций, которым соответствуют константы, объявленные в файле `include/config.php`: `ADD_PRODUCT` (добавление товара в корзину), `REMOVE_PRODUCT` (удаление товара из корзины), `UPDATE_PRODUCTS_QUANTITIES` (изменение количеств товаров в корзине), `SAVE_PRODUCT_FOR_LATER` (откладывание товара для оплаты в будущем) и `MOVE_PRODUCT_TO_CART` (возврат отложенного товара для немедленной оплаты).

Прежде чем двигаться дальше, давайте перечислим основные шаги, которые нужно сделать, чтобы реализовать пользовательский интерфейс корзины покупателя.

1. Модифицировать кнопки Add to Cart для работы с нашей системой корзины покупателя.
2. Добавить поле корзины покупателя в шаблон `store_front.tpl` вместо кнопки View Cart.
3. Изменить метод `CheckRequest()` из класса `Link` в файле `presentation/link.php`, чтобы он мог распознавать параметр строки запроса `CartAction`.
4. Реализовать компонентный шаблон `cart_details`.

Обновление кнопок Add to Cart

Нам нужно изменить код в файле `products_list.tpl` так, чтобы для каждого отображаемого товара отображалась кнопка Add to Cart со ссылкой в формате, приведенном выше (с параметром `CartAction` в строке запроса).

Упражнение: создание механизма добавления товаров в корзину

1. Добавьте следующий код в конец файла `include/config.php`.

```
// Операции с корзиной покупателя
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. Если в главе 9 вы реализовали поддержку PayPal на сайте, удалите код из файла `presentation/store_front.php`, выполняющий перенаправления после щелчка на кнопке Add to Cart. Откройте файл `presentation/store_front.php` и *удалите* следующий фрагмент кода из метода `init()`.

```
// Создаем ссылку для возврата в магазин в корзине покупателя PayPal
if (!isset ($_GET['AddProduct']))
```

```
{
  /* Сохраняем текущий запрос, необходимый для работы ссылки в paypal,
     возвращающей в магазин */
  $_SESSION['paypal_continue_shopping'] =
    Link::Build(str_replace(VIRTUAL_LOCATION, '',
                            $_SERVER['REQUEST_URI']));
...
...
  // Выполняем перенаправление на страницу корзины PayPal
  header('HTTP/1.1 302 Found');
  header('Location: ' . $paypal_url);

  // Очищаем выходной буфер и завершаем выполнение
  flush();
  ob_flush();
  ob_end_clean();
  exit();
}
```

3. Откройте файл `presentation/link.php` и вставьте в метод `CheckRequest()` класса `Link` строку, выделенную полужирным шрифтом в следующем листинге.

```
// Выполняет перенаправление по корректному URL в случае необходимости
public static function CheckRequest()
{
  $proper_url = '';

if (isset ($_GET['Search']) || isset($_GET['SearchResults']) ||
    isset ($_GET['CartAction']))
{
  return ;
}
```

4. В том же файле `presentation/link.php` удалите метод `ToAddProduct()` и добавьте в класс `Link` новый метод, `ToCart()`,который добавляет ссылки Add Product и View Cart.

```
// Создаем ссылки для корзины покупателя
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. Откройте файл `presentation/products_list.php` и найдите в коде метода `init()` класса `ProductList` следующий код.

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

Замените этот код таким:

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

6. Откройте файл `presentation/product.php` и найдите в коде метода `init()` класса `Product` следующий код.

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

Замените этот код таким:

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

Как это работает: добавление ссылок на товары

Мы создали кнопки Add to Cart, содержащие ссылки на файл `index.php` с дополнительным параметром `CartAction` в строке запроса. Внеся это изменение, попробуйте открыть сайт в браузере, чтобы убедиться, что кнопки находятся на своих местах, хотя вы не сможете протестировать их работу до завершения модернизации уровня представления.

Если вы откроете какую-нибудь страницу каталога и щелкнете на кнопке Add to Cart для какого-нибудь товара, браузер попытается выполнить переход по URL примерно такого вида:

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

Параметр `CartAction` в строке запроса указывает, какую операцию с корзиной хочет выполнить пользователь. Операцию добавления нового товара обозначает код 1. На данный момент эта ссылка просто открывает основную страницу сайта, поскольку код не знает, как интерпретировать параметр `CartAction`.

Краткое отображение содержимого корзины на страницах сайта

Вместо кнопок View Cart, предоставляемых PayPal, мы создадим компонент сайта, который будет кратко сообщать о содержимом корзины и содержать ссылку View Details для подробного просмотра этого содержимого. Пример такого компонента на странице сайта показан на рис. 12.3. Создайте компонентный шаблон `cart_summary` для этого компонента, выполнив следующее упражнение.

Упражнение: краткое отображение содержимого корзины

1. Для начала удалите кнопки View Cart. Найдите приведенный ниже фрагмент кода в файле `presentation/templates/store_front.tpl` и удалите этот фрагмент.

```
<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. В тот же файл добавьте ссылку на компонент, отвечающий за отображение содержимого корзины.

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

3. Откройте файл `presentation/store_front.php` и вставьте в него фрагменты кода, выделенные полужирным шрифтом в следующем листинге.

```
<?php
class StoreFront
{
  public $mSiteUrl;
  // Определяем файл шаблона для содержимого страницы
  public $mContentsCell = 'first_page_contents.tpl';
  // Определяем файл шаблона для ячейки категорий
  public $mCategoriesCell = 'blank.tpl';
  // Определяем файл шаблона для ячейки содержимого корзины
  public $mCartSummaryCell = 'blank.tpl';
  // Заголовок страницы
  public $mPageTitle;
...
  public function init()
  {
...
    // Загружаем сведения о товаре на страницу товара
    if (isset ($_GET['ProductId']))
      $this->mContentsCell = 'product.tpl';

    // Загружаем страницу с результатами поиска, если выполнялся поиск
    elseif (isset ($_GET['SearchResults']))
      $this->mContentsCell = 'search_results.tpl';

    // Загружаем шаблоны для отображения содержимого корзины
    if (isset ($_GET['CartAction']))
      $this->mContentsCell = 'cart_details.tpl';
    else
      $this->mCartSummaryCell = 'cart_summary.tpl';

    // Загружаем заголовок страницы
    $this->mPageTitle = $this->_GetPageTitle();
  }
```

4. Создайте в папке `presentation` новый файл объекта представления с именем `cart_summary.php` и поместите в него такой код.

```
<?php
// Класс, кратко отображающий содержимое корзины
class CartSummary
{
  // Public-переменные, доступные в шаблоне Smarty
  public $mTotalAmount;
  public $mItems;
  public $mLinkToCartDetails;
  public $mEmptyCart;

  // Конструктор класса
  public function __construct()
  {
    /* Вычисляем общую стоимость товаров в корзине
```

```
       без учета налогов и цены доставки */
    $this->mTotalAmount = ShoppingCart::GetTotalAmount();
    // Получаем список товаров в корзине
    $this->mItems = ShoppingCart::GetCartProducts(GET_CART_PRODUCTS);
    if (empty($this->mItems))
      $this->mEmptyCart = true;
    else
      $this->mEmptyCart = false;
    $this->mLinkToCartDetails = Link::ToCart();
  }
}
?>
```

5. Создайте в папке `presentation/templates` новый файл с именем `cart_summary.tpl` и поместите в него такой код.

```
{* 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. Откройте файл `.htaccess` и добавьте в него такие две строки.

```
# Переписываем страницы с содержимым корзины
RewriteRule ^cart-details/?$ index.php?CartAction [L]
```

7. Добавьте следующие стили в файл `tshirtshop.css` в папке `styles`.

```
.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;
}
```

Как это работает: краткое отображение содержимого корзины

Если вы заново откроете сайт TShirtShop в браузере, то увидите поле для отображения содержимого корзины в левой части страниц каталога. Пока что вы еще не можете добавлять товары в корзину, поскольку нужно сначала создать страницу для отображения сведений о ней. Полностью протестировать только что созданный компонент вы сможете после выполнения следующего упражнения — в нем мы создадим недостающую страницу.

Отображение сведений о корзине

Сейчас попытки щелкнуть на кнопках Add to Cart и View Cart на страницах сайта приводят к генерации ошибок, поскольку мы не создали компонентного шаблона `cart_details`, отображающего сведения о корзине покупателя. Чтобы создать новый компонентный шаблон, сначала мы создадим файл `cart_details.tpl` в папке `presentation/templates`, а затем — файл объекта представления `cart_details.php` в папке `presentation`, в котором будет содержаться класс `CartDetails`.

Упражнение: отображение сведений о корзине

1. Модифицируйте код в файле `presentation/store_front.php` для поддержки возврата в магазин со страницы сведений о корзине. Добавьте код, выделенный полужирным шрифтом в следующем листинге, в метод `init()` класса `StoreFront`.

```
public function init()
{
  $_SESSION['link_to_store_front'] =
    Link::Build(str_replace(VIRTUAL_LOCATION, '', getenv('REQUEST_URI')));

  // Создаем ссылку для возврата в каталог
  if (!isset ($_GET['CartAction']))
  $_SESSION['link_to_last_page_loaded'] =
$_SESSION['link_to_store_front'];

  // Загружаем подробные сведения об отделе на страницу отдела
  if (isset ($_GET['DepartmentId']))
```

2. Создайте в папке `presentation` новый файл объекта представления с именем `cart_details.php` и поместите в него следующий код.

```
<?php
// Класс, отвечающий за работу корзины покупателя
class CartDetails
{
  // Public-переменные, доступные в шаблоне Smarty
  public $mCartProducts;
  public $mSavedCartProducts;
```

```
  public $mTotalAmount;
  public $mIsCartNowEmpty = 0; // Корзина пуста?
  public $mIsCartLaterEmpty = 0; // Список отложенных товаров пуст?
  public $mLinkToContinueShopping;
  public $mUpdateCartTarget;

  // Private-атрибуты
  private $_mItemId;
  private $_mCartAction;

  // Конструктор класса
  public function __construct()
  {
    if (isset ($_GET['CartAction']))
      $this->_mCartAction = $_GET['CartAction'];
    else
      trigger_error('CartAction not set', E_USER_ERROR);

    // Эти операции с корзиной требуют корректного идентификатора товара
    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);

    // Задаем цель для ссылки, возвращающей в каталог
    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 ();

        // Получаем выбранные атрибуты товаров, если они есть
        foreach ($_POST as $key => $value)
        {
          // Если в массиве POST есть элементы, начинающиеся с "attr_"
          if (substr($key, 0, 5) == 'attr_')
          {
            // Получаем имя и значение выбранного атрибута
            $selected_attributes[] = substr($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:
      // Не делаем ничего
      break;
  }

  /* Вычисляем общую стоимость товаров в корзине
     без учета налогов и цены доставки */
  $this->mTotalAmount = ShoppingCart::GetTotalAmount();

  // Получаем список товаров в корзине
  $this->mCartProducts =
    ShoppingCart::GetCartProducts(GET_CART_PRODUCTS);

  // Получаем список товаров, отложенных для оплаты в будущем
  $this->mSavedCartProducts =
    ShoppingCart::GetCartProducts(GET_CART_SAVED_PRODUCTS);

  // Проверяем, не пуста ли корзина
  if (count($this->mCartProducts) == 0)
    $this->mIsCartNowEmpty = 1;

  // Проверяем, не пуст ли список отложенных товаров
  if (count($this->mSavedCartProducts) == 0)
    $this->mIsCartLaterEmpty = 1;

  // Создаем ссылки для операций с корзиной
```

```
    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. Создайте в папке `presentation/templates` файл `cart_details.tpl` и поместите в него следующий код.

```
{* 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. Откройте файл `presentation/link.php` и модифицируйте метод `ToCart()` класса `Link` так, как показано ниже.

```
// Создаем ссылки для корзины покупателя
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. Добавьте следующие стили в файл `tshirtshop.css` в папке `styles`.

```
.cart-subtotal td {
  border: none;
  margin: 10px 0px;
  width: 100%;
}

.yui-b div form.cart-form {
  margin: 0;
  padding: 0;
}
```

Мы только что создали код корзины покупателя, который будет работать с посетителями магазина. Можете протестировать его работу, чтобы убедиться, что все в порядке. Попробуйте добавлять товары в корзину, изменять их количество и удалять их из корзины.

Как это работает: корзина покупателя

Операции, которые можно выполнять с корзиной покупателя, обозначаются константами, определенными в файле `include/config.php`. Это константы `ADD_PRODUCT`, `REMOVE_PRODUCT`, `UPDATE_PRODUCTS_QUANTITIES`, `SAVE_PRODUCT_FOR_LATER` и `MOVE_PRODUCT_TO_CART`. Заметьте, что мы не определили специальной константы для просмотра содержимого корзины, поэтому если значение параметра `CartAction` не равно ни одной из этих констант или просто отсутствует, сайт просто отображает содержимое корзины.

Каждая операция с корзиной, кроме просмотра содержимого и обновления, полагается на параметр строки запроса `ItemId` (если этот параметр не задан, генерируется ошибка). Если строка запроса корректна, вызывается соответствующий метод уровня логики приложения.

Администрирование корзины покупателя

Создав работающую корзину покупателя, нам нужно разобраться с еще двумя проблемами. Обе эти проблемы относятся к администрированию сайта. Вот они.

- Как удалить из каталога товар, который есть в существующих корзинах покупателей?

- Как подсчитать или удалить элементы в старых корзинах покупателей? Эта возможность должна присутствовать в административной части сайта, иначе таблица `shopping_cart` в базе данных будет постоянно расти, заполняясь старыми, забытыми корзинами покупателей.

Удаление товаров, находящихся в корзинах покупателей

Страница администрирования товаров позволяет полностью удалять выбранные товары из каталога. Прежде чем удалять товар из каталога, нужно удалить его из всех корзин покупателей.

Обновите хранимую процедуру `catalog_delete_product` в базе данных tshirtshop, выполнив следующие действия.

1. С помощью phpMyAdmin подключитесь к базе данных tshirtshop и выполните операторы SQL, приведенные в следующих шагах. Перед их выполнением не забудьте задать символы `$$` в качестве разделителя.
2. Выполните следующий оператор SQL, чтобы удалить старую версию хранимой процедуры `catalog_delete_product` из базы данных.

```
-- Удаляет старую хранимую процедуру catalog_delete_product
DROP PROCEDURE catalog_delete_product$$
```

3. Выполните следующий оператор SQL, чтобы создать новую версию этой процедуры.

```
-- Создает хранимую процедуру catalog_delete_product
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$$
```

Создание страницы администрирования корзин покупателей

Вторая проблема, связанная с корзинами покупателей, состоит в том, что на данный момент у нас нет никакой возможности удалять из базы данных старые корзины. На сайте с большим количеством посетителей таблица `shopping_cart` может быстро вырасти до огромных размеров.

В текущей версии нашего приложения идентификаторы корзин хранятся на машине клиента в течение семи дней. Соответственно, мы можем предположить, что корзины, не обновлявшиеся в течение десяти дней, можно безбоязненно удалять из базы данных.

В следующем упражнении мы создадим простую страницу для администрирования корзин покупателей, на которой администратор сможет узнать, сколько корзин хранится в таблице `shopping_cart`, и удалять корзины при необходимости. Как будет выглядеть эта страница, показано на рис. 12.5.

Самое интересное здесь — операторы SQL, которые будут удалять корзины, не обновлявшиеся в течение десяти дней или более. Эти операторы не так просты, как вам, вероятно, представляется. Возможно, вы считали, что можно просто удалять все записи из таблицы, в столбце `added_on` которых указана дата более десяти дней назад. Но такой подход не работает с корзинами, к которым посетители возвращаются многократно (скажем, положив первый товар две недели назад, еще два — неделю назад, еще один — три дня назад и т.д.). Если последнее изменение в корзине произошло недавно, мы не имеем права удалять ни один из элементов содержимого

корзины, даже если некоторые из них очень старые. Другими словами, нужно или удалять корзину целиком, или вообще ее не трогать. Возраст корзины в целом определяется по дате последнего добавления или изменения любого товара в ней.

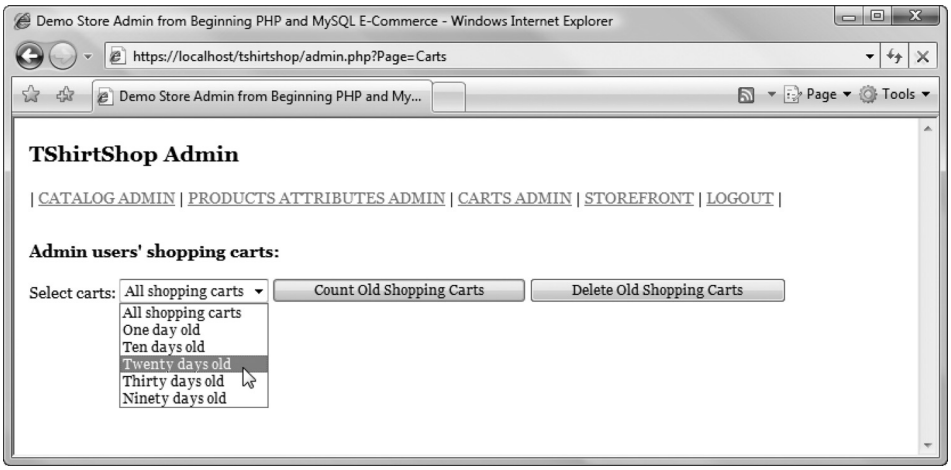


Рис. 12.5. Страница администрирования корзин покупателей

Итак, посмотрим, что нам нужно сделать, чтобы создать страницу администрирования.

Упражнение: создание страницы администрирования корзин покупателей

1. С помощью phpMyAdmin выполните операторы SQL из следующего шага, чтобы создать в базе данных хранимые процедуры для работы с корзинами. Не забудьте задать символы $$ в качестве разделителя.
2. Выполните следующие операторы SQL для создания новых хранимых процедур.

```
-- Создаем хранимую процедуру shopping_cart_count_old_carts
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$$

-- Создаем хранимую процедуру shopping_cart_delete_old_carts
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. Добавьте следующие методы в класс `ShoppingCart` в файле `business/shopping_cart.php`.

```
// Подсчитываем старые корзины в базе данных
public static function CountOldShoppingCarts($days)
{
  // Составляем SQL-запрос
  $sql = 'CALL shopping_cart_count_old_carts(:days)';

  // Создаем массив параметров
  $params = array (':days' => $days);

  // Выполняем запрос и возвращаем результат
  return DatabaseHandler::GetOne($sql, $params);
}

// Удаляем старые корзины
public static function DeleteOldShoppingCarts($days)
{
  // Составляем SQL-запрос
  $sql = 'CALL shopping_cart_delete_old_carts(:days)';

  // Создаем массив параметров
  $params = array (':days' => $days);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}
```

4. Добавьте ссылку на файл `shopping_cart.php` в файле `admin.php`.

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

5. Создайте в папке `presentation` новый файл объекта представления с именем `admin_carts.php` и поместите в него такой код.

```
<?php
// Класс, отвечающий за администрирование корзин покупателей
class AdminCarts
{
  // Public-переменные, доступные в шаблоне smarty
  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-переменные
  public $_mAction = '';

  // Конструктор класса
  public function __construct()
  {
    foreach ($_POST as $key => $value)
      // При щелчке на кнопке отправки...
      if (substr($key, 0, 6) == 'submit')
      {
        // Получаем область действия кнопки
        $this->_mAction = substr($key, strlen('submit_'), strlen($key));
```

```
        // Получаем выбранное количество дней
        if (isset ($_POST['days']))
          $this->mSelectedDaysNumber = (int) $_POST['days'];
        else
          trigger_error('days value not set');
      }

    $this->mLinkToCartsAdmin = Link::ToCartsAdmin();
  }

  public function init()
  {
    // При подсчете корзин...
    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] .
        ').';
    }

    // При удалении корзин...
    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. Создайте в папке `presentation/templates` файл `admin_carts.tpl` и поместите в него такой код.

```
{* admin_carts.tpl *}
{load_presentation_object filename="admin_carts" assign="obj"}
<form action="{$obj->mLinkToCartsAdmin}" method="post">
  <h3>Admin users&#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. Откройте файл `presentation/link.php` и добавьте в конец класса `Link` следующий метод.

```
// Создает ссылку на страницу администрирования корзин
public static function ToCartsAdmin()
{
  return self::ToAdmin('Page=Carts');
}
```

8. Модифицируйте файл `presentation/templates/admin_menu.tpl` — добавьте в него ссылку на страницу администрирования корзин.

```
<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. Откройте файл `presentation/admin_menu.php` и добавьте в него строки, создающие новую ссылку в меню.

```
<?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. Добавьте две новые строки в конец метода `init()` класса `StoreAdmin` в файле `presentation/store_admin.php`. Эти строки обеспечивают загрузку файла `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';
  }
 }
}
?>
```

Как это работает: страница администрирования корзин

Всю основную работу по администрированию корзин выполняют две хранимые процедуры, которые мы создали в этом упражнении: `shopping_cart_count_old_carts` и `shopping_cart_delete_old_carts`. Обе эти процедуры получают в качестве параметра число дней, по которому определяют, какие корзины — старые, а какие — нет.

Возраст корзины определяется возрастом элемента, который добавлялся или изменялся последним. Вот условие, которое определяет, будет ли конкретная корзина считаться старой.

```
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);
```

Два вложенных запроса необходимы, чтобы обойти ограничение MySQL, не позволяющее выбирать данные из таблицы в процессе обновления.

Резюме

Из этой главы вы узнали, как хранить в базе данных сведения о корзинах покупателей и как организовать их обработку в своем приложении. Вероятно, самым интересным моментом было сохранение идентификаторов корзин в cookie-файлах на машинах клиентов — до сих пор в этой книге мы не делали ничего подобного.

Пройдя через процесс создания корзины, начиная от базы данных и заканчивая инструментами администрирования, мы столкнулись с несколькими новыми проблемами.

В главе 13 мы воспользуемся технологией под названием AJAX для улучшения функциональности каталога товаров и корзины. В главе 14 мы добавим к корзине функциональность оплаты покупок и начнем собирать статистическую информацию о покупках. Это позволит реализовать интересные механизмы наподобие динамических рекомендаций товаров, которыми мы займемся в главе 15.

Глава 13

Усовершенствование сайта с помощью AJAX

В этой главе мы улучшим работу корзины покупателя на нашем сайте с помощью технологии, ставшей знаменитой в 2005 году. Она называется AJAX и позволяет делать веб-приложения более удобными в использовании.

Хотя сама по себе AJAX — тема не слишком обширная, по крайней мере по сравнению с некоторыми языками программирования, изучить ее непросто. В этой главе мы продемонстрируем, как AJAX используется в веб-приложениях, но если вас заинтересуют подробности, обратитесь к другим книгам, например *Ajax на практике* (Дейв Крейн, Бер Бибо, Джордон Сонневельд, пер. с англ., ИД "Вильямс", 2008 г.) и *Ajax в действии* (Дейв Крейн, Эрик Паскарелло, Даррен Джеймс, пер. с англ., ИД "Вильямс", 2006).

Эта глава содержит очень сжатое введение в основы AJAX. С помощью AJAX мы

- реализуем возможность добавлять в корзину покупателя новые товары, не загружая заново веб-страницу с помощью кнопок Add to Cart (это позволит обновлять содержимое корзины гораздо быстрее, без замирания приложения на время обновления);
- Уменьшим время реакции корзины на действия посетителя, например изменение количества товаров, их удаление из корзины или сохранение в разделе Save for later.

Вероятно, эти усовершенствования покажутся вам не слишком значительными. Но они улучшат впечатления посетителей от работы с вашим сайтом и позволят посетителям убедиться, что ваш сайт — современный, а не созданный в прошлом веке.

Краткое введение в AJAX

Основное преимущество, которое может дать веб-приложениям AJAX, — уменьшение времени реакции. Время реакции — это время, которое пройдет от момента совершения пользователем какого-то действия в интерфейсе приложения до момента, когда пользователь увидит результат своего действия.

Сам термин "AJAX" появился в 2005 году — его предложил Джесси Джеймс Гарретт (Jesse James Garrett; `http://www.jjg.net/about/`). Этот термин представляет собой аббревиатуру от "Asynchronous JavaScript and XML". Он показывает, что в данной технологии используется язык JavaScript. Приложение, использующее AJAX, генерирует обращения к серверу и получает данные по мере необходимости, при этом веб-страница обновляется по частям, без ее полной загрузки заново. AJAX часто упоминается в связи с Web 2.0 (`http://en.wikipedia.org/wiki/Web_2`) как одна из технологий, позволяющих создавать веб-сайты нового поколения, поддерживающие коллективную работу и обмен информацией.

С технической точки зрения единственная "новая" вещь в AJAX — это название. Все технологии, которые в сумме образуют AJAX, существовали и использовались уже довольно длительное время. Для выполнения асинхронных обращений к серверу (отправления запросов в фоновом режиме) мы используем объект `XMLHttpRequest` языка JavaScript (и еще несколько менее известных элементов). Для обновления частей веб-страницы в браузере пользователя согласно получаемым данным мы используем объектную модель документа (Document Object Model — DOM) в сочетании с каскадными таблицами стилей (Cascading Style Sheets — CSS). А для кодирования информации при ее передаче между клиентом и сервером и обратно мы используем XML или другие форматы данных, например JavaScript Object Notation (JSON). Чуть ниже мы рассмотрим каждый из этих компонентов подробнее, но сначала взгляните на общую схему работы веб-страницы, использующей AJAX. Эта схема показана на рис. 13.1.

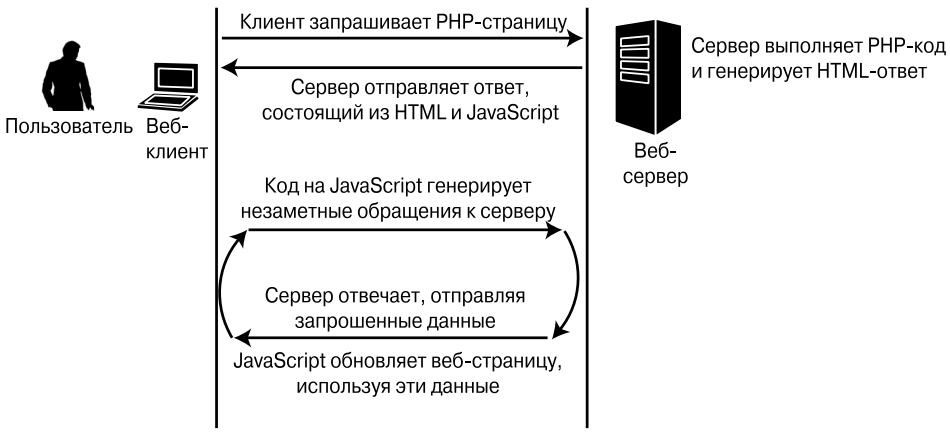


Рис. 13.1. Схема работы AJAX с асинхронными вызовами

JavaScript

JavaScript — язык программирования, поддерживаемый всеми современными веб-браузерами, а потому идеально подходящий для реализации клиентских частей разнообразных приложений. Код на JavaScript можно вставлять прямо в HTML-страницы или вызывать из таких страниц, и он может выполняться при загрузке страницы или в качестве реакции на определенные события (например, щелчок на ссылке). Изначально JavaScript использовался в основном для реализации различных графических эффектов и анимации на веб-страницах, но с течением времени с его помощью стали решать все более серьезные задачи.

Хотя хорошее знание JavaScript вам, определенно, не помешает, в этой главе мы рассмотрим все, что потребуется для улучшения нашего проекта. Для тех, кто желает изучить этот язык более подробно, существует масса учебников, в том числе онлайн-курсы.

- `http://www.echoecho.com/javascript.htm`
- `http://www.webteacher.com/javascript/`
- `http://www.w3schools.com/js/default.asp`
- `http://en.wikipedia.org/wiki/JavaScript`

DOM

DOM (Document Object Model — объектная модель документа) — это API, позволяющий считывать и изменять иерархически структурированные документы, прежде всего HTML и XML. Компоненты DOM доступны во многих языках программирования; в контексте AJAX DOM-интерфейс языка JavaScript позволяет считывать, проверять, изменять и создавать HTML-элементы веб-страниц. Чаще всего, получив ответ сервера на асинхронный запрос, код на JavaScript модифицирует страницу в окне браузера, чтобы уведомить пользователя о результатах запроса.

JavaScript предоставляет доступный по умолчанию объект `document`, являющийся DOM-представлением текущей страницы. Можно, например, добавлять содержимое в элемент `<body>` текущей страницы с помощью метода `document.write()` или получать значения элементов с помощью метода `document.getElementById()`. Например, редактировать содержимое элемента `<div>` с именем `cart-summary` можно с помощью такой строки кода:

```
document.getElementById("cart-summary").innerHTML = response;
```

ВНИМАНИЕ

Существует важный момент, о котором следует помнить, применяя JavaScript для генерации содержимого страницы: клиенты, не поддерживающие JavaScript, просто не увидят содержимого, которое он генерирует. Поэтому важное содержимое страниц не должно генерироваться сценариями на JavaScript. Другими словами, общий смысл страниц должен оставаться понятным, даже если сценарии не работают. В частности, сценарии не выполняются, когда веб-страницы просматриваются поисковыми системами, например, Google, Yahoo или MSN. Если эти поисковые системы не увидят содержимого ваших страниц, поскольку не выполнят сценарии, рейтинги этих страниц вряд ли будут высокими.

Если вы хотите узнать больше о DOM, осмелимся порекомендовать следующие ресурсы.

- `http://www.quirksmode.org/dom/intro.html` (вводный учебный курс по DOM)
- `http://www.javascriptkit.com/javatutors/dom.shtml` (еще один учебный курс)
- `http://www.topxml.com/learning/games/b/default.asp` (довольно интересная и полезная игра с DOM)
- `http://krook.org/jsdom/` (справочник по DOM)

Объект `XMLHttpRequest`

`XMLHttpRequest` — это объект, позволяющий использовать асинхронные запросы к серверу в коде на JavaScript. Это позволяет инициировать HTTP-запросы и по-

лучать ответы от сервера в фоновом режиме, без отправки страницы на сервер целиком. Обычно при получении ответа от сервера отображаемое содержимое страницы изменяется с помощью DOM, и таким образом реализуются интерактивная функциональность и визуальные эффекты на основе свежих данных с сервера. Именно для этого и предназначен AJAX.

`XMLHttpRequest` был реализован Microsoft в 1999 году как объект ActiveX в Internet Explorer и постепенно стал де-факто стандартом для всех браузеров — встроенная поддержка для этого объекта есть во всех современных браузерах, кроме Internet Explorer версии 6.

Давайте посмотрим, как `XMLHttpRequest` работает. Вам нужно знать, как создавать этот объект, инициализировать его с требуемыми параметрами, инициировать запрос к серверу и получать результаты запроса.

Создание объектов XMLHttpRequest

В Internet Explorer версии 6 и более старых версиях объект `XMLHttpRequest` реализуется в виде элемента управления ActiveX и создается следующим оператором:

```
xmlHttp = new ActiveXObject("Microsoft.XMLHttp");
```

В Internet Explorer 7 и других веб-браузерах, включая Firefox, Opera и Safari, `XMLHttpRequest` является встроенным объектом и создается так:

```
xmlHttp = new XMLHttpRequest();
```

Код для создания объектов `XMLHttpRequest`, работающий во всех браузерах, мы оформим в виде функции на JavaScript. Эта функция создает экземпляр встроенного объекта, если он доступен, или экземпляр элемента управления ActiveX `Microsoft.XmlHttpRequest` в Internet Explorer 6 или более старых версий.

```
// Создает экземпляр XMLHttpRequest
function createXmlHttpRequestObject()
{
  // В xmlHttp будет храниться ссылка на объект XMLHttpRequest
  var xmlHttp;
  // Пытаемся создать экземпляр встроенного объекта XMLHttpRequest
  try
  {
    // Создаем объект XMLHttpRequest
    xmlHttp = new XMLHttpRequest();
  }
  catch(e)
  {
    // Предполагаем, что используется IE6 или более старый
    try
    {
      xmlHttp = new ActiveXObject("Microsoft.XMLHttp");
    }
    catch(e) { }
  }
  // Возвращаем ссылку на созданный объект или
  // выводим сообщение об ошибке
  if (!xmlHttp)
    alert("Error creating the XMLHttpRequest object.");
  else
    return xmlHttp;
}
```

ЗАМЕЧАНИЕ

`Microsoft.XMLHttp` — это самая старая версия библиотеки ActiveX `XMLHttp`, но существует больше версий и разновидностей этой библиотеки, чем вы можете себе представить. Каждый товар Microsoft, включая Internet Explorer и MDAC, поставлялся со своей версией этого элемента управления ActiveX, со своим собственным именем. В нашем коде мы применим механизм, который будет автоматически обнаруживать самую свежую версию `XMLHttp`, установленную на клиентском компьютере, но только если на этом компьютере установлен Internet Explorer 6 или более старая версия этого браузера. В противном случае мы будем просто использовать встроенный объект `XMLHttpRequest`.

Функция `createXmlHttpRequestObject()` использует конструкцию языка JavaScript `try-catch`. Это удобный и эффективный механизм обработки исключений, первоначально появившийся в объектно-ориентированном программировании, а сейчас доступный и в PHP 5. *Исключение* (exception) — это объект, представляющий ошибку, возникшую во время выполнения кода. Исключения могут перехватываться и обрабатываться и при успешной обработке не блокируют выполнение программы, в которой они возникают.

ЗАМЕЧАНИЕ

Исключение распространяется вверх по стеку вызовов из точки своего возникновения вплоть до среды, в которой выполняется программа. Стек вызовов — это последовательность методов, вызывающих друг друга в процессе работы программы. Поэтому, если метод `A()` вызывает метод `B()`, который, в свою очередь, вызывает метод `C()`, стек будет содержать три метода. Если исключение возникнет в методе `C()`, его можно перехватить и обработать конструкцией `try-catch` в этом методе, и оно не выйдет за пределы метода. Если исключение не будет перехвачено и обработано в методе, оно распространится в метод `B()` и далее по стеку. Последний слой в стеке — веб-браузер клиента, который может обрадовать пользователя невразумительным сообщением о неперехваченном исключении в программе.

С помощью конструкции `try-catch` можно перехватывать исключения и обрабатывать их локально, не позволяя им распространяться до браузера клиента. Вот синтаксис этой конструкции.

```
try
{
  // Код, в котором может возникнуть исключение
}
catch (e)
{
  // Этот блок выполняется, если в блоке try возникает исключение
  // (сведения об исключении доступны в параметре e)
}
```

Если при выполнении кода в блоке `try` возникает исключение (ошибка), управление немедленно передается блоку `catch`. Если исключений в блоке `try` не возникает, код в блоке `catch` вообще не выполняется.

Как нужно обрабатывать конкретные исключения, зависит от ситуации. Иногда ошибки можно просто игнорировать, иногда можно выставлять в коде какие-нибудь флаги или выводить сообщения об ошибках, которые сможет прочитать пользователь. В нашем случае мы сначала пытаемся создать объект `XMLHttpRequest` как встроенный объект браузера.

```
// Пытаемся создать экземпляр встроенного объекта XMLHttpRequest
try
{
```

```
  // Создаем объект XMLHttpRequest
  xmlHttp = new XMLHttpRequest();
}
```

Internet Explorer 7, Firefox, Opera, Safari и другие браузеры выполнят этот фрагмент кода без проблем, и никаких ошибок при этом не возникнет, поскольку в них `XMLHttpRequest` является встроенным объектом. Однако в Internet Explorer 6 и более старых версиях этого браузера `XMLHttpRequest` не является встроенным объектом, и начнет выполняться код в блоке `catch`. В этом блоке мы создаем объект `XMLHttpRequest` как элемент управления ActiveX.

```
catch(e)
{
  // Предполагаем, что используется IE6 или более старый
  try
  {
    xmlHttp = new ActiveXObject("Microsoft.XMLHttp");
  }
  catch(e) { }
}
```

Есть и другие способы создания объектов `XMLHttpRequest`. Например, можно воспользоваться возможностью JavaScript под названием *обнаружение объектов* (object detection). Эта возможность позволяет проверить, поддерживается ли заданный объект браузером, и работает следующим образом.

```
if (window.XMLHttpRequest)
{
  xmlHttp = new XMLHttpRequest();
}
```

В конце функции `createXmlHttpRequest()` мы проверяем, указывает ли `xmlHttp` на корректный экземпляр `XMLHttpRequest`.

```
// Возвращаем ссылку на созданный объект или
// выводим сообщение об ошибке
if (!xmlHttp)
  alert("Error creating the XMLHttpRequest object.");
else
  return xmlHttp;
```

Здесь мы воспользовались обратным вариантом обнаружения объекта — Java Script возвращает значение `true` для ссылки на существующий корректный объект, например, `xmlHttp`. Соответственно, `!xmlHttp` будет возвращать `true` для несуществующего или некорректного объекта (в том числе переменной, которой не присвоено значение).

Использование XMLHttpRequest

Создав объект `XMLHttpRequest`, мы можем использовать его для выполнения асинхронных запросов к серверу. В табл. 13.1 перечислены `public`-свойства и методы этого объекта.

Таблица 13.1. `Public`-свойства и методы объектов `XMLHttpRequest`

Метод/свойство	Назначение
`abort`	Останавливает текущий запрос
`getAllResponseHeaders()`	Возвращает все заголовки запроса в виде строки

Окончание табл. 13.1

Метод/свойство	Назначение
`getResponseHeader("headerLabel")`	Возвращает один заголовок запроса в виде строки
`open("method", "URL"[, asyncFlag[, "userName"[, "password"]]])`	Инициализирует параметры запроса
`send(content)`	Выполняет HTTP-запрос
`setRequestHeader("label","value")`	Задает заголовок HTTP-запроса
`onreadystatechange`	Задает функцию — обработчик изменений состояния запроса
`readyState`	Возвращает состояние запроса: `0` — не инициализирован `1` — загружается `2` — загружен `3` — в интерактивном состоянии `4` — завершен
`responseText`	Возвращает ответ сервера в виде строки
`responseXml`	Возвращает ответ сервера в виде XML-документа, с которым можно работать с помощью методов DOM
`status`	Возвращает состояние запроса
`statusText`	Возвращает сообщение о состоянии запроса

Для каждого запроса мы будем использовать методы `open()` и `send()`. Метод `open()` конфигурирует запрос, *не отправляя его*. У этого метода есть два обязательных параметра и несколько необязательных. В первом параметре указывается способ отправки данных на сервер — обычно ему присваивается значение `GET` или `POST`. Второй параметр — URL, по которому отправляется запрос. URL может быть как полным (абсолютным), так и относительным. Если в URL не указан ресурс, доступный по протоколу HTTP, первый параметр игнорируется.

Третий параметр метода `open()`, который называется `async`, указывает, должен ли запрос обрабатываться асинхронно: значение `true` указывает, что код продолжает выполняться после метода `send()` без ожидания ответа от сервера; значение `false` заставляет сценарий ожидать ответа от сервера, прежде чем выполняться дальше. Чтобы разрешить асинхронные запросы, лежащие в основе AJAX, нужно присвоить параметру `async` значение `true` и обрабатывать события `onreadystatechange`, сообщающие о прогрессе в обработке запроса на сервере.

Сконфигурировав запрос методом `open()`, можно отправить его серверу с помощью метода `send()`.При использовании запросов `GET` параметры передаются в строке запроса, например, `http://localhost/ajax/test.php?param1=x&param2=y`. Такой запрос передает серверу два параметра — `param1` со значением `x` и `param2` со значением `y`.

```
// Вызываем сервер для выполнения серверной операции
xmlHttp.open("GET", "http://localhost/ajax/test.php?param1=x&param2=y",
true);
xmlHttp.onreadystatechange = handleRequestStateChange;
xmlHttp.send(null);
```

При использовании запросов POST строка запроса задается как параметр метода send() вместо ее присоединения к URL запроса, например, так.

```
// Вызываем сервер для выполнения серверной операции
xmlHttp.open("POST", "http://localhost/ajax/test.php", true);
xmlHttp.onreadystatechange = handleRequestStateChange;
xmlHttp.send("param1=x&param2=y");
```

Эти два фрагмента кода дают одинаковые результаты. С практической точки зрения различия между запросами GET и POST следующие.

- GET позволяет легче отлаживать программы, поскольку запросы GET можно имитировать с помощью обычного браузера и видеть, что генерирует сервер в ответ на эти запросы.
- Запросы POST — единственный эффективный способ передачи на сервер больших блоков данных, которые нельзя поместить в строку запроса при использовании GET.
- Запросы GET предназначаются для запроса с сервера каких-либо данных, а POST — для сообщения об изменениях. Например, поисковые системы отправляют запросы GET для получения данных из сети, но никогда не используют запросы POST. Если вы воспользуетесь запросом GET для отправки данных на сервер и поисковая система узнает адрес серверного сценария, она сможет начать модифицировать ваши данные. Вряд ли вам это понравится.

Вот минимальный вариант функции process(), выполняющей асинхронные запросы с помощью GET.

```
function process()
{
  // Вызываем сервер для выполнения серверной операции
  xmlHttp.open("GET", "server_script.php", true);
  xmlHttp.onreadystatechange = handleRequestStateChange;
  xmlHttp.send(null);
}
```

У такой реализации есть следующие недостатки.

- Функция process() может выполняться, даже если в xmlHttp нет ссылки на корректный экземпляр XMLHttpRequest. Это может произойти, например, если браузер пользователя не поддерживает объектов XMLHttpRequest. В таком случае будет сгенерировано исключение, механизма перехвата для которого в этой реализации нет. Другие наши усилия по обработке исключений не очень нам помогут, если мы не оснастим и эту функцию нормальной обработкой исключений.
- Эта функция не защищена от других ошибок, которые могут возникнуть. Например, как вы увидите позже в этой главе, некоторые браузеры генерируют исключения, если им не нравится сервер, с которым вы пытаетесь работать через объект XMLHttpRequest.

Более эффективной версией process() будет следующая.

```
// Выполняет запрос к серверу и назначает функцию-обработчик
function process()
{
  // Работаем, только если xmlHttp - корректный объект
  if (xmlHttp)
  {
    // Пытаемся связаться с сервером
    try
```

```
    {
      // Инициируем запрос
      xmlHttp.open("GET", "server_script.php", true);
      xmlHttp.onreadystatechange = handleRequestStateChange;
      xmlHttp.send(null);
    }
    // Сообщаем пользователю об ошибке, если произошел сбой
    catch (e)
    {
      alert("Can't connect to server:\n" + e.toString());
    }
  }
}
```

Если `xmlHttp` равно `null` (или `false`), функция не отображает еще одно сообщение об ошибке — мы предполагаем, что сообщение о ней уже вывела функция `createXmlHttpRequestObject()`. Но мы выведем сообщение, если возникнут проблемы при обращении к серверу.

Обработка ответа сервера

При выполнении асинхронного запроса в коде, например в таком, как приведенный выше, выполнение не замирает после отправки запроса с помощью `xmlHttp.send()`; вместо этого код продолжает выполняться. Функция `process()`, показанная выше, определяет функцию `handleRequestStateChange()` как обработчик изменения состояния запроса.

Свойство `readyState` может принимать одно из следующих значений, соответствующих стадиям выполнения запроса.

- `0` — не инициализирован
- `1` — загружается
- `2` — загрузился
- `3` — интерактивен
- `4` — завершен

Обычно метод-обработчик (в нашем случае — `handleRequestStateChange()`) вызывается многократно — каждый раз, когда выполнение запроса входит в новую стадию. Однако не стоит полагаться на веб-браузер в генерации событий `readystatechange` для всех состояний запроса, кроме состояния `4` (выполнение запроса завершено).

Смысл состояний, думаю, понятен из их названий, за исключением состояния `3`. Интерактивное состояние запроса — это промежуточное состояние, когда ответ на него принят частично. В наших AJAX-приложениях мы будем пользоваться только состоянием `4`, означающим, что ответ с сервера получен полностью.

В следующем фрагменте кода показан типичный вариант реализации `handleRequestStateChange()`. Полужирным шрифтом выделено место, непосредственно отвечающее за чтение ответа сервера.

```
// Функция, выполняемая при изменении состояния запроса
function handleRequestStateChange()
{
  // Продолжаем выполнение, только если процесс завершен
  if (xmlHttp.readyState == 4)
  {
    // Продолжаем, только если HTTP-код "OK"
    if (xmlHttp.status == 200)
    {
```

```
      // Считываем ответ
      response = xmlHttp.responseText;
      // Что-то делаем с ответом
      // ...
    }
  }
}
```

Прежде чем пытаться прочитать ответ сервера, этот код проверяет код состояния ответа. Если это код 200 (предусмотренный стандартом HTTP код, означающий успешное выполнение запроса), значит, можно обрабатывать ответ. Есть и другие коды состояния, например часто используются код 404 (требуемый ресурс не найден) и 500 (внутренняя ошибка сервера).

Для обработки ошибок, возникающих при соединении с сервером и получении ответа, тоже можно применять конструкции `try-catch`. Улучшенная версия `handleRequestStateChange()` будет выглядеть так.

```
// Функция, выполняемая при изменении состояния запроса
function handleRequestStateChange()
{
  // Продолжаем выполнение, только если процесс завершен
  if (xmlHttp.readyState == 4)
  {
    // Продолжаем, только получив HTTP-код состояния "OK"
    if (xmlHttp.status == 200)
    {
      try
      {
        // Получаем ответ
        response = xmlHttp.responseText;
        // Что-то делаем с ответом
        // ...
      }
      catch(e)
      {
        // Отображаем сообщение о состоянии запроса
        alert("Error reading the response: " + e.toString());
      }
    }
    else
    {
      // Отображаем сообщение о состоянии запроса
      alert("There was a problem retrieving the data:\n" +
xmlHttp.statusText);
    }
  }
}
```

На практике, разумеется, стоит выводить более внятные сообщения об ошибках. Кроме того, нужно писать приложение так, чтобы оно оставалось работоспособным, даже если в коде на JavaScript возникнет ошибка. Давайте посмотрим, как это можно сделать.

Написание устойчивого к сбоям кода

В идеальном случае современное веб-приложение должно использовать AJAX везде, где это позволяет улучшить его реакцию на действия пользователя, но в то же время сайт должен оставаться работоспособным, даже если машина пользователя вообще не поддерживает JavaScript. В нашем случае мы хотим, чтобы у поль-

зователей была возможность работать с корзинами, даже если в их браузерах нет поддержки JavaScript.

Когда пользователь щелкает на кнопке Add to Cart, мы сначала пытаемся выполнить фоновое обращение к серверу с помощью AJAX, чтобы обновить содержимое корзины на сервере. Если при этом обращении возникает ошибка или если Java Script недоступен, мы отправляем на сервер форму и добавляем товар в корзину без использования AJAX. Такой подход позволит улучшить впечатления от работы с сайтом для большинства пользователей и одновременно пользоваться сайтом даже пользователям, браузеры которых не поддерживают JavaScript.

Вероятно, вы удивитесь, узнав это, но на самом деле такого результата несложно добиться. Вот примеры использующих AJAX ссылок Save for later и Remove для корзины.

```
<a href="{$obj->mCartProducts[i].save}"
    onclick="return executeCartAction(this);">Save for later</a>
<a href="{$obj->mCartProducts[i].remove}"
    onclick="return executeCartAction(this);">Remove</a>
```

В обоих ссылках при возникновении события `click` выполняется функция JavaScript. Если она возвращает значение `false`, переход по ссылке не происходит. Если функция возвращает `true` или вообще не выполняется, то происходит переход по ссылке, и содержимое корзины изменяется без использования AJAX.

Кнопка Add to Cart реализована аналогично ссылкам, только в этом случае функция JavaScript вызывается при отправке формы:

```
onsubmit="return addProductToCart(this);">
```

На рис. 13.2 показано, что происходит, когда пользователь щелкает на кнопке Add to Cart. Процесс в целом аналогичен обновлению содержимого корзины, которое мы также реализуем в этой главе. Как видно из рисунка, конечный результат будет одинаковым независимо от того, используется JavaScript или нет. Разница — в скорости реакции приложения; при использовании AJAX пользователю не придется ждать несколько секунд, пока будет обновляться страница.

Всегда ли полезен AJAX

Теоретическую часть этой главы мы закончим предупреждением: AJAX может не только улучшить впечатления пользователей от работы с вашим веб-сайтом, но и ухудшить их, если использовать его неправильно. Обычно AJAX лучше использовать вместе с традиционными платформами разработки веб-сайтов, а не вместо этих платформ.

Например, если только к вашему приложению не предъявляются особые требования, лучше всего позволить пользователям перемещаться по страницам с помощью обычных гиперссылок. Веб-браузеры изначально создавались именно для такой навигации, и пользователи к ней привыкли.

AJAX может дать вашим приложениям следующие преимущества.

- Позволяет создавать приложения, быстрее реагирующие на действия пользователя.
- Поощряет создание структур, помогающих разработчикам не изобретать уже существующие компоненты и механизмы для решения стандартных задач.
- Использует существующие технологии и возможности, поддерживаемые всеми современными веб-браузерами.
- Задействует многие из навыков, которыми обладают разработчики.

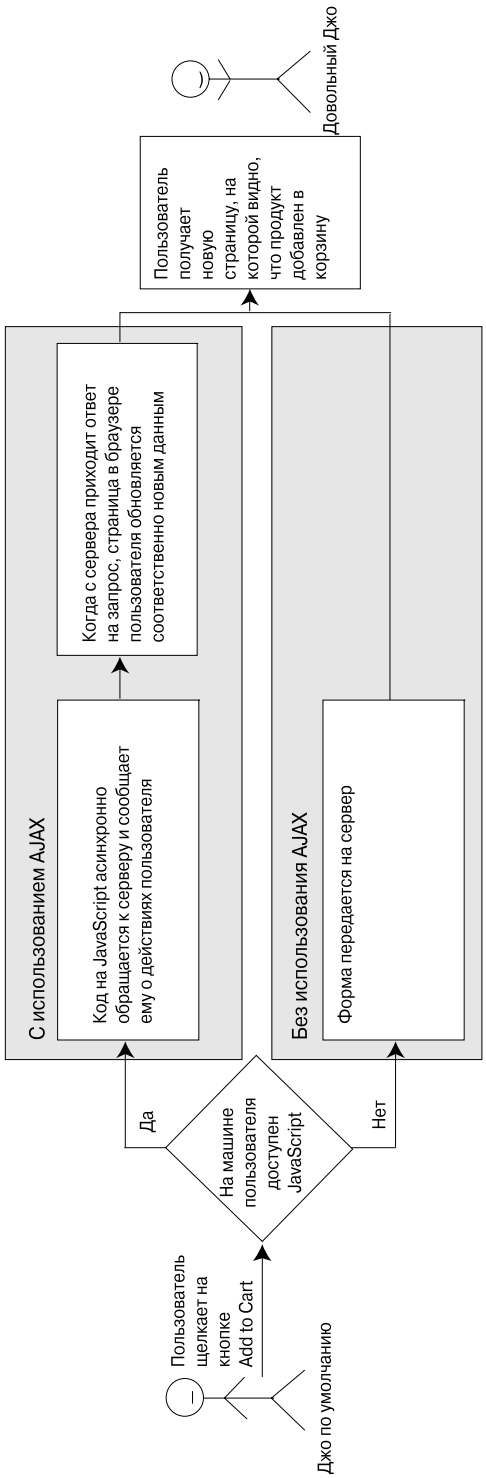


Рис. 13.2. Схема приложения, устойчивого к сбоям AJAX

А вот проблемы, которые могут возникнуть при использовании AJAX.

- *Доступен не всем пользователям.* Поддержку JavaScript в браузере можно отключить, и AJAX перестанет работать.
- *Легко может применяться там, где он не нужен.* Только опыт в разработке веб-приложений и знание того, для чего вы разрабатываете каждое конкретное приложение, позволят вам решить, нужно ли использовать AJAX.
- *Может сделать содержимое сайта невидимым для поисковых систем.* Поисковые системы не индексируют содержимое, которое генерируется динамически с помощью JavaScript, поскольку они не выполняют сценарии на JavaScript, просматривая веб-страницы. Если оптимизация под поисковые системы для вас важна, не используйте AJAX для генерации содержимого страниц, которое должно индексироваться. В нашем случае поисковые системы не должны обращать внимания на корзины покупателей, так что это ограничение мы можем игнорировать.
- *Усложняет навигацию и создание закладок.* Обычно AJAX-приложения работают внутри страниц, URL которых не изменяется в ответ на действия пользователя, и в этом случае закладку можно создать только на этот URL. Соответственно, кнопки возврата на предыдущую или следующую страницу могут работать непредсказуемо, если только ваше AJAX-приложение не предусматривает обеспечения их корректной работы. Чтобы добиться этого, можно воспользоваться, например, платформой Really Simple History (`http://codingparadise.org/projects/dhtml_history/README.html`). Она позволяет создавать закладки, динамически добавляя метки с помощью кода на Java Script, например, в виде `http://www.example.com/my-ajax-app.html#Page2`. Кроме того, вам придется написать код, который будет сохранять и загружать состояния приложения по параметру метки. Но это сильно усложнит приложение и сделает его менее стабильным в работе.

Итак, вы предупреждены о потенциальных проблемах. Давайте посмотрим, как применить AJAX в TShirtShop.

Создание корзины покупателя с применением AJAX

Реализацию корзин покупателя с помощью AJAX мы разделим на две стадии. Сначала мы реализуем поддержку AJAX для кнопок Add to Cart, обновляющих содержимое корзины без загрузки заново всей страницы в браузере. После этого мы займемся всеми остальными элементами корзины.

На рис. 13.3 показано поле содержимого корзины на странице сайта в момент обновления. Как видите, в этом поле отображается надпись Updating..., сообщающая пользователю о том, что в данный момент содержимое корзины обновляется. На рис. 13.4 показана корзина покупателя во время удаления товара.

Применение AJAX для добавления товаров в корзину

На рис. 13.2 вы уже видели, что мы собираемся создать. Сама по себе реализация сложностей не представляет. Мы модифицируем шаблоны `product.tpl` и `products_list.tpl` так, чтобы после щелчка на кнопке Add to Cart вызывалась функция JavaScript `addProductToCart()`.

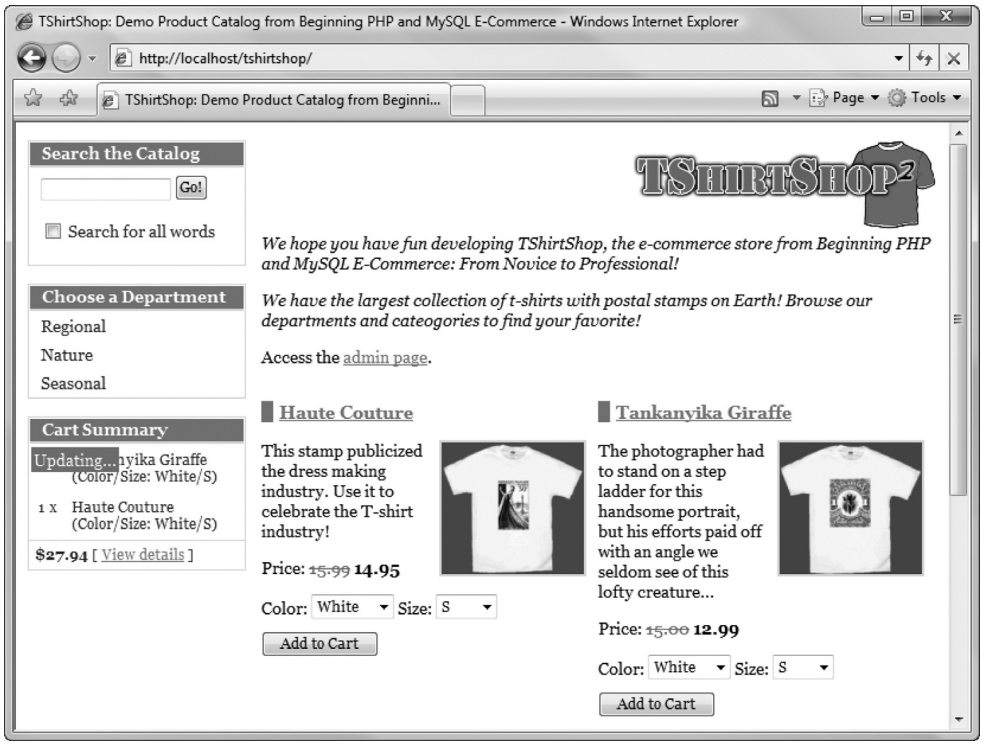


Рис. 13.3. Добавление товара в корзину

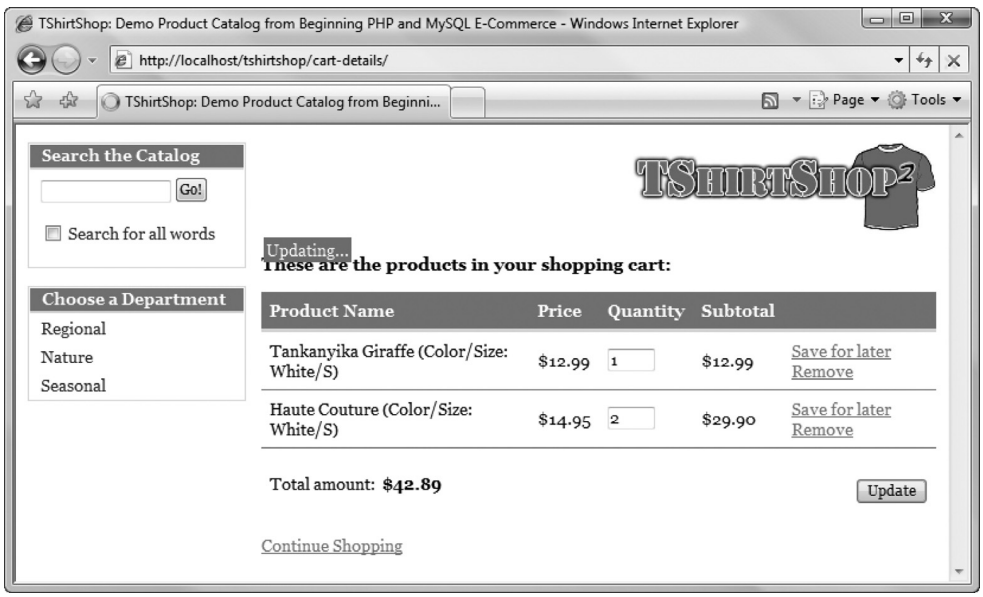


Рис. 13.4. Удаление товара из корзины

После этого мы создадим ajax.js — довольно объемный файл JavaScript, в котором будет содержаться весь код, нужный для поддержки новой функциональности. В частности, в этом файле будут храниться следующие функции.

- createXmlHttpRequest(), создающая объект XMLHttpRequest.
- handleError(message), отображающая сообщения об ошибках, генерируемые другими функциями, и вызывающая механизмы, не использующие AJAX при определенном значении переменной showErrors.
- addProductToCart(), вызываемая из HTML-страниц для добавления товара в корзину.
- handleAddProductToCart() — обработчик событий, генерируемых при обработке запроса.
- postAddProductToCartProcess(), считывающая ответ сервера на запрос и обновляющая поле содержимого корзины на страницах.

Кроме того, мы внесем в свой проект ряд мелких изменений, позволяющих использовать или поддерживать новые возможности. Выполните следующее упражнение, а затем мы обсудим все более подробно.

Упражнение: ускорение добавления товаров в корзину

1. Откройте файл presentation/templates/product.tpl и добавьте в него код, выделенный полужирным шрифтом в следующем листинге. Кроме того, замените имя поля ввода на add_to_cart.

```
{* Форма Add to Cart *}
<form class="add-product-form" target="_self" method="post"
  action="{$obj->mProduct.link_to_add_product}"
  onsubmit="return addProductToCart(this);">

{* Генерируем список значений атрибутов *}
<p class="attributes">

{* Просматриваем список атрибутов и их значений *}
{section name=k loop=$obj->mProduct.attributes}

...

{/section}
</p>

{* Добавляем кнопку для отправки формы *}
<p>
  <input type="submit" name="add_to_cart" value="Add to Cart" />
</p>
</form>
```

2. Откройте файл presentation/templates/products_list.tpl и внесите в него изменения, выделенные полужирным шрифтом в следующем листинге.

```
{* Форма Add to Cart *}
<form class="add-product-form" target="_self" method="post"
  action="{$obj->mProducts[k].link_to_add_product}"
  onsubmit="return addProductToCart(this);">

{* Генерируем список значений атрибутов *}
<p class="attributes">
```

```
{* Просматриваем список атрибутов и их значений *}
{section name=l loop=$obj->mProducts[k].attributes}

...

{/section}
</p>

{* Добавляем кнопку для отправки формы *}
<p>
  <input type="submit" name="add_to_cart" value="Add to Cart" />
</p>
</form>
```

3. Создайте в корневой папке проекта (папке `tshirtshop`) папку `scripts`, добавьте в нее файл с именем `ajax.js` и поместите в этот файл такой код.

```
// Хранит экземпляр объекта XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();

// Отображаем сообщения об ошибках (если true) или отказываемся
// от использования AJAX (если false)
var showErrors = true;

// Содержит ссылку или форму, отправленную посетителем на сервер
var actionObject = '';

// Создает экземпляр XMLHttpRequest
function createXmlHttpRequestObject()
{
  // Будет хранить объект XMLHttpRequest
  var xmlHttp;

  // Создаем объект XMLHttpRequest
  try
  {
    // Пытаемся создать встроенный объект XMLHttpRequest
    xmlHttp = new XMLHttpRequest();
  }
  catch(e)
  {
    // Если используется IE6 или более старый браузер
    var XmlHttpVersions = new Array(
      "MSXML2.XMLHTTP.6.0", "MSXML2.XMLHTTP.5.0", "MSXML2.XMLHTTP.4.0",
      "MSXML2.XMLHTTP.3.0", "MSXML2.XMLHTTP", "Microsoft.XMLHTTP");

    // Перебираем идентификаторы, пока один из них не сработает
    for (i = 0; i < XmlHttpVersions.length && !xmlHttp; i++)
    {
      try
      {
        // Пытаемся создать объект XMLHttpRequest
        xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
      }
      catch (e) {} // Игнорируем потенциальные ошибки
    }
  }

  // Если объект XMLHttpRequest успешно создан, возвращаем его
  if (xmlHttp)
```

```
  {
    return xmlHttp;
  }
  // Если произошла ошибка, передаем ее handleError
  else
  {
    handleError("Error creating the XMLHttpRequest object.");
  }
}

// Отображает сообщение об ошибке или отказывается от использования AJAX
function handleError($message)
{
  // Игнорирует ошибки, если значение showErrors равно false
  if (showErrors)
  {
    // Отображаем сообщение об ошибке
    alert("Error encountered: \n" + $message);
    return false;
  }
  // Отказываемся от использования AJAX
  else if (!actionObject.tagName)
  {
    return true;
  }
  // Отказываемся от использования AJAX и выполняем переход по ссылке
  else if (actionObject.tagName == 'A')
  {
    window.location = actionObject.href;
  }
  // Отказываемся от использования AJAX и отправляем на сервер форму
  else if (actionObject.tagName == 'FORM')
  {
    actionObject.submit();
  }
}

// Добавляет товар в корзину
function addProductToCart(form)
{
  // Отображаем сообщение "Updating"
  document.getElementById('updating').style.visibility = 'visible';

  // Переходим к классической отправке форм, если XMLHttpRequest недосту-
пен
  if (!xmlHttp) return true;

  // Создаем URL для асинхронного открытия
  request = form.action + '&AjaxRequest';
  params = '';

  // Получаем выбранные атрибуты
  formSelects = form.getElementsByTagName('SELECT');
  if (formSelects)
  {
    for (i = 0; i < formSelects.length; i++)
    {
      params += '&' + formSelects[i].name + '=';
      selected_index = formSelects[i].selectedIndex;
      params += encodeURIComponent(formSelects[i][selected_index].text);
    }
```

```
  }

  // Пытаемся связаться с сервером
  try
  {
    // Продолжаем, только если объект XMLHttpRequest свободен
    if (xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
    {
      // Делаем запрос к серверу для проверки извлеченных данных
      xmlHttp.open("POST", request, true);
      xmlHttp.setRequestHeader("Content-Type",
        "application/x-www-form-urlencoded");
      xmlHttp.onreadystatechange = addToCartStateChange;
      xmlHttp.send(params);
    }
  }
  catch (e)
  {
    // Обрабатываем ошибку
    handleError(e.toString());
  }

  // Останавливаем классическую отправку формы, если AJAX успешно отрабо-
тал
  return false;
}

// Функция, получающая HTTP-ответ
function addToCartStateChange()
{
  // Когда значение readyState равно 4, мы считываем ответ сервера
  if (xmlHttp.readyState == 4)
  {
    // Продолжаем, только если HTTP-код состояния означает "OK"
    if (xmlHttp.status == 200)
    {
      try
      {
        updateCartSummary();
      }
      catch (e)
      {
        handleError(e.toString());
      }
    }
    else
    {
      handleError(xmlHttp.statusText);
    }
  }
}

// Обрабатываем ответ сервера
function updateCartSummary()
{
  // Считываем ответ
  response = xmlHttp.responseText;
  // Ошибка сервера?
  if (response.indexOf("ERRNO") >= 0 || response.indexOf("error") >= 0)
  {
    handleError(response);
```

```
  }
  else
  {
    // Извлекаем содержимое элемента div cart_summary
    var cartSummaryRegEx = /^<div class="box" id="cart-summary">?_
   ([\s\S]*)<\/div>$/m;
    matches = cartSummaryRegEx.exec(response);
    response = matches[1];

    // Обновляем поле содержимого корзины и убираем сообщение об обновле-
нии
    document.getElementById("cart-summary").innerHTML = response;

    // Убираем сообщение "Updating..."
    document.getElementById('updating').style.visibility = 'hidden';
  }
}
```

4. Откройте файл `presentation/templates/store_front.tpl` и добавьте в него ссылку на файл `ajax.js`.

```
<html>
  <head>
    <title>{$obj->mPageTitle}</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <link href="{$obj->mSiteUrl}tshirtshop.css" type="text/css"
     rel="stylesheet" />
    <script type="text/javascript"
     src="{$obj->mSiteUrl}scripts/ajax.js"></script>
  </head>
```

5. Модифицируйте файл `index.php` так, как показано в следующем листинге.

```
// Загружаем уровень логики приложения
require_once BUSINESS_DIR . 'catalog.php';
require_once BUSINESS_DIR . 'shopping_cart.php';

// Коррекция URL
Link::CheckRequest();

// Обработка AJAX-запросов
if (isset ($_GET['AjaxRequest']))
{
  // Заголовки отправляются, чтобы предотвратить кэширование в браузерах
  header('Expires: Fri, 25 Dec 1980 00:00:00 GMT'); // Устаревшее время
  header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
  header('Cache-Control: no-cache, must-revalidate');
  header('Pragma: no-cache');
  header('Content-Type: text/html');

  if (isset ($_GET['CartAction']))
  {
    $cart_action = $_GET['CartAction'];

    if ($cart_action == ADD_PRODUCT)
    {
      require_once PRESENTATION_DIR . 'cart_details.php';

      $cart_details = new CartDetails();
      $cart_details->init();
```

```
      $application->display('cart_summary.tpl');
    }
  }
  else
    trigger_error('CartAction not set', E_USER_ERROR);
}
else
{
  // Отображаем сообщение
  $application->display('store_front.tpl');
}

// Закрываем соединение с базой данных
DatabaseHandler::Close();
```

6. Откройте файл `presentation/cart_details.php` и внесите следующие изменения в блок `switch` метода `init()`.

```
    if (count($selected_attributes) > 0)
      $attributes = implode('/', $selected_attributes) . ': ' .
                    implode('/', $selected_attribute_values);

    ShoppingCart::AddProduct($this->_mItemId, $attributes);

    if (!isset ($_GET['AjaxRequest']))
      header('Location: ' . $this->mLinkToContinueShopping);
    else
      return;

    break;
  case REMOVE_PRODUCT:
    ShoppingCart::RemoveProduct($this->_mItemId);

    if (!isset ($_GET['AjaxRequest']))
      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]);

    if (!isset ($_GET['AjaxRequest']))
      header('Location: ' . Link::ToCart());

    break;
  case SAVE_PRODUCT_FOR_LATER:
    ShoppingCart::SaveProductForLater($this->_mItemId);

    if (!isset ($_GET['AjaxRequest']))
      header('Location: ' . Link::ToCart());

    break;
  case MOVE_PRODUCT_TO_CART:
    ShoppingCart::MoveProductToCart($this->_mItemId);
    if (!isset ($_GET['AjaxRequest']))
      header('Location: ' . Link::ToCart());

    break;
  default:
    // Не делаем ничего
    break;
}
```

7. Модифицируйте метод `CheckRequest()` класса `Link` в файле `link.php` так, как показано ниже. Нам не нужно проверять ссылку при ответе на запрос AJAX.

```
public static function CheckRequest()
{
  $proper_url = '';

  if (isset ($_GET['Search']) || isset($_GET['SearchResults']) ||
      isset ($_GET['CartAction']) || isset ($_GET['AjaxRequest']))
  {
    return ;
  }
```

8. Откройте файл `presentation/templates/cart_summary.tpl` и внесите в него изменения, показанные ниже.

```
{* cart_summary.tpl *}
{load_presentation_object filename="cart_summary" assign="obj"}
{* Start cart summary *}
<div class="box" id="cart-summary">
  <p class="box-title">Cart Summary</p>
  <div id="updating">Updating...</div>
{if $obj->mEmptyCart}
  <p class="empty-cart">Your shopping cart is empty!</p>
{else}
```

9. Добавьте в файл `tshirtshop.css` в папке `styles` определение нового стиля.

```
#updating {
  background-color: #ff0000;
  border: none;
  color: #ffffff;
  margin: 2px;
  padding: 2px;
  visibility: hidden;
  position: absolute;
  width: 70px;
}
```

10. Откройте сайт в окне браузера и убедитесь, что кнопки Add to Cart нормально работают, используя AJAX.

Как это работает

Мы изменили имя HTML-элемента, отвечающего за отправку формы, на `add_to_cart`, чтобы ее можно было отправлять из сценария.

```
{/section}
</p>

{* Добавляем кнопку для отправки формы *}
<p>
  <input type="submit" name="add_to_cart" value="Add to Cart" />
</p>
</form>
```

У нас есть переменная `showErrors`, от значения которой зависит, как будут обрабатываться ошибки. Если это значение равно `true`, то с помощью функции `alert()` будут выводиться отладочные сообщения об ошибках. Это не слишком дружественный к пользователю способ обработки ошибок, но он удобен при отладке приложения. Если значение `showErrors` — `false`, сообщения не выводятся. Вместо этого активизируется механизм отправки форм на сервер без использования JavaScript.

```
// Отображаем сообщения об ошибках (если true) или отказываемся
// от использования AJAX (если false)
var showErrors = true;
```

Эта переменная используется в функции `handleError()`, которая вызывается каждый раз, когда в коде на JavaScript происходит ошибка. Если `showErrors` установлена равной `true`, функция просто отображает сведения об ошибке.

```
// Отображает сообщение об ошибке или отказывается от использования AJAX
function handleError($message)
{
  // Игнорирует ошибки, если значение showErrors равно false
  if (showErrors)
  {
    // Отображаем сообщение об ошибке
    alert("Error encountered: \n" + $message);
    return false;
  }
```

Если значение `showErrors` — `false`, функция не отображает сообщений об ошибках. Вместо этого код пытается выполнить запрошенное пользователем действие без использования AJAX. В коде применяется переменная `actionObject` — форма, отправленная пользователем, или ссылка, на которой щелкнул пользователь. Если значение этой переменной не задано, значит, ошибка произошла в функции JavaScript, указанной в атрибуте `onclick` или `onsubmit`. В этом случае нужно просто вернуть значение `true`, и будет выполнено исходное запрошенное действие.

```
// Отказываемся от использования AJAX
else if (!actionObject.tagName)
{
  return true;
}
```

Если `actionObject` — ссылка, значит, ошибка произошла после того, как пользователь на ней щелкнул, и мы перенаправляем запрос по URL этой ссылки.

```
// Отказываемся от использования AJAX и выполняем переход по ссылке
else if (actionObject.tagName == 'A')
{
  window.location = actionObject.href;
}
```

Если `actionObject` — форма, значит, ошибка произошла после отправки формы, и мы перенаправляем запрос по ее URL.

```
  // Отказываемся от использования AJAX и отправляем на сервер форму
  else if (actionObject.tagName == 'FORM')
  {
    actionObject.submit();
  }
```

Файл `index.php` теперь может работать и с запросами, использующими AJAX. При выполнении асинхронного запроса к `index.php` в строку запроса добавляется параметр `AjaxRequest`, чтобы `index.php` знал, как обработать запрос. Сначала задаются требуемые значения заголовков, чтобы гарантировать, что запрос не будет кэширован.

```
if (isset ($_GET['AjaxRequest']))
{
  // Заголовки отправляются, чтобы предотвратить кэширование в браузерах
  header('Expires: Fri, 25 Dec 1980 00:00:00 GMT'); // Устаревшее время
  header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
  header('Cache-Control: no-cache, must-revalidate');
  header('Pragma: no-cache');
  header('Content-Type: text/html');
```

После этого для добавления товара в корзину используется код, созданный нами ранее. Для вывода используется шаблон `cart_summary.tpl`, и код на JavaScript обновит содержимое поля корзины с помощью функций DOM.

```
  if (isset ($_GET['CartAction']))
  {
    $cart_action = $_GET['CartAction'];

    if ($cart_action == ADD_PRODUCT)
    {
      require_once PRESENTATION_DIR . 'cart_details.php';

      $cart_details = new CartDetails();
      $cart_details->init();

      $application->display('cart_summary.tpl');
    }
  }
  else
    trigger_error('CartAction not set', E_USER_ERROR);
}
```

Функция `updateCartSummary()` в файле `ajax.js` отвечает за чтение HTML-разметки, сгенерированной шаблоном `cart_summary.tpl`, и вставку этой разметки в поле корзины. Сначала функция проверяет, не произошло ли ошибки при обработке запроса. Если ошибки есть, вызывается функция `handleError()`, которая или выводит сообщения об этих ошибках, или задействует механизмы, не использующие AJAX.

```
// Обрабатываем ответ сервера
function updateCartSummary()
{
  // Считываем ответ
  response = xmlHttp.responseText;
  // Ошибка сервера?
  if (response.indexOf("ERRNO") >= 0 || response.indexOf("error") >= 0)
  {
    handleError(response);
  }
```

Если в ответе сервера ошибок нет, мы предполагаем, что в нем содержится HTML-разметка с новым содержимым корзины. Ответ будет выглядеть примерно так.

```
{* Start cart summary *}
<div class="box" id="cart-summary">
  <p class="box-title">Cart Summary</p>
  <div id="updating">Updating...</div>
...
</div>
```

Код на JavaScript должен взять содержимое корневого элемента `<div>` и вставить это содержимое в элемент `<div>` шаблона `cart_summary`. Для этого мы используем регулярное выражение `cartSummaryRegEx`.

```
else
{
  // Извлекаем содержимое элемента div cart_summary
  var cartSummaryRegEx = /^<div class="box" id="cart-summary">?_
 ([\s\S]*)<\/div>$/m;
  matches = cartSummaryRegEx.exec(response);
  response = matches[1];
```

Вместо этого мы могли бы использовать методы XML DOM или метод `substring` типа `string`. Последний вариант быстрее и проще всего в реализации, но ему недостает гибкости. Если формат передаваемых сервером данных или имя элемента `<div>` изменится, этот вариант перестанет работать. Вот код такого варианта:

```
response = response.substring(25, response.length - 7);
```

Получив требуемую строку, `updateCartSummary()` заменяет этой строкой содержимое элемента `cart-summary`, таким образом обновляя сведения о содержимом корзины на странице. При этом скрывается надпись `Updating...`

```
    // Обновляем поле содержимого корзины и убираем сообщение об обновлении
    document.getElementById("cart-summary").innerHTML = response;

    // Убираем сообщение "Updating..."
    document.getElementById('updating').style.visibility = 'hidden';
  }
}
```

Модернизация корзины покупателя с помощью AJAX

Улучшенный вариант корзины покупателя будет использовать AJAX для операций обновления содержимого корзины. При удалении товаров из корзины, откладывании товаров для оплаты в будущем или изменении количества товаров все нужные операции будут выполняться в фоновом режиме. Во время асинхронного запроса будет отображаться надпись `Updating...`, свидетельствующая о том, что обновление выполняется (как на рис. 13.4).

Как и при добавлении товаров в корзину, обновление содержимого может выполняться и без использования AJAX. Если на компьютере пользователя JavaScript не поддерживается, механизм обновления будет работать, но не в фоновом режиме, а с помощью классического отправления форм.

Выполните следующее упражнение, чтобы реализовать корзину покупателя с поддержкой AJAX.

Упражнение: добавление поддержки AJAX в корзину покупателя

1. Начнем с файла шаблона `presentation\templates\cart_details.tpl` — он должен вызывать функции JavaScript после щелчков на кнопках. Откройте этот файл и внесите в него изменения, выделенные полужирным шрифтом в следующем листинге.

```
{* cart_details.tpl *}
{load_presentation_object filename="cart_details" assign="obj"}
<div id="updating">Updating...</div>
{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}"
 onsubmit="return executeCartAction(this);">
  <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}"
         onclick="return executeCartAction(this);">Save for later</a>
        <a href="{$obj->mCartProducts[i].remove}"
         onclick="return executeCartAction(this);">Remove</a>
      </td>
    </tr>
  {/section}
  </table>
  <table class="cart-subtotal">
    <tr>
      <td>
        <p>
          Total amount: 
          <font class="price">${$obj->mTotalAmount}</font>
        </p>
      </td>
      <td align="right">
        <input type="submit" name="update" value="Update" />
      </td>
    </tr>
  </table>
</form>
{/if}
{if ($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="{$obj->mSavedCartProducts[j].move}"
       onclick="return executeCartAction(this);">Move to cart</a>
      <a href="{$obj->mSavedCartProducts[j].remove}"
       onclick="return executeCartAction(this);">Remove</a>
    </td>
  </tr>
  {/section}
</table>
{/if}
```

```
{if $obj->mLinkToContinueShopping}
<p><a href="{$obj->mLinkToContinueShopping}">Continue Shopping </a></p>
{/if}
```

2. Откройте файл `ajax.js` и добавьте в него следующие функции.

```
// Вызывается при обновлениях содержимого корзины
function executeCartAction(obj)
{
  // Отображаем сообщение "Updating..."
  document.getElementById('updating').style.visibility = 'visible';

  // Переходим к классической отправке форм, если XMLHttpRequest недоступен
  if (!xmlHttp) return true;

  // Сохраняем ссылку на объект
  actionObject = obj;

  // Инициализируем ответ и параметры
  response = '';
  params = '';

  // Если была активизирована ссылка, получаем ее атрибут href
  if (obj.tagName == 'A')
  {
    url = obj.href + '&AjaxRequest';
  }
  // Если была отправлена форма, получаем ее элементы
  else
  {
    url = obj.action + '&AjaxRequest';
    formElements = obj.getElementsByTagName('INPUT');

    if (formElements)
    {
      for (i = 0; i < formElements.length; i++)
      {
        params += '&' + formElements[i].name + '=';
        params += encodeURIComponent(formElements[i].value);
      }
    }
  }
  // Пытаемся связаться с сервером
  try
  {
    // Отправляем запрос, только если объект XMLHttpRequest не занят
    if (xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
    {
      xmlHttp.open("POST", url, true);
      xmlHttp.setRequestHeader("Content-Type",
                               "application/x-www-form-urlencoded");

      xmlHttp.onreadystatechange = cartActionStateChange;
      xmlHttp.send(params);
    }
  }
  catch (e)
  {
    // Обрабатываем ошибку
    handleError(e.toString());
  }

  // Останавливаем классическую отправку формы, если AJAX успешно отработал
```

```
  return false;
}

// Функция, получающая HTTP-ответ
function cartActionStateChange()
{
  // Когда readyState равно 4, считываем ответ сервера
  if (xmlHttp.readyState == 4)
  {
    // Продолжаем, только если получен HTTP-код состояния "OK"
    if (xmlHttp.status == 200)
    {
      try
      {
        // Считываем ответ
        response = xmlHttp.responseText;

        // Ошибка сервера?
        if (response.indexOf("ERRNO") >= 0 || response.indexOf("error") >= 0)
        {
          handleError(response);
        }
        else
        {
          // Обновляем корзину
          document.getElementById("contents").innerHTML = response;
          // Скрываем сообщение "Updating..."
          document.getElementById('updating').style.visibility = 'hidden';
        }
      }
      catch (e)
      {
        // Обрабатываем ошибку
        handleError(e.toString());
      }
    }
    else
    {
      // Обрабатываем ошибку
      handleError(xmlHttp.statusText);
    }
  }
}
```

3. Внесите в файл `index.php` следующие изменения.

```
  if ($cart_action == ADD_PRODUCT)
  {
    require_once PRESENTATION_DIR . 'cart_details.php';

    $cart_details = new CartDetails();
    $cart_details->init();

    $application->display('cart_summary.tpl');
  }
  else
  {
    $application->display('cart_details.tpl');
  }
}
else
  trigger_error('CartAction not set', E_USER_ERROR);
```

4. Откройте приложение в браузере и убедитесь, что корзина работает как полагается.

Как это работает: корзина с использованием AJAX

Большую часть кода, обеспечивающего работу AJAX, мы написали ранее в этой главе, и в данном упражнении наша задача была простой.

Во-первых, мы модифицировали шаблон корзины покупателя (файл `cart_details.tpl`), определив в нем обработчик события `click` для ссылок Save for later и кнопок Remove, а также обработчик события `submit` для формы. Эти обработчики (при наличии поддержки JavaScript) выполняются до того, как браузер может отправить форму или выполнить переход по активизированной ссылке.

Во-вторых, мы создали следующие функции на JavaScript для обработчиков событий.

- `executeCartAction()`, вызываемую со страницы корзины для выполнения операций с содержимым этой корзины
- `handleExecuteCartAction()` — обработчик для серверных вызовов операций с корзиной
- `postExecuteCartActionProcess()` — для считывания ответа сервера на запросы и обновления страницы

И наконец мы обновили файл `index.php`, чтобы он возвращал HTML-разметку корзины по запросам AJAX. Вот и все — теперь корзина использует AJAX, и пользователям гораздо удобнее с ней работать.

Резюме

Итак, теперь наш сайт использует AJAX. Эта глава была довольно длинной (и, определенно, не простой), но из нее вы узнали немало того, что пригодится вам в последующих главах. Разумеется, у AJAX гораздо больше возможностей, чем мы рассмотрели в этой главе, но и уже сделанное нами заметно улучшит впечатление посетителей от нашего сайта.

Глава 14

Прием заказов от покупателей

Наша новенькая использующая AJAX корзина покупателя полностью готова к работе, за одним важным исключением: посетители не могут размещать заказы. А ведь именно ради этого мы затевали весь проект! Именно решению этой проблемы и посвящена данная глава. Собственно решение мы разделим на две стадии.

- Сначала мы реализуем пользовательскую часть механизма размещения заказов. Точнее говоря, мы добавим кнопку Place Order на страницу корзины покупателя. После щелчка на этой кнопке должен создаваться заказ для PayPal, в котором будут перечислены заказанные товары из корзины покупателя (не забывайте: пока что наш сайт не выполняет финансовые транзакции самостоятельно).
- Затем мы реализуем простую страницу администрирования заказов, чтобы администраторы сайта могли просматривать и обрабатывать подлежащие выполнению заказы.

Код для каждой из частей мы рассмотрим как обычно — начнем с уровня базы данных, затем перейдем к уровню логики приложения и закончим уровнем представления (пользовательским интерфейсом).

Реализация системы приема заказов

Вся система приема заказов построена вокруг вышеупомянутой кнопки Place Order. На рис. 14.1 показано, как эта кнопка будет выглядеть на странице сайта после того, как мы модифицируем компонентный шаблон `cart_details` позже в этой главе.

Да, кнопка выглядит не слишком впечатляюще для элемента, которому посвящена целая глава. Но после щелчка на ней приложение выполняет множество замысловатых операций. Не забывайте, что на данном этапе мы не отслеживаем, кто размещает заказ, а сохраняем детали заказа в базе данных. Это позволит нам реализовать механизм дополнительных предложений (по принципу "посетители, купившие данный товар, также купили вот это"). Мы реализуем этот механизм в главе 15.

В целом после щелчка на кнопке Place Order должно произойти следующее.

- Сначала мы сохраним заказ в базе данных. Для этого мы создадим в ней пару новых таблиц (`orders` и `order_detail`) и напишем код, сохраняющий заказы в этих таблицах.

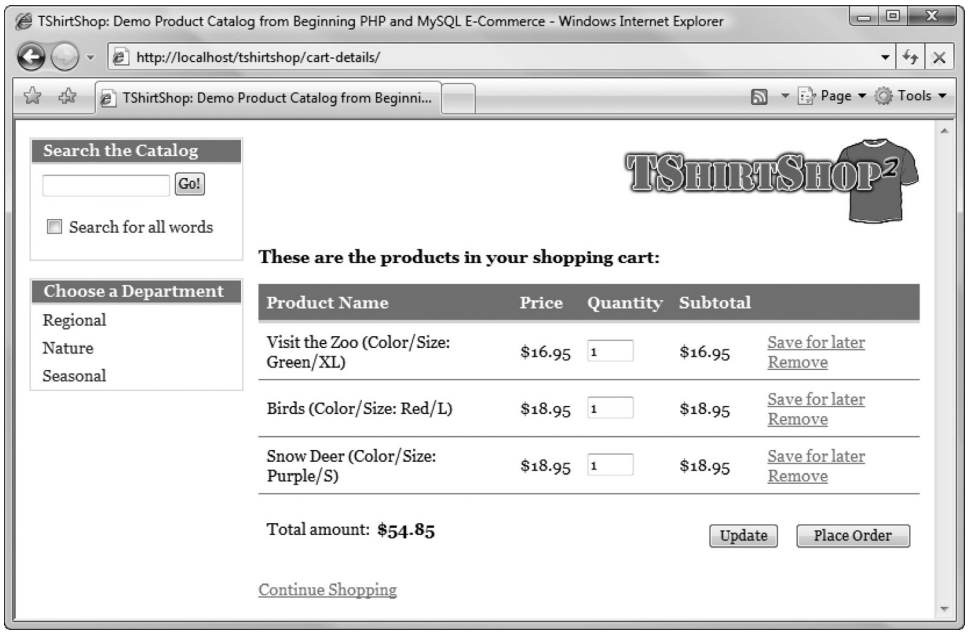


Рис. 14.1. Корзина покупателя с кнопкой Place Order

- Затем мы очистим корзину покупателя. После размещения заказа корзина должна опустеть. Возможно, покупатель захочет отменить заказ на этапе оплаты, а мы не хотим, чтобы наша база данных раздувалась от хранимых в ней отмененных заказов. PayPal и другие службы работы с платежами предлагают механизмы программного уведомления об оплате заказов.
- И наконец мы отправляем посетителя на страницу PayPal, чтобы заплатить за заказ.

ЗАМЕЧАНИЕ

Для обработки платежей мы все еще используем стороннюю программу. Корзина покупателя, предоставляемая PayPal, нам уже не нужна, поскольку в предыдущих главах мы создали собственную. Но мы будем использовать возможность покупки отдельных товаров PayPal — Single Item Purchases — для получения платежей.

При использовании сторонних служб работы с платежами возникает довольно серьезная проблема: посетитель может отменить заказ, находясь на странице оплаты на сайте PayPal. В результате в базе данных могут быть сохранены неоплаченные заказы. Очевидно, нам понадобится система подтверждения оплаты, а также структура в базе данных, в которой будет храниться статус каждого заказа.

Мы реализуем простую систему подтверждения оплаты. Каждая служба приема платежей, включая PayPal, может отправлять сообщения о подтверждении платежей. Мы позволим администраторам нашего сайта проверять, какие заказы были оплачены, а какие — нет.

ЗАМЕЧАНИЕ

PayPal и ее конкуренты предлагают автоматические системы информирования о совершении платежей. Но в этой книге мы не будем подробно анализировать работу данных систем — если хотите, почитайте самостоятельно документацию по этим системам. Например, для PayPal такая документация содержится в руководстве *Order Management Integration Guide*, которое на момент написания этой книги доступно по адресу `http://www.paypal.com/en_US/pdf/PP_OrderManagement_IntegrationGuide.pdf`.

Сохранение деталей заказов

Как упоминалось выше, мы начнем реализацию нового механизма с создания всех необходимых структур данных. Вероятно, это вас не удивит. Вы уже знаете, что при реализации любого механизма нужно сначала определить, с какими данными он должен работать.

Информация о заказах, которую мы хотим сохранять, состоит из двух частей.

- *Сведения о заказе в целом.* Когда заказ был сформирован? Были ли товары отправлены покупателю, и если да, то когда? Каков текущий статус заказа? Все эти сведения мы будем хранить в таблице `orders`, каждая запись в которой будет соответствовать одному заказу.
- *Сведения о товарах, включенных в заказ.* Какие товары входят в заказ? Эту информацию мы будем хранить в таблице `order_detail`, в которой каждая запись будет соответствовать одному товару из заказа. Значит, каждой записи в таблице `orders` может соответствовать множество записей в таблице `order_detail`, и между этими таблицами будет существовать отношение "один-ко-многим" (если вы забыли, что такое отношения, перечитайте главу 5, в которой мы их рассматривали).

ПОДСКАЗКА

До сих пор мы использовали в качестве имен таблиц исключительно существительные в единственном числе (`shopping_cart`, `department` и т.д.). Но сейчас нам придется сделать исключение из этого правила для таблицы `orders`. Дело в том, что `ORDER` — это ключевое слово языка SQL. Чтобы избежать путаницы в коде, мы лучше нарушим правило именования таблиц, чем используем в качестве имени ключевое слово.

В таблице `orders` хранятся сведения о заказах в целом, а в таблице `order_detail` — сведения о товарах, включенных в заказы. Мы создадим эти таблицы в следующем упражнении.

Упражнение: создание таблиц `orders` и `order_detail`

1. Загрузите phpMyAdmin, выберите базу данных `tshirtshop` и откройте страницу нового SQL-запроса.
2. Выполните следующий код, создающий таблицу `orders` в базе данных.

```
-- Создаем таблицу orders
CREATE TABLE `orders` (
  `order_id`          INT             NOT NULL  AUTO_INCREMENT,
  `total_amount`      NUMERIC(10, 2)  NOT NULL  DEFAULT 0.00,
  `created_on`        DATETIME        NOT NULL,
```

```
  `shipped_on`       DATETIME,
  `status`           INT            NOT NULL  DEFAULT 0,
  `comments`         VARCHAR(255),
  `customer_name`    VARCHAR(100),
  `shipping_address` VARCHAR(255),
  `customer_email`   VARCHAR(50),
  PRIMARY KEY (`order_id`)
);
```

3. Выполните следующий код, создающий таблицу `order_detail` в базе данных.

```
-- Создаем таблицу order_detail
CREATE TABLE `order_detail` (
  `item_id`      INT            NOT NULL  AUTO_INCREMENT,
  `order_id`     INT            NOT NULL,
  `product_id`   INT            NOT NULL,
  `attributes`   VARCHAR(1000)  NOT NULL,
  `product_name` VARCHAR(100)   NOT NULL,
  `quantity`     INT            NOT NULL,
  `unit_cost`    NUMERIC(10, 2) NOT NULL,
  PRIMARY KEY (`item_id`),
  KEY `idx_order_detail_order_id` (`order_id`)
);
```

Создав таблицы, давайте подробнее рассмотрим их структуру и отношения.

Как это работает: таблица `orders`

В таблице `orders` содержатся две категории информации: сведения о самом заказе (первые шесть полей) и сведения о посетителе, сделавшем заказ (последние три поля). Хранение сведений о посетителе в таблице `orders` (в полях `customer_name`, `shipping_address` и `customer_email`) необязательно. Администратор сайта может воспользоваться этими сведениями, если они помогут ему при обработке заказов, но пока что нас не слишком волнует, кто именно разместил заказ — нам интересно только, какие товары в него входят. На третьей стадии проекта (см. главу 16) мы перейдем к хранению сведений о покупателях в отдельной таблице.

Имена полей поясняют их назначение: `order_id` — это ключ таблицы, `total_amount` — общая стоимость заказа, `created_on` и `shipped_on` — даты создания заказа и отправки товаров (последний поддерживает значения `NULL`, означающие, что товары еще не были отправлены).

В поле `status` хранится целое число, которое может принимать следующие значения.

- `0`: заказ *размещен*. Это начальное состояние любого заказа после щелчка на кнопке Place Order на странице корзины покупателя.
- `1`: заказ *подтвержден*. Администратор помечает заказ как подтвержденный, получив подтверждение оплаты.
- `2`: заказ *выполнен*. Администратор помечает заказ как выполненный, когда товары отправляются покупателю. Одновременно в поле `shipped_on` задается дата отправки товаров.
- `3`: заказ *отменен*. Обычно администратор помечает заказ как отмененный, если заказ был размещен (с помощью кнопки Place Order), но не был оплачен, или в некоторых других случаях, когда требуется отменить заказ.

В следующей таблице показано несколько записей из таблицы `orders`.

order_id	total_amount	created_on	shipped_on	status	comments	customer_name	shipping_address	customer_email
1	30.94	2007-12-09 17:50:04	2007-12-09 20:31:00	2				
2	37.00	2007-12-09 19:56:17	*NULL*	1				
3	34.85	2007-12-09 20:03:03	*NULL*	0				

Как это работает: таблица `order_detail`

В следующей таблице приведено несколько примеров записей из таблицы `order_detail`.

Item_id	order_id	product_id	attributes	product_name	quantity	unit_cost
1	1	87	Color/Size: White/S	Christmas Tree	1	17.95
2	1	51	Color/Size: Yellow/L	Tankanyyika Giraffe	1	12.99
3	2	94	Color/Size: Orang/M	Swede Santa	1	18.50
4	2	85	Color/Size: Red/L	Altar Piece	1	18.50
5	3	36	Color/Size: Green/XL	Visit the Zoo	1	16.95
6	3	97	Color/Size: Red/L	Birds	1	18.95
7	3	80	Color/Size: Purple/S	Snow Deer	1	18.95

Каждая запись в таблице `order_detail` соответствует заказанному товару, относящемуся к заказу с идентификатором `order_id`. Формируя ключ для этой таблицы, мы сталкиваемся с тем же ограничением, что и при создании таблицы `shopping_cart`. Вообще говоря, ключ мог бы состоять из полей `order_id`, `product_id`, `attributes`, но это запрещено MySQL, и мы создали дополнительное поле `item_id` для формирования ключа.

Кроме того, в таблице `order_detail` хранятся название товара, его атрибуты, количество заказанных единиц и цена (столбец `unit_cost`). Возможно, вы не понимаете, зачем мы сохраняем эту информацию здесь, если у нас уже есть поле `product_id`, по которому можно найти все нужные сведения. Но следует помнить, что информацию о заказах мы сохраняем как часть истории работы сайта. Идентификаторы товаров, их названия и цены могут меняться, но в таблице `order_detail` будут храниться данные, бывшие действительными на момент создания заказа. В столбце `product_id` хранится идентификатор товара, по которому можно найти этот товар (если он еще есть в базе данных).

Реализация уровня данных

Нам необходимо добавить в уровень данных две новые хранимые процедуры. Первая из них — процедура `shopping_cart_create_order`, создающая заказ на товары, находящиеся в корзине покупателя. Вторая — процедура `shopping_cart_empty`, очищающая корзину после размещения заказа.

В следующем упражнении мы создадим эти процедуры, начиная с `shopping_cart_empty`, поскольку она вызывается из `shopping_cart_create_order`.

Упражнение: создание хранимых процедур

1. С помощью phpMyAdmin создайте хранимые процедуры, описанные в последующих шагах. Не забывайте задавать символы `$$` в качестве разделителя перед выполнением кода на каждом шаге.
2. Выполните следующий оператор, создающий хранимую процедуру `shopping_cart_empty`. Когда посетитель разместит заказ, эта хранимая процедура будет вызываться из `shopping_cart_create_order` для очистки корзины.

```
-- Создаем хранимую процедуру shopping_cart_empty
CREATE PROCEDURE shopping_cart_empty(IN inCartId CHAR(32))
BEGIN
  DELETE FROM shopping_cart WHERE cart_id = inCartId;
END$$
```

3. Выполните следующий оператор, создающий хранимую процедуру `shopping_cart_create_order`. Эта хранимая процедура вызывается, когда посетитель решает заказать товары, находящиеся в его корзине, и щелкает на кнопке Place Order. Задача хранимой процедуры — создать новый заказ из товаров, находящихся в корзине. Для этого нужно добавить одну запись в таблицу `orders` и несколько (по одной на каждый вид товара в корзине) — в таблицу `order_detail`.

```
-- Создаем хранимую процедуру shopping_cart_create_order
CREATE PROCEDURE shopping_cart_create_order(IN inCartId CHAR(32))
BEGIN
  DECLARE orderId INT;

  -- Вставляем в таблицу orders новую запись и получаем новый
  -- идентификатор записи
  INSERT INTO orders (created_on) VALUES (NOW());
  -- Получаем новый идентификатор
  SELECT LAST_INSERT_ID() INTO orderId;

  -- Вставляем сведения о заказе в таблицу order_detail
  INSERT INTO order_detail (order_id, product_id, attributes,
                            product_name, quantity, unit_cost)
  SELECT orderId, p.product_id, sc.attributes, p.name, sc.quantity,
  COALESCE(NULLIF(p.discounted_price, 0), p.price) AS unit_cost
  FROM        shopping_cart sc
  INNER JOIN  product p
                ON sc.product_id = p.product_id
  WHERE       sc.cart_id = inCartId AND sc.buy_now;

  -- Сохраняем общую сумму заказа
  UPDATE orders
  SET total_amount = (SELECT SUM(unit_cost * quantity)
                      FROM order_detail
                      WHERE order_id = orderId)
  WHERE order_id = orderId;

  -- Очищаем корзину покупателя
  CALL shopping_cart_empty(inCartId);

  -- Возвращаем идентификатор заказа
  SELECT orderId;
END$$
```

Как это работает: хранимые процедуры `shopping_cart_empty` и `shopping_cart_create_order`

Первая операция, которую выполняет хранимая процедура `shopping_cart_create_order`, — добавление новой записи в таблицу `orders`. Это необходимо сделать в начале процедуры, чтобы узнать, какой идентификатор (значение в столбце `order_id`) будет сгенерирован для нового заказа. Не забывайте, что столбец `order_id` имеет атрибут `AUTO_INCREMENT`, и его значение для каждой записи генерируется автоматически базой данных. Мы считываем это значение после вставки новой записи.

```
-- Вставляем в таблицу orders новую запись и получаем новый
-- идентификатор записи
INSERT INTO orders (created_on) VALUES (NOW());
```

```
-- Получаем новый идентификатор
SELECT LAST_INSERT_ID() INTO orderId;
```

Это стандартный метод получения свежесозданного идентификатора. После выполнения оператора INSERT можно получить последнее сгенерированное значение в столбце с атрибутом AUTO_INCREMENT, воспользовавшись функцией LAST_INSERT_ID(). Сама по себе эта функция довольно проста, но если вас интересуют подробности, можете найти их в документации по адресу http://dev.mysql.com/doc/refman/5.1/en/getting-unique-id.html.

Итак, мы получаем значение, возвращаемое функцией LAST_INSERT_ID(), сохраняем его в переменной с именем orderId и используем для добавления записей в таблицу order_detail. Данные для этих записей мы выбираем из таблиц product и shopping_cart. Мы получаем список товаров и их количеств из таблицы shopping_cart, а их имена и цены — из таблицы product. После этого мы сохраняем полученные данные в таблице order_detail для каждого товара.

```
-- Вставляем сведения о заказе в таблицу order_detail
INSERT INTO order_detail (order_id, product_id, attributes,
                          product_name, quantity, unit_cost)
SELECT orderId, p.product_id, sc.attributes, p.name, sc.quantity,
COALESCE(NULLIF(p.discounted_price, 0), p.price) AS unit_cost
FROM        shopping_cart sc
INNER JOIN  product p
              ON sc.product_id = p.product_id
WHERE       sc.cart_id = inCartId AND sc.buy_now;
```

ПОДСКАЗКА

Создавая соединение таблиц shopping_cart и product, мы получаем значение product_id из таблицы product, но его можно с тем же успехом получать и из таблицы shopping_cart — мы выполняем соединение по столбцу product_id.

Хранимая процедура также вычисляет общую стоимость заказа, умножая цену единицы каждого товара на его количество и вычисляя сумму произведений. Результат сохраняется в столбце total_amount.

```
-- Сохраняем общую сумму заказа
UPDATE orders
SET total_amount = (SELECT SUM(unit_cost * quantity)
                    FROM order_detail
                    WHERE order_id = orderId)
WHERE order_id = orderId;
```

В конце своей работы хранимая процедура очищает корзину покупателя, вызывая хранимую процедуру shopping_cart_empty, и возвращает идентификатор заказа.

```
-- Очищаем корзину покупателя
CALL shopping_cart_empty(inCartId);

-- Возвращаем идентификатор заказа
SELECT orderId;
```

Реализация уровня логики приложения

На уровне логики приложения механизм приема заказов состоит из единственного метода. Добавьте этот метод в класс ShoppingCart в файле business/shopping_cart.php.

```
// Создаем новый заказ
public static function CreateOrder()
{
```

```
  // Составляем SQL-запрос
  $sql = 'CALL shopping_cart_create_order(:cart_id)';

  // Создаем массив параметров
  $params = array (':cart_id' => self::GetCartId());

  // Выполняем запрос и возвращаем результат
  return DatabaseHandler::GetOne($sql, $params);
}
```

Этот метод вызывает хранимую процедуру `shopping_cart_create_order`, которую мы только что рассмотрели, и возвращает идентификатор внесенного в базу данных заказа.

Реализация уровня представления

Наконец мы увидим, как механизм приема заказов работает на практике. Кнопка Place Order — это единственный элемент управления, относящийся к данному механизму. Давайте сначала добавим эту кнопку в шаблон `cart_details`, а затем реализуем ее функциональность.

Упражнение: уровень представления — прием заказов

1. Модифицируйте код в файле `presentation/templates/cart_details.tpl`, добавив в него описание кнопки, выделенное полужирным шрифтом в следующем листинге.

```
<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>
    <td align="right">
      <input type="submit" name="place_order" value="Place Order"
        onclick="placingOrder=true;" />
    </td>
  </tr>
</table>
```

2. Внесите небольшое изменение в файл `ajax.js`, чтобы гарантировать, что щелчки на только что созданной нами кнопке не будут обрабатываться JavaScript. Если помните, в главе 13 мы добавили в форму корзины покупателя обработчик событий `onsubmit`, чтобы операции с корзиной выполнялись асинхронно, если это возможно. Но нам не нужно выполнять асинхронно никаких операций с помощью кнопки Place Order. Эта кнопка должна отправлять форму на сервер и обеспечивать переход к странице приема платежа. Внесите в код в файле `ajax.js` изменения, выделенные полужирным шрифтом в следующем листинге.

```
// Хранит экземпляр объекта XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();

// Отображаем сообщения об ошибках (если true) или отказываемся
```

```
// от использования AJAX (если false)
var showErrors = true;

// Содержит ссылку или форму, отправленную посетителем на сервер
var actionObject = '';

// Устанавливается в true после щелчка на кнопке Place Order,
// в противном случае - false
var placingOrder = false;

// Создает экземпляр XMLHttpRequest
function createXmlHttpRequestObject()
```

3. Код обработчика `onsubmit` будет проверять значение переменной `placingOrder`. Это значение будет равно `true`, если был выполнен щелчок на кнопке Place Order, и обработчик в этом случае также будет возвращать значение `true`, чтобы форма отправлялась на сервер как обычно, без использования AJAX. Кроме того, не нужно отправлять на сервер элемент формы `place_order` во время AJAX-запросов на обновление содержимого корзины. Внесите в функцию `executeCartAction()` в файле `ajax.js` изменения, выделенные полужирным шрифтом в следующем листинге.

```
// Вызывается при обновлениях содержимого корзины
function executeCartAction(obj)
{

  // Возвращаемся к классической отправке формы при создании заказа
  if (placingOrder) return true;

  // Отображаем сообщение "Updating..."
  document.getElementById('updating').style.visibility = 'visible';

  // Переходим к классической отправке форм, если XMLHttpRequest недосту-
пен
  if (!xmlHttp) return true;

  // Сохраняем ссылку на объект
  actionObject = obj;

  // Инициализируем ответ и параметры
  response = '';
  params = '';

  // Если была активизирована ссылка, получаем ее атрибут href
  if (obj.tagName == 'A')
  {
    url = obj.href + '&AjaxRequest';
  }
  // Если была отправлена форма, получаем ее элементы
  else
  {
    url = obj.action + '&AjaxRequest';
    formElements = obj.getElementsByTagName('INPUT');

    if (formElements)
    {
      for (i = 0; i < formElements.length; i++)
      {
        if (formElements[i].name != 'place_order')
        {
```

```
          params += '&' + formElements[i].name + '=';
          params += encodeURIComponent(formElements[i].value);
        }
      }
    }
  }
```

4. Сделайте кнопку Place Order работающей. Поскольку ее работа зависит от компании, которая будет обрабатывать ваши платежи, вам, вероятно, придется адаптировать приведенный ниже код к требованиям этой компании. В следующем примере воспользуйтесь услугами PayPal. В файле `config.php` присвойте константам значения, приведенные ниже. *Не забудьте заменить* `youremail@example.com` *тем адресом электронной почты, который вы указали, регистрируясь на PayPal.*

```
// Конфигурация PayPal
define('PAYPAL_URL',
       'https://www.paypal.com/xclick/business=youremail@example.com');
define('PAYPAL_EMAIL', 'youremail@example.com');
define('PAYPAL_CURRENCY_CODE', 'USD');
define('PAYPAL_RETURN_URL', 'http://www.example.com');
define('PAYPAL_CANCEL_RETURN_URL', 'http://www.example.com');
```

5. Добавьте код, выделенный полужирным шрифтом в следующем листинге, в метод `init()` класса `CartDetails` в файле `presentation/cart_details.php`.

```
// Вычисляем общую сумму для корзины
$this->mTotalAmount = ShoppingCart::GetTotalAmount();

// Если нажата кнопка Place Order...
if(isset ($_POST['place_order']))
{
  // Создаем заказ и получаем его идентификатор
  $order_id = ShoppingCart::CreateOrder();

  // Здесь будет храниться ссылка на PayPal
  $redirect =
    PAYPAL_URL . '&item_name=TShirtShop Order ' .
                 urlencode('#') . $order_id .
    '&item_number=' . $order_id .
    '&amount=' . $this->mTotalAmount .
    '&currency_code=' . PAYPAL_CURRENCY_CODE .
    '&return=' . PAYPAL_RETURN_URL .
    '&cancel_return=' . PAYPAL_CANCEL_RETURN_URL;

  // Перенаправление на страницу оплаты
  header('Location: ' . $redirect);
  exit();
}
  // Получаем список товаров в корзине
  $this->mCartProducts =
    ShoppingCart::GetCartProducts(GET_CART_PRODUCTS);
```

6. Теперь кнопка Place Order должна работать. Проверьте ее работу, добавив в корзину несколько товаров и щелкнув на этой кнопке. Корзина должна очиститься, а затем должен произойти переход на страницу PayPal, примерно такую, как на рис. 14.2.

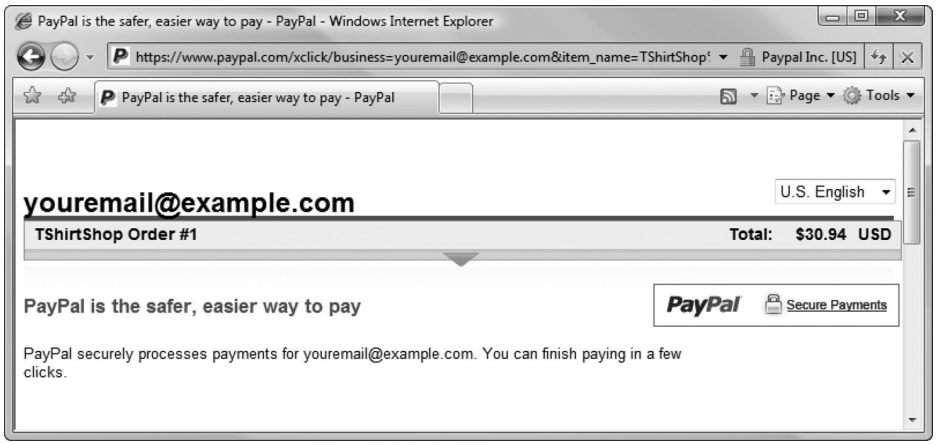


Рис. 14.2. Страница приема платежей PayPal

Как это работает: размещение заказов

Когда посетитель щелкает на кнопке Place Order, происходят два важных действия. Во-первых, в базе данных создается новый заказ, — это делает метод `CreateOrder()` класса `ShoppingCart`, вызывающий хранимую процедуру `shopping_cart_create_order`. Хранимая процедура заносит в базу данных сведения о составе заказа и возвращает идентификатор этого заказа.

```
// Создаем заказ и получаем его идентификатор
$order_id = ShoppingCart::CreateOrder();
```

Во-вторых, выполняется перенаправление на страницу приема платежей, которая позволяет осуществить платеж на сумму, равную общей стоимости заказа.

Администрирование заказов

Итак, посетитель только что разместил заказ. И что теперь?

Предоставив посетителю возможность оплатить заказ, вы должны гарантировать, что он получит заказанные товары. Сайту нужна хорошо продуманная система администрирования заказов, позволяющая администраторам просматривать статус заказов, ожидающих выполнения.

ЗАМЕЧАНИЕ

В данной главе мы стремимся не создать идеальную систему администрирования заказов, а получить нечто простое и достаточно эффективное для решения этой задачи.

Часть сайта, отвечающая за администрирование заказов, будет состоять из двух компонентных шаблонов — `admin_orders` и `admin_order_details`.

Когда администратор щелкнет на ссылке ORDERS ADMIN, страница `admin.php` загрузит компонентный шаблон `admin_orders`, позволяющий фильтровать заказы по заданным критериям. При первой загрузке этот шаблон будет выглядеть так, как показано на рис. 14.3.

После щелчка на одной из кнопок Go! будут отображены заказы, соответствующие заданным критериям, как на рис. 14.4.

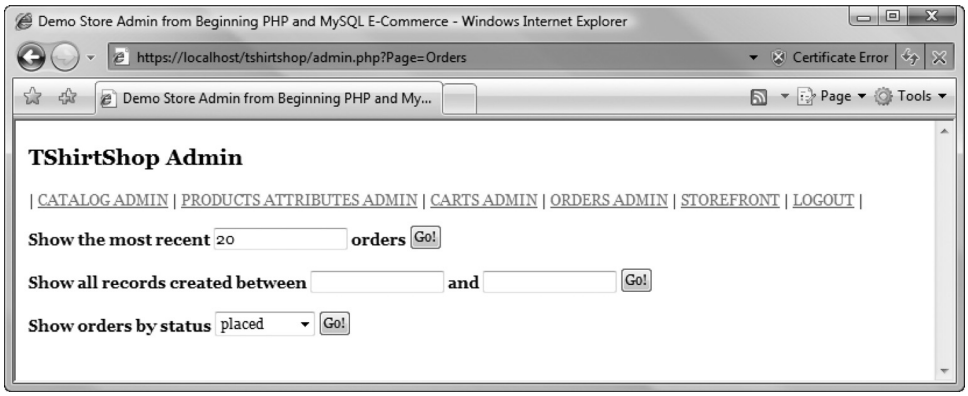


Рис. 14.3. Страница администрирования заказов

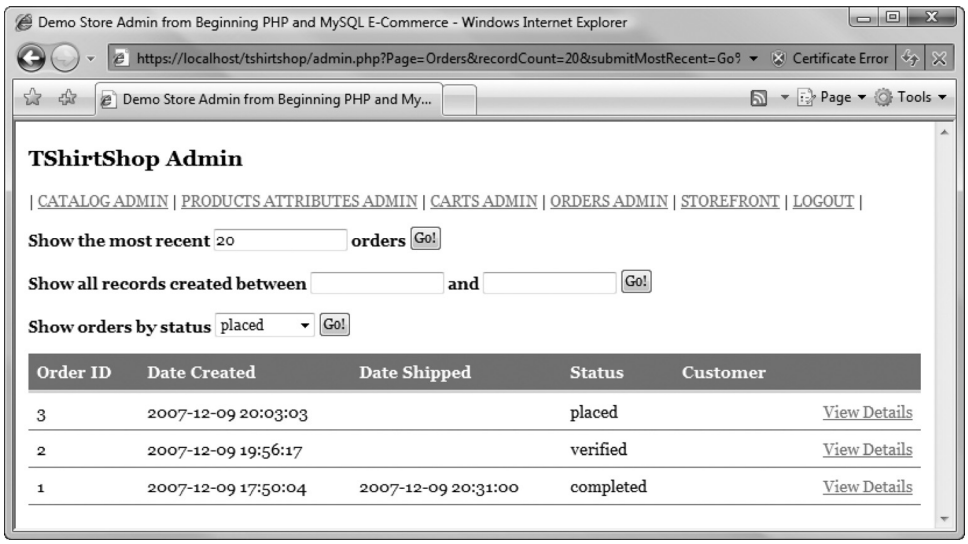


Рис. 14.4. Выбранные из базы данных заказы на странице администрирования

После щелчка на ссылке View details для любого из заказов открывается страница, на которой можно просматривать и изменять информацию о данном заказе (эта страница показана на рис. 14.5).

Прежде чем создавать компонентные шаблоны `admin_orders` и `admin_order_details`, нужно модифицировать шаблон `admin.php` для загрузки новых шаблонов, а также организовать отображение ссылки ORDERS ADMIN в файле `admin_menu.tpl`.

Выполните следующее упражнение, чтобы заложить основы механизма администрирования заказов, который мы создадим далее в этой главе. Учтите, что после выполнения этого упражнения файл `admin.php` будет неработоспособным, поскольку в нем будет ссылка на файл уровня логики приложения, который мы создадим позже.

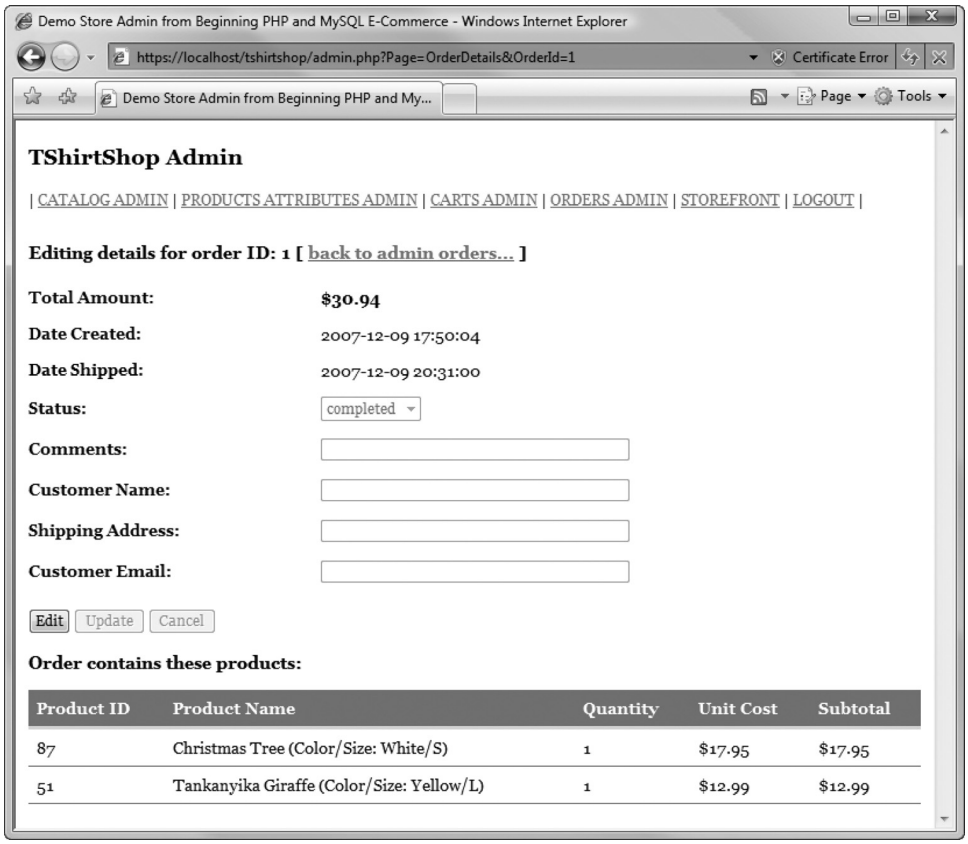


Рис. 14.5. Сведения о конкретном заказе

Упражнение: подготовка к созданию страницы администрирования заказов

1. В файл `admin.php` добавьте ссылку на файл `business/orders.php`, который мы создадим позже.

```
// Загружаем код уровня логики приложения
require_once BUSINESS_DIR . 'catalog.php';
require_once BUSINESS_DIR . 'shopping_cart.php';
require_once BUSINESS_DIR . 'orders.php';
```

2. В файле `presentation/store_admin.php` в конец метода `init()` класса `StoreAdmin` добавьте код, выделенный полужирным шрифтом в следующем листинге. Этот код будет загружать шаблоны `admin_orders.tpl` и `admin_order_details.tpl`.

```
    elseif ($admin_page == 'ProductDetails')
      $this->mContentsCell = 'admin_product_details.tpl';
    elseif ($admin_page == 'Carts')
      $this->mContentsCell = 'admin_carts.tpl';
    elseif ($admin_page == 'Orders')
      $this->mContentsCell = 'admin_orders.tpl';
    elseif ($admin_page == 'OrderDetails')
      $this->mContentsCell = 'admin_order_details.tpl';
  }
```

```
  }
}
?>
```

3. Добавьте в класс `Link` в файле `presentation/link.php` два новых метода.

```
// Создает ссылку на страницу администрирования заказов
public static function ToOrdersAdmin()
{
  return self::ToAdmin('Page=Orders');
}

// Создает ссылку на страницу администрирования деталей заказов
public static function ToOrderDetailsAdmin($orderId)
{
  $link = 'Page=OrderDetails&OrderId=' . $orderId;
  return self::ToAdmin($link);
}
```

4. Откройте файл `presentation/admin_menu.php` и добавьте в класс `AdminMenu` код для отображения ссылки ORDERS ADMIN в меню администрирования, выделенный полужирным шрифтом в следующем листинге.

```
<?php
class AdminMenu
{
  public $mLinkToStoreAdmin;
  public $mLinkToAttributesAdmin;
  public $mLinkToCartsAdmin;
  public $mLinkToOrdersAdmin;
  public $mLinkToStoreFront;
  public $mLinkToLogout;

  public function __construct()
  {
    $this->mLinkToStoreAdmin = Link::ToAdmin();
    $this->mLinkToAttributesAdmin = Link::ToAttributesAdmin();
    $this->mLinkToCartsAdmin = Link::ToCartsAdmin();
    $this->mLinkToOrdersAdmin = Link::ToOrdersAdmin();

    if (isset ($_SESSION['link_to_store_front']))
      $this->mLinkToStoreFront = $_SESSION['link_to_store_front'];
    else
      $this->mLinkToStoreFront = Link::ToIndex();

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

5. Внесите в файл `presentation/templates/admin_menu.tpl` изменения, необходимые для отображения ссылки на страницу администрирования заказов. Кроме того, эти изменения позволят применять к меню стилевое оформление.

```
<p class="menu"> |
  <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->mLinkToOrdersAdmin}">ORDERS ADMIN</a> |
  <a href="{$obj->mLinkToStoreFront}">STOREFRONT</a> |
  <a href="{$obj->mLinkToLogout}">LOGOUT</a> |
```

6. Откройте файл `tshirtshop.css` в папке `styles` и добавьте в него следующее определение стиля.

```
.menu {
  font-size: 93%;
}
```

Как это работает: подготовка к созданию страницы администрирования заказов

В этом упражнении мы не создали в проекте ничего такого, что можно было бы протестировать, — мы только подготовили его к реализации механизмов администрирования заказов. Кроме того, мы добавили в файл `tshirtshop.css` новый стиль, который позволит аккуратно размещать меню администрирования на странице. Учтите, что если вы сейчас откроете сайт в браузере, то при попытке загрузки страницы `admin.php` произойдет ошибка — на этой странице есть ссылка на несуществующий файл `business/orders.php`. Его мы создадим в следующем упражнении.

Отображение существующих заказов

Следующие несколько страниц посвящены созданию компонентного шаблона `admin_orders` и используемых в нем элементов уровня логики приложения и уровня данных. Этот шаблон позволяет администраторам просматривать существующие заказы. Поскольку список заказов может со временем стать очень длинным, важно иметь возможность фильтровать его по заданным критериям, выбирая только некоторые заказы.

Администратор сможет задавать фильтрацию заказов по следующим критериям.

- Отображение заказов, сделанных последними
- Отображение заказов, сделанных в определенный период
- Отображение заказов с выбранным статусом

Мы создадим все необходимое для этого в следующем упражнении.

Упражнение: создание страницы администрирования заказов

1. Начните с создания необходимых хранимых процедур в базе данных. С помощью phpMyAdmin создайте хранимые процедуры, описанные в следующих шагах. Не забывайте задавать символы `$$` в качестве разделителя перед выполнением кода на каждом шаге.
2. Выполните приведенный ниже код, который создаст в базе данных хранимую процедуру `orders_get_most_recent_orders`. Эта хранимая процедура будет возвращать заказы, сделанные последними. Раздел `LIMIT` оператора `SELECT` ограничивает количество возвращаемых записей числом, указанным во входном параметре `inHowMany`. Раздел `ORDER BY DESC`, упорядочивающий отобранные записи по убыванию, гарантирует, что новые заказы будут указываться в списке первыми.

```
-- Создаем хранимую процедуру orders_get_most_recent_orders
CREATE PROCEDURE orders_get_most_recent_orders(IN inHowMany INT)
BEGIN
  PREPARE statement FROM
    "SELECT order_id, total_amount, created_on,
            shipped_on, status, customer_name
     FROM orders
     ORDER BY created_on DESC
```

```
    LIMIT ?";

  SET @p1 = inHowMany;

  EXECUTE statement USING @p1;
END$$
```

3. Выполните приведенный ниже код, чтобы создать хранимую процедуру `orders_get_orders_between_dates`. Эта хранимая процедура будет возвращать заказы, сделанные между датами, заданными в параметрах `inStartDate` и `inEndDate`. Результаты будут располагаться в порядке убывания дат.

```
-- Создаем хранимую процедуру orders_get_orders_between_dates
CREATE PROCEDURE orders_get_orders_between_dates(
  IN inStartDate DATETIME, IN inEndDate DATETIME)
BEGIN
  SELECT   order_id, total_amount, created_on,
           shipped_on, status, customer_name
  FROM     orders
  WHERE    created_on >= inStartDate AND created_on <= inEndDate
  ORDER BY created_on DESC;
END$$
```

4. Выполните приведенный ниже код, чтобы создать хранимую процедуру `orders_get_orders_by_status`. Она будет возвращать заказы со статусом, заданным в параметре `inStatus`.

```
-- Создаем хранимую процедуру orders_get_orders_by_status
CREATE PROCEDURE orders_get_orders_by_status(IN inStatus INT)
BEGIN
  SELECT   order_id, total_amount, created_on,
           shipped_on, status, customer_name
  FROM     orders
  WHERE    status = inStatus
  ORDER BY created_on DESC;
END$$
```

5. На уровне логики приложения администрирование заказов будет обеспечиваться классом `Orders`, методы которого будут обращаться к только что созданным нами хранимым процедурам. Этот класс прямолинеен, и в его коде нет ничего заслуживающего внимания. Просто создайте в папке `business` файл `orders.php` и добавьте в него следующий код.

```
<?php
// Класс уровня логики приложения для администрирования заказов
class Orders
{
  public static $mOrderStatusOptions = array ('placed',    // 0
                                              'verified',  // 1
                                              'completed', // 2
                                              'canceled'); // 3
  // Получаем $how_many последних заказов
  public static function GetMostRecentOrders($how_many)
  {
    // Составляем SQL-запрос
    $sql = 'CALL orders_get_most_recent_orders(:how_many)';

    // Создаем массив параметров
    $params = array (':how_many' => $how_many);

    // Выполняем запрос и возвращаем результаты
    return DatabaseHandler::GetAll($sql, $params);
```

```
  }

  // Получаем заказы, сделанные между заданными датами
  public static function GetOrdersBetweenDates($startDate, $endDate)
  {
    // Составляем SQL-запрос
    $sql = 'CALL orders_get_orders_between_dates(:start_date, :end_date)';

    // Создаем массив параметров
    $params = array (':start_date' => $startDate, ':end_date' => $endDate);

    // Выполняем запрос и возвращаем результаты
    return DatabaseHandler::GetAll($sql, $params);
  }

  // Получаем заказы с выбранным статусом
  public static function GetOrdersByStatus($status)
  {
    // Составляем SQL-запрос
    $sql = 'CALL orders_get_orders_by_status(:status)';

    // Создаем массив параметров
    $params = array (':status' => $status);

    // Выполняем запрос и возвращаем результаты
    return DatabaseHandler::GetAll($sql, $params);
  }
}
?>
```

6. Создайте собственно компонентный шаблон `admin_orders`. Создайте файл `admin_orders.tpl` в папке `presentation/templates` и поместите в него следующий код.

```
{* admin_orders.tpl *}
{load_presentation_object filename="admin_orders" assign="obj"}
{if $obj->mErrorMessage}<p class="error">{$obj->mErrorMessage}</p>{/if}
<form method="get" action="{$obj->mLinkToAdmin}">
  <input name="Page" type="hidden" value="Orders" />
  <p>
    <font class="bold-text">Show the most recent</font>
    <input name="recordCount" type="text" value="{$obj->mRecordCount}" />
    <font class="bold-text">orders</font>
    <input type="submit" name="submitMostRecent" value="Go!" />
  </p>
  <p>
    <font class="bold-text">Show all records created between</font>
    <input name="startDate" type="text" value="{$obj->mStartDate}" />
    <font class="bold-text">and</font>
    <input name="endDate" type="text" value="{$obj->mEndDate}" />
    <input type="submit" name="submitBetweenDates" value="Go!" />
  </p>
  <p>
    <font class="bold-text">Show orders by status</font>
    {html_options name="status" options=$obj->mOrderStatusOptions
     selected=$obj->mSelectedStatus}
    <input type="submit" name="submitOrdersByStatus" value="Go!" />
  </p>
</form>
{if $obj->mOrders}
<table class="tss-table">
  <tr>
    <th>Order ID</th>
```

```
    <th>Date Created</th>
    <th>Date Shipped</th>
    <th>Status</th>
    <th>Customer</th>
    <th> </th>
  </tr>
  {section name=i loop=$obj->mOrders}
    {assign var=status value=$obj->mOrders[i].status}
  <tr>
    <td>{$obj->mOrders[i].order_id}</td>
    <td>{$obj->mOrders[i].created_on|date_format:"%Y-%m-%d %T"}</td>
    <td>{$obj->mOrders[i].shipped_on|date_format:"%Y-%m-%d %T"}</td>
    <td>{$obj->mOrderStatusOptions[$status]}</td>
    <td>{$obj->mOrders[i].customer_name}</td>
    <td align="right">
      <a href="{$obj->mOrders[i].link_to_order_details_admin}">View De-
tails</a>
    </td>
  </tr>
  {/section}
</table>
{/if}
```

7. Создайте в папке `presentation` файл `admin_orders.php` и поместите в него следующий код.

```
<?php
// Класс уровня представления, отвечающий за администрирование заказов
class AdminOrders
{
  // Public-переменные, доступные в шаблоне smarty
  public $mOrders;
  public $mStartDate;
  public $mEndDate;
  public $mRecordCount = 20;
  public $mOrderStatusOptions;
  public $mSelectedStatus = 0;
  public $mErrorMessage = '';
  public $mLinkToAdmin;

  // Конструктор класса
  public function __construct()
  {
    /* Сохраняем ссылку на текущую страницу в сеансовой переменной
       link_to_orders_admin; она понадобится нам для создания ссылки
       "back to admin orders ..." на страницах деталей заказов */
    $_SESSION['link_to_orders_admin'] =
      Link::Build(str_replace(VIRTUAL_LOCATION, '', getenv('REQUEST_URI')));

    $this->mLinkToAdmin = Link::ToAdmin();
    $this->mOrderStatusOptions = Orders::$mOrderStatusOptions;
  }

  public function init()
  {
    // Если используется фильтр "Показывать x последних заказов"...
    if (isset ($_GET['submitMostRecent']))
    {
      // Если количество записей - не корректное целое число,
      // выводим сообщение об ошибке
      if ((string)(int)$_GET['recordCount'] == (string)$_GET['recordCount'])
      {
```

```
    $this->mRecordCount = (int)$_GET['recordCount'];
      $this->mOrders = Orders::GetMostRecentOrders($this->mRecordCount);
  }
  else
    $this->mErrorMessage = $_GET['recordCount'] . ' is not a number.';
}

/* Если используется фильтр "Показать заказы, сделанные между
   date_1 и date_2"... */
if (isset ($_GET['submitBetweenDates']))
{
  $this->mStartDate = $_GET['startDate'];
  $this->mEndDate = $_GET['endDate'];

  // Проверяем корректность начальной даты
  if (($this->mStartDate == '') ||
      ($timestamp = strtotime($this->mStartDate)) == -1)
    $this->mErrorMessage = 'The start date is invalid. ';
  else
    // Преобразуем дату в формат YYYY/MM/DD HH:MM:SS
    $this->mStartDate =
      strftime('%Y/%m/%d %H:%M:%S', strtotime($this->mStartDate));

  // Проверяем корректность конечной даты
  if (($this->mEndDate == '') ||
      ($timestamp = strtotime($this->mEndDate)) == -1)
    $this->mErrorMessage .= 'The end date is invalid.';
  else
    // Преобразуем дату в формат YYYY/MM/DD HH:MM:SS
    $this->mEndDate =
      strftime('%Y/%m/%d %H:%M:%S', strtotime($this->mEndDate));

  // Проверяем правильность следования дат
  if ((empty ($this->mErrorMessage)) &&
      (strtotime($this->mStartDate) > strtotime($this->mEndDate)))
    $this->mErrorMessage .=
      'The start date should be more recent than the end date.';

  // Если ошибок нет, получаем заказы, сделанные между двумя датами
  if (empty($this->mErrorMessage))
    $this->mOrders = Orders::GetOrdersBetweenDates(
                       $this->mStartDate, $this->mEndDate);
}

// Если используется фильтр "Показать заказы с выбранным статусом"...
if (isset ($_GET['submitOrdersByStatus']))
{
  $this->mSelectedStatus = $_GET['status'];
  $this->mOrders = Orders::GetOrdersByStatus($this->mSelectedStatus);
}

if (is_array($this->mOrders) && count($this->mOrders) == 0)
  $this->mErrorMessage =
    'No orders found matching your searching criteria!';

// Создаем ссылку View Details
for ($i = 0; $i < count($this->mOrders); $i++)
{
  $this->mOrders[$i]['link_to_order_details_admin'] =
    Link::ToOrderDetailsAdmin($this->mOrders[$i]['order_id']);
}
```

```
  }
}
?>
```

8. Загрузите файл `admin.php` в браузер и введите имя пользователя и пароль, когда их у вас спросят. Щелкните на ссылке ORDERS ADMIN, а затем на одной из кнопок Go!, и в окне браузера появится нечто похожее на рис. 14.3.

Как это работает: чтение заказов из базы данных

Каждая из кнопок Go! вызывает один из методов уровня логики приложения (в классе `Orders`) и заполняет таблицу информацией о выбранных заказах.

При обработке запроса мы проверяем, корректные ли данные ввел пользователь. После щелчка на первой кнопке Go! проверяется корректность введенного числа (это число указывает, сколько записей нужно вывести). Кроме того, мы проверяем корректность дат, введенных в текстовые поля Start Date и End Date.

Сначала мы обрабатываем данные с помощью функции `strtotime()`, которая просматривает строку и преобразует ее в дату и время в формате UNIX. Эта функция удобна тем, что понимает, например, входные значения наподобие `now`, `tomorrow`, `last Friday`, `next Tuesday` и др.

Полученная в результате дата обрабатывается функцией `strftime()`, которая преобразует ее в формат `YYYY/MM/DD HH:MM:SS`. Еще раз посмотрите на код, который обрабатывает значения даты и времени.

```
// Проверяем корректность начальной даты
if (($this->mStartDate == '') ||
    ($timestamp = strtotime($this->mStartDate)) == -1)
  $this->mErrorMessage = 'The start date is invalid. ';
else
  // Преобразуем дату в формат YYYY/MM/DD HH:MM:SS
  $this->mStartDate =
    strftime('%Y/%m/%d %H:%M:%S', strtotime($this->mStartDate));
```

ЗАМЕЧАНИЕ

Если вам интересно, какие входные значения понимает функция `strtotime()`, обратитесь по адресу `http://www.php.net/strtotime`. Информация о функции `strftime()` доступна по адресу `http://www.php.net/strftime`. Кроме того, небольшой учебный курс по использованию функции `strtotime()` доступен по адресу `http://www.programmers-corner.com/tutorial/80`.

За исключением этой части, код в шаблоне `admin_orders` вполне тривиален и не содержит ничего нового для вас.

Отображение деталей заказов

В этом разделе мы создадим компонентный шаблон `admin_order_details`, который позволит администраторам редактировать детали отдельных заказов. Самые необходимые операции с заказами — это изменение их статуса с размещенного на подтвержденный при получении платежа и с подтвержденного на выполненный при отправке товаров. Как выглядит шаблон `admin_order_details` в действии, показано на рис. 14.3.

Администратор сайта помечает заказ как подтвержденный, когда PayPal сообщает, что за заказ получена оплата. Заказ помечается как выполненный, когда товары из заказа отправляются покупателю. Администратор может также пометить заказ как отмененный, если, например, PayPal не сообщает об оплате этого заказа в течение определенного времени.

Кнопки Edit, Update и Cancel на странице работы с деталями заказа позволяют администраторам вручную редактировать все элементы заказа. После щелчка на кнопке Edit элементы, в которых на странице отображаются данные, становятся доступными для редактирования.

Теперь вам понятно, что должен делать шаблон `admin_order_details`. Давайте создадим его, как обычно, начав с уровня данных.

Упражнение: администрирование деталей заказов

1. С помощью phpMyAdmin создайте в базе данных `tshirtshop` три хранимые процедуры из последующих шагов — `orders_get_order_info`, `orders_get_order_details` и `orders_update_order`. Не забывайте задавать символы `$$` в качестве разделителя перед выполнением SQL-операторов на каждом шаге.
2. Выполните следующий оператор SQL, создающий хранимую процедуру `orders_get_order_info`. Эта хранимая процедура возвращает данные, необходимые для заполнения формы в компонентном шаблоне `admin_order_details`, например суммы заказов, даты их создания, даты отправки и т.д. Как отображаются эти данные, показано на рис. 14.4.

```
-- Создаем хранимую процедуру orders_get_order_info
CREATE PROCEDURE orders_get_order_info(IN inOrderId INT)
BEGIN
  SELECT order_id, total_amount, created_on, shipped_on, status,
         comments, customer_name, shipping_address, customer_email
  FROM   orders
  WHERE  order_id = inOrderId;
END$$
```

3. Выполните следующий оператор SQL, создающий хранимую процедуру `orders_get_order_details`. Она возвращает список товаров, входящих в заданный заказ. Возвращаемыми данными заполняется таблица деталей заказа, расположенная в нижней части страницы.

```
-- Создаем хранимую процедуру orders_get_order_details
CREATE PROCEDURE orders_get_order_details(IN inOrderId INT)
BEGIN
  SELECT order_id, product_id, attributes, product_name,
         quantity, unit_cost, (quantity * unit_cost) AS subtotal
  FROM   order_detail
  WHERE  order_id = inOrderId;
END$$
```

4. Выполните следующий оператор SQL, создающий хранимую процедуру `orders_update_order`. Она вызывается, когда администратор желает обновить заказ в базе данных после его редактирования.

```
-- Создаем хранимую процедуру orders_update_order
CREATE PROCEDURE orders_update_order(IN inOrderId INT, IN inStatus INT,
  IN inComments VARCHAR(255), IN inCustomerName VARCHAR(50),
  IN inShippingAddress VARCHAR(255), IN inCustomerEmail VARCHAR(50))
BEGIN
  DECLARE currentStatus INT;

  SELECT status
  FROM   orders
  WHERE  order_id = inOrderId
  INTO   currentStatus;
```

```
  IF inStatus != currentStatus AND (inStatus = 0 OR inStatus = 1) THEN
    UPDATE orders SET shipped_on = NULL WHERE order_id = inOrderId;
  ELSEIF inStatus != currentStatus AND inStatus = 2 THEN
    UPDATE orders SET shipped_on = NOW() WHERE order_id = inOrderId;
  END IF;

  UPDATE orders
  SET    status = inStatus, comments = inComments,
         customer_name = inCustomerName,
         shipping_address = inShippingAddress,
         customer_email = inCustomerEmail
  WHERE  order_id = inOrderId;
END$$
```

5. На уровне логики приложения работа компонентного шаблона `admin_order_details` обеспечивается несколькими методами класса `Orders`, которые нужно добавить в код этого класса в файле `business/orders.php`.

```
// Считывает детали выбранного заказа
public static function GetOrderInfo($orderId)
{
  // Составляем SQL-запрос
$sql = 'CALL orders_get_order_info(:order_id)';

  // Создаем массив параметров
  $params = array (':order_id' => $orderId);

  // Выполняем запрос и возвращаем результаты
return DatabaseHandler::GetRow($sql, $params);
}
// Получаем товары, относящиеся к заданному заказу
public static function GetOrderDetails($orderId)
{
  // Составляем SQL-запрос
  $sql = 'CALL orders_get_order_details(:order_id)';

  // Создаем массив параметров
  $params = array (':order_id' => $orderId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetAll($sql, $params);
}

// Обновляем детали заказа в базе данных
public static function UpdateOrder($orderId, $status, $comments,
$customerName, $shippingAddress, $customerEmail)
{
  // Составляем SQL-запрос
  $sql = 'CALL orders_update_order(:order_id, :status, :comments,
                :customer_name, :shipping_address, :customer_email)';

  // Создаем массив параметров
  $params = array (':order_id' => $orderId,
                   ':status' => $status,
                   ':comments' => $comments,
                   ':customer_name' => $customerName,
                   ':shipping_address' => $shippingAddress,
                   ':customer_email' => $customerEmail);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}
```

6. Создайте собственно пользовательский интерфейс для администрирования деталей заказов. Уровень представления как раз и будет состоять из компонентного шаблона `admin_order_details`. Создайте в папке `presentation/templates` файл шаблона `admin_order_details.tpl` и поместите в этот файл следующий код.

```
{* admin_order_details.tpl *}
{load_presentation_object filename="admin_order_details" assign="obj"}
<form method="get" action="{$obj->mLinkToAdmin}">
  <h3>
    Editing details for order ID:
    {$obj->mOrderInfo.order_id} [
    <a href="{$obj->mLinkToOrdersAdmin}">back to admin orders...</a> ]
  </h3>
  <input type="hidden" name="Page" value="OrderDetails" />
  <input type="hidden" name="OrderId"
   value="{$obj->mOrderInfo.order_id}" />
  <table class="borderless-table">
    <tr>
      <td class="bold-text">Total Amount: </td>
      <td class="price">
        ${$obj->mOrderInfo.total_amount}
      </td>
    </tr>
    <tr>
      <td class="bold-text">Date Created: </td>
      <td>
        {$obj->mOrderInfo.created_on|date_format:"%Y-%m-%d %T"}
      </td>
    </tr>
    <tr>
      <td class="bold-text">Date Shipped: </td>
      <td>
        {$obj->mOrderInfo.shipped_on|date_format:"%Y-%m-%d %T"}
      </td>
    </tr>
    <tr>
      <td class="bold-text">Status: </td>
      <td>
        <select name="status"
         {if ! $obj->mEditEnabled} disabled="disabled" {/if} >
          {html_options options=$obj->mOrderStatusOptions
           selected=$obj->mOrderInfo.status}
        </select>
      </td>
    </tr>
    <tr>
      <td class="bold-text">Comments: </td>
      <td>
        <input name="comments" type="text" size="50"
         value="{$obj->mOrderInfo.comments}"
         {if ! $obj->mEditEnabled} disabled="disabled" {/if} />
      <td>
    </tr>
    <tr>
      <td class="bold-text">Customer Name: </td>
      <td>
        <input name="customerName" type="text" size="50"
         value="{$obj->mOrderInfo.customer_name}"
         {if ! $obj->mEditEnabled} disabled="disabled" {/if} />
      <td>
    </tr>
```

```
    <tr>
      <td class="bold-text">Shipping Address: </td>
      <td>
        <input name="shippingAddress" type="text" size="50"
         value="{$obj->mOrderInfo.shipping_address}"
         {if ! $obj->mEditEnabled} disabled="disabled" {/if} />
      </td>
    </tr>
    <tr>
      <td class="bold-text">Customer Email: </td>
      <td>
        <input name="customerEmail" type="text" size="50"
         value="{$obj->mOrderInfo.customer_email}"
         {if ! $obj->mEditEnabled} disabled="disabled" {/if} />
      </td>
    </tr>
  </table>
  <p>
    <input type="submit" name="submitEdit" value="Edit"
     {if $obj->mEditEnabled} disabled="disabled" {/if} />
    <input type="submit" name="submitUpdate" value="Update"
     {if ! $obj->mEditEnabled} disabled="disabled" {/if} />
    <input type="submit" name="submitCancel" value="Cancel"
     {if ! $obj->mEditEnabled} disabled="disabled" {/if} />
  </p>
  <h3>Order contains these products:</h3>
  <table class="tss-table">
    <tr>
      <th>Product ID</th>
      <th>Product Name</th>
      <th>Quantity</th>
      <th>Unit Cost</th>
      <th>Subtotal</th>
    </tr>
  {section name=i loop=$obj->mOrderDetails}
    <tr>
      <td>{$obj->mOrderDetails[i].product_id}</td>
      <td>
        {$obj->mOrderDetails[i].product_name}
        ({$obj->mOrderDetails[i].attributes})
      </td>
      <td>{$obj->mOrderDetails[i].quantity}</td>
      <td>${$obj->mOrderDetails[i].unit_cost}</td>
      <td>${$obj->mOrderDetails[i].subtotal}</td>
    </tr>
  {/section}
  </table>
</form>
```

7. Создайте в папке `presentation` файл `admin_order_details.php` и поместите в него следующий код.

```
<?php
// Класс уровня представления, отвечающий за администрирование деталей заказов
class AdminOrderDetails
{
  // Public-переменные, доступные в шаблоне smarty
  public $mOrderId;
  public $mOrderInfo;
  public $mOrderDetails;
  public $mEditEnabled;
  public $mOrderStatusOptions;
```

```
  public $mLinkToAdmin;
  public $mLinkToOrdersAdmin;

  // Конструктор класса
  public function __construct()
  {
    // Получаем ссылку на предыдущую страницу из сеансовой переменной
    $this->mLinkToOrdersAdmin = $_SESSION['link_to_orders_admin'];

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

    // Мы получаем идентификатор заказа в строке запроса
    if (isset ($_GET['OrderId']))
      $this->mOrderId = (int) $_GET['OrderId'];
    else
      trigger_error('OrderId paramater is required');

    $this->mOrderStatusOptions = Orders::$mOrderStatusOptions;
  }

  // Инициализируем элементы класса
  public function init()
  {
    if (isset ($_GET['submitUpdate']))
    {
      Orders::UpdateOrder($this->mOrderId, $_GET['status'],
        $_GET['comments'], $_GET['customerName'], $_GET['shippingAddress'],
        $_GET['customerEmail']);
    }

    $this->mOrderInfo = Orders::GetOrderInfo($this->mOrderId);
    $this->mOrderDetails = Orders::GetOrderDetails($this->mOrderId);

    // Значение, указывающее, включен или отключен режим редактирования
    if (isset ($_GET['submitEdit']))
      $this->mEditEnabled = true;
    else
      $this->mEditEnabled = false;
  }
}
?>
```

8. Добавьте в базу данных несколько заказов, а затем откройте страницу администрирования и щелкните на ссылке ORDERS ADMIN в меню. На загрузившейся странице щелкните на кнопке Go!, чтобы просмотреть заказы, и в строке одного из заказов щелкните на кнопке View Details. Должна загрузиться страница администрирования деталей заказа, работа которой описана ранее в этой главе.

Как это работает: администрирование деталей заказов

Два только что созданных вами файла — `admin_order_details.tpl` и `admin_order_details.php` — позволяют просматривать и редактировать детали отдельных заказов.

Конструктор класса `AdminOrderDetails` (метод `__construct()`) проверяет наличие параметра `OrderId` в строке запроса, поскольку без значения этого параметра шаблон не может работать.

```
// Конструктор класса
public function __construct()
{
  // Получаем ссылку на предыдущую страницу из сеансовой переменной
  $this->mLinkToOrdersAdmin = $_SESSION['link_to_orders_admin'];

  $this->mLinkToAdmin = Link::ToAdmin();
```

```
  // Мы получаем идентификатор заказа в строке запроса
  if (isset ($_GET['OrderId']))
    $this->mOrderId = (int) $_GET['OrderId'];
  else
    trigger_error('OrderId paramater is required');

  $this->mOrderStatusOptions = Orders::$mOrderStatusOptions;
}
```

Метод `init()` реагирует на действия пользователя и вызывает различные методы уровня логики приложения для выполнения требуемых операций с заказами. Он заполняет форму данными, которые получает от методов `Orders::GetOrderInfo()` и `Orders::GetOrderDetails()`.

Поле `$mEditEnabled` класса `AdminOrderDetails` указывает, находится ли страница в режиме редактирования заказа. Значение этого поля определяется значением параметра `submitEdit` в строке запроса. В режиме редактирования доступны все текстовые поля и кнопки Update и Cancel. В режиме просмотра эти элементы недоступны, т.е. они не реагируют на действия пользователя.

Резюме

В этой главе мы достигли многого. В две стадии мы снабдили наш сайт механизмом приема заказов от покупателей и системой администрирования этих заказов. Мы добавили на страницу корзины покупателя кнопку Place Order, щелкнув на которой, посетитель может заказать товары, находящиеся в корзине. Система администрирования, созданная нами, позволяет администраторам сайта просматривать и обрабатывать заказы покупателей.

Поскольку данные о заказах хранятся в базе данных, мы можем статистически их анализировать и выполнять подсчеты количества проданных товаров. Из следующей главы вы узнаете, как реализовать систему динамических рекомендаций, предлагающую посетителям дополнительные товары на основе тех, которые они уже выбрали.

Глава 15

Предложение дополнительных товаров

Одно из основных преимуществ электронных магазинов по сравнению с обычными — возможность подстраиваться под предпочтения каждого отдельного посетителя и предлагать товары, которые, вероятнее всего, будут ему интересны. Если сайт сможет предлагать посетителям дополнительные товары, возможно, посетители в конечном итоге купят больше, чем изначально планировали. Вы, без сомнения, видели этот механизм в действии на многих сайтах электронных магазинов, и тому есть причина — он заметно увеличивает прибыль.

В этой главе мы реализуем на сайте TShirtShop простой, но эффективный механизм предложения дополнительных товаров. Точнее говоря, мы

- добавим предложения дополнительных товаров на страницы товаров (эти предложения будут указывать на товары, которые покупались вместе с выбранным),
- добавим предложения товаров на страницу корзины (эти предложения будут указывать на товары, которые покупались вместе с находящимися в корзине).

Сам по себе этот механизм довольно прост, но поиск *оптимального* механизма предложения дополнительных товаров — нетривиальная задача.

Увеличение продаж с помощью динамических рекомендаций

Есть несколько способов увеличения продаж путем дополнительных предложений. Их эффективность зависит от того, какие товары вы продаете в своем магазине. Вот основные приемы.

- *Продажа более дорогих товаров.* Этот метод основан на предложении посетителю более высокоуровневых (и, соответственно, более дорогих) товаров, чем те, которые он выбрал. Например, в ресторане можно предложить посетителю взять порцию большего размера — это может заметно увеличить доходы.
- *Продажа дополнительных товаров.* Этот метод предлагает посетителям дополнительные товары, исходя из анализа тех товаров, которые они купили или просмотрели.

- *Рекламируемые товары на главной странице.* TShirtShop уже позволяет администраторам сайта выбирать товары, которые должны отображаться на главной странице и страницах отделов.

В этой главе мы реализуем систему динамических рекомендаций для продажи как более дорогих, так и дополнительных товаров. Поскольку база данных `tshirtshop` уже хранит сведения о том, какие товары были проданы, в этой главе мы реализуем механизм "посетители, купившие данный товар, также купили вот это". Его преимущество состоит в том, что он не требует постоянной поддержки от администраторов. Наш сайт будет автоматически помогать нам увеличивать продажи, не требуя для этого никаких усилий с нашей стороны.

Как уже упоминалось, мы создадим систему динамических рекомендаций для корзины покупателя и для страниц со сведениями о товарах. После того как мы добавим в наш проект все необходимые элементы, на страницах со сведениями о товарах будут отображаться списки рекомендуемых товаров, как на рис. 15.1.

На странице корзины покупателя будет присутствовать похожий список рекомендаций (см. рис. 15.2).



Рис. 15.1. Список рекомендуемых товаров на странице сведений о товаре

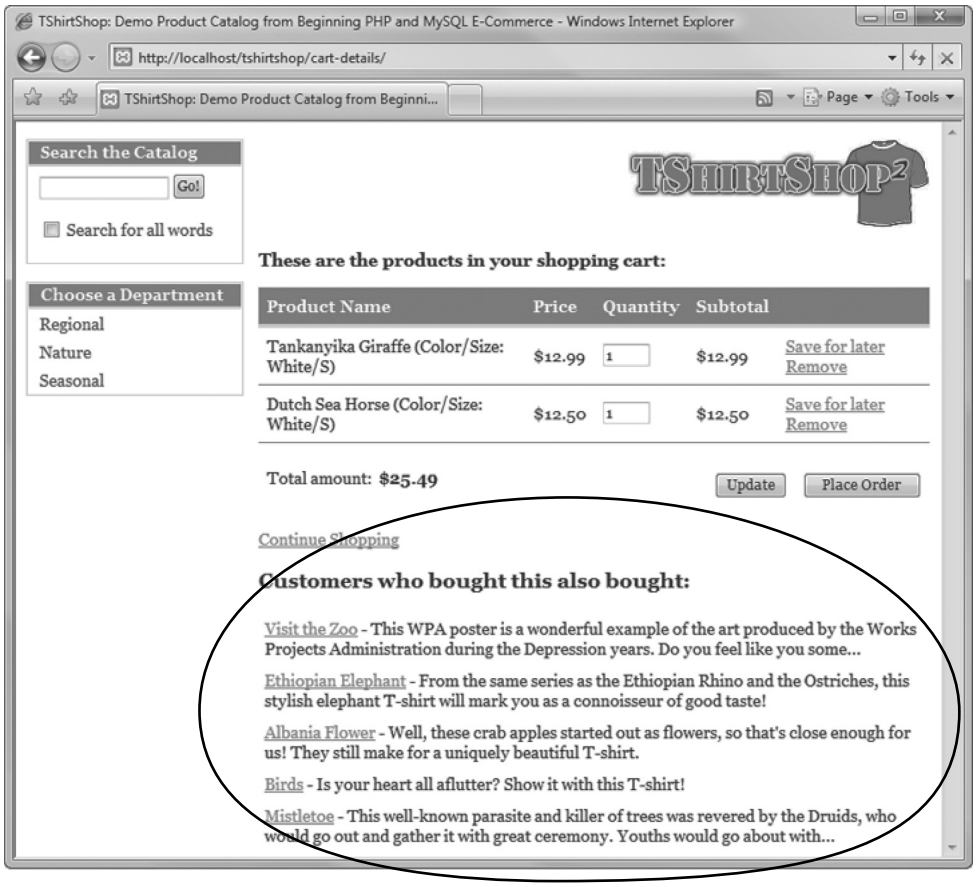


Рис. 15.2. Список рекомендуемых товаров на странице корзины покупателя

Выбор рекомендаций из базы данных

С технической точки зрения основная сложность в системе рекомендаций — это составление запросов, которые будут выбирать из базы данных рекомендуемые товары. Написание запроса `SELECT` для поиска товаров, продававшихся вместе с заданным товаром (для рекомендации на странице товара) или вместе с заданным набором товаров (для рекомендации на странице корзины), — задача не слишком сложная, но и тривиальной ее назвать нельзя.

Начнем с рекомендаций для отдельных товаров. Нам нужно выяснить, какие товары покупались вместе с заданным товаром.

В нашей таблице `order_detail` содержатся данные по всем сделанным в магазине заказам. Чтобы определить, какие товары заказывались вместе с заданным, нужно будет соединить два экземпляра таблицы `order_detail` по их полям `order_id`. Соединение нескольких экземпляров одной и той же таблицы ничем не отличается от соединения разных таблиц, в которых есть одинаковые столбцы (если вы подзабыли, что такое соединение, можете вернуться к главе 5 и перечитать раздел “Соединения таблиц данных”.

Мы будем соединять два экземпляра таблицы `order_detail` (обозначенные `od1` и `od2`) по столбцу `order_id` и выбирать нужные записи по значению в столбце `product_id` таблицы `od1`. В результате из таблицы `od2` будут выбраны все товары, участвовавшие в заказах вместе с заданным товаром. Чтобы четче уяснить себе, как будет работать такой запрос, взгляните на рис. 15.3. На нем показаны два заказа (1 и 2), в каждом из которых заказано по 3 товара (1, 2, 3 и 2, 5, 7 соответственно). Выбирая заказ, в котором заказывался товар 3, мы легко выясняем, какие товары заказывались вместе с ним.

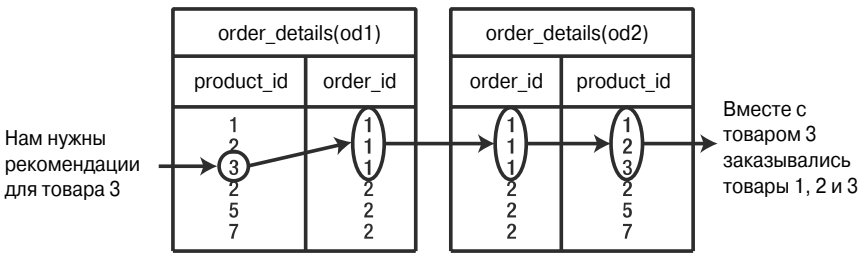


Рис. 15.3. Поиск товаров для рекомендаций в таблице `order_details`

Давайте посмотрим, как будет выглядеть такой запрос, реализованный в виде оператора SQL. Приведенный ниже оператор извлекает из таблицы все идентификаторы товаров (столбец `product_id`), заказанных вместе с товаром, `product_id` которого равен 4.

```
SELECT od2.product_id
FROM   order_detail od1
JOIN   order_detail od2
         ON od1.order_id = od2.order_id
WHERE od1.product_id = 4;
```

Этот оператор вернет длинный список идентификаторов товаров, в котором будет фигурировать и товар с идентификатором 4. Вот пример такого списка.

```
product_id
----------
4
5
10
43
4
5
10
23
25
28
4
10
12
14
43
```

Для начала такой результат неплох, но до идеала ему далеко. Во-первых, было бы неплохо, если бы этот список был отсортирован по частоте совместных заказов товаров вместе с заданным товаром. Чем чаще товары покупаются совместно, тем более обоснованной будет рекомендация. Как выполнить такую сортировку, мы покажем вам немного ниже.

Во-вторых, в списке не должен присутствовать сам товар, по которому мы выполняем поиск: он уже заказан, и рекомендовать его незачем. Эту проблему решить несложно — достаточно добавить в раздел WHERE еще одно правило.

```
SELECT od2.product_id
FROM   order_detail od1
JOIN   order_detail od2
         ON od1.order_id = od2.order_id
WHERE  od1.product_id = 4 AND od2.product_id != 4;
```

С помощью этого запроса мы получим список товаров, аналогичный приведенному выше, но товара с идентификатором 4 в нем уже не будет.

```
product_id
----------
5
10
43
5
10
23
25
28
10
12
14
43
```

Список заметно сократился, но в нем есть повторяющиеся записи (если товары заказывались вместе с заданным много раз, то и в результатах запроса они будут упоминаться многократно). Чтобы рекомендовать только товары, которые заказывались вместе с заданным чаще всего, нужно отсортировать список по частоте упоминания в нем разных элементов. Мы сделаем это, сгруппировав результаты предыдущего запроса по product_id и расположив их в порядке убывания в зависимости от частоты упоминания в списке. Частота будет указываться в столбце rank, значения для которого будут вычисляться в ходе выполнения запроса. Сам код запроса приведен ниже.

```
SELECT od2.product_id, COUNT(od2.product_id) AS rank
FROM   order_detail od1
JOIN   order_detail od2
         ON od1.order_id = od2.order_id
WHERE  od1.product_id = 4 AND od2.product_id != 4
GROUP BY od2.product_id
ORDER BY rank DESC;
```

ПОДСКАЗКА

Если между COUNT и последующим выражением будет присутствовать пробел, MySQL будет выдавать сообщения об ошибке. Будьте внимательны.

Теперь запрос будет возвращать список, в котором товары, чаще всего заказывавшиеся совместно с заданным, будут располагаться первыми.

```
product_id rank
---------- ----
        10    3
         5    2
        43    2
        23    1
```

```
        25    1
        28    1
        12    1
        14    1
```

Если саму частоту совместных заказов отображать не нужно, то функцию COUNT можно использовать прямо в разделе ORDER BY оператора SELECT. Кроме того, чтобы ограничить количество выводимых записей, можно воспользоваться ключевым словом LIMIT. Если вам нужно только 5 первых записей, подойдет такой запрос.

```
SELECT   od2.product_id
FROM     order_detail od1
JOIN     order_detail od2
           ON od1.order_id = od2.order_id
WHERE    od1.product_id = 4 AND od2.product_id != 4
GROUP BY od2.product_id
ORDER BY COUNT(od2.product_id) DESC
LIMIT 5;
```

Результаты его выполнения будут следующими.

```
product_id
----------
        10
         5
        43
        23
        25
```

ЗАМЕЧАНИЕ

Учтите, что для вас этот запрос может вернуть данные, отличающиеся от приведенных выше, даже если в вашей базе данных будут такие же данные, как у нас. Если критерии упорядочения в разделе ORDER BY недостаточно четки, СУБД будет выбирать метод сортировки, наиболее удобный с ее точки зрения. На практике это значит, что вы получите записи, которые были найдены первыми. Например, в нашем случае есть пять товаров со значением 1 в столбце rank (12, 14, 23, 25, 28). Когда мы выполнили этот запрос для своей базы данных, мы получили товары 23 и 25, но вы можете получить другие товары. Впрочем, это не слишком важно — если у нас много товаров с одинаковой частотой совместных заказов, выбирать из них может и сама СУБД.

Поскольку результаты выполнения запроса в виде столбца чисел не слишком понятны непосвященным наблюдателям, неплохо будет также выводить названия выбранных товаров. Для этого придется добавить в соединение таблицу product и выбрать требуемые значения из ее столбца product_name.

```
SELECT   od2.product_id, od2.product_name
FROM     order_detail od1
JOIN     order_detail od2 ON od1.order_id = od2.order_id
JOIN     product p ON od2.product_id = p.product_id
WHERE    od1.product_id = 4 AND od2.product_id != 4
GROUP BY od2.product_id
ORDER BY COUNT(od2.product_id) DESC
LIMIT 5;
```

Результаты, возвращаемые этим запросом, могут выглядеть примерно так.

```
product_id   name
----------   -----------------
        10   Haute Couture
         5   Marianne
        43   Equatorial Rhino
```

```
        23   Italian Airmail
        25   Romulus & Remus
```

Кроме того, можно учитывать при составлении рекомендаций только заказы, сделанные за n последних дней. Для этого придется использовать еще одно соединение с таблицей `orders`, в которой есть столбец `created_on`. Приведенный ниже запрос генерирует рекомендации только по заказам, созданным в течение последних 30 дней.

```
SELECT   od2.product_id, od2.product_name
FROM     order_detail od1
JOIN     order_detail od2 ON od1.order_id = od2.order_id
JOIN     orders o ON od1.order_id = o.order_id
WHERE    od1.product_id = 4 AND od2.product_id != 4
         AND DATE_SUB(NOW(), INTERVAL 30 DAY) < o.created_on
GROUP BY od2.product_id
ORDER BY COUNT(od2.product_id) DESC
LIMIT 5;
```

ЗАМЕЧАНИЕ

Мы не будем использовать в TShirtShop фильтрацию заказов по датам, но о такой возможности следует помнить. С течением времени в вашей базе данных будет накапливаться все больше заказов, и стоит ограничивать количество заказов, используемое для генерации рекомендаций — в противном случае рекомендации могут оказаться неактуальными, поскольку будут основаны на устаревших данных. Кроме того, если использовать для генерации рекомендаций все доступные заказы, с течением времени эта генерация будет требовать все больших затрат времени. В конце концов, стоит ли рекомендовать в июле покупать футболки с новогодней символикой? А как будут себя чувствовать ваши посетители, получая бессмысленные рекомендации после длинных пауз на обработку данных? Вероятнее всего, они вообще перестанут обращать внимание на рекомендации. Если вы решите использовать такой прием, выполнение запросов можно будет ускорить, добавив индекс для столбца `created_on`.

Чтобы получить рекомендации по товарам из корзины покупателя, мы применяем практически тот же метод, но выполняем поиск не по одному товару, а по нескольким поочередно. Это требует более сложного запроса, но никаких принципиальных трудностей не вызывает.

И последнее замечание перед тем, как мы начнем писать код. Предложенная нами реализация — всего лишь одна из бесконечного множества возможных. И в целом в программировании, и при работе с базами данных в частности, одного и того же результата можно достичь разными способами. Запрос, генерирующий рекомендации для товаров, тоже можно составить по-разному. Первое, что приходит в голову, — это использовать подзапросы вместо соединений таблиц.

На уровне данных механизм рекомендаций будет состоять из двух хранимых процедур: `catalog_get_recommendations` и `shopping_cart_get_recommendations`. В дополнение к данным, которые мы приводили выше, эти хранимые процедуры будут также извлекать названия и описания товаров, которые и будут показаны посетителю.

Упражнение: хранимые процедуры для механизма рекомендаций товаров

1. С помощью phpMyAdmin добавьте в базу данных `tshirtshop` хранимые процедуры, описанные в следующих шагах данного упражнения. Не забудьте задавать символы `$$` в качестве разделителя перед выполнением кода из каждого шага.
2. Выполните следующий оператор, создающий в базе данных хранимую процедуру `catalog_get_recommendations`.

```
-- Создаем хранимую процедуру catalog_get_recommendations
CREATE PROCEDURE catalog_get_recommendations(
  IN inProductId INT, IN inShortProductDescriptionLength INT)
BEGIN
  PREPARE statement FROM
    "SELECT od2.product_id, od2.product_name,
            IF(LENGTH(p.description) <= ?, p.description,
               CONCAT(LEFT(p.description, ?), '...')) AS description
     FROM     order_detail od1
     JOIN     order_detail od2 ON od1.order_id = od2.order_id
     JOIN     product p ON od2.product_id = p.product_id
     WHERE    od1.product_id = ? AND
              od2.product_id != ?
     GROUP BY od2.product_id
     ORDER BY COUNT(od2.product_id) DESC
     LIMIT 5";

  SET @p1 = inShortProductDescriptionLength;
  SET @p2 = inProductId;

  EXECUTE statement USING @p1, @p1, @p2, @p2;
END$$
```

3. Выполните следующий оператор, создающий в базе данных хранимую процедуру shopping_cart_get_recommendations.

```
-- Создаем хранимую процедуру shopping_cart_get_recommendations
CREATE PROCEDURE shopping_cart_get_recommendations(
  IN inCartId CHAR(32), IN inShortProductDescriptionLength INT)
BEGIN
  PREPARE statement FROM
    "-- Возвращает товары, присутствующие в списке заказов
     SELECT   od1.product_id, od1.product_name,
              IF(LENGTH(p.description) <= ?, p.description,
                 CONCAT(LEFT(p.description, ?), '...')) AS description
     FROM     order_detail od1
     JOIN     order_detail od2
                ON od1.order_id = od2.order_id
     JOIN     product p
                ON od1.product_id = p.product_id
     JOIN     shopping_cart
                ON od2.product_id = shopping_cart.product_id
     WHERE    shopping_cart.cart_id = ?
              -- Среди рекомендаций не должно быть товаров,
              -- уже присутствующих в корзине покупателя
              AND od1.product_id NOT IN
              (-- Возвращает товары, присутствующие
               -- в корзине покупателя
               SELECT product_id
               FROM   shopping_cart
               WHERE  cart_id = ?)
     -- Группируем по product_id, чтобы подсчитать частоту встречаемости
     GROUP BY od1.product_id
     -- Упорядочиваем по убыванию частоты
     ORDER BY COUNT(od1.product_id) DESC
     LIMIT 5";

  SET @p1 = inShortProductDescriptionLength;
  SET @p2 = inCartId;

  EXECUTE statement USING @p1, @p1, @p2, @p2;
END$$
```

Как это работает: получение рекомендаций товаров из базы данных

Только что созданные хранимые процедуры просто реализуют методики, рассмотренные нами выше. В других редакциях данной книги (для PostgreSQL и SQL Server) мы рассматривали также альтернативные варианты реализации этих хранимых процедур — на основе подзапросов. В MySQL использование подзапросов невозможно из-за ограничений на применение `LIMIT`, описанных в статье по адресу `http://dev.mysql.com/doc/refman/5.1/en/subquery-errors.html`. Если в последующих версиях MySQL эти ограничения будут сняты, можете попробовать переписать приведенные выше хранимые процедуры с помощью подзапросов. В некоторых случаях это позволяет заметно повысить производительность.

Реализация рекомендаций для товаров

В следующем упражнении мы напишем код уровня логики приложения и уровня представления для системы рекомендации товаров.

Уровень логики приложения будет состоять из двух методов с одним названием `GetRecommendations()` одного — в классе `Catalog` и одного — в классе `ShoppingCart`. Первый генерирует рекомендации для страниц с описаниями товаров, а второй — для страницы корзины покупателя.

Упражнение: реализация рекомендаций для товаров

1. Добавьте в класс `Catalog` в файле `business/catalog.php` следующий метод.

```
// Получаем рекомендации для товаров
public static function GetRecommendations($productId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_recommendations(
                :product_id, :short_product_description_length)';

  // Создаем массив параметров
  $params = array (':product_id' => $productId,
                   ':short_product_description_length' =>
                     SHORT_PRODUCT_DESCRIPTION_LENGTH);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetAll($sql, $params);
}
```

2. Добавьте в класс `ShoppingCart` в файле `business/shopping_cart.php` следующий метод.

```
// Получаем рекомендации для товаров из корзины покупателя
public static function GetRecommendations()
{
  // Составляем SQL-запрос
  $sql = 'CALL shopping_cart_get_recommendations(
                :cart_id, :short_product_description_length)';

  // Создаем массив параметров
  $params = array (':cart_id' => self::GetCartId(),
                   ':short_product_description_length' =>
                     SHORT_PRODUCT_DESCRIPTION_LENGTH);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetAll($sql, $params);
}
```

3. Займитесь уровнем представления. Модифицируйте компонентные шаблоны `product` и `cart_details` для отображения рекомендаций. Откройте файл `presentation/product.php` и добавьте в класс `Product` новый элемент с именем `$mRecommendations`.

```
// Public-переменные для использования в шаблоне Smarty
public $mProduct;
public $mProductLocations;
public $mLinkToContinueShopping;
public $mLocations;
public $mEditActionTarget;
public $mShowEditButton;
public $mRecommendations;
```

4. Теперь нужно организовать запись рекомендованных товаров в `$mRecommendations` и создать ссылки на их страницы. Модифицируйте метод `init()` класса `Product` так, как показано ниже:

```
$this->mLocations = Catalog::GetProductLocations($this->_mProductId);
// Генерируем ссылку Add to Cart link
$this->mProduct['link_to_add_product'] =
  Link::ToCart(ADD_PRODUCT, $this->_mProductId);

// Получаем рекомендации товаров
$this->mRecommendations =
  Catalog::GetRecommendations($this->_mProductId);

// Генерируем ссылки на рекомендованные товары
for ($i = 0; $i < count($this->mRecommendations); $i++)
  $this->mRecommendations[$i]['link_to_product'] =
    Link::ToProduct($this->mRecommendations[$i]['product_id']);

// Генерируем ссылки на страницы отдела и категориии
for ($i = 0; $i < count($this->mLocations); $i++)
{
```

5. Чтобы завершить реализацию системы рекомендаций для страниц товаров, обновите шаблон `product.tpl`. Добавьте приведенный ниже код в конец файла `presentation/templates/product.tpl`. После этого можно загрузить сайт TShirtShop в браузер и открыть страницу какого-нибудь товара. Если этот товар ранее заказывался вместе с другими товарами, эти товары будут отображены в списке рекомендаций.

```
{if $obj->mRecommendations}
<h2>Customers who bought this also bought:</h2>
<ol>
  {section name=m loop=$obj->mRecommendations}
  <li>
    {strip}
    <a href="{$obj->mRecommendations[m].link_to_product}">
      {$obj->mRecommendations[m].product_name}
    </a>
    {/strip}
    <span class="list"> - {$obj->mRecommendations[m].description}</span>
  </li>
  {/section}
</ol>
{/if}
```

6. Откройте файл `tshirtshop.css` и добавьте в него следующий стиль:

```
.list { color: #000000; }
```

7. А теперь займемся корзиной покупателя. Модифицируйте компонентный шаблон cart_details для отображения рекомендаций на странице корзины. Для начала откройте файл cart_details.php в папке presentation и добавьте переменную $mRecommmendations в определение класса CartDetails:

```
<?php
// Класс, отвечающий за работу корзины покупателя
class CartDetails
{
  // Public-переменные, доступные в шаблоне Smarty
  public $mCartProducts;
  public $mSavedCartProducts;
  public $mTotalAmount;
  public $mIsCartNowEmpty = 0; // Корзина пуста?
  public $mIsCartLaterEmpty = 0; // Список отложенных товаров пуст?
  public $mLinkToContinueShopping;
  public $mUpdateCartTarget;
  public $mRecommendations;
```

8. Обеспечьте получение рекомендаций товаров в переменной $mRecommendations и сгенерируйте ссылки на страницы этих товаров. Добавьте код, выделенный полужирным шрифтом в следующем листинге, в конец метода init() класса CartDetails.

```
    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']);
    }

    // Получаем рекомендации для товаров
    $this->mRecommendations =
      ShoppingCart::GetRecommendations();

    // Генерируем ссылки на рекомендованные товары
    for ($i = 0; $i < count($this->mRecommendations); $i++)
      $this->mRecommendations[$i]['link_to_product'] =
        Link::ToProduct($this->mRecommendations[$i]['product_id']);
  }
}
?>
```

9. Обновите шаблон cart_details для отображения списка рекомендаций. Добавьте следующий код в конец файла presentation/templates/cart_details.tpl.

```
{if $obj->mRecommendations}
<h2>Customers who bought this also bought:</h2>
<ol>
  {section name=m loop=$obj->mRecommendations}
  <li>
    {strip}
    <a href="{$obj->mRecommendations[m].link_to_product}">
      {$obj->mRecommendations[m].product_name}
    </a>
    {/strip}
    <span class="list"> - {$obj->mRecommendations[m].description}</span>
  </li>
```

```
  {/section}
</ol>
{/if}
```

10. Загрузите сайт TShirtShop в браузер и создайте несколько заказов (в каждом из заказов должно быть более одного товара). После этого на страницах товаров и на странице корзины покупателя должны появиться рекомендации товаров, как показано на рис. 15.1 и 15.2.

Резюме

Отображение рекомендаций товаров — эффективный способ повышения продаж, и мы успешно реализовали механизм рекомендаций в этой короткой главе. Самое сложное здесь было — составить SQL-запрос, возвращающий товары для рекомендаций. Мы разобрались, как должен работать такой запрос, и успешно его создали.

Следующая глава начинает третью стадию разработки нашего проекта. В ней мы займемся созданием системы учетных записей пользователей.

Часть III

Третий этап разработки

Глава 16

Учетные записи клиентов

К данному моменту у нас есть хотя и несложный, но работающий сайт, который использует PayPal для приема платежей и подтверждения заказов. На третьей стадии разработки мы продвинемся дальше — избавимся от необходимости работать с PayPal и тем самым получим возможность гораздо эффективнее контролировать заказы. Это не так сложно, как вам может показаться, но нужно быть внимательными, чтобы сделать все правильно.

В этой главе мы заложим фундамент собственного механизма приема платежей — создадим систему учетных записей клиентов.

Чтобы сделать сайты электронных магазинов более дружественными по отношению к клиентам, данные наподобие номеров кредитных карточек обычно хранятся в базах данных, чтобы клиентам не приходилось при оформлении каждого заказа вводить всю информацию заново. Система учетных записей клиентов как раз и предназначена для организованного ввода и хранения этой информации. Ее интерфейс будет представлять собой специальные веб-страницы с полями для ввода данных.

Кроме этих веб-страниц нужно издать еще кое-что. Учтите, что, например, хранить незашифрованными в базе данных номера кредитных карточек и другую информацию о клиентах рискованно: ею могут воспользоваться злоумышленники, просто украв сервер или взломав ваш сайт. Чтобы ограничить нежелательный доступ к этой информации, проще всего хранить ее зашифрованной и расшифровывать только при необходимости. Мы создадим небольшую библиотеку, которая будет содержать все нужные для этого функции.

Важно также создать защищенные от перехвата данных каналы связи сервера с клиентами. Простые формы HTML и протокол HTTP не обеспечивают защиты передаваемых через Интернет данных от перехвата. Поэтому мы применим SSL и HTTPS-соединения.

В этой главе мы

- создадим систему для работы с учетными записями клиентов,
- реализуем классы для системы безопасности,
- добавим в TShirtShop функциональность учетных записей,
- создадим страницу приема платежей.

Хранение учетных записей клиентов

Организовать систему для работы с учетными записями можно самыми разными способами. В целом она должна содержать следующие компоненты.

- Механизм аутентификации, разрешающий клиентам доступ к защищенным частям сайта
- Механизм поддержки сеансов, который "помнит", что клиент прошел аутентификацию, и предоставляет этому клиенту соответствующие права до окончания сеанса работы с нашим сайтом
- Механизмы проверки на всех защищенных страницах, которые не позволяют клиенту просматривать эти страницы без аутентификации

Сначала мы рассмотрим общие принципы реализации всего этого на сайте TShirtShop.

Схема учетных записей клиентов TShirtShop

Простейший способ определить, прошел ли посетитель аутентификацию — проверять, есть ли его идентификатор в сеансовой переменной. Можно проверять значение этой переменной при попытках открыть защищенные страницы и выдавать пользователю сообщение, если он не прошел аутентификацию. Сохранять значение в сеансовой переменной может страница аутентификации, если пользователь успешно пройдет аутентификацию. Когда пользователь завершает сеанс работы с сайтом, можно просто удалить значение из сеансовой переменной.

Чтобы пройти аутентификацию, пользователь должен сообщить свое имя (мы будем вместо имени использовать адрес электронной почты, поскольку он наверняка будет уникальным) и пароль. Передавать эту информацию через Интернет в незашифрованном виде рискованно, поскольку ее можно легко перехватить. Позже в этой главе мы обсудим варианты безопасной передачи данных через Интернет, но пока что давайте сосредоточимся на аутентификации, механизм которой не зависит от способа передачи данных.

Еще один момент, связанный с безопасностью, — это способ хранения имен пользователей и паролей на сервере. Если хранить такие сведения незашифрованными, то они будут слишком легкой добычей для злоумышленников. Вместо пароля можно хранить его *хеш-код* (hash code). Хеш — это строка, полученная из пароля путем выполнения определенных операций. Из этой строки невозможно восстановить исходный пароль. Чтобы проверить, правильно ли пользователь ввел пароль, мы просто сгенерируем хеш-код введенного пароля и сравним его с хранимым на сервере. Если хеш-коды совпадут, значит, пароль введен правильно, и пользователь прошел аутентификацию.

На сайте также должен присутствовать механизм регистрации новых пользователей. В результате регистрации информация о новом пользователе должна быть добавлена в базу данных, включая его адрес электронной почты и хеш-код пароля.

Для реализации всего вышеперечисленного в нашем проекте понадобится сделать следующее.

- Создать две таблицы в базе данных: `customer` — для хранения сведений о пользователях и `shipping_region` — для хранения регионов, в которые нужно будет отправлять заказы
- Реализовать на уровне данных и уровне логики приложения методы, которые будут работать с этими таблицами

- Модифицировать компонентный шаблон `cart_details`, который теперь будет перенаправлять пользователя на страницу приема платежей, реализованную с помощью нового компонентного шаблона `checkout_info`
- Создать компонентный шаблон `customer_login` для страницы аутентификации
- Создать компонентный шаблон `customer_details` для регистрации новых пользователей и редактирования сведений в учетных записях пользователей
- Создать компонентный шаблон `customer_credit_card`, позволяющий пользователям вводить сведения о кредитных картах
- Создать компонентный шаблон `customer_credit_card`, позволяющий пользователям вводить адреса доставки

Создание таблиц `customer` и `shipping_region`

Чтобы создать таблицы `customer` и `shipping_region`, выполните следующее упражнение.

Упражнение: создание таблиц в базе данных

1. Загрузите phpMyAdmin, откройте базу данных `tshirtshop` и перейдите на страницу нового запроса SQL.
2. Выполните следующий запрос, чтобы создать в базе данных таблицу `shipping_region`.

```
-- Создаем таблицу shipping_region
CREATE TABLE `shipping_region` (
  `shipping_region_id` INT NOT NULL AUTO_INCREMENT,
  `shipping_region` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`shipping_region_id`)
);
```

3. Добавьте в таблицу `shipping_region` значения `"Please select"`, `"US / Canada"`, `"Europe"` и `"Rest of the World"`. Строка `"Please select"` должна содержать значение 1 в столбце `shipping_region_id` (это важно!). Выполните следующий оператор SQL, чтобы добавить эти значения в таблицу.

```
-- Заполняем таблицу shipping_region
INSERT INTO `shipping_region` (`shipping_region_id`, `shipping_region`)
VALUES
       (1, 'Please Select') , (2, 'US / Canada'),
       (3, 'Europe'), (4, 'Rest of World');
```

4. Выполните следующий оператор, чтобы добавить в базу данных таблицу `customer`.

```
-- Создаем таблицу customer
CREATE TABLE `customer` (
  `customer_id`        INT           NOT NULL  AUTO_INCREMENT,
  `name`               VARCHAR(50)   NOT NULL,
  `email`              VARCHAR(100)  NOT NULL,
  `password`           VARCHAR(50)   NOT NULL,
  `credit_card`        TEXT,
  `address_1`          VARCHAR(100),
  `address_2`          VARCHAR(100),
  `city`               VARCHAR(100),
  `region`             VARCHAR(100),
  `postal_code`        VARCHAR(100),
```

```
  `country`             VARCHAR(100),
  `shipping_region_id` INTEGER       NOT NULL   DEFAULT 1,
  `day_phone`           VARCHAR(100),
  `eve_phone`           VARCHAR(100),
  `mob_phone`           VARCHAR(100),
  PRIMARY KEY (`customer_id`),
  UNIQUE KEY `idx_customer_email` (`email`),
  KEY `idx_customer_shipping_region_id` (`shipping_region_id`)
);
```

Информация о кредитных картах будет храниться на сервере в зашифрованном виде, чтобы усложнить нежелательный доступ к ней. Однако, в отличие от паролей, эту информацию нужно будет расшифровывать при работе с заказами, поэтому ее нельзя просто хешировать (из хеш-кодов нельзя восстановить исходные данные). Шифрование информации о кредитных картах будет обеспечивать уровень логики приложения, которым мы займемся ниже.

Реализация классов, отвечающих за безопасность

На данном этапе защищать необходимо два компонента сведений о пользователях.

- Пароли
- Информацию о кредитных картах

Для их защиты мы создадим на уровне логики приложения новые классы, которые будут храниться в папке `business` в следующих файлах.

- `password_hasher.php`. Содержит класс `PasswordHasher` со статическим методом `Hash()`, возвращающим хеш-код заданного пароля.
- `secure_card.php`. Содержит класс `SecureCard`, предназначенный для работы с кредитными картами и шифрования сведений о них. Он обеспечивает как шифрование, так и расшифровку данных.
- `symmetric_crypt.php`. Содержит класс `SymmetricCrypt`, используемый классом `SecureCard` для операций шифрования и расшифровки. Это значит, что при необходимости изменить метод шифрования менять придется только код в классе `SymmetricCrypt` — класс `SecureCard` можно будет оставить в покое.

Сначала мы рассмотрим хеширование, а затем — шифрование.

Реализация функциональности хеширования на уровне логики приложения

Хеширование позволяет получить код, однозначно указывающий на хешируемый объект. Существует много алгоритмов, преобразующих байтовый массив в хеш-код. Среди часто используемых алгоритмов следует упомянуть MD5 (Message Digest 5), генерирующий 128-битовый хеш-код. К сожалению, широкое применение этого алгоритма привело к появлению специальных инструментов для подбора хеш-кодов, генерируемых с его помощью.

Еще один популярный алгоритм хеширования — SHA1 (Secure Hashing Algorithm 1), генерирующий 160-битовые хеш-коды. Этот алгоритм считается более

надежным, чем MD5, но он медленнее работает. Если вас интересует более подробная информация о MD5 и SHA1, прежде всего обратитесь к Википедии.

В TShirtShop мы будем использовать алгоритм SHA1, хотя вместо него можно с легкостью задействовать любой другой. Чтобы создать класс `PasswordHasher`, выполните инструкции из следующего упражнения.

Упражнение: реализация класса `PasswordHasher`

Чтобы создать класс `PasswordHasher`, выполните следующие действия.

1. Добавьте две приведенные ниже строки в конец файла `include/config.php`. Во второй строке определено произвольное значение (можете изменить его, если хотите), добавляемое к задаваемым паролям перед хешированием.

```
// Произвольное значение, добавляемое в строку перед хешированием
define('HASH_PREFIX', 'K1-');
```

2. Создайте файл `password_hasher.php` в папке `business` и вставьте в него следующий код.

```
<?php
class PasswordHasher
{
  public static function Hash($password, $withPrefix = true)
  {
    if ($withPrefix)
      $hashed_password = sha1(HASH_PREFIX . $password);
    else
      $hashed_password = sha1($password);

    return $hashed_password;
  }
}
?>
```

3. Создайте простую тестовую страницу для проверки работы класса `PasswordHasher`. Создайте в папке `tshirtshop` файл `test_hasher.php` и поместите в него следующий код.

```
<?php
if (isset ($_GET['to_be_hashed']))
{
  require_once 'include/config.php';
  require_once BUSINESS_DIR . 'password_hasher.php';

  $original_string = $_GET['to_be_hashed'];

  echo 'The hash of "' . $original_string . '" is ' .
       PasswordHasher::Hash($original_string, false);

  echo '<br />';

  echo '... and the hash of "' . HASH_PREFIX . $original_string .
       '" (secret prefix concatenated with password) is ' .
       PasswordHasher::Hash($original_string, true);
}
?>

<br /><br />
<form action="test_hasher.php">
```

```
  Write your password:
  <input type="text" name="to_be_hashed" /><br />
  <input type="submit" value="Hash it" />
</form>
```

4. Загрузите страницу `test_hasher.php` в браузер, введите в текстовое поле пароль, для которого хотите сгенерировать хеш-код, и щелкните на кнопке Hash it. Результат должен быть таким, как на рис. 16.1.

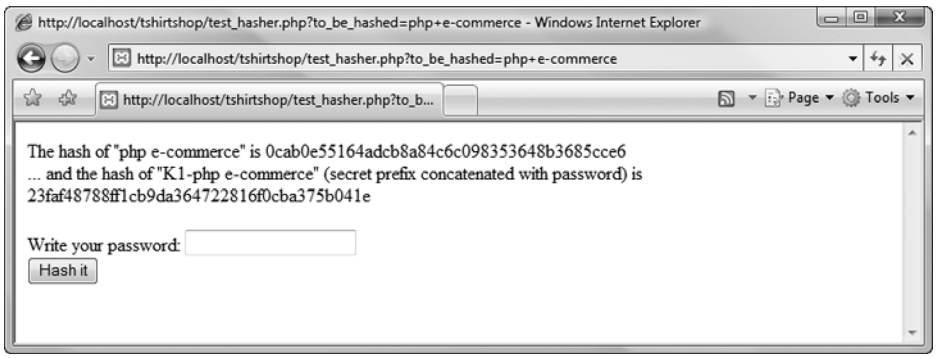


Рис. 16.1. Тестирование функции хеширования паролей

Как это работает: хеширование паролей

Код класса `PasswordHasher` не содержит ничего сложного. По умолчанию статический метод `Hash()` возвращает хеш-значение пароля, в начало которого добавлен заданный секретный префикс.

Возможно, вам интересно, зачем добавлять к паролю префикс перед хешированием. Ответ прост: чтобы повысить надежность паролей, усложнив их подбор. Если вашу базу данных украдут, воры могут попытаться подобрать пароли по хеш-значениям, перебирая в качестве паролей слова из словаря и сравнивая получающиеся хеш-коды с хеш-кодами из базы данных. Если какие-то два хеш-кода совпадут, значит, совпадают и пароли, для которых они сгенерированы. Добавляя секретный префикс к заданным пользователями паролям, мы усложняем подбор паролей с помощью словарной атаки (т.е. перебора слов из словаря).

Страница `test_hasher.php` служит только для тестирования класса `PasswordHasher`.

ЗАМЕЧАНИЕ

Можно реализовать хеширование и на уровне базы данных с помощью функций MySQL `PASSWORD()`, `MD5()` и `SHA1()`. Например, чтобы посмотреть, как работает функция `SHA1()`, выполните следующий оператор SQL:

```
SELECT SHA1('freedom');
```

Но у такого подхода есть недостаток. Если сервер базы данных находится не на том же компьютере, что и веб-сервер, пароли будут передаваться с веб-сервера на сервер базы данных по сети в незашифрованном виде, если вы не организуете связь между серверами с помощью SSL. Кроме того, подход с хешированием паролей в PHP-коде более гибкий, поскольку не полагается на использование MySQL. Функции шифрования MySQL описаны в его документации, доступной по адресу `http://dev.mysql.com/doc/refman/5.1/en/encryption-functions.html`.

Реализация функциональности шифрования на уровне логики приложения

Существует множество методов шифрования данных. Эта область сейчас бурно развивается, и новые методы появляются чуть ли не каждый день. Универсального метода шифрования не существует, но для каждой ситуации можно подобрать свой удобный метод. В целом все методы шифрования делятся на две категории.

- *Симметричные методы.* Методы, в которых один и тот же ключ используется и для шифрования, и для расшифровки.
- *Асимметричные методы.* Методы, в которых для шифрования и расшифровки используются разные ключи. Ключ для шифрования обычно называется *открытым ключом* (public key), а для расшифровки — *закрытым ключом* (private key). Закрытым ключом нельзя зашифровать данные — он пригоден только для расшифровки данных, зашифрованных открытым ключом. Открытый и закрытый ключи математически связаны между собой и всегда генерируются вместе. Закрытый ключ нельзя получить из открытого, как и открытый из закрытого. Если у вас есть пара из открытого и закрытого ключей, вы можете разослать открытый ключ своим корреспондентам, и они будут шифровать с его помощью предназначенные для вас сообщения. Поскольку закрытый ключ будет известен только вам, только вы сможете расшифровывать эти сообщения.

Хотя асимметричное шифрование более надежно, оно требует гораздо больших ресурсов для шифрования и расшифровки. Симметричное шифрование выполняется быстрее, но оно менее надежно — при его использовании отправитель информации должен знать ключ, а передать этот ключ отправителю так, чтобы он не был перехвачен, не так-то просто.

Асимметричное шифрование позволяет устранить эту проблему за счет использования пар ключей. Закрытый ключ никому не нужно сообщать, поэтому посторонним лицам будет гораздо труднее прочитать зашифрованные сообщения. Однако из-за медлительности работы методов асимметричного шифрования обычно используется комбинированный метод: асимметричное шифрование применяется только для передачи через Интернет ключей для симметричного шифрования, после чего ключи для симметричного шифрования используются для шифрования всех сообщений.

В TShirtShop положение несколько проще, чем при передаче сообщений через Интернет. Достаточно зашифровывать данные перед их сохранением в базе данных и расшифровывать при чтении из базы данных. Для этого вполне подойдет обычный симметричный алгоритм.

ЗАМЕЧАНИЕ

В любом случае мы незаметно для себя используем асимметричное шифрование — оно применяется в протоколе HTTPS.

Как и для хеширования, для симметричного и асимметричного шифрования существует много алгоритмов. В библиотеке PHP `mcrypt` есть реализации всех наиболее распространенных алгоритмов симметричного шифрования. Для асимметричного шифрования стандартной библиотеки PHP пока нет, но если оно вам понадобится, можете воспользоваться семейством программ PGP (Pretty Good Privacy; `http://www.pgp.com`) или GnuPG (`http://www.gnupg.org`).

Наиболее распространенные алгоритмы асимметричного шифрования — DSA (Digital Signature Algorithm — алгоритм цифровой подписи) и RSA (по фамилиям

изобретателей: Rivest, Shamir и Adleman). DSA можно использовать только для создания цифровых подписей под документами для проверки их подлинности, а RSA более универсален (хотя работает медленнее, чем DSA). В настоящее время DSA используется правительством США как стандарт цифровой подписи. Алгоритмы DSA и RSA реализованы в семействах программ PGP и GnuPG.

Среди популярных алгоритмов симметричного шифрования, доступных в библиотеке `mcrypt`, можно упомянуть DES (Data Encryption Standard), 3DES (Triple DES), RC2 (Ron's Code или Rivest Cipher; создан Роном Райвестом (Ron Rivest)) и Rijndael (по фамилиям создателей: Daemen и Rijmen).

DES и Rijndael

DES до недавнего времени был стандартным алгоритмом шифрования, но сейчас на смену ему постепенно приходят другие алгоритмы. В нем используется 64-битовый ключ, но непосредственно в шифровании участвуют всего 56 бит (остальные 8 бит используются только для контроля). Ключ такой длины можно подобрать с помощью современных компьютеров за короткое время.

Triple DES и RC2 — это модификации DES, предназначенные для повышения стойкости к взлому путем подбора ключа. Triple DES использует трехкратное шифрование DES с эффективной длиной ключа 168 бит. RC2 может использовать ключи длиной до 128 бит (более длинные ключи доступны в вариантах RC3, RC4 и т.д.).

Rijndael — это радикально отличающийся от DES алгоритм шифрования, который на данный момент используется как стандарт AES (Advanced Encryption System — усовершенствованная система шифрования). Он был выбран в качестве стандарта из нескольких алгоритмов, выступавших кандидатами на роль замены для DES, и постепенно получает все большее распространение.

Процедуры для шифрования и расшифровки данных несколько сложнее, чем для хеширования. Функции из библиотеки `mcrypt` оптимизированы для работы с чистыми данными, поэтому нам придется немного повозиться с преобразованием данных. Кроме того, нужно будет задать ключ и начальный вектор (IV) для шифрования и расшифровки. Потребность в начальном векторе вызвана природой алгоритмов — блоки данных обычно шифруются последовательно друг за другом, и вычисление шифра для каждого блока требует определенных данных о предыдущем блоке. Поскольку у первого блока предыдущего просто нет, вместо предыдущего блока используется начальный вектор. Для шифрования по алгоритму AES (Rijndael_128) вектор и ключ должны быть длиной 32 бит (256 бит).

ЗАМЕЧАНИЕ

Если вам интересно, как работает AES, обратитесь к Википедии — в статье по адресу `http://ru.wikipedia.org/Advanced_Encryption_Standard` содержится подробное описание алгоритма.

В целом для шифрования строки необходимо выполнить следующие операции.

1. Создать 32-байтовый уникальный начальный вектор.
2. Преобразовать начальный вектор (хранимый в виде шестнадцатеричной строки) в байтовый массив.
3. Зашифровать строку с помощью алгоритма AES, передав ему начальный вектор в виде байтового массива.
4. Преобразовать полученные зашифрованные данные из байтового массива в шестнадцатеричную строку.

Для расшифровки используется похожая процедура.

1. Преобразовать начальный вектор (хранимый в виде шестнадцатеричной строки) в байтовый массив аналогично п. 2 для процедуры шифрования.

2. Преобразовать строку, подлежащую расшифровке, в байтовый массив.
3. Расшифровать строку из предыдущего шага, применяя начальный вектор к байтовому массиву.

В следующем приложении мы будем использовать алгоритм AES, но при желании можно задать другой алгоритм в классе `SymmetricCrypt`.

Упражнение: реализация класса `SymmetricCrypt`

1. Добавьте в папку `business` файл `symmetric_crypt.php` и поместите в него следующий код.

```
<?php
class SymmetricCrypt
{
  // Ключ шифрования/расшифровки
  private static $_msSecretKey = 'From Dusk Till Dawn';

  // Начальный вектор
  private static $_msHexaIv = 'c7098adc8d6128b5d4b4f7b2fe7f7f05';

  // Используем алгоритм AES
  private static $_msCipherAlgorithm = MCRYPT_RIJNDAEL_128;

  /* Функция шифрует текстовую строку, полученную в качестве параметра,
     и возвращает результат в шестнадцатеричной форме */
  public static function Encrypt($plainString)
  {
    // Упаковка SymmetricCrypt::_msHexaIv в двоичную строку
    $binary_iv = pack('H*', self::$_msHexaIv);

    // Шифрование $plainString
    $binary_encrypted_string = mcrypt_encrypt(
                                self::$_msCipherAlgorithm,
                                self::$_msSecretKey,
                                $plainString,
                                MCRYPT_MODE_CBC,
                                $binary_iv);

    // Преобразование $binary_encrypted_string в шестнадцатеричную форму
    $hexa_encrypted_string = bin2hex($binary_encrypted_string);

    return $hexa_encrypted_string;
  }

  /* Функция расшифровывает шестнадцатеричную строку, полученную
     в качестве параметра, и возвращает результат в шестнадцатеричной фор-
ме */
  public static function Decrypt($encryptedString)
  {
    // Упаковка Symmetric::_msHexaIv в двоичную строку
    $binary_iv = pack('H*', self::$_msHexaIv);

    // Преобразование строки в байтовый массив
    $binary_encrypted_string = pack('H*', $encryptedString);

    // Расшифровка строки $binary_encrypted_string
    $decrypted_string = mcrypt_decrypt(
                          self::$_msCipherAlgorithm,
                          self::$_msSecretKey,
```

```
                                $binary_encrypted_string,
                                MCRYPT_MODE_CBC,
                                $binary_iv);

    return $decrypted_string;
  }
}
?>
```

2. Добавьте в папку `tshirtshop` тестовый файл `test_encryption.php` и поместите в него следующий код.

```
<?php
if (isset ($_GET['my_string']))
{
  require_once 'include/config.php';
  require_once BUSINESS_DIR . 'symmetric_crypt.php';

  $string = $_GET['my_string'];

  echo 'The string is:<br />' . $string . '<br /><br />';

  $encrypted_string = SymmetricCrypt::Encrypt($string);

  echo 'Encrypted string: <br />' . $encrypted_string . '<br /><br />';

  $decrypted_string = SymmetricCrypt::Decrypt($encrypted_string);

  echo 'Decrypted string:<br />' . $decrypted_string;
}
?>

<br /><br />
<form action="test_encryption.php">
  Enter string to encrypt:
  <input type="text" name="my_string" /><br />
  <input type="submit" value="Encrypt" />
</form>
```

3. Загрузите файл `test_encryption.php` в браузер и задайте строку для шифрования (как показано на рис. 16.2).

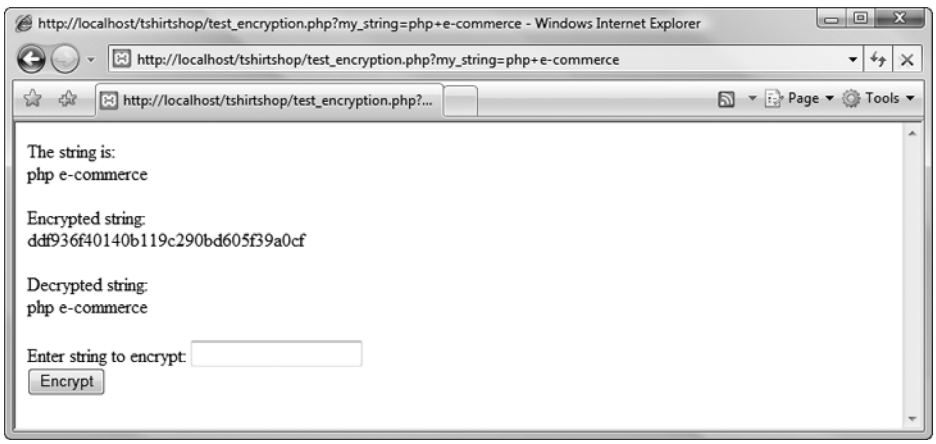


Рис. 16.2. Тестирование функций шифрования и расшифровки

ЗАМЕЧАНИЕ

Обычно сразу после установки XAMPP не сконфигурирован на использование библиотеки `mcrypt`, и, попытавшись выполнить это упражнение, вы получите сообщение об ошибке при вызове функции `mcrypt_encrypt()`. Чтобы сконфигурировать PHP на загрузку библиотеки `mcrypt`, раскомментируйте в файле `php.ini` строку

; extension=php_mcrypt.dll

(Для этого удалите точку с запятой в начале строки.) Файл `php.ini` находится в папке `C:\xampp\apache\bin` в Windows или в папке `/opt/lampp/etc` в Linux. Внеся изменения в файл `php.ini`, перезапустите Apache.

ВНИМАНИЕ

Длина зашифрованной строки будет всегда кратна 32 байт. Если исходная строка была короче 32 байт, перед шифрованием она дополняется нулями. Об этом следует помнить, поскольку расшифрованная строка может оказаться не точно идентичной исходной. В нашем проекте мы будем шифровать XML-данные, находящиеся между тегами, поэтому дополнительные нули в конце строк нам не помешают.

Как это работает: функциональность шифрования на уровне логики приложения

В классе `SymmetricCrypt` есть два статических метода — `Encrypt()` и `Decrypt()`, для шифрования и расшифровки данных соответственно, а также ряд статических полей для хранения параметров шифрования.

```
// Ключ шифрования/расшифровки
private static $_msSecretKey = 'From Dusk Till Dawn';

// Начальный вектор
private static $_msHexaIv = 'c7098adc8d6128b5d4b4f7b2fe7f7f05';

// Используем алгоритм AES
private static $_msCipherAlgorithm = MCRYPT_RIJNDAEL_128;
```

Ключ шифрования для алгоритма AES имеет длину 16 байт. Библиотека `mcrypt` позволяет использовать ключи меньшей длины, но это приведет к снижению надежности шифра. Начальный вектор должен иметь длину точно 16 байт; он хранится в программе в виде шестнадцатеричной строки длиной 32 знака (каждому байту соответствуют 2 знака). Переменным `$_msSecretKey` и `$_msHexaIv` в коде, приведенном выше, присвоены временные значения (вам лучше их изменить, работая над проектом).

Метод `Encrypt()` сначала преобразует начальный вектор из шестнадцатеричной строки в байтовый массив, поскольку именно байтовый массив нужен функции `mcrypt_encrypt()`, которая непосредственно выполняет шифрование:

```
// Упаковка SymmetricCrypt::_msHexaIv в двоичную строку
$binary_iv = pack('H*', self::$_msHexaIv);
```

Преобразование строки в массив выполняет функция `pack()` (документация по этой функции доступна по адресу `http://www.php.net/pack`).

Вызов функции `mcrypt_encrypt()` выглядит так.

```
// Шифрование $plainString
$binary_encrypted_string = mcrypt_encrypt(
                              self::$_msCipherAlgorithm,
                              self::$_msSecretKey,
                              $plainString,
                              MCRYPT_MODE_CBC,
                              $binary_iv);
```

Именно при этом вызове данные шифруются по алгоритму AES. Смысл параметров, передаваемых функции `mcrypt_encrypt()`, вполне очевиден, но, если вам интересно, документация по этой функции доступна по адресу `http://www.php.net/mcrypt`. Параметр `MCRYPT_MODE_CBC` приказывает применять цепной метод шифрования, т.е. использовать зашифрованные блоки данных в шифровании последующих блоков (за исключением первого блока — для его шифрования используется начальный вектор).

В конце зашифрованная строка преобразуется в шестнадцатеричный формат, с которым проще работать (например, сохранять в базе данных или в файле конфигурации).

```
// Преобразование $binary_encrypted_string в шестнадцатеричную форму
$hexa_encrypted_string = bin2hex($binary_encrypted_string);
```

Метод `Decrypt()` похож на `Encrypt()`. Первое, что он делает, — преобразует начальный вектор в байтовый массив точно так же, как метод `Encrypt()`.

Поскольку метод `Encrypt()` возвращает зашифрованную строку в шестнадцатеричном виде, входным параметром `Decrypt()` также является шестнадцатеричная строка. Она преобразуется в байтовый массив для передачи методу `mcrypt_decrypt()`.

```
// Преобразование строки в байтовый массив
$binary_encrypted_string = pack('H*', $encryptedString);

// Расшифровка строки $binary_encrypted_string
$decrypted_string = mcrypt_decrypt(
                        self::$_msCipherAlgorithm,
                        self::$_msSecretKey,
                        $binary_encrypted_string,
                        MCRYPT_MODE_CBC,
                        $binary_iv);

return $decrypted_string;
```

Тестовый файл `test_encryption.php` просто позволяет шифровать и расшифровывать вводимые строки, чтобы проверить работу класса `SymmetricCrypt`. Код в этом файле очень прост, и мы не будем его обсуждать.

Итак, теперь у нас есть класс `SymmetricCrypt`, и осталось только создать класс `SecureCard`, чтобы завершить нашу систему для работы с кредитными картами.

Хранение информации о кредитных картах с помощью класса `SecureCard`

В следующем упражнении мы создадим класс `SecureCard`, который будет работать с информацией о кредитных картах. В нем будет использоваться функциональность, реализованная нами в предыдущих двух упражнениях, — мы будем шифровать данные о кредитных картах, чтобы гарантировать безопасность их хранения.

Упражнение: создание класса `SecureCard`

1. Создайте в папке `business` файл `secure_card.php` и поместите в него следующий код.

```
<?php
// Представляет кредитную карту
class SecureCard
{
  // Private-элементы, содержащие сведения о кредитной карте
  private $_mIsDecrypted = false;
```

```
private $_mIsEncrypted = false;
private $_mCardHolder;
private $_mCardNumber;
private $_mIssueDate;
private $_mExpiryDate;
private $_mIssueNumber;
private $_mCardType;
private $_mEncryptedData;
private $_mXmlCardData;

// Конструктор класса
public function __construct()
{
  // Конструктор пуст
}

// Расшифровываем данные
public function LoadEncryptedDataAndDecrypt($newEncryptedData)
{
  $this->_mEncryptedData = $newEncryptedData;
  $this->DecryptData();
}

// Шифруем данные
public function LoadPlainDataAndEncrypt($newCardHolder, $newCardNumber,
                                        $newIssueDate, $newExpiryDate,
                                        $newIssueNumber, $newCardType)
{
  $this->_mCardHolder = $newCardHolder;
  $this->_mCardNumber = $newCardNumber;
  $this->_mIssueDate = $newIssueDate;
  $this->_mExpiryDate = $newExpiryDate;
  $this->_mIssueNumber = $newIssueNumber;
  $this->_mCardType = $newCardType;
  $this->EncryptData();
}

// Создаем XML-документ с данными о карте
private function CreateXml()
{
  // Помещаем данные в XML-документ
  $xml_card_data = &$this->_mXmlCardData;
  $xml_card_data = new DOMDocument();

  $document_root = $xml_card_data->createElement('CardDetails');

  $child = $xml_card_data->createElement('CardHolder');
  $child = $document_root->appendChild($child);
  $value = $xml_card_data->createTextNode($this->_mCardHolder);
  $value = $child->appendChild($value);

  $child = $xml_card_data->createElement('CardNumber');
  $child = $document_root->appendChild($child);
  $value = $xml_card_data->createTextNode($this->_mCardNumber);
  $value = $child->appendChild($value);

  $child = $xml_card_data->createElement('IssueDate');
  $child = $document_root->appendChild($child);
  $value = $xml_card_data->createTextNode($this->_mIssueDate);
  $value = $child->appendChild($value);
```

```
    $child = $xml_card_data->createElement('ExpiryDate');
    $child = $document_root->appendChild($child);
    $value = $xml_card_data->createTextNode($this->_mExpiryDate);
    $value = $child->appendChild($value);

    $child = $xml_card_data->createElement('IssueNumber');
    $child = $document_root->appendChild($child);
    $value = $xml_card_data->createTextNode($this->_mIssueNumber);
    $value = $child->appendChild($value);

    $child = $xml_card_data->createElement('CardType');
    $child = $document_root->appendChild($child);
    $value = $xml_card_data->createTextNode($this->_mCardType);
    $value = $child->appendChild($value);

    $document_root = $xml_card_data->appendChild($document_root);
  }

  // Извлекаем данные о карте из  XML-документа
  private function ExtractXml($decryptedData)
  {
    $xml = simplexml_load_string($decryptedData);
    $this->_mCardHolder = (string) $xml->CardHolder;
    $this->_mCardNumber = (string) $xml->CardNumber;
    $this->_mIssueDate = (string) $xml->IssueDate;
    $this->_mExpiryDate = (string) $xml->ExpiryDate;
    $this->_mIssueNumber = (string) $xml->IssueNumber;
    $this->_mCardType = (string) $xml->CardType;
  }

  // Зашифровываем данные о кредитной карте
  private function EncryptData()
  {
    // Помещаем данные в XML-документ
    $this->CreateXml();

    // Шифруем данные
    $this->_mEncryptedData =
      SymmetricCrypt::Encrypt($this->_mXmlCardData->saveXML());

    // Устанавливаем флаг
    $this->_mIsEncrypted = true;
  }

  // Расшифровываем данные о карте и извлекаем их из XML-документа
  private function DecryptData()
  {
    // Расшифровываем данные
    $decrypted_data = SymmetricCrypt::Decrypt($this->_mEncryptedData);

    // Извлекаем данные из XML-документа
    $this->ExtractXml($decrypted_data);

    // Устанавливаем флаг расшифровки
    $this->_mIsDecrypted = true;
  }

  public function __get($name)
  {
    if ($name == 'EncryptedData')
    {
```

```
      if ($this->_mIsEncrypted)
        return $this->_mEncryptedData;
      else
        throw new Exception('Data not encrypted');
    }
    elseif ($name == 'CardNumberX')
    {
      if ($this->_mIsDecrypted)
        return 'XXXX-XXXX-XXXX-' .
          substr($this->_mCardNumber, strlen($this->_mCardNumber) - 4, 4);
      else
        throw new Exception('Data not decrypted');
    }
    elseif (in_array($name, array ('CardHolder', 'CardNumber', 'IssueDate',
                                   'ExpiryDate', 'IssueNumber', 'CardType')))

    {
      $name = '_m' . $name;

      if ($this->_mIsDecrypted)
        return $this->$name;
      else
        throw new Exception('Data not decrypted');
    }
    else
    {
      throw new Exception('Property ' . $name . ' not found');
    }
  }
}
?>
```

2. Создайте в папке `tshirtshop` тестовый файл `test_card.php` и поместите в него следующий код.

```
<?php
require_once 'include/config.php';
require_once BUSINESS_DIR . 'symmetric_crypt.php';
require_once BUSINESS_DIR . 'secure_card.php';

$card_holder = 'John Doe';
$card_number = '1234567890123456';
$expiry_date = '01/09';
$issue_date = '01/06';
$issue_number = 100;
$card_type = 'Mastercard';

echo '<br />Credit card data:<br />' .
     $card_holder . ', ' . $card_number . ', ' .
     $issue_date . ', ' . $expiry_date . ', ' .
     $issue_number . ', ' . $card_type . '<br />';

$credit_card = new SecureCard();

try
{
  $credit_card->LoadPlainDataAndEncrypt($card_holder, $card_number,
                  $issue_date, $expiry_date, $issue_number, $card_type);

  $encrypted_data = $credit_card->EncryptedData;
}
```

```
catch(Exception $e)
{
  echo '<font color="red">Exception: ' . $e->getMessage() . '</font>';

  exit();
}

echo '<br />Encrypted data:<br />' . $encrypted_data . '<br />';

$our_card = new SecureCard();

try
{
  $our_card->LoadEncryptedDataAndDecrypt($encrypted_data);

  echo '<br/>Decrypted data:<br/>' .
       $our_card->CardHolder . ', ' . $our_card->CardNumber . ', ' .
       $our_card->IssueDate . ', ' . $our_card->ExpiryDate . ', ' .
       $our_card->IssueNumber . ', ' . $our_card->CardType;
}
catch(Exception $e)
{
  echo '<font color="red">Exception: ' . $e->getMessage() . '</font>';

  exit();
}
?>
```

3. Загрузите страницу `test_card.php` в браузере и посмотрите на результаты. Они должны быть примерно такими, как на рис. 16.3. При желании можете заменить данные в этом файле какими-нибудь другими.

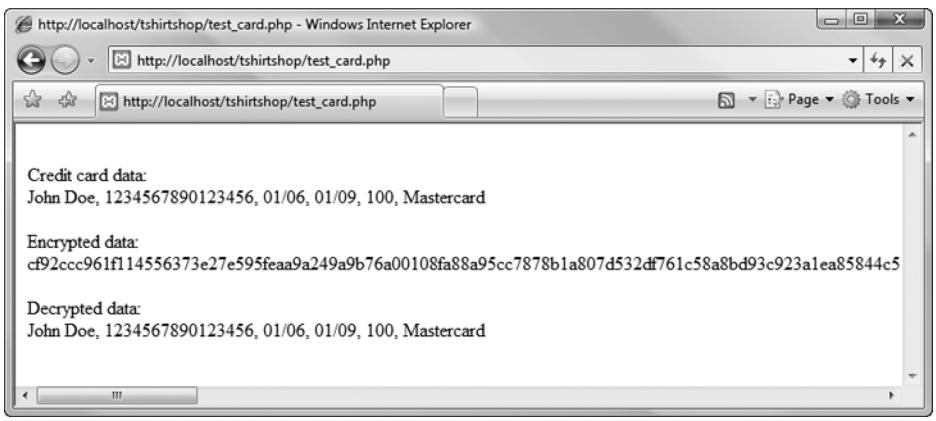


Рис. 16.3. Шифрование и расшифровка данных о кредитной карте

Как это работает: класс `SecureCard`

В этом упражнении кода немного больше, чем в предыдущих, но весь этот код довольно прост. Сначала мы объявляем переменные, в которых храним сведения о кредитных картах, в виде отдельных значений, в виде зашифрованных строк и в виде промежуточного XML-документа. Кроме того, мы объявляем булевы флаги, которые указывают, была ли информация успешно зашифрована или расшифрована.

```
<?php
// Представляет кредитную карту
```

```
class SecureCard
{
  // Private-элементы, содержащие сведения о кредитной карте
  private $_mIsDecrypted = false;
  private $_mIsEncrypted = false;
  private $_mCardHolder;
  private $_mCardNumber;
  private $_mIssueDate;
  private $_mExpiryDate;
  private $_mIssueNumber;
  private $_mCardType;
  private $_mEncryptedData;
  private $_mXmlCardData;
```

Следующими указаны два важных метода, доступных извне класса. Метод `LoadEncryptedDataAndDecrypt()` получает зашифрованную строку и расшифровывает ее, а `LoadPlainDataAndEncrypt()` получает данные о кредитной карте и зашифровывает их.

```
// Расшифровываем данные
public function LoadEncryptedDataAndDecrypt($newEncryptedData)
{
  $this->_mEncryptedData = $newEncryptedData;
  $this->DecryptData();
}

// Шифруем данные
public function LoadPlainDataAndEncrypt($newCardHolder, $newCardNumber,
                                        $newIssueDate, $newExpiryDate,
                                        $newIssueNumber, $newCardType)
{
  $this->_mCardHolder = $newCardHolder;
  $this->_mCardNumber = $newCardNumber;
  $this->_mIssueDate = $newIssueDate;
  $this->_mExpiryDate = $newExpiryDate;
  $this->_mIssueNumber = $newIssueNumber;
  $this->_mCardType = $newCardType;
  $this->EncryptData();
}
```

Основную работу выполняют `private`-методы `EncryptData()` и `DecryptData()`, которые мы рассмотрим чуть позже. А сейчас давайте посмотрим, что делают два метода, занимающихся упаковкой и распаковкой данных в XML-формате.

XML — удобный, основанный на тегах формат, позволяющий хранить разнообразную информацию. Класс `SecureCard` хранит сведения о каждой кредитной карте в XML-документе примерно такого вида.

```
<?xml version="1.0"?>
<CardDetails>
  <CardHolder>John Doe</CardHolder>
  <CardNumber>1234567890123456</CardNumber>
  <IssueDate>01/06</IssueDate>
  <ExpiryDate>01/09</ExpiryDate>
  <IssueNumber>100</IssueNumber>
  <CardType>Mastercard</CardType>
</CardDetails>
```

Для работы с XML-данными мы используем класс `DOMDocument`; он знает, как читать, записывать и изменять XML-документы. Объектная модель документа (Document Object Model — DOM) — самый распространенный и универсальный API для работы с XML.

ПОДСКАЗКА

Стандарт DOM разработан и поддерживается World Wide Web Consortium; его официальная страница — `http://www.w3.org/DOM/`.

Поддержка DOM в PHP 5 позволяет легко читать, создавать, редактировать и сохранять XML-документы. Эта поддержка полностью соответствует стандарту DOM. В нашем коде есть метод `CreateXml()`, использующий DOM для создания XML-документа со структурой, показанной выше. Чтобы создать такой документ, мы последовательно создаем его узлы и присваиваем им значения.

```
// Создаем XML-документ с данными о карте
private function CreateXml()
{
  // Помещаем данные в XML-документ
  $xml_card_data = &$this->_mXmlCardData;
  $xml_card_data = new DOMDocument();

  $document_root = $xml_card_data->createElement('CardDetails');

  $child = $xml_card_data->createElement('CardHolder');
  $child = $document_root->appendChild($child);
  $value = $xml_card_data->createTextNode($this->_mCardHolder);
  $value = $child->appendChild($value);

...

  $document_root = $xml_card_data->appendChild($document_root);
}
```

Для чтения XML-документа можно использовать объект `DOMDocument`, но в методе `ExtractXml()` мы предпочли воспользоваться новой возможностью PHP 5, которая называется SimpleXML. Хотя эта возможность и не так универсальна, как `DOMDocument`, она позволяет легко считывать данные из XML в структуру данных.

```
// Извлекаем данные о карте из  XML-документа
private function ExtractXml($decryptedData)
{
  $xml = simplexml_load_string($decryptedData);
  $this->_mCardHolder = (string) $xml->CardHolder;
  $this->_mCardNumber = (string) $xml->CardNumber;
  $this->_mIssueDate = (string) $xml->IssueDate;
  $this->_mExpiryDate = (string) $xml->ExpiryDate;
  $this->_mIssueNumber = (string) $xml->IssueNumber;
  $this->_mCardType = (string) $xml->CardType;
}
```

Сначала метод `EncryptData()` вызывает метод `CreateXml()`, чтобы упаковать сведения о карте в XML-формат.

```
// Зашифровываем данные о кредитной карте
private function EncryptData()
{
  // Помещаем данные в XML-документ
  $this->CreateXml();
```

Затем созданный XML-документ шифруется и сохраняется в переменной `$_mEncryptedData`:

```
// Шифруем данные
$this->_mEncryptedData =
  SymmetricCrypt::Encrypt($this->_mXmlCardData->saveXML());
```

Наконец мы устанавливаем флаг `$_mIsEncrypted`, указывающий, что данные зашифрованы.

```
  // Устанавливаем флаг
  $this->_mIsEncrypted = true;
}
```

Метод `DecryptData()` выполняет обратную последовательность операций — расшифровывает данные, извлекает сведения о кредитной карте из XML-документа в поля класса и устанавливает флаг `$_mIsDecrypted`, указывающий, что данные расшифрованы.

```
// Расшифровываем данные о карте и извлекаем их из XML-документа
private function DecryptData()
{
  // Расшифровываем данные
  $decrypted_data = SymmetricCrypt::Decrypt($this->_mEncryptedData);

  // Извлекаем данные из XML-документа
  $this->ExtractXml($decrypted_data);

  // Устанавливаем флаг расшифровки
  $this->_mIsDecrypted = true;
}
```

Далее определяем несколько полей класса. Начиная с PHP 5, в классе можно определить `public`-метод `__get()`, который вызывается автоматически при попытке обратиться к методу (или к полю), который не объявлен в классе. Например, в фрагменте кода

```
$card = new SecureCard();
$encrypted = $card->EncryptedData;
```

будет вызван метод `__get()`, поскольку в классе `SecureCard` нет поля `EncryptedData`. В методе `__get()` можно выяснить, к какому полю или методу программа попыталась обратиться, и вернуть значение поля или выполнить действия соответствующего метода. Этот прием особенно полезен, если нужно вычислять какие-либо значения "на лету".

В нашем случае метод `__get()` будет обрабатывать обращения к восьми несуществующим полям класса. Первое такое поле — `EncryptedData`, значение которого возвращается, только если флаг `$_mIsEncrypted` установлен равным `true`.

```
public function __get($name)
{
  if ($name == 'EncryptedData')
  {
    if ($this->_mIsEncrypted)
      return $this->_mEncryptedData;
    else
      throw new Exception('Data not encrypted');
  }
```

Следующее свойство — `CardNumberX`, которое должно содержать номер карты, в котором все цифры, кроме последней, заменены символом "X". Это свойство будет использоваться, когда понадобится сообщить пользователю, какую карту он будет применять, но не показывать номер карты целиком.

```
elseif ($name == 'CardNumberX')
{
  if ($this->_mIsDecrypted)
    return 'XXXX-XXXX-XXXX-' .
      substr($this->_mCardNumber, strlen($this->_mCardNumber) - 4, 4);
  else
    throw new Exception('Data not decrypted');
}
```

Остальные шесть свойств (`CardHolder`, `CardNumber`, `IssueDate`, `ExpiryDate`, `IssueNumber` и `CardType`) обрабатываются одним блоком кода:

```
  elseif (in_array($name, array ('CardHolder', 'CardNumber', 'IssueDate',
                                 'ExpiryDate', 'IssueNumber', 'CardType')))
```

```
  {
    $name = '_m' . $name;

    if ($this->_mIsDecrypted)
      return $this->$name;
    else
      throw new Exception('Data not decrypted');
  }
  else
  {
    throw new Exception('Property ' . $name . ' not found');
  }
}
```

Заметьте, что все эти данные доступны, только если установлен флаг `$_mIsDecrypted`; в противном случае попытка доступа к ним приведет к генерации исключения.

Кроме того, заметьте, что данные будут недоступны после шифрования. Те данные, которые использовались для инициализации объекта SecureCard, доступны только в зашифрованном виде. Это несколько избыточная мера защиты, поскольку класс `SecureCard` предназначен для шифрования и расшифровки данных о кредитных картах, а не для постоянного хранения этих данных. После того как данные зашифрованы, нам уже не нужно обращаться к незашифрованным данным — нас интересует только зашифрованная строка.

ЗАМЕЧАНИЕ

Прежде чем перейти к коду клиентской части приложения, стоит упомянуть один важный момент, который вы, вероятно, уже заметили. В приведенном выше коде не проверяется корректность введенных сведений о кредитной карте. Этот код может работать, даже если в качестве данных будут выступать пустые строки. Это позволяет классу быть максимально гибким. В любом случае данные о кредитных картах удобнее проверять непосредственно на странице, на которой пользователь их вводит, или вообще не проверять. Это не так опасно, как вам, возможно, кажется: если эти данные будут неправильными, платеж просто не пройдет, и мы сообщим об этом пользователю, предложив ему ввести новые данные. Разумеется, есть еще проблема форматирования данных (например, срок действия карты обычно задается в формате ММ/ГГ), но ее тоже можно решить вне класса `SecureCard`.

Тестовая страница (`test_card.php`) для этого класса просто отображает сведения о кредитной карте в зашифрованном и незашифрованном виде. Как видите, в зашифрованном виде строка с данными получается довольно длинной, и в базе данных для нее отведен столбец соответствующей величины. Как видно из тестового примера, шифрование и расшифровка выполняются правильно, и мы можем вплотную заняться системой учетных записей пользователей.

Добавление учетных записей пользователей в TShirtShop

Прежде чем реализовать визуальную часть функциональности учетных записей, давайте посмотрим, чего мы хотим достичь в оставшейся части этой главы.

Нам нужно, чтобы на главной странице сайта присутствовала форма для аутентификации пользователей. Кроме того, нужно предоставить пользователям возможность регистрироваться на сайте и редактировать свои учетные записи. Мы создадим компонентный шаблон для формы аутентификации и разместим его на главной странице, над полем поиска, как показано на рис. 16.4.

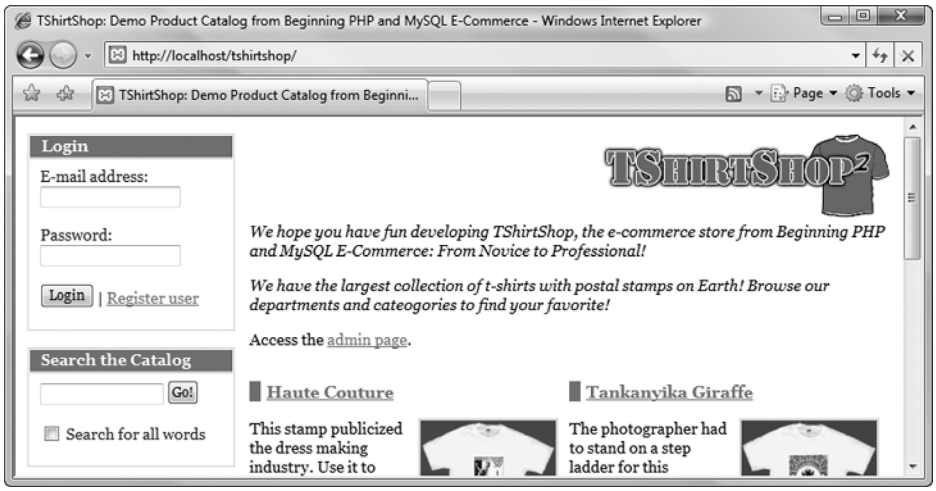


Рис. 16.4. Главная страница TShirtShop с формой аутентификации

Страница регистрации нового пользователя будет выглядеть так, как показано на рис. 16.5.

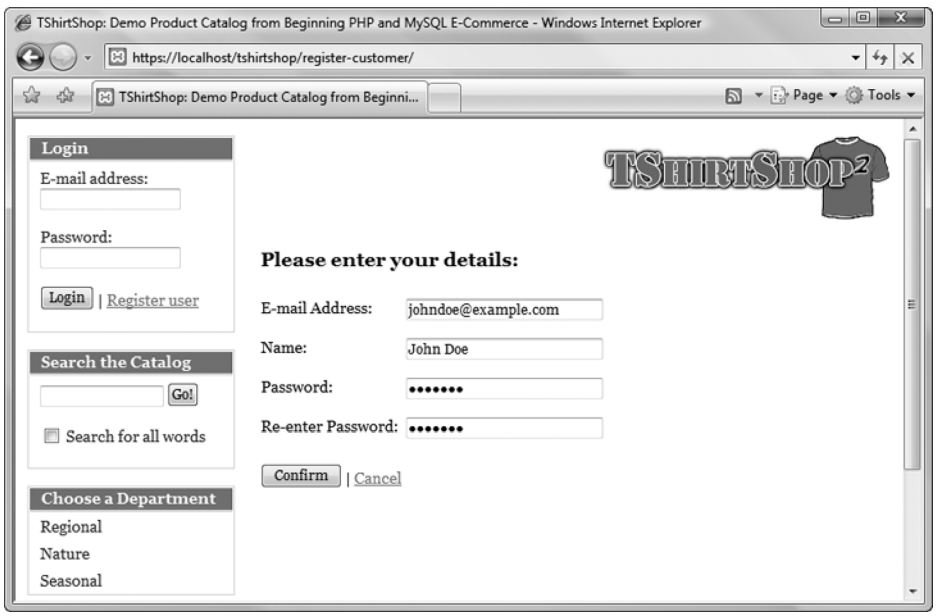


Рис. 16.5. Страница регистрации нового пользователя

После того как пользователь пройдет аутентификацию, новый компонентный шаблон будет отображать имя пользователя и несколько ссылок для работы с учетной записью, как показано на рис. 16.6.

Щелчок на ссылке Add CC Details позволяет открывать страницу, показанную на рис. 16.7.

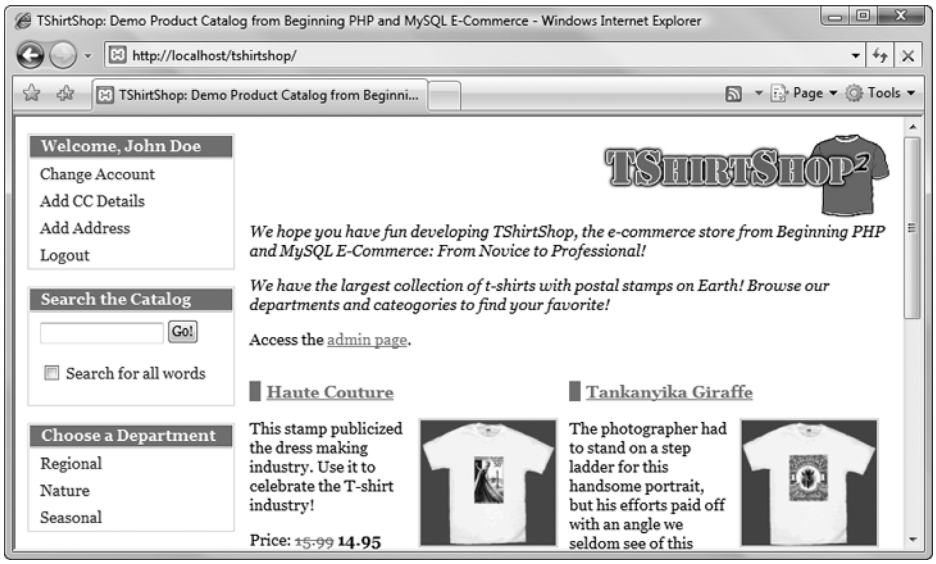


Рис. 16.6. Главная страница TShirtshop после того, как пользователь прошел аутентификацию

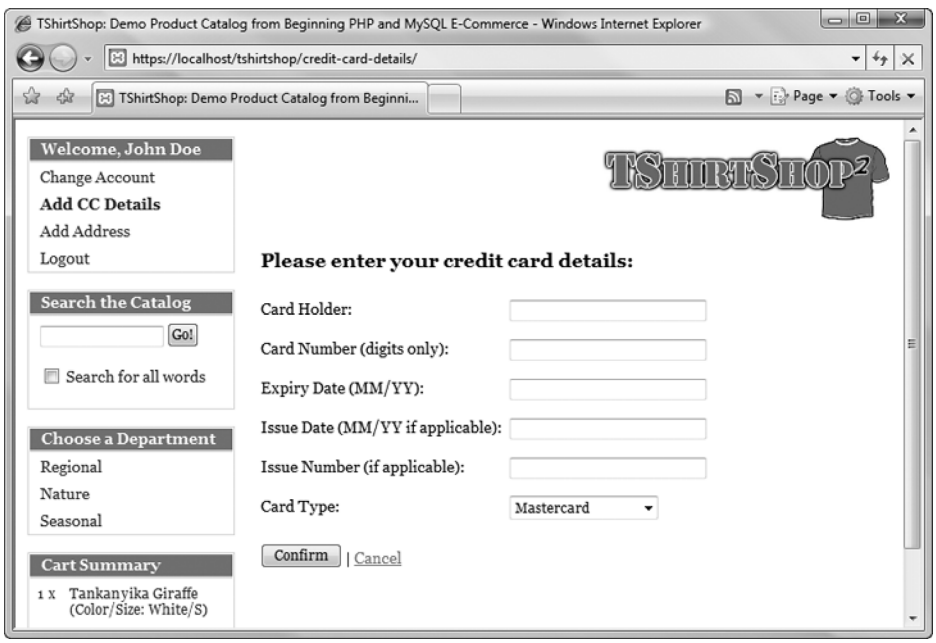


Рис. 16.7. Страница для редактирования сведений о кредитной карте

Похожая форма будет отображена после щелчка на ссылке Add Address. Если пользователь уже ввел данные о кредитной карте и задал свой адрес, соответствующие ссылки изменяются на Change CC Details и Change Address.

Как всегда, новую функциональность мы начнем реализовать с уровня данных.

Реализация уровня данных

В следующем упражнении мы создадим набор хранимых процедур для работы с учетными записями пользователей и разберемся, как эти процедуры работают.

Упражнение: создание хранимых процедур для работы с учетными записями

1. С помощью phpMyAdmin выполните операторы SQL, приведенные далее в этом упражнении. Не забудьте задавать символы `$$` в качестве разделителя, прежде чем выполнять оператор из каждого шага упражнения.
2. Выполните следующий оператор, чтобы создать хранимую процедуру `customer_get_login_info`.

```
-- Создаем хранимую процедуру customer_get_login_info
CREATE PROCEDURE customer_get_login_info(IN inEmail VARCHAR(100))
BEGIN
  SELECT customer_id, password FROM customer WHERE email = inEmail;
END$$
```

   Когда пользователь пытается пройти аутентификацию, нужно проверить, правильный ли он ввел пароль. Хранимая процедура `customer_get_login_info` возвращает идентификатор пользователя и хешированный пароль для пользователя с заданным адресом электронной почты.

3. Выполните следующий оператор, чтобы создать хранимую процедуру `customer_add`.

```
-- Создаем хранимую процедуру customer_add
CREATE PROCEDURE customer_add(IN inName VARCHAR(50),
  IN inEmail VARCHAR(100), IN inPassword VARCHAR(50))
BEGIN
  INSERT INTO customer (name, email, password)
         VALUES (inName, inEmail, inPassword);

  SELECT LAST_INSERT_ID();
END$$
```

   Эта хранимая процедура будет вызываться при регистрации на сайте нового пользователя. Она возвращает идентификатор пользователя, который можно будет сохранить в сеансовой переменной.

4. Выполните следующий оператор, чтобы создать хранимую процедуру `customer_get_customer`.

```
-- Создаем хранимую процедуру customer_get_customer
CREATE PROCEDURE customer_get_customer(IN inCustomerId INT)
BEGIN
  SELECT customer_id, name, email, password, credit_card,
         address_1, address_2, city, region, postal_code, country,
         shipping_region_id, day_phone, eve_phone, mob_phone
  FROM   customer
  WHERE  customer_id = inCustomerId;
END$$
```

   Эта хранимая процедура возвращает все сведения о пользователе с заданным идентификатором.

5. Выполните следующий оператор, чтобы создать хранимую процедуру `customer_update_account`.

```
-- Создаем хранимую процедуру customer_update_account
CREATE PROCEDURE customer_update_account(IN inCustomerId INT,
  IN inName VARCHAR(50), IN inEmail VARCHAR(100),
  IN inPassword VARCHAR(50), IN inDayPhone VARCHAR(100),
  IN inEvePhone VARCHAR(100), IN inMobPhone VARCHAR(100))
BEGIN
  UPDATE customer
  SET    name = inName, email = inEmail,
         password = inPassword, day_phone = inDayPhone,
         eve_phone = inEvePhone, mob_phone = inMobPhone
  WHERE  customer_id = inCustomerId;
END$$
```

Эта хранимая процедура обновляет в базе данных все сведения об учетной записи заданного пользователя.

6. Выполните следующий оператор, чтобы создать хранимую процедуру `customer_update_credit_card`.

```
-- Создаем хранимую процедуру customer_update_credit_card
CREATE PROCEDURE customer_update_credit_card(
  IN inCustomerId INT, IN inCreditCard TEXT)
BEGIN
  UPDATE customer
  SET    credit_card = inCreditCard
  WHERE  customer_id = inCustomerId;
END$$
```

Хранимая процедура `customer_update_credit_card` изменяет только значение в столбце `credit_card` таблицы `customer`. Это все, что требуется сделать, поскольку в данном столбце хранится зашифрованная версия XML-документа со всеми сведениями о кредитной карте.

7. Выполните следующий оператор, чтобы создать хранимую процедуру `customer_get_shipping_regions`.

```
-- Создаем хранимую процедуру customer_get_shipping_regions
CREATE PROCEDURE customer_get_shipping_regions()
BEGIN
  SELECT shipping_region_id, shipping_region FROM shipping_region;
END$$
```

Эта хранимая процедура возвращает из базы данных список регионов доставки для страницы с адресом пользователя.

8. Выполните следующий оператор, чтобы создать хранимую процедуру `customer_update_address`.

```
-- Создаем хранимую процедуру customer_update_address
CREATE PROCEDURE customer_update_address(IN inCustomerId INT,
  IN inAddress1 VARCHAR(100), IN inAddress2 VARCHAR(100),
  IN inCity VARCHAR(100), IN inRegion VARCHAR(100),
  IN inPostalCode VARCHAR(100), IN inCountry VARCHAR(100),
  IN inShippingRegionId INT)
BEGIN
  UPDATE customer
  SET    address_1 = inAddress1, address_2 = inAddress2, city = inCity,
         region = inRegion, postal_code = inPostalCode,
         country = inCountry, shipping_region_id = inShippingRegionId
  WHERE  customer_id = inCustomerId;
END$$
```

Эта хранимая процедура обновляет адрес пользователя в базе данных.

Реализация уровня логики приложения

В папке business создайте файл customer.php. В нем будет размещаться класс Customer. Код этого класса довольно большой — он обращается к уровню данных, обрабатывая запросы, поступающие от уровня представления. Поместите в файл business/customer.php следующий код.

```
<?php
// Класс уровня логики приложения, содержащий функциональность для работы с
// учетными записями пользователей
class Customer
{
  // Проверяет наличие переменной customer_id в сеансе
  public static function IsAuthenticated()
  {
    if (!(isset ($_SESSION['tshirtshop_customer_id'])))
      return 0;
    else
      return 1;
  }

  // Возвращает значение customer_id и пароль для пользователя с
  // адресом электронной почты $email
  public static function GetLoginInfo($email)
  {
    // Составляем SQL-запрос
    $sql = 'CALL customer_get_login_info(:email)';

    // Создаем массив параметров
    $params = array (':email' => $email);

    // Выполняем запрос и возвращаем результаты
    return DatabaseHandler::GetRow($sql, $params);
  }

  public static function IsValid($email, $password)
  {
    $customer = self::GetLoginInfo($email);

    if (empty ($customer['customer_id']))
      return 2;

    $customer_id = $customer['customer_id'];
    $hashed_password = $customer['password'];

    if (PasswordHasher::Hash($password) != $hashed_password)
      return 1;
    else
    {
      $_SESSION['tshirtshop_customer_id'] = $customer_id;

      return 0;
    }
  }

  public static function Logout()
  {
    unset($_SESSION['tshirtshop_customer_id']);
  }
```

```
public static function GetCurrentCustomerId()
{
  if (self::IsAuthenticated())
    return $_SESSION['tshirtshop_customer_id'];
  else
    return 0;
}

/* Добавляет новую учетную запись пользователя, аутентифицирует
его, если $addAndLogin равна true и возвращает customer_id */
public static function Add($name, $email, $password, $addAndLogin = true)
{
  $hashed_password = PasswordHasher::Hash($password);

  // Составляем SQL-запрос
  $sql = 'CALL customer_add(:name, :email, :password)';

  // Создаем массив параметров
  $params = array (':name' => $name, ':email' => $email
                   ':password' => $hashed_password);

  // Выполняем запрос и получаем customer_id
  $customer_id = DatabaseHandler::GetOne($sql, $params);

  if ($addAndLogin)
    $_SESSION['tshirtshop_customer_id'] = $customer_id;

  return $customer_id;
}

public static function Get($customerId = null)
{
  if (is_null($customerId))
    $customerId = self::GetCurrentCustomerId();

  // Составляем SQL-запрос
  $sql = 'CALL customer_get_customer(:customer_id)';

  // Создаем массив параметров
  $params = array (':customer_id' => $customerId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetRow($sql, $params);
}

public static function UpdateAccountDetails($name, $email, $password,
                         $dayPhone, $evePhone, $mobPhone,
                         $customerId = null)
{
  if (is_null($customerId))
    $customerId = self::GetCurrentCustomerId();

  $hashed_password = PasswordHasher::Hash($password);

  // Составляем SQL-запрос
  $sql = 'CALL customer_update_account(:customer_id, :name, :email,
                :password, :day_phone, :eve_phone, :mob_phone)';

  // Создаем массив параметров
  $params = array (':customer_id' => $customerId, ':name' => $name,
                   ':email' => $email, ':password' => $hashed_password,
```

```
                    ':day_phone' => $dayPhone, ':eve_phone' => $evePhone,
                    ':mob_phone' => $mobPhone);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

public static function DecryptCreditCard($encryptedCreditCard)
{
  $secure_card = new SecureCard();
  $secure_card->LoadEncryptedDataAndDecrypt($encryptedCreditCard);

  $credit_card = array();
  $credit_card['card_holder'] = $secure_card->CardHolder;
  $credit_card['card_number'] = $secure_card->CardNumber;
  $credit_card['issue_date'] = $secure_card->IssueDate;
  $credit_card['expiry_date'] = $secure_card->ExpiryDate;
  $credit_card['issue_number'] = $secure_card->IssueNumber;
  $credit_card['card_type'] = $secure_card->CardType;
  $credit_card['card_number_x'] = $secure_card->CardNumberX;

  return $credit_card;
}

public static function GetPlainCreditCard()
{
  $customer_data = self::Get();

  if (!(empty ($customer_data['credit_card'])))
    return self::DecryptCreditCard($customer_data['credit_card']);
  else
    return array('card_holder' => '', 'card_number' => '',
                 'issue_date' => '', 'expiry_date' => '',
                 'issue_number' => '', 'card_type' => '',
                 'card_number_x' => '');
}

public static function UpdateCreditCardDetails($plainCreditCard,
                                                $customerId = null)
{
  if (is_null($customerId))
    $customerId = self::GetCurrentCustomerId();

  $secure_card = new SecureCard();
  $secure_card->LoadPlainDataAndEncrypt($plainCreditCard['card_holder'],
    $plainCreditCard['card_number'], $plainCreditCard['issue_date'],
    $plainCreditCard['expiry_date'], $plainCreditCard['issue_number'],
  $plainCreditCard['card_type']);
  $encrypted_card = $secure_card->EncryptedData;

  // Составляем SQL-запрос
  $sql = 'CALL customer_update_credit_card(:customer_id, :credit_card)';

  // Создаем массив параметров
  $params = array (':customer_id' => $customerId,
                   ':credit_card' => $encrypted_card);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}

public static function GetShippingRegions()
{
```

```
    // Составляем SQL-запрос
    $sql = 'CALL customer_get_shipping_regions()';

    // Выполняем запрос и возвращаем результаты
    return DatabaseHandler::GetAll($sql);
  }

  public static function UpdateAddressDetails($address1, $address2, $city,
                          $region, $postalCode, $country,
                          $shippingRegionId, $customerId = null)
  {
    if (is_null($customerId))
      $customerId = self::GetCurrentCustomerId();

    // Составляем SQL-запрос
    $sql = 'CALL customer_update_address(:customer_id, :address_1,
                  :address_2, :city, :region, :postal_code, :country,
                  :shipping_region_id)';

    // Создаем массив параметров
    $params = array (':customer_id' => $customerId,
                     ':address_1' => $address1, ':address_2' => $address2,
                     ':city' => $city, ':region' => $region,
                     ':postal_code' => $postalCode,
                     ':country' => $country,
                     ':shipping_region_id' => $shippingRegionId);

    // Выполняем запрос
    DatabaseHandler::Execute($sql, $params);
  }
}
?>
```

Реализация уровня представления

Уровень представления для системы работы с учетными записями будет состоять из следующих компонентных шаблонов.

- `customer_login`. Поля для аутентификации.
- `customer_logged`. Когда пользователь пройдет аутентификацию, этот шаблон будет выводиться вместо `customer_login`, отображая имя пользователя и ссылки для управления учетной записью и завершения сеанса.
- `customer_details`. Шаблон для регистрации нового пользователя или редактирования сведений об уже зарегистрированном.
- `customer_address`. Шаблон для добавления или редактирования информации об адресах пользователей.
- `customer_credit_card`. Шаблон для добавления или редактирования информации о кредитных картах.

Следующее упражнение — одно из самых длинных в книге — посвящено созданию всех только что перечисленных шаблонов.

Упражнение: создание компонентных шаблонов

1. Создайте в папке `presentation/templates` файл шаблона `customer_login.tpl` и поместите в него следующий код.

```
{* customer_login.tpl *}
{load_presentation_object filename="customer_login" assign="obj"}
<div class="box">
  <p class="box-title">Login</p>
  <form method="post" action="{$obj->mLinkToLogin}">
    {if $obj->mErrorMessage}<p class="error">{$obj-
>mErrorMessage}</p>{/if}
    <p>
      <label for="email">E-mail address:</label>
      <input type="text" maxlength="50" name="email" size="22"
       value="{$obj->mEmail}" />
    </p>
    <p>
      <label for="password">Password:</label>
      <input type="password" maxlength="50" name="password" size="22" />
    </p>
    <p>
      <input type="submit" name="Login" value="Login" /> |
      <a href="{$obj->mLinkToRegisterCustomer}">Register user</a>
    </p>
  </form>
</div>
```

2. Создайте в папке `presentation` файл объекта представления `customer_login.php` и поместите в него следующий код.

```
<?php
class CustomerLogin
{
  // Public-переменные
  public $mErrorMessage;
  public $mLinkToLogin;
  public $mLinkToRegisterCustomer;
  public $mEmail = '';
  // Конструктор класса
  public function __construct()
  {
    if (USE_SSL == 'yes' && getenv('HTTPS') != 'on')
      $this->mLinkToLogin =
        Link::Build(str_replace(VIRTUAL_LOCATION, '',
getenv('REQUEST_URI')),
                    'https');
    else
      $this->mLinkToLogin =
        Link::Build(str_replace(VIRTUAL_LOCATION, '',
getenv('REQUEST_URI')));

    $this->mLinkToRegisterCustomer = Link::ToRegisterCustomer();
  }

  public function init()
  {
    // Проверяем, выполнялась ли аутентификация
    if (isset ($_POST['Login']))
    {
      // Получаем статус аутентификации
      $login_status = Customer::IsValid($_POST['email'],
$_POST['password']);

      switch ($login_status)
      {
        case 2:
```

```
          $this->mErrorMessage = 'Unrecognized Email.';
          $this->mEmail = $_POST['email'];

          break;
        case 1:
          $this->mErrorMessage = 'Unrecognized password.';
          $this->mEmail = $_POST['email'];

          break;
        case 0:
          $redirect_to_link =
            Link::Build(str_replace(VIRTUAL_LOCATION, '',
                                    getenv('REQUEST_URI')));

          header('Location:' . $redirect_to_link);

          exit();
      }
    }
  }
}
?>
```

3. Создайте в папке `presentation/templates` файл шаблона `customer_logged.tpl` и поместите в него следующий код.

```
{* customer_logged.tpl *}
{load_presentation_object filename="customer_logged" assign="obj"}
<div class="box">
  <p class="box-title">Welcome, {$obj->mCustomerName}</p>
  <ul>
    <li>
      <a {if $obj->mSelectedMenuItem eq 'account'} class="selected" {/if}
         href="{$obj->mLinkToAccountDetails}">
        Change Account
      </a>
    </li>
    <li>
      <a {if $obj->mSelectedMenuItem eq 'credit-card'} class="selected"
{/if}
         href="{$obj->mLinkToCreditCardDetails}">
        {$obj->mCreditCardAction} CC Details
      </a>
    </li>
    <li>
      <a {if $obj->mSelectedMenuItem eq 'address'} class="selected" {/if}
         href="{$obj->mLinkToAddressDetails}">
        {$obj->mAddressAction} Address
      </a>
    </li>
    <li>
      <a href="{$obj->mLinkToLogout}">
        Logout
      </a>
    </li>
  </ol>
</div>
```

4. Создайте в папке `presentation` файл объекта представления `customer_logged.php` и поместите в него следующий код.

```
<?php
class CustomerLogged
{
  // Public-атрибуты
  public $mCustomerName;
  public $mCreditCardAction = 'Add';
  public $mAddressAction = 'Add';
  public $mLinkToAccountDetails;
  public $mLinkToCreditCardDetails;
  public $mLinkToAddressDetails;
  public $mLinkToLogout;
  public $mSelectedMenuItem;

  // Конструктор класса
  public function __construct()
  {
    $this->mLinkToAccountDetails = Link::ToAccountDetails();
    $this->mLinkToCreditCardDetails = Link::ToCreditCardDetails();
    $this->mLinkToAddressDetails = Link::ToAddressDetails();

    $this->mLinkToLogout = Link::Build('index.php?Logout');

    if (isset ($_GET['AccountDetails']))
      $this->mSelectedMenuItem = 'account';
    elseif (isset ($_GET['CreditCardDetails']))
      $this->mSelectedMenuItem = 'credit-card';
    elseif (isset ($_GET['AddressDetails']))
      $this->mSelectedMenuItem = 'address';
  }

  public function init()
  {
    if (isset ($_GET['Logout']))
    {
      Customer::Logout();

      header('Location:' . $_SESSION['link_to_last_page_loaded']);

      exit();
    }

    $customer_data = Customer::Get();
    $this->mCustomerName = $customer_data['name'];

    if (!(empty ($customer_data['credit_card'])))
      $this->mCreditCardAction = 'Change';

    if (!(empty ($customer_data['address_1'])))
      $this->mAddressAction = 'Change';
  }
}
?>
```

5. Создайте в папке `presentation/templates` файл шаблона `customer_details.tpl` и поместите в него следующий код.

```
{* customer_details.tpl *}
{load_presentation_object filename="customer_details" assign="obj"}
<form method="post" action="{$obj->mLinkToAccountDetails}">
  <h2>Please enter your details:</h2>
  <table class="customer-table">
    <tr>
```

```
    <td>E-mail Address:</td>
    <td>
      <input type="text" name="email" value="{$obj->mEmail}"
      {if $obj->mEditMode}readonly="readonly"{/if} size="32" />
      {if $obj->mEmailAlreadyTaken}
      <p class="error">A user with that e-mail address already exists.</p>
      {/if}
      {if $obj->mEmailError}
      <p class="error">You must enter an e-mail address.</p>
      {/if}
    </td>
  </tr>
  <tr>
    <td>Name:</td>
    <td>
      <input type="text" name="name" value="{$obj->mName}" size="32" />
      {if $obj->mNameError}
      <p class="error">You must enter your name.</p>
      {/if}
    </td>
  </tr>
  <tr>
    <td>Password:</td>
    <td>
      <input type="password" name="password" size="32" />
      {if $obj->mPasswordError}
      <p class="error">You must enter a password.</p>
      {/if}
    </td>
  </tr>
  <tr>
    <td>Re-enter Password:</td>
    <td>
      <input type="password" name="passwordConfirm" size="32" />
      {if $obj->mPasswordConfirmError}
      <p class="error">You must re-enter your password.</p>
      {elseif $obj->mPasswordMatchError}
      <p class="error">You must re-enter the same password.</p>
      {/if}
    </td>
  </tr>
  {if $obj->mEditMode}
  <tr>
    <td>Day phone:</td>
    <td>
      <input type="text" name="dayPhone" value="{$obj->mDayPhone}"
       size="32" />
    </td>
  </tr>
  <tr>
    <td>Eve phone:</td>
    <td>
      <input type="text" name="evePhone" value="{$obj->mEvePhone}"
       size="32" />
    </td>
  </tr>
  <tr>
    <td>Mob phone:</td>
    <td>
      <input type="text" name="mobPhone" value="{$obj->mMobPhone}"
       size="32" />
```

```
      </td>
    </tr>
    {/if}
  </table>
  <input type="submit" name="sended" value="Confirm" /> |
  <a href="{$obj->mLinkToCancelPage}">Cancel</a>
</form>
```

6. Создайте в папке `presentation` файл объекта представления `customer_details.php` и поместите в него следующий код.

```
<?php
class CustomerDetails
{
  // Public-атрибуты
  public $mEditMode = 0;
  public $mEmail;
  public $mName;
  public $mPassword;
  public $mDayPhone = null;
  public $mEvePhone = null;
  public $mMobPhone = null;
  public $mNameError = 0;
  public $mEmailAlreadyTaken = 0;
  public $mEmailError = 0;
  public $mPasswordError = 0;
  public $mPasswordConfirmError = 0;
  public $mPasswordMatchError = 0;
  public $mLinkToAccountDetails;
  public $mLinkToCancelPage;

  // Private-атрибуты
  private $_mErrors = 0;

  // Class constructor
  public function __construct()
  {
    // Проверяем, работаем ли мы с новым пользователем или
    // с уже прошедшим аутентификацию
    if (Customer::IsAuthenticated())
      $this->mEditMode = 1;

    if ($this->mEditMode == 0)
      $this->mLinkToAccountDetails = Link::ToRegisterCustomer();
    else
      $this->mLinkToAccountDetails = Link::ToAccountDetails();

    // Задаем ссылку для отмены
    if (isset ($_SESSION['customer_cancel_link']))
      $this->mLinkToCancelPage = $_SESSION['customer_cancel_link'];
    else
      $this->mLinkToCancelPage = Link::ToIndex();

    // Проверяем, выполнялась ли отправка данных
    if (isset ($_POST['sended']))
    {
      // Имя должно быть задано
      if (empty ($_POST['name']))
      {
        $this->mNameError = 1;
        $this->_mErrors++;
      }
```

```
      else
        $this->mName = $_POST['name'];

      if ($this->mEditMode == 0 && empty ($_POST['email']))
      {
        $this->mEmailError = 1;
        $this->_mErrors++;
      }
      else
        $this->mEmail = $_POST['email'];

      // Пароль не может быть пустым
      if (empty ($_POST['password']))
      {
        $this->mPasswordError = 1;
        $this->_mErrors++;
      }
      else
        $this->mPassword = $_POST['password'];

      // Поле подтверждения пароля не может быть пустым
      if (empty ($_POST['passwordConfirm']))
      {
        $this->mPasswordConfirmError = 1;
        $this->_mErrors++;
      }
      else
        $password_confirm = $_POST['passwordConfirm'];

      // Значения в полях пароля и подтверждения должны совпадать
      if (!isset ($password_confirm) ||
        $this->mPassword != $password_confirm)
      {
        $this->mPasswordMatchError = 1;
          $this->_mErrors++;
      }

      if ($this->mEditMode == 1)
      {
        if (!empty ($_POST['dayPhone']))
          $this->mDayPhone = $_POST['dayPhone'];
        if (!empty ($_POST['evePhone']))
          $this->mEvePhone = $_POST['evePhone'];
        if (!empty ($_POST['mobPhone']))
          $this->mMobPhone = $_POST['mobPhone'];
      }
    }
  }

  public function init()
  {
    // Если данные были отправлены и в них нет ошибок
    if ((isset ($_POST['sended'])) && ($this->_mErrors == 0))
    {
      // Проверяем, есть ли зарегистрированный пользователь
      // с таким адресом электронной почты...
      $customer_read = Customer::GetLoginInfo($this->mEmail);
      /* ...если есть, а мы создаем новую учетную запись,
         то сообщаем об ошибке*/
      if ((!(empty ($customer_read['customer_id']))) &&
        ($this->mEditMode == 0))
```

```
      {
        $this->mEmailAlreadyTaken = 1;
        return;
      }

      // Работаем с новым пользователем или обновляем сведения о
      // зарегистрированном ранее
      if ($this->mEditMode == 0)
        Customer::Add($this->mName, $this->mEmail, $this->mPassword);
      else
        Customer::UpdateAccountDetails($this->mName, $this->mEmail,
          $this->mPassword, $this->mDayPhone, $this->mEvePhone,
          $this->mMobPhone);

      header('Location:' . $this->mLinkToCancelPage);

      exit();
    }

    if ($this->mEditMode == 1 && !isset ($_POST['sended']))
    {
      // Редактируем сведения о зарегистрированном пользователе
      $customer_data = Customer::Get();

      $this->mName     = $customer_data['name'];
      $this->mEmail    = $customer_data['email'];
      $this->mDayPhone = $customer_data['day_phone'];
      $this->mEvePhone = $customer_data['eve_phone'];
      $this->mMobPhone = $customer_data['mob_phone'];
    }
  }
}
?>
```

7. Создайте в папке `presentation/templates` файл шаблона `customer_address.tpl` и поместите в него следующий код.

```
{* customer_address.tpl *}
{load_presentation_object filename="customer_address" assign="obj"}
<form method="post" action="{$obj->mLinkToAddressDetails}">
  <h2>Please enter your address details:</h2>
  <table class="customer-table">
    <tr>
      <td>Address 1:</td>
      <td>
        <input type="text" name="address1" value="{$obj->mAddress1}"
         size="32" />
        {if $obj->mAddress1Error}
        <p class="error">You must enter an address.</p>
        {/if}
      </td>
    </tr>
    <tr>
      <td>Address 2:</td>
      <td>
        <input type="text" name="address2" value="{$obj->mAddress2}"
         size="32" />
      </td>
    </tr>
    <tr>
      <td>Town/City:</td>
      <td>
```

```
        <input type="text" name="city" value="{$obj->mCity}"
         size="32" />
        {if $obj->mCityError}
        <p class="error">You must enter a city.</p>
        {/if}
      </td>
    </tr>
    <tr>
      <td>Region/State:</td>
      <td>
        <input type="text" name="region" value="{$obj->mRegion}"
         size="32" />
        {if $obj->mRegionError}
        <p class="error">You must enter a region/state.</p>
        {/if}
      </td>
    </tr>
    <tr>
      <td>Postal Code/ZIP:</td>
      <td>
        <input type="text" name="postalCode" value="{$obj->mPostalCode}"
         size="32" />
        {if $obj->mPostalCodeError}
        <p class="error">You must enter a postal code/ZIP.</p>
        {/if}
      </td>
    </tr>
    <tr>
      <td>Country:</td>
      <td>
        <input type="text" name="country" value="{$obj->mCountry}"
         size="32" />
        {if $obj->mCountryError}
        <p class="error">You must enter a country.</p>
        {/if}
      </td>
    </tr>
    <tr>
      <td>Shipping region:</td>
      <td>
        <select name="shippingRegion">
          {html_options options=$obj->mShippingRegions
           selected=$obj->mShippingRegion}
        </select>
        {if $obj->mShippingRegionError}
        <p class="error">You must select a shipping region.</p>
        {/if}
      </td>
    </tr>
  </table>
  <input type="submit" name="sended" value="Confirm" /> |
  <a href="{$obj->mLinkToCancelPage}">Cancel</a>
</form>
```

8. Создайте в папке `presentation` файл объекта представления `customer_address.php` и поместите в него следующий код.

```
<?php
class CustomerAddress
{
  // Public-атрибуты
  public $mAddress1 = '';
```

```
public $mAddress2 = '';
public $mCity = '';
public $mRegion = '';
public $mPostalCode = '';
public $mCountry = '';
public $mShippingRegion = '';
public $mShippingRegions = array ();
public $mAddress1Error = 0;
public $mCityError = 0;
public $mRegionError = 0;
public $mPostalCodeError = 0;
public $mCountryError = 0;
public $mShippingRegionError = 0;
public $mLinkToAddressDetails;
public $mLinkToCancelPage;

// Private-атрибуты
private $_mErrors = 0;

// Конструктор класса
public function __construct()
{
  // Задаем цель действия
  $this->mLinkToAddressDetails = Link::ToAddressDetails();

  // Задаем страницу для отмены
  if (isset ($_SESSION['customer_cancel_link']))
    $this->mLinkToCancelPage = $_SESSION['customer_cancel_link'];
  else
    $this->mLinkToCancelPage = Link::ToIndex();

  // Проверяем, отправлялись ли данные
  if (isset ($_POST['sended']))
  {
    // Адрес 1 не может быть пустым
    if (empty ($_POST['address1']))
    {
      $this->mAddress1Error = 1;
      $this->_mErrors++;
    }
    else
      $this->mAddress1 = $_POST['address1'];

    if (isset ($_POST['address2']))
      $this->mAddress2 = $_POST['address2'];

    if (empty ($_POST['city']))
    {
      $this->mCityError = 1;
      $this->_mErrors++;
    }
    else
      $this->mCity = $_POST['city'];

    if (empty ($_POST['region']))
    {
      $this->mRegionError = 1;
      $this->_mErrors++;
    }
    else
      $this->mRegion = $_POST['region'];
```

```
      if (empty ($_POST['postalCode']))
      {
        $this->mPostalCodeError = 1;
        $this->_mErrors++;
      }
      else
        $this->mPostalCode = $_POST['postalCode'];

      if (empty ($_POST['country']))
      {
        $this->mCountryError = 1;
        $this->_mErrors++;
      }
      else
        $this->mCountry = $_POST['country'];

      if ($_POST['shippingRegion'] == 1)
      {
        $this->mShippingRegionError = 1;
        $this->_mErrors++;
      }
      else
        $this->mShippingRegion = $_POST['shippingRegion'];
    }
  }

  public function init()
  {
    $shipping_regions = Customer::GetShippingRegions();

    foreach ($shipping_regions as $item)
      $this->mShippingRegions[$item['shipping_region_id']] =
        $item['shipping_region'];

    if (!isset ($_POST['sended']))
    {
      $customer_data = Customer::Get();

      if (!(empty ($customer_data)))
      {
        $this->mAddress1       = $customer_data['address_1'];
        $this->mAddress2       = $customer_data['address_2'];
        $this->mCity           = $customer_data['city'];
        $this->mRegion         = $customer_data['region'];
        $this->mPostalCode     = $customer_data['postal_code'];
        $this->mCountry        = $customer_data['country'];
        $this->mShippingRegion = $customer_data['shipping_region_id'];
      }
    }
    elseif ($this->_mErrors == 0)
    {
      Customer::UpdateAddressDetails($this->mAddress1, $this->mAddress2,
        $this->mCity, $this->mRegion, $this->mPostalCode,
        $this->mCountry, $this->mShippingRegion);

      header('Location:' . $this->mLinkToCancelPage);

      exit();
    }
  }
}
?>
```

9. Создайте в папке presentation/templates файл шаблона customer_credit_card.tpl и поместите в него следующий код.

```
{* customer_credit_card.tpl *}
{load_presentation_object filename="customer_credit_card" assign="obj"}
<form method="post" action="{$obj->mLinkToCreditCardDetails}">
  <h2>Please enter your credit card details:</h2>
  <table class="customer-table">
    <tr>
      <td>Card Holder:</td>
      <td>
        <input type="text" name="cardHolder" size="32"
         value="{$obj->mPlainCreditCard.card_holder}" />
        {if $obj->mCardHolderError}
        <p class="error">You must enter a card holder.</p>
        {/if}
      </td>
    </tr>
    <tr>
      <td>Card Number (digits only):</td>
      <td>
        <input type="text" name="cardNumber" size="32"
         value="{$obj->mPlainCreditCard.card_number}" />
        {if $obj->mCardNumberError}
        <p class="error">You must enter a card number.</p>
        {/if}
      </td>
    </tr>
    <tr>
      <td>Expiry Date (MM/YY):</td>
      <td>
        <input type="text" name="expDate" size="32"
         value="{$obj->mPlainCreditCard.expiry_date}" />
        {if $obj->mExpDateError}
        <p class="error">You must enter an expiry date</p>
        {/if}
      </td>
    </tr>
    <tr>
      <td>Issue Date (MM/YY if applicable):</td>
      <td>
        <input type="text" name="issueDate" size="32"
         value="{$obj->mPlainCreditCard.issue_date}" />
      </td>
    </tr>
    <tr>
      <td>Issue Number (if applicable):</td>
      <td>
        <input type="text" name="issueNumber" size="32"
         value="{$obj->mPlainCreditCard.issue_number}" />
      </td>
    </tr>
    <tr>
      <td>Card Type:</td>
      <td>
        <select name="cardType">
          {html_options options=$obj->mCardTypes
           selected=$obj->mPlainCreditCard.card_type}
        </select>
        {if $obj->mCardTypesError}
        <p class="error">You must enter a card type.</p>
```

```
        {/if}
      </td>
    </tr>
  </table>
  <input type="submit" name="sended" value="Confirm" /> |
  <a href="{$obj->mLinkToCancelPage}">Cancel</a>
</form>
```

10. Создайте в папке `presentation` файл объекта представления `customer_credit_card.php` и поместите в него следующий код.

```
<?php
class CustomerCreditCard
{
  // Public-атрибуты
  public $mCardHolderError;
  public $mCardNumberError;
  public $mExpDateError;
  public $mCardTypesError;
  public $mPlainCreditCard;
  public $mCardTypes;
  public $mLinkToCreditCardDetails;
  public $mLinkToCancelPage;

  // Private-атрибуты
  private $_mErrors = 0;

  public function __construct()
  {
    $this->mPlainCreditCard = array('card_holder' => '',
      'card_number' => '', 'issue_date' => '', 'expiry_date' => '',
      'issue_number' => '', 'card_type' => '', 'card_number_x' => '');

    // Задаем цель действия
    $this->mLinkToCreditCardDetails = Link::ToCreditCardDetails();

    // Задаем страницу для отмены
    if (isset ($_SESSION['customer_cancel_link']))
      $this->mLinkToCancelPage = $_SESSION['customer_cancel_link'];
    else
      $this->mLinkToCancelPage = Link::ToIndex();

    $this->mCardTypes = array ('Mastercard' => 'Mastercard',
      'Visa' => 'Visa', 'Mastercard' => 'Mastercard',
      'Switch' => 'Switch', 'Solo' => 'Solo',
      'American Express' => 'American Express');

     // Проверяем, отправлялись ли данные
     if (isset ($_POST['sended']))
     {
       // Инициализация и проверка данных
       if (empty ($_POST['cardHolder']))
       {
         $this->mCardHolderError = 1;
         $this->_mErrors++;
       }
       else
         $this->mPlainCreditCard['card_holder'] = $_POST['cardHolder'];

       if (empty ($_POST['cardNumber']))
       {
         $this->mCardNumberError = 1;
```

```
          $this->_mErrors++;
        }
        else
          $this->mPlainCreditCard['card_number'] = $_POST['cardNumber'];

        if (empty ($_POST['expDate']))
        {
          $this->mExpDateError = 1;
          $this->_mErrors++;
        }
        else
          $this->mPlainCreditCard['expiry_date'] = $_POST['expDate'];

      if (isset ($_POST['issueDate']))
        $this->mPlainCreditCard['issue_date'] = $_POST['issueDate'];

      if (isset ($_POST['issueNumber']))
        $this->mPlainCreditCard['issue_number'] = $_POST['issueNumber'];

      $this->mPlainCreditCard['card_type'] = $_POST['cardType'];

      if (empty ($this->mPlainCreditCard['card_type']))
      {
        $this->mCardTypeError = 1;
        $this->_mErrors++;
      }
    }
  }

  public function init()
  {
    if (!isset ($_POST['sended']))
    {
      // Получаем информацию о кредитной карте
      $this->mPlainCreditCard = Customer::GetPlainCreditCard();
    }
    elseif ($this->_mErrors == 0)
    {
      // Обновляем информацию о кредитной карте
      Customer::UpdateCreditCardDetails($this->mPlainCreditCard);

      header('Location:' . $this->mLinkToCancelPage);

      exit();
    }
  }
}
?>
```

11. Откройте файл `presentation/link.php` и добавьте в класс `Link` четыре новых метода.

```
// Создает ссылку на страницу регистрации пользователей
public static function ToRegisterCustomer()
{
  return self::Build('register-customer/', 'https');
}

// Создает ссылку на страницу обновления сведений о пользователе
public static function ToAccountDetails()
{
  return self::Build('account-details/', 'https');
```

```
}

// Создает ссылку на страницу обновления сведений о кредитной карте
public static function ToCreditCardDetails()
{
  return self::Build('credit-card-details/', 'https');
}

// Создает ссылку на страницу обновления сведений об адресе пользователя
public static function ToAddressDetails()
{
  return self::Build('address-details/', 'https');
}
```

12. В том же классе `Link` модифицируйте метод `CheckRequest()`, добавив в него фрагмент кода, выделенный полужирным шрифтом в приведенном ниже листинге.

```
public static function CheckRequest()
{
$proper_url = '';
if (isset ($_GET['Search']) || isset($_GET['SearchResults']) ||
    isset ($_GET['CartAction']) || isset ($_GET['AjaxRequest']) ||
    isset ($_POST['Login']) || isset ($_GET['Logout']) ||
    isset ($_GET['RegisterCustomer']) ||
    isset ($_GET['AddressDetails']) ||
    isset ($_GET['CreditCardDetails']) ||
    isset ($_GET['AccountDetails']))
{
  return ;
}
```

13. Откройте файл `.htaccess` и добавьте в него следующие правила для переписывания ссылок.

```
  # Переписываем страницы с содержимым корзины
  RewriteRule ^cart-details/?$ index.php?CartAction [L]

  # Переписываем страницы регистрации пользователей
  RewriteRule ^register-customer/?$ index.php?RegisterCustomer [L]

  # Переписываем страницы со сведениями об адресах
  RewriteRule ^address-details/?$ index.php?AddressDetails [L]

  # Переписываем страницы со сведениями о кредитных картах
  RewriteRule ^credit-card-details/?$ index.php?CreditCardDetails [L]

  # Переписываем страницы со сведениями об учетных записях
  RewriteRule ^account-details/?$ index.php?AccountDetails [L]
</IfModule>
```

14. Добавьте в файл `index.php` ссылки на класс хеширования паролей, класс симметричного шифрования, класс для работы с кредитными картами и класс учетных записей.

```
// Загружаем уровень логики приложения
require_once BUSINESS_DIR . 'catalog.php';
require_once BUSINESS_DIR . 'catalog.php';
require_once BUSINESS_DIR . 'shopping_cart.php';
require_once BUSINESS_DIR . 'password_hasher.php';
require_once BUSINESS_DIR . 'symmetric_crypt.php';
require_once BUSINESS_DIR . 'secure_card.php';
require_once BUSINESS_DIR . 'customer.php';
```

15. Откройте файл presentation/store_front.php и добавьте в класс StoreFront определение нового элемента, как показано в приведенном ниже листинге.

```
<?php
class StoreFront
{
  public $mSiteUrl;
  // Определяем файл шаблона для содержимого страницы
  public $mContentsCell = 'first_page_contents.tpl';
  // Определяем файл шаблона для ячейки категорий
  public $mCategoriesCell = 'blank.tpl';
  // Определяем файл шаблона для ячейки содержимого корзины
  public $mCartSummaryCell = 'blank.tpl';
  // Определяем файл шаблона для полей аутентификации
  public $mLoginOrLoggedCell = 'customer_login.tpl';
  // Заголовок страницы
  public $mPageTitle;
```

16. Вставьте в конец метода init() класса StoreFront новый код, как показано в приведенном ниже листинге.

```
    // Загружаем шаблоны для отображения содержимого корзины
    if (isset ($_GET['CartAction']))
      $this->mContentsCell = 'cart_details.tpl';
    else
      $this->mCartSummaryCell = 'cart_summary.tpl';

    if (Customer::IsAuthenticated())
      $this->mLoginOrLoggedCell = 'customer_logged.tpl';

    if (isset ($_GET['RegisterCustomer']) ||
        isset ($_GET['AccountDetails']))
      $this->mContentsCell = 'customer_details.tpl';
    elseif (isset ($_GET['AddressDetails']))
      $this->mContentsCell = 'customer_address.tpl';
    elseif (isset ($_GET['CreditCardDetails']))
      $this->mContentsCell = 'customer_credit_card.tpl';

    // Загружаем заголовок страницы
    $this->mPageTitle = $this->_GetPageTitle();
  }
```

17. В тот же метод init() вставьте еще один фрагмент кода, как показано в приведенном ниже листинге.

```
public function init()
{
  $_SESSION['link_to_store_front'] =
    Link::Build(str_replace(VIRTUAL_LOCATION, '', getenv('REQUEST_URI')));

  // Создаем ссылку "continue shopping"
  if (!isset ($_GET['CartAction']) && !isset($_GET['Logout']) &&
      !isset($_GET['RegisterCustomer']) &&
      !isset($_GET['AddressDetails']) &&
      !isset($_GET['CreditCardDetails']) &&
      !isset($_GET['AccountDetails']))
    $_SESSION['link_to_last_page_loaded'] =
$_SESSION['link_to_store_front'];

  // Создаем ссылку "cancel" для страницы со сведениями о пользователе
  if (!isset($_GET['Logout']) &&
```

```
      !isset($_GET['RegisterCustomer']) &&
      !isset($_GET['AddressDetails']) &&
      !isset($_GET['CreditCardDetails']) &&
      !isset($_GET['AccountDetails']))
    $_SESSION['customer_cancel_link'] = $_SESSION['link_to_store_front'];

  // Загружаем подробные сведения об отделе на страницу отдела
  if (isset ($_GET['DepartmentId']))
```

18. Добавьте в файл `presentation/templates/store_front.tpl` одну новую строку:

```
<div class="yui-b">
  {include file=$obj->mLoginOrLoggedCell}
  {include file="search_box.tpl"}
  {include file="departments_list.tpl"}
  {include file=$obj->mCategoriesCell}
```

19. Добавьте в файл `styles/tshirtshop.css` новый стиль.

```
.customer-table td {
  border: none;
  padding-left: 0;
  vertical-align: top;
}
```

20. Загрузите сайт в браузер и убедитесь в том, что все работает так, как показано на рис. 16.4 – 16.7.

Создание страницы оплаты

Теперь можно приступить к созданию страницы оплаты. Эта страница похожа на страницу корзины (компонентный шаблон `cart_details`), поскольку отображает список выбранных товаров, но кроме того, на ней отображается дополнительная информация, например адрес доставки и тип кредитной карты. Чтобы открыть страницу оплаты, пользователю нужно будет щелкнуть на ссылке Checkout на странице корзины (мы добавим эту ссылку на страницу в следующем упражнении).

Приблизительный вид страницы оплаты показан на рис. 16.8.

Если пользователь попытается выполнить оплату, не указав необходимые сведения, кнопка Place Order будет недоступна и появится сообщение о том, каких данных не хватает. Пример такого сообщения показан на рис. 16.9.

Как вы помните, мы уже создали страницы, позволяющие пользователю вводить и редактировать сведения о кредитных картах и адресах доставки. А теперь давайте создадим компонентный шаблон `checkout_info`, отвечающий за страницу, показанную на рис. 16.8.

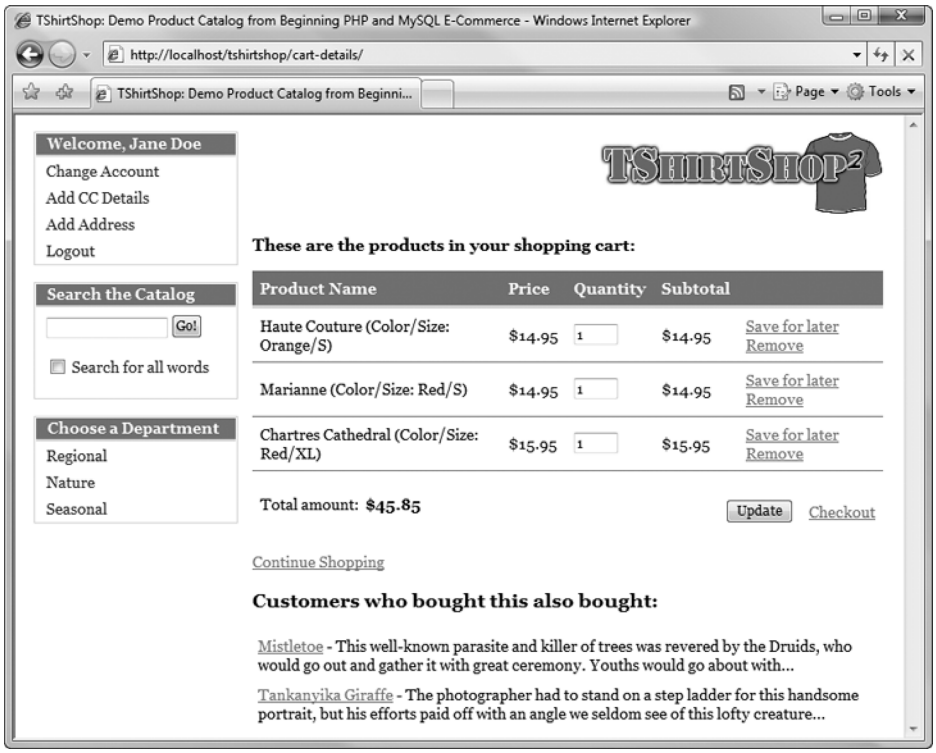


Рис. 16.8. Страница оплаты

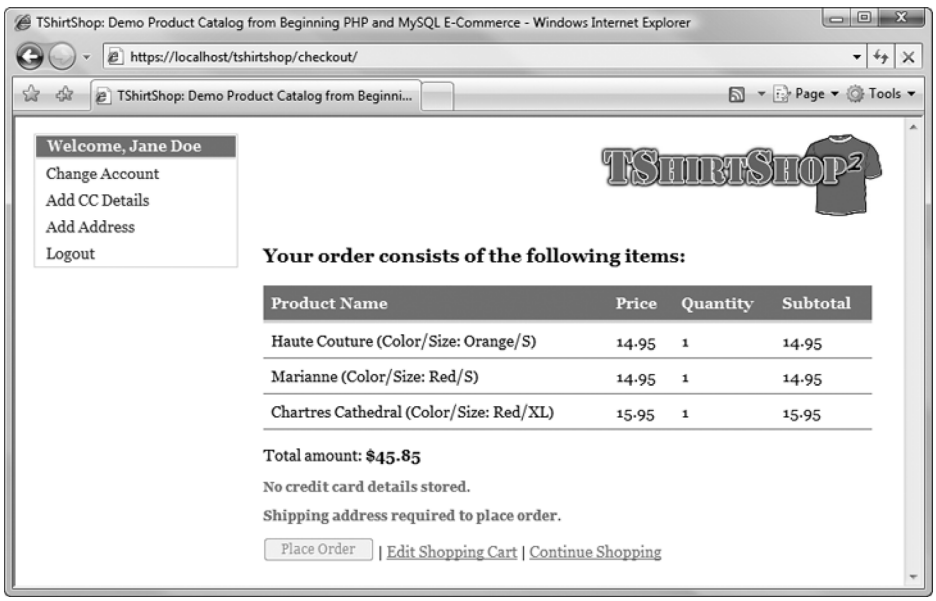


Рис. 16.9. Сообщение об отсутствии необходимых данных

Упражнение: создание компонентного шаблона checkout_info

1. Создайте в папке presentation/templates файл checkout_info.tpl и поместите в него следующий код.

```
{* checkout_info.tpl *}
{load_presentation_object filename="checkout_info" assign="obj"}
<form method="post" action="{$obj->mLinkToCheckout}">
  <h2>Your order consists of the following items:</h2>
  <table class="tss-table">
    <tr>
      <th>Product Name</th>
      <th>Price</th>
      <th>Quantity</th>
      <th>Subtotal</th>
    </tr>
  {section name=i loop=$obj->mCartItems}
    <tr>
      <td>{$obj->mCartItems[i].name} ({$obj-
>mCartItems[i].attributes})</td>
      <td>{$obj->mCartItems[i].price}</td>
      <td>{$obj->mCartItems[i].quantity}</td>
      <td>{$obj->mCartItems[i].subtotal}</td>
    </tr>
  {/section}
  </table>
  <p>Total amount: <font class="price">${$obj->mTotalAmount}</font></p>
  {if $obj->mNoCreditCard == 'yes'}
  <p class="error">No credit card details stored.</p>
  {else}
  <p>{$obj->mCreditCardNote}</p>
  {/if}
  {if $obj->mNoShippingAddress == 'yes'}
  <p class="error">Shipping address required to place order.</p>
  {else}
  <p>
    Shipping address: <br />
     {$obj->mCustomerData.address_1}<br />
    {if $obj->mCustomerData.address_2}
       {$obj->mCustomerData.address_2}<br />
    {/if}
     {$obj->mCustomerData.city}<br />
     {$obj->mCustomerData.region}<br />
     {$obj->mCustomerData.postal_code}<br />
     {$obj->mCustomerData.country}<br /><br />
    Shipping region: {$obj->mShippingRegion}
  </p>
  {/if}
  <input type="submit" name="place_order" value="Place Order"
    {$obj->mOrderButtonVisible} /> |
  <a href="{$obj->mLinkToCart}">Edit Shopping Cart</a> |
  <a href="{$obj->mLinkToContinueShopping}">Continue Shopping</a>
</form>
```

2. Создайте в папке presentation файл checkout_info.php и поместите в него следующий код.

```
<?php
// Класс, отвечающий за страницу оплаты
class CheckoutInfo
```

```
{
  // Public-атрибуты
  public $mCartItems;
  public $mTotalAmount;
  public $mCreditCardNote;
  public $mOrderButtonVisible;
  public $mNoShippingAddress = 'no';
  public $mNoCreditCard = 'no';
  public $mPlainCreditCard;
  public $mShippingRegion;
  public $mLinkToCheckout;
  public $mLinkToCart;
  public $mLinkToContinueShopping;

  // Конструктор класса
  public function __construct()
  {
    $this->mLinkToCheckout = Link::ToCheckout();
    $this->mLinkToCart = Link::ToCart();
    $this->mLinkToContinueShopping =
$_SESSION['link_to_last_page_loaded'];
  }

  public function init()
  {
    // Задаем элементы для использования в шаблоне Smarty
    $this->mCartItems = ShoppingCart::GetCartProducts(GET_CART_PRODUCTS);
    $this->mTotalAmount = ShoppingCart::GetTotalAmount();
    $this->mCustomerData = Customer::Get();

    // Если была нажата кнопка Place Order, сохраняем заказ в базу дан-
ных...
    if(isset ($_POST['place_order']))
    {
      // Создаем заказ и получаем для него ID
      $order_id = ShoppingCart::CreateOrder();

      // Здесь будет ссылка на PayPal
      $redirect =
        PAYPAL_URL . '&item_name=TShirtShop Order ' .
        urlencode('#') . $order_id .
        '&item_number=' . $order_id .
        '&amount=' . $this->mTotalAmount .
        '&currency_code=' . PAYPAL_CURRENCY_CODE .
        '&return=' . PAYPAL_RETURN_URL .
        '&cancel_return=' . PAYPAL_CANCEL_RETURN_URL;

      // Перенаправление на страницу оплаты
      header('Location: ' . $redirect);

      exit();
    }

    // Заказы размещать могут только пользователи, указавшие все сведения
    if (empty ($this->mCustomerData['credit_card']))
    {
      $this->mOrderButtonVisible = 'disabled="disabled"';
      $this->mNoCreditCard = 'yes';
    }
    else
    {
```

```
      $this->mPlainCreditCard = Customer::DecryptCreditCard(
                                  $this->mCustomerData['credit_card']);

      $this->mCreditCardNote = 'Credit card to use: ' .
                               $this->mPlainCreditCard['card_type'] .
                               '<br />Card number: ' .
                               $this->mPlainCreditCard['card_number_x'];
    }

    if (empty ($this->mCustomerData['address_1']))
    {
      $this->mOrderButtonVisible = 'disabled="disabled"';
      $this->mNoShippingAddress = 'yes';
    }
    else
    {
      $shipping_regions = Customer::GetShippingRegions();
      foreach ($shipping_regions as $item)
        if ($item['shipping_region_id'] ==
            $this->mCustomerData['shipping_region_id'])
      $this->mShippingRegion = $item['shipping_region'];
    }
  }
}
?>
```

3. Создайте в папке `presentation/templates` файл `checkout_not_logged.tpl` и поместите в него следующий код.

```
{* checkout_not_logged.tpl *}
<h3>
  You must be logged in to CHECKOUT <br />
  If you don't have an account please register <br />
</h3>
```

Этот шаблон отображает сообщение о том, что пользователи, не прошедшие аутентификацию, не могут оплачивать заказы (без аутентификации пользователь не сможет щелкнуть на ссылке Checkout, но сможет попытаться набрать URL страницы оплаты в поле адреса в браузере).

4. Откройте файл `presentation/link.php` и добавьте в конец класса `Link` следующий метод.

```
// Создает ссылку на страницу оплаты
public static function ToCheckout()
{
  return self::Build('checkout/', 'https');
}
```

5. В том же файле модифицируйте метод `CheckRequest()` класса `Link`, добавив в него строку, выделенную полужирным шрифтом в приведенном ниже листинге.

```
if (isset ($_GET['Search']) || isset($_GET['SearchResults']) ||
    isset ($_GET['CartAction']) || isset ($_GET['AjaxRequest']) ||
    isset ($_POST['Login']) || isset ($_GET['Logout']) ||
    isset ($_GET['RegisterCustomer']) ||
    isset ($_GET['AddressDetails']) ||
    isset ($_GET['CreditCardDetails']) ||
    isset ($_GET['AccountDetails']) || isset ($_GET['Checkout']))
{
  return ;
}
```

6. Откройте файл .htaccess из корневой папки проекта и добавьте в него новое правило для переписывания ссылок.

```
  # Переписываем страницы со сведениями об учетных записях
  RewriteRule ^account-details/?$ index.php?AccountDetails [L]

  # Переписываем страницы оплаты
  RewriteRule ^checkout/?$ index.php?Checkout [L]
</IfModule>
```

7. Откройте файл presentation/store_front.php и модифицируйте метод init() класса StoreFront так, как показано ниже, чтобы он правильно генерировал ссылки Continue Shopping.

```
  // Создаем ссылку "continue shopping"
if (!isset ($_GET['CartAction']) && !isset($_GET['Logout']) &&
    !isset($_GET['RegisterCustomer']) &&
    !isset($_GET['AddressDetails']) &&
    !isset($_GET['CreditCardDetails']) &&
    !isset($_GET['AccountDetails']) &&
    !isset($_GET['Checkout']))
  $_SESSION['link_to_last_page_loaded'] =
$_SESSION['link_to_store_front'];

// Создаем ссылку "cancel" для страницы со сведениями о пользователе
if (!isset($_GET['Logout']) &&
    !isset($_GET['RegisterCustomer']) &&
    !isset($_GET['AddressDetails']) &&
    !isset($_GET['CreditCardDetails']) &&
    !isset($_GET['AccountDetails']))
  $_SESSION['customer_cancel_link'] = $_SESSION['link_to_store_front'];

// Загружаем подробные сведения об отделе на страницу отдела
if (isset ($_GET['DepartmentId']))
{
  $this->mContentsCell = 'department.tpl';
  $this->mCategoriesCell = 'categories_list.tpl';
}
```

8. В том же файле (presentation/store_front.php) добавьте новый элемент в определение класса StoreFront.

```
  // Определяем файл шаблона для ячейки содержимого корзины
  public $mCartSummaryCell = 'blank.tpl';
  // Определяем файл шаблона для полей аутентификации
  public $mLoginOrLoggedCell = 'customer_login.tpl';
  // Управляет отображением навигационных полей (списка отделов и т.д.)
  public $mHideBoxes = false;
  // Заголовок страницы
  public $mPageTitle;
```

9. В том же классе StoreFront вставьте в код метода init() фрагмент кода, выделенный полужирным шрифтом в приведенном ниже листинге.

```
  if (isset ($_GET['RegisterCustomer']) ||
      isset ($_GET['AccountDetails']))
    $this->mContentsCell = 'customer_details.tpl';
  elseif (isset ($_GET['AddressDetails']))
    $this->mContentsCell = 'customer_address.tpl';
  elseif (isset ($_GET['CreditCardDetails']))
    $this->mContentsCell = 'customer_credit_card.tpl';

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

```
  {
    if (Customer::IsAuthenticated())
      $this->mContentsCell = 'checkout_info.tpl';
    else
      $this->mContentsCell = 'checkout_not_logged.tpl';

    $this->mHideBoxes = true;
  }

    // Загружаем заголовок страницы
    $this->mPageTitle = $this->_GetPageTitle();
  }
```

10. Модифицируйте фрагмент кода в файле `presentation/templates/store_front.tpl` так, как показано ниже, чтобы на странице оплаты отображалось или поле аутентификации, или сообщение о пройденной аутентификации.

```
        <div class="yui-b">
          {include file=$obj->mLoginOrLoggedCell}
          {if !$obj->mHideBoxes}
            {include file="search_box.tpl"}
            {include file="departments_list.tpl"}
            {include file=$obj->mCategoriesCell}
            {include file=$obj->mCartSummaryCell}
          {/if}
        </div>
      </div>
    </div>
  </body>
</html>
```

11. Модифицируйте код в файле `presentation/templates/cart_details.tpl`, чтобы он перенаправлял пользователей на страницу `checkout_info`, а не на сайт PayPal. Кнопка Place Order превратится в ссылку Checkout.

```
...
  <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>
      {if $obj->mShowCheckoutLink}
      <td align="right">
        <a href="{$obj->mLinkToCheckout}">Checkout</a>
      </td>
      {/if}
    </tr>
  </table>
</form>
{/if}
...
```

12. В файле `presentation/cart_details.php` добавьте в определение класса `CartDetails` два новых `public`-элемента.

```
...
  public $mRecommendations;
  public $mLinkToCheckout;
  public $mShowCheckoutLink = false;

  // Private-атрибуты
  private $_mItemId;
...
```

13. Обновите метод `init()` в классе `CartDetails`, удалив из него код для PayPal и вставив вместо него код, выделенный полужирным шрифтом в приведенном ниже листинге (можете удалять код для PayPal безбоязненно — выше мы уже поместили код для ссылки на PayPal в файл `checkout_info.php`).

```
...
    /* Вычисляем общую стоимость товаров в корзине
       без учета налогов и цены доставки */
    $this->mTotalAmount = ShoppingCart::GetTotalAmount();

    // Отображаем ссылку Checkout для корзины
    if ($this->mTotalAmount != 0 && Customer::IsAuthenticated())
    {
      $this->mLinkToCheckout = Link::ToCheckout();
      $this->mShowCheckoutLink = true;
    }
```

14. Теперь все готово, и можно насладиться результатом. Откройте сайт в браузере, пройдите аутентификацию, поместите в корзину несколько товаров и щелкните на ссылке `Checkout` на странице корзины. В браузере должна открыться страница, похожая на изображенную на рис. 16.9.

Как это работает: компонентный шаблон `checkout_info`

В методе `init()` класса `CheckoutInfo` мы сначала проверяем, щелкнул ли пользователь на кнопке Place Order. Если щелкнул, сохраняем заказ в базе данных и перенаправляем пользователя на домашнюю страницу.

```
// Если была нажата кнопка Place Order, сохраняем заказ в базу данных...
if(isset ($_POST['place_order']))
{
  // Создаем заказ и получаем для него ID
  $order_id = ShoppingCart::CreateOrder();

  // Здесь будет ссылка на PayPal
  $redirect =
    PAYPAL_URL . '&item_name=TShirtShop Order ' .
    urlencode('#') . $order_id .
    '&item_number=' . $order_id .
    '&amount=' . $this->mTotalAmount .
    '&currency_code=' . PAYPAL_CURRENCY_CODE .
    '&return=' . PAYPAL_RETURN_URL .
    '&cancel_return=' . PAYPAL_CANCEL_RETURN_URL;

  // Перенаправление на страницу оплаты
  header('Location: ' . $redirect);

  exit();
}
```

Кроме того, мы объявляем несколько переменных, которые будут использоваться в шаблоне.

```
// Задаем элементы для использования в шаблоне Smarty
$this->mCartItems = ShoppingCart::GetCartProducts(GET_CART_PRODUCTS);
$this->mTotalAmount = ShoppingCart::GetTotalAmount();
$this->mCustomerData = Customer::Get();
```

Если пользователь не указал адрес доставки или сведения о кредитной карте, мы выводим соответствующие сообщения и блокируем кнопку Place Order. Если все нужные сведения есть, мы расшифровываем информацию о кредитной карте и отображаем ее тип и последние четыре цифры номера.

```
// Заказы размещать могут только пользователи, указавшие все сведения
if (empty ($this->mCustomerData['credit_card']))
{
  $this->mOrderButtonVisible = 'disabled="disabled"';
  $this->mNoCreditCard = 'yes';
}
else
{
  $this->mPlainCreditCard = Customer::DecryptCreditCard(
                              $this->mCustomerData['credit_card']);

  $this->mCreditCardNote = 'Credit card to use: ' .
                           $this->mPlainCreditCard['card_type'] .
                           '<br />Card number: ' .
                           $this->mPlainCreditCard['card_number_x'];
}

if (empty ($this->mCustomerData['address_1']))
{
  $this->mOrderButtonVisible = 'disabled="disabled"';
  $this->mNoShippingAddress = 'yes';
}
```

Остальной код не содержит ничего особенно интересного.

Защита конфиденциальных сведений с помощью SSL

Создавая механизмы администрирования каталога, мы упоминали о том, что для передачи конфиденциальных данных между сервером и браузером клиента лучше использовать соединения, защищенные с помощью SSL. Впрочем, в том случае использовать SSL было необязательно, поскольку можно было разрешить только локальный доступ к механизмам администрирования.

Однако теперь SSL станет для нас необходимым — мы не можем допустить пересылку сведений о кредитных картах через Интернет в незашифрованном виде. Если вы следовали инструкциям из главы 10, страницы со сведениями о пользователях уже должны быть защищены. Не забудьте, что применение SSL на вашем сайте настраивается в файле `include/config.php`.

Нам нужно сделать так, чтобы страницы с конфиденциальной информацией были доступны *только через SSL*. Например, если пользователь попытается открыть в браузере страницу `http://localhost/tshirtshop/credit-card-details/`, сайт должен перенаправить его по адресу `https://localhost/tshirtshop/credit-card-details/` (при условии, что пользователь прошел аутентификацию). Таким образом, нужно защитить страницу оплаты, страницу аутентификации, страницу регистрации новых пользователей и страницу редактирования учетных записей.

Разумеется, защищать с помощью SSL *все* страницы сайта бессмысленно. Во-первых, это сильно увеличит нагрузку на сервер, поскольку все данные, которые он будет получать и отправлять, будут зашифрованными, а шифрование — весьма сложная математическая процедура. Во-вторых, доступные только через SSL страницы не видны поисковым системам, а нам нужно, чтобы поисковые системы могли просматривать каталог товаров на нашем сайте.

Если сайт настроен на работу с SSL так, как описано в главе 10, достаточно будет внести несколько изменений в код, чтобы защитить нужные страницы от доступа через HTTP. Сначала добавьте новый метод в конец класса `StoreFront` (в файле `presentation/store_front.php`).

```
// Обращаемся к защищенной странице?
private function _IsSensitivePage()
{
  if (isset($_GET['RegisterCustomer']) ||
      isset($_GET['AccountDetails']) ||
      isset($_GET['CreditCardDetails']) ||
      isset($_GET['AddressDetails']) ||
      isset($_GET['Checkout']) ||
      isset($_POST['Login']))
    return true;

  return false;
}
```

Затем добавьте в конструктор класса `StoreFront` код, выделенный полужирным шрифтом в приведенном ниже листинге.

```
// Конструктор класса
public function __construct()
{
  $is_https = false;

  // К странице обращаются через HTTPS-соединение?
  if (getenv('HTTPS') == 'on')
    $is_https = true;

  // Используем HTTPS для доступа к страницам с
  // конфиденциальной информацией
  if ($this->_IsSensitivePage() && $is_https == false &&
      USE_SSL != 'no')
  {
    $redirect_to =
      Link::Build(str_replace(VIRTUAL_LOCATION, '',
getenv('REQUEST_URI')),
                  'https');

    header ('Location: '. $redirect_to);

    exit();
  }

  // Не используем HTTPS для страниц, не
  // содержащих конфиденциальной информации
  if (!$this->_IsSensitivePage() && $is_https == true)
  {
    $redirect_to =
      Link::Build(str_replace(VIRTUAL_LOCATION, '',
getenv('REQUEST_URI')));
```

```
    header ('Location: '. $redirect_to);
    exit();
  }
  $this->mSiteUrl = Link::Build('');
}
```

Теперь сайт будет автоматически перенаправлять пользователей, прошедших аутентификацию, по адресам, использующим HTTPS.

Резюме

В этой главе мы реализовали систему, которая позволяет пользователям создавать на нашем сайте личные учетные записи и сохранять в них сведения, необходимые для оформления заказов. При этом мы рассмотрели различные аспекты системы учетных записей, включая шифрование конфиденциальных данных и использование защищенных соединений.

Мы начали с создания в базе данных новой таблицы, предназначенной для хранения информации о пользователях.

Затем мы добавили на уровень логики приложения классы, отвечающие за шифрование конфиденциальных данных, и класс для хранения сведений о кредитных картах.

После этого мы применили данные классы для создания страниц аутентификации, регистрации и редактирования учетных записей. И наконец мы защитили страницы с конфиденциальными данными, разрешив доступ к ним только с помощью SSL.

Глава 17

Хранение заказов

Наш проект электронного магазина постепенно обретает законченные очертания. Мы добавили в него механизмы работы с учетными записями посетителей, а также механизмы сохранения в базе данных адресов доставки заказов и сведений о кредитных картах. Но пока что мы не используем эту информацию в системе отслеживания заказов, которую создали на втором этапе разработки, — мы вообще не связываем заказ с учетной записью посетителя, сделавшего заказ.

В этой главе мы внесем в проект изменения, необходимые для связывания заказов и учетных записей посетителей. Это позволит нам с помощью базы данных отслеживать процесс обработки заказов, сделанных конкретными посетителями. Кроме того, мы заложим основы конвейера для их обработки и для выполнения операций с кредитными картами.

Также мы рассмотрим вопросы налогообложения покупок и транспортных расходов. Учитывать налоги и транспортные расходы можно разными способами, но мы рассмотрим простейший из них и заложим основы для последующего создания более эффективного механизма.

Эта глава состоит из трех частей, посвященных соответственно

- созданию механизма размещения заказов через учетные записи пользователей,
- модификации механизмов администрирования для интеграции новых возможностей,
- учету налогов и транспортных расходов.

В следующей главе мы приступим к реализации более изощренной системы, и код, который мы напишем в этой главе, будет полностью задействован только в следующей. Поэтому некоторые из нововведений, которые мы воплотим в жизнь ниже, могут сначала показаться ненужными. Однако в дальнейшем они докажут свою необходимость.

Добавление заказов в учетные записи пользователей

Чтобы посетители сайта могли размещать заказы, нужно внести в наш механизм размещения заказов несколько изменений. Сейчас заказы, которые мы сохраняем в базе данных, никак не связаны с учетными записями пользователей. В этом разделе мы позволим пользователям создавать заказы и размещать их через свои учетные записи.

Сначала мы модифицируем базу данных, чтобы она могла хранить информацию о заказах конкретных посетителей. Для этого сначала изменим структуру таблицы `orders`, а затем — хранимую процедуру `shopping_cart_create_order`.

ПРЕДУПРЕЖДЕНИЕ

Изменения, которые мы внесем в следующем упражнении в таблицу `orders`, сделают ее несовместимой с теми данными, которые хранятся в ней сейчас, и эти данные будут удалены. Если вы хотите их сохранить, создайте резервную копию базы данных, прежде чем выполнять упражнение.

Вот изменения, которые нужно внести в таблицу `orders`.

- Удалить все данные, хранящиеся в ней сейчас.
- Удалить столбцы `customer_name`, `shipping_address` и `customer_email`.
- Добавить столбцы `customer_id`, `auth_code` и `reference`. Поле `customer_id` будет ссылаться на таблицу `customer`, указывая, какой посетитель сделал конкретный заказ. Остальные два поля нужны для работы с кредитными картами и будут подробно описаны в главе 20.

Кроме того, мы модифицируем хранимую процедуру `shopping_cart_create_order` соответственно изменениям в таблице `orders`. Выполните следующее упражнение, чтобы внести изменения в таблицу `orders` и хранимую процедуру `shopping_cart_create_order`.

Упражнение: добавление заказов в учетные записи

1. Запустите phpMyAdmin, откройте базу данных `tshirtshop` и перейдите на страницу нового SQL-запроса.

ПРЕДУПРЕЖДЕНИЕ

Создайте резервную копию базы данных, прежде чем выполнять последующие инструкции, поскольку они приведут к удалению данных из таблиц `order_detail` и `orders`.

2. С помощью страницы нового SQL-запроса выполните следующие операторы, чтобы удалить данные из таблиц `order_detail` и `orders`:

```
-- Удаляем все записи из таблицы order_detail
TRUNCATE TABLE order_detail;

-- Удаляем все записи из таблицы orders
TRUNCATE TABLE orders;
```

3. Удалите поля `customer_name`, `shipping_address` и `customer_email` из таблицы `orders`. Они там больше не нужны, поскольку хранившиеся в них данные теперь будут храниться в таблице `customer`. Для удаления полей выполните следующий оператор SQL.

```
-- Удаляем столбцы customer_name, shipping_address и customer_email
-- из таблицы orders
ALTER TABLE `orders` DROP COLUMN `customer_name`,
                     DROP COLUMN `shipping_address`,
                     DROP COLUMN `customer_email`;
```

4. Добавьте в таблицу `customer` новые поля (`customer_id`, `auth_code` и `reference`), а затем создайте индекс для поля `customer_id`. Чтобы сделать это, выполните следующие операторы SQL.

```
-- Добавляем три новых поля: customer_id, auth_code и reference.
ALTER TABLE `orders` ADD COLUMN `customer_id` INT,
                     ADD COLUMN `auth_code` VARCHAR(50),
```

```
                    ADD COLUMN `reference` VARCHAR(50);

-- Добавляем в таблицу orders новый индекс
CREATE INDEX `idx_orders_customer_id` ON `orders` (`customer_id`);
```

5. Удалите старый вариант хранимой процедуры `shopping_cart_create_order` и создайте вместо него новый (не забудьте предварительно задать символы `$$` в качестве разделителя).

```
-- Удаляем хранимую процедуру shopping_cart_create_order
DROP PROCEDURE shopping_cart_create_order$$

-- Создаем хранимую процедуру shopping_cart_create_order
CREATE PROCEDURE shopping_cart_create_order(IN inCartId CHAR(32),
  IN inCustomerId INT)
BEGIN
  DECLARE orderId INT;

  -- Вставляем в таблицу orders новую запись и получаем идентификатор
  -- нового заказа
  INSERT INTO orders (created_on, customer_id) VALUES (NOW(),
inCustomerId);
  -- Получаем новый идентификатор заказа
  SELECT LAST_INSERT_ID() INTO orderId;

  -- Вставляем подробности заказа в таблицу order_detail
  INSERT INTO order_detail (order_id, product_id, attributes,
                            product_name, quantity, unit_cost)
  SELECT      orderId, p.product_id, sc.attributes, p.name, sc.quantity,
              COALESCE(NULLIF(p.discounted_price, 0), p.price) AS
unit_cost
  FROM        shopping_cart sc
  INNER JOIN  product p
                ON sc.product_id = p.product_id
  WHERE       sc.cart_id = inCartId AND sc.buy_now;

  -- Сохраняем общую стоимость заказа
  UPDATE orders
  SET    total_amount = (SELECT SUM(unit_cost * quantity)
                         FROM   order_detail
                         WHERE  order_id = orderId)
  WHERE  order_id = orderId;

  -- Очищаем корзину
  CALL shopping_cart_empty(inCartId);

  -- Возвращаем идентификатор заказа
  SELECT orderId;
END$$
```

6. Замените старый метод `CreateOrder()` класса `ShoppingCart` в файле `business/shopping_cart.php` таким.

```
// Создаем новый заказ
public static function CreateOrder($customerId)
{
  // Составляем SQL-запрос
  $sql = 'CALL shopping_cart_create_order(:cart_id, :customer_id)';

  // Создаем массив параметров
  $params = array (':cart_id' => self::GetCartId(),
                   ':customer_id' => $customerId);
```

```
  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetOne($sql, $params);
}
```

7. Модифицируйте метод `init()` класса `CheckoutInfo` в файле `presentation/checkout_info.php` так, как показано в следующем листинге.

```
  public function init()
  {
    // Задаем элементы для использования в шаблоне Smarty
    $this->mCartItems = ShoppingCart::GetCartProducts(GET_CART_PRODUCTS);
    $this->mTotalAmount = ShoppingCart::GetTotalAmount();
    $this->mCustomerData = Customer::Get();

    // Если была нажата кнопка Place Order, сохраняем заказ в базу данных...
    if(isset ($_POST['place_order']))
    {
      // Создаем заказ и получаем для него ID
      $order_id =
ShoppingCart::CreateOrder(Customer::GetCurrentCustomerId());

      // Здесь будет ссылка на PayPal
      $redirect =
        PAYPAL_URL . '&item_name=TShirtShop Order ' .
        urlencode('#') . $order_id .
        '&item_number=' . $order_id .
        '&amount=' . $this->mTotalAmount .
        '&currency_code=' . PAYPAL_CURRENCY_CODE .
        '&return=' . PAYPAL_RETURN_URL .
        '&cancel_return=' . PAYPAL_CANCEL_RETURN_URL;

      // Перенаправление на страницу оплаты
      header('Location: ' . $redirect);

      exit();
    }
```

8. Попробуйте создать заказ или два с помощью новой системы, чтобы убедиться, что она работает. Для этого вам понадобится пройти аутентификацию и указать все требуемые данные, чтобы заказ прошел проверку.

ЗАМЕЧАНИЕ

На данный момент страница администрирования заказов не работает. Ее нужно модифицировать, чтобы она могла взаимодействовать с изменившейся базой данных.

Как это работает: добавление заказов в учетные записи

Код, с которым мы работали в этом упражнении, очень прост и едва ли заслуживает подробного рассмотрения. Функции обработки заказов на уровне логики приложения и уровне представления теперь принимают в качестве параметра идентификатор посетителя, создавшего заказ.

Позже, написав дополнительный код, мы сможем предоставлять пользователям дополнительную информацию, например отправлять письма с уведомлениями об отправке заказанных товаров. Но сейчас нам нужно заняться администрированием заказов.

Администрирование заказов

Итак, теперь наш сайт отслеживает связи между посетителями и заказами, которые делают эти посетители. Теперь нам нужно обновить механизм администрирования заказов, поскольку старый перестал работать. Чтобы сделать это, нужно внести изрядное количество изменений в уровень данных и уровень логики приложения, создав новые структуры данных и код для доступа к ним. Из-за большого объема изменений мы рассмотрим по отдельности уровень данных, уровень логики приложения и уровень представления.

Изменение уровня данных

На уровне данных нужно изменить код ряда хранимых процедур и добавить несколько новых. Вот процедуры, которые нужно модифицировать.

- `orders_get_most_recent_orders`
- `orders_get_orders_between_dates`
- `orders_get_orders_by_status`
- `orders_get_order_info`
- `orders_update_order`

А вот три новые хранимые процедуры.

- `orders_get_by_customer_id`
- `orders_get_order_short_details`
- `customer_get_customers_list`

Упражнение: модификация уровня данных

1. С помощью phpMyAdmin поочередно выполняйте операторы SQL из каждого шага упражнения. Не забывайте задавать символы `$$` в качестве разделителя перед выполнением каждого шага.
2. Выполните следующие операторы SQL, чтобы удалить старый вариант хранимой процедуры `orders_get_most_recent_orders` и создать новый.

```
-- Удаляем хранимую процедуру orders_get_most_recent_orders
DROP PROCEDURE orders_get_most_recent_orders$$

-- Создаем хранимую процедуру orders_get_most_recent_orders
CREATE PROCEDURE orders_get_most_recent_orders(IN inHowMany INT)
BEGIN
  PREPARE statement FROM
    "SELECT     o.order_id, o.total_amount, o.created_on,
                o.shipped_on, o.status, c.name
     FROM       orders o
     INNER JOIN customer c
                  ON o.customer_id = c.customer_id
     ORDER BY   o.created_on DESC
     LIMIT      ?";

  SET @p1 = inHowMany;

  EXECUTE statement USING @p1;
END$$
```

3. Чтобы удалить старый вариант хранимой процедуры `orders_get_orders_between_dates` и создать новый, выполните такие операторы SQL.

```
-- Удаляем хранимую процедуру orders_get_orders_between_dates
DROP PROCEDURE orders_get_orders_between_dates$$

-- Создаем хранимую процедуру orders_get_orders_between_dates
CREATE PROCEDURE orders_get_orders_between_dates(
  IN inStartDate DATETIME, IN inEndDate DATETIME)
BEGIN
  SELECT     o.order_id, o.total_amount, o.created_on,
             o.shipped_on, o.status, c.name
  FROM       orders o
  INNER JOIN customer c
               ON o.customer_id = c.customer_id
  WHERE      o.created_on >= inStartDate AND o.created_on <= inEndDate
  ORDER BY   o.created_on DESC;
END$$
```

4. Чтобы удалить старый вариант хранимой процедуры `orders_get_orders_by_status` и создать новый, выполните такие операторы SQL.

```
-- Удаляем хранимую процедуру orders_get_orders_by_status
DROP PROCEDURE orders_get_orders_by_status$$

-- Создаем хранимую процедуру orders_get_orders_by_status
CREATE PROCEDURE orders_get_orders_by_status(IN inStatus INT)
BEGIN
  SELECT     o.order_id, o.total_amount, o.created_on,
             o.shipped_on, o.status, c.name
  FROM       orders o
  INNER JOIN customer c
               ON o.customer_id = c.customer_id
  WHERE      o.status = inStatus
  ORDER BY   o.created_on DESC;
END$$
```

5. Удалите старую хранимую процедуру `orders_get_order_info` и создайте новую, выполнив следующий код.

```
-- Удаляем хранимую процедуру orders_get_order_info
DROP PROCEDURE orders_get_order_info$$

-- Создаем хранимую процедуру orders_get_order_info
CREATE PROCEDURE orders_get_order_info(IN inOrderId INT)
BEGIN
  SELECT order_id, total_amount, created_on, shipped_on, status,
         comments, customer_id, auth_code, reference
  FROM   orders
  WHERE  order_id = inOrderId;
END$$
```

6. Удалите старую хранимую процедуру `orders_update_order` и создайте новую, выполнив следующий код.

```
-- Удаляем хранимую процедуру orders_update_order
DROP PROCEDURE orders_update_order$$

-- Создаем хранимую процедуру orders_update_order
CREATE PROCEDURE orders_update_order(IN inOrderId INT, IN inStatus INT,
  IN inComments VARCHAR(255), IN inAuthCode VARCHAR(50),
  IN inReference VARCHAR(50))
BEGIN
```

```
  DECLARE currentStatus INT;

  SELECT status
  FROM   orders
  WHERE  order_id = inOrderId
  INTO   currentStatus;

  IF inStatus != currentStatus AND (inStatus = 0 OR inStatus = 1) THEN
    UPDATE orders SET shipped_on = NULL WHERE order_id = inOrderId;
  ELSEIF inStatus != currentStatus AND inStatus = 2 THEN
    UPDATE orders SET shipped_on = NOW() WHERE order_id = inOrderId;
  END IF;

  UPDATE orders
  SET    status = inStatus, comments = inComments,
         auth_code = inAuthCode, reference = inReference
  WHERE  order_id = inOrderId;
END$$
```

7. Выполните следующий оператор SQL, чтобы создать хранимую процедуру `orders_get_orders_by_customer_id`.

```
-- Создаем хранимую процедуру orders_get_by_customer_id
CREATE PROCEDURE orders_get_by_customer_id(IN inCustomerId INT)
BEGIN
  SELECT     o.order_id, o.total_amount, o.created_on,
             o.shipped_on, o.status, c.name
  FROM       orders o
  INNER JOIN customer c
               ON o.customer_id = c.customer_id
  WHERE      o.customer_id = inCustomerId
  ORDER BY   o.created_on DESC;
END$$
```

8. Выполните следующий оператор SQL, чтобы создать хранимую процедуру `orders_get_order_short_details`.

```
-- Создаем хранимую процедуру orders_get_order_short_details
CREATE PROCEDURE orders_get_order_short_details(IN inOrderId INT)
BEGIN
  SELECT     o.order_id, o.total_amount, o.created_on,
             o.shipped_on, o.status, c.name
  FROM       orders o
  INNER JOIN customer c
               ON o.customer_id = c.customer_id
  WHERE      o.order_id = inOrderId;
END$$
```

9. Выполните следующий оператор SQL, чтобы создать хранимую процедуру `customers_get_customers_list`.

```
-- Создаем хранимую процедуру customer_get_customers_list
CREATE PROCEDURE customer_get_customers_list()
BEGIN
  SELECT customer_id, name FROM customer ORDER BY name ASC;
END$$
```

Изменение уровня логики приложения

На уровне логики приложения изменений будет меньше, чем на уровне данных, но все же они будут. Нам понадобится обновить метод `UpdateOrder()` класса `Orders` и добавить в классы `Orders` и `Customers` три новых метода — `GetByCustomerId()`,

GetOrdersShortDetails() и GetCustomersList(). Эти методы будут обеспечивать новую функциональность механизма администрирования и использоваться в шаблоне admin_orders.tpl на уровне представления.

Упражнение: обновление уровня логики приложения

1. Добавьте в класс Orders в файле business/orders.php метод GetByCustomerId().

```
// Возвращает все заказы, сделанные указанным пользователем
public static function GetByCustomerId($customerId)
{
  // Составляем SQL-запрос
  $sql = 'CALL orders_get_by_customer_id(:customer_id)';

  // Создаем массив параметров
  $params = array (':customer_id' => $customerId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetAll($sql, $params);
}
```

2. Добавьте в тот же класс метод GetOrderShortDetails().

```
// Возвращает подробные сведения о заказе
public static function GetOrderShortDetails($orderId)
{
  // Составляем SQL-запрос

  $sql = 'CALL orders_get_order_short_details(:order_id)';

  // Создаем массив параметров
  $params = array (':order_id' => $orderId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetAll($sql, $params);
}
```

3. Замените старый вариант метода Update() класса Orders таким.

```
// Обновляет подробности заказа
public static function UpdateOrder($orderId, $status, $comments,
$authCode, $reference)
{
  // Составляем SQL-запрос
  $sql = 'CALL orders_update_order(:order_id, :status, :comments,
                                   :auth_code, :reference)';

  // Создаем массив параметров
  $params = array (':order_id' => $orderId,
                   ':status' => $status,
                   ':comments' => $comments,
                   ':auth_code' => $authCode,
                   ':reference' => $reference);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}
```

4. Добавьте новый метод — GetCustomersList() — в класс Customer в файле business/customer.php.

```
// Возвращает список всех имен пользователей с их идентификаторами
public static function GetCustomersList()
{
  // Составляем SQL-запрос
  $sql = 'CALL customer_get_customers_list()';

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetAll($sql);
}
```

Изменение уровня представления

Теперь нужно модифицировать уровень представления, чтобы он использовал новые возможности, предоставляемые уровнем логики приложения и уровнем данных. Пока что не будем сильно менять интерфейс администрирования заказов, поскольку все равно придется основательно его переделать, когда мы займемся новой системой обработки заказов.

На рис. 17.1 показана страница, описанная шаблоном `admin_templates`. Она позволяет администраторам фильтровать присутствующие в базе данных заказы по нескольким критериям.

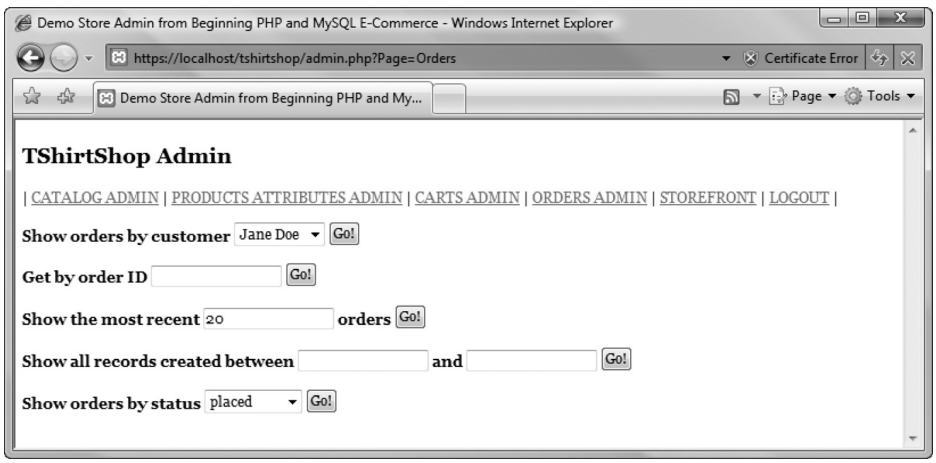


Рис. 17.1. Страница, формируемая шаблоном `admin_templates`

Независимо от критериев фильтрации, которые мы выберем, мы получим в результате список заказов. На рис. 17.2 показан пример заказа, соответствующего выбранным критериям.

Компонентный шаблон `admin_order_details` генерирует страницы наподобие показанной на рис. 17.3. Обратите внимание на указанные стоимость доставки и налоги (мы добавим код, который их вычисляет, позже в этой главе).

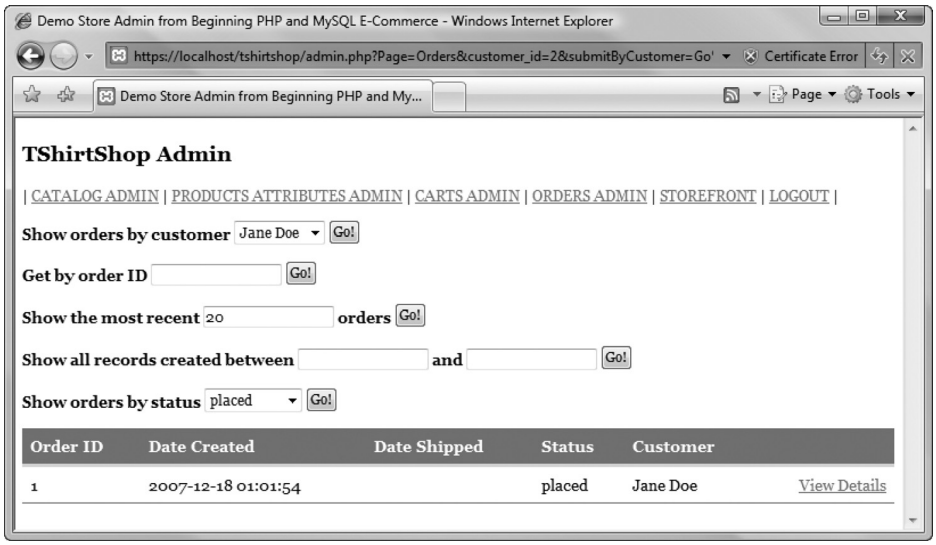


Рис. 17.2. Страница администрирования заказов

Упражнение: обновление уровня представления

1. Добавьте код, выделенный полужирным шрифтом в следующем листинге, в файл `presentation/templates/admin_orders.tpl`.

```
{* admin_orders.tpl *}
{load_presentation_object filename="admin_orders" assign="obj"}
{if $obj->mErrorMessage}<p class="error">{$obj->mErrorMessage}</p>{/if}
<form method="get" action="{$obj->mLinkToAdmin}">
  <input name="Page" type="hidden" value="Orders" />
  <p>
    <font class="bold-text">Show orders by customer</font>
    <select name="customer_id">
    {section name=i loop=$obj->mCustomers}
      <option value="{$obj->mCustomers[i].customer_id}"
       {if $obj->mCustomers[i].customer_id == $obj->mCustomerId}
         selected="selected"
       {/if}>
        {$obj->mCustomers[i].name}
      </option>
    {/section}
    </select>
    <input type="submit" name="submitByCustomer" value="Go!" />
  </p>
  <p>
    <font class="bold-text">Get by order ID</font>
    <input name="orderId" type="text" value="{$obj->mOrderId}" />
    <input type="submit" name="submitByOrderId" value="Go!" />
  </p>
  <p>
    <font class="bold-text">Show the most recent</font>
    <input name="recordCount" type="text" value="{$obj->mRecordCount}" />
...
```

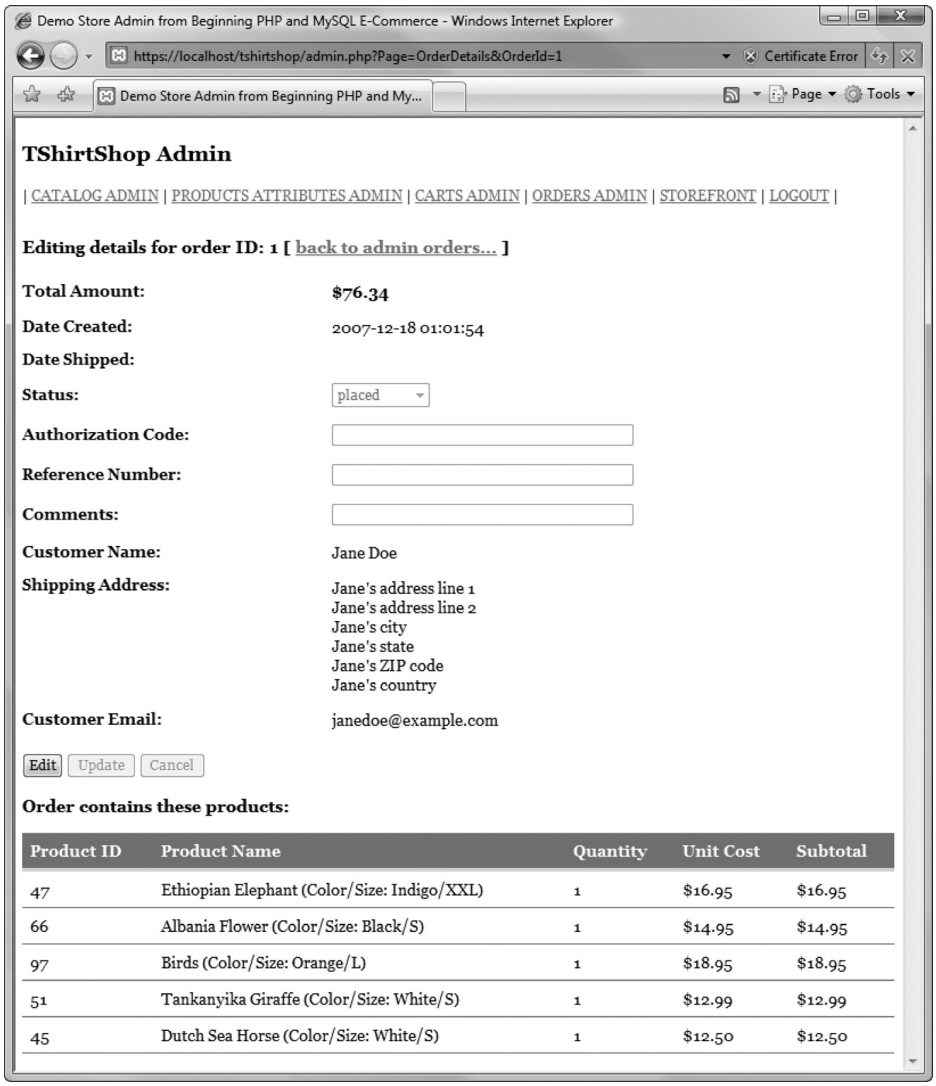


Рис. 17.3. Администрирование элементов заказов

```
{section name=i loop=$obj->mOrders}
  {assign var=status value=$obj->mOrders[i].status}
<tr>
  <td>{$obj->mOrders[i].order_id}</td>
  <td>{$obj->mOrders[i].created_on|date_format:"%Y-%m-%d %T"}</td>
  <td>{$obj->mOrders[i].shipped_on|date_format:"%Y-%m-%d %T"}</td>
  <td>{$obj->mOrderStatusOptions[$status]}</td>
  <td>{$obj->mOrders[i].name}</td>
  <td align="right">
    <a href="{$obj->mOrders[i].link_to_order_details_admin}">
            View Details</a>
  </td>
</tr>
```

```
  {/section}
</table>
{/if}
```

2. Добавьте выделенные полужирным шрифтом элементы в определение класса AdminOrders в файле presentation/admin_orders.php.

```
public $mLinkToAdmin;
public $mCustomers;
public $mCustomerId;
public $mOrderId;
```

3. Добавьте код, выделенный полужирным шрифтом в следующем листинге, в метод init() класса AdminOrders в файле presentation/admin_orders.php.

```
// Если используется фильтр "Показать заказы с выбранным статусом"...
if (isset ($_GET['submitOrdersByStatus']))
{
  $this->mSelectedStatus = $_GET['status'];
  $this->mOrders = Orders::GetOrdersByStatus($this->mSelectedStatus);
}

// Если используется фильтр "Показать заказы выбранного посетителя"...
if (isset ($_GET['submitByCustomer']))
{
  if (empty ($_GET['customer_id']))
    $this->mErrorMessage = 'No customer has been selected';
  else
  {
    $this->mCustomerId = $_GET['customer_id'];
    $this->mOrders = Orders::GetByCustomerId($this->mCustomerId);
  }
}

// Если используется фильтр "Показать заказ по идентификатору"...
if (isset ($_GET['submitByOrderId']))
{
  if (empty ($_GET['orderId']))
    $this->mErrorMessage = 'You must enter an order ID.';
  else
  {
    $this->mOrderId = $_GET['orderId'];
    $this->mOrders = Orders::GetOrderShortDetails($this->mOrderId);
  }
}

$this->mCustomers = Customer::GetCustomersList();

if (is_array($this->mOrders) && count($this->mOrders) == 0)
  $this->mErrorMessage =
    'No orders found matching your searching criteria!';
```

4. Добавьте выделенный полужирным шрифтом элемент в определение класса AdminOrderDetails в файле presentation/admin_order_details.php.

```
public $mLinkToAdmin;
public $mLinkToOrdersAdmin;
public $mCustomerInfo;
```

5. Модифицируйте код обновления заказа в методе init() класса AdminOrderDetails так, как показано ниже.

```
// Инициализируем элементы класса
public function init()
{
  if (isset ($_GET['submitUpdate']))
  {
    Orders::UpdateOrder($this->mOrderId, $_GET['status'],
      $_GET['comments'], $_GET['authCode'], $_GET['reference']);
  }
```

6. В тот же метод добавьте строку, которая будет считывать данные о пользователе, разместившем заказ.

```
$this->mOrderInfo = Orders::GetOrderInfo($this->mOrderId);
$this->mOrderDetails = Orders::GetOrderDetails($this->mOrderId);
$this->mCustomerInfo = Customer::Get($this->mOrderInfo['customer_id']);

// Значение, указывающее, включен или отключен режим редактирования if
(isset ($_GET['submitEdit']))
```

7. Модифицируйте файл presentation/templates/admin_order_details.tpl так, как показано в листинге ниже:

```
  <tr>
    <td class="bold-text">Status: </td>
    <td>
      <select name="status"
       {if ! $obj->mEditEnabled} disabled="disabled" {/if} >
        {html_options options=$obj->mOrderStatusOptions
         selected=$obj->mOrderInfo.status}
    </select>
    </td>
  </tr>
  <tr>
    <td class="bold-text">Authorization Code: </td>
    <td>
      <input name="authCode" type="text" size="50"
       value="{$obj->mOrderInfo.auth_code}"
       {if ! $obj->mEditEnabled} disabled="disabled" {/if} />
    <td>
  </tr>
  <tr>
    <td class="bold-text">Reference Number: </td>
    <td>
      <input name="reference" type="text" size="50"
       value="{$obj->mOrderInfo.reference}"
       {if ! $obj->mEditEnabled} disabled="disabled" {/if} />
    <td>
  </tr>
  <tr>
    <td class="bold-text">Comments: </td>
    <td>
      <input name="comments" type="text" size="50"
       value="{$obj->mOrderInfo.comments}"
       {if ! $obj->mEditEnabled} disabled="disabled" {/if} />
    <td>
  </tr>
  <tr>
    <td class="bold-text">Customer Name: </td>
    <td>{$obj->mCustomerInfo.name}</td>
  </tr>
  <tr>
    <td class="bold-text" valign="top">Shipping Address: </td>
```

```
    <td>
      {$obj->mCustomerInfo.address_1}<br />
      {if $obj->mCustomerInfo.address_2}
        {$obj->mCustomerInfo.address_2}<br />
      {/if}
      {$obj->mCustomerInfo.city}<br />
      {$obj->mCustomerInfo.region}<br />
      {$obj->mCustomerInfo.postal_code}<br />
      {$obj->mCustomerInfo.country}<br />
    </td>
  </tr>
  <tr>
    <td class="bold-text">Customer Email: </td>
    <td>{$obj->mCustomerInfo.email}</td>
  </tr>
</table>
```

8. Добавьте ссылки на файлы с классами симметричного шифрования, кредитных карт и данных пользователей в файл `admin.php`.

```
require_once BUSINESS_DIR . 'shopping_cart.php';
require_once BUSINESS_DIR . 'orders.php';
require_once BUSINESS_DIR . 'symmetric_crypt.php';
require_once BUSINESS_DIR . 'secure_card.php';
require_once BUSINESS_DIR . 'customer.php';
```

9. Загрузите сайт в браузер и убедитесь, что добавленный код работает так, как показано на рис. 17.1–17.3.

Как это работает: изменения уровня представления

Прежде чем воспользоваться изменениями, которые мы внесли в проект, придется выполнить еще несколько упражнений. После этого механизм работы с посетителями будет полностью готов к работе, и нам останется только реализовать конвейер обработки заказов и систему работы с кредитными картами. А сейчас займемся поддержкой налогов и стоимости доставки.

Налоги и стоимость доставки

В большинстве сайтов электронных магазинов существует возможность учитывать при составлении заказов стоимость доставки и налоги, которые нужно заплатить. Разумеется, это не всегда необходимо. Например, магазинам, торгующим электронными книгами, не нужно учитывать стоимость доставки, поскольку вся доставка сводится к передаче файла через Интернет. Однако мы реализуем в нашем проекте возможность учета дополнительных расходов в заказах.

На самом деле реализовать такую возможность очень просто — в базовом варианте. Но если мы захотим сделать соответствующий механизм достаточно эффективным, он может быстро превратиться в весьма объемную часть проекта. В этой главе мы создадим только базовый вариант, который вы позже сможете при желании расширить. Сначала давайте обсудим проблемы.

Налоги

История взаимоотношений электронных магазинов с налоговыми службами весьма бурная. Во времена, когда первые электронные магазины только появились, государства просто не понимали, что с этих магазинов можно брать дань. Соответ-

ственно, многие электронные магазины тогда вообще не учитывали налоги в заказах. Особенно часто такое встречалось при заказах в магазинах, расположенных в других странах, и проблемы начинались, только если купленный товар застревал на таможне, поскольку за него не были заплачены пошлины.

Когда электронные магазины стали более распространенными, государства поняли, сколько денег они теряют, не облагая эти магазины поборами, и быстро перешли от замешательства к ярости. Развив бурную деятельность, каждое государство сначала создало свою схему налогообложения для бизнеса в Интернете. Но сейчас эти схемы стали более-менее похожими, и страсти относительно улеглись.

Основной момент в налогообложении при торговле в Интернете — наличие субъекта налогообложения. Это значит, что, покупая товар в заграничном электронном магазине, в большинстве ситуаций вы можете не волноваться о налогах. Если вы, например, живете в одной из стран Европейского Союза и покупаете что-то в другой, вам, вероятно, придется заплатить налог. Юридические вопросы такого рода довольно запутаны, и мы не станем разбираться в их тонкостях.

В целом величина налога определяется тремя факторами.

- Тем, откуда и куда нужно доставить товар
- Законами государства-отправителя и государства-получателя
- Категорией, к которой принадлежит купленный товар

В одних государствах эти проблемы решать проще, в других — сложнее. Например, в Великобритании можно применить налог на добавочную стоимость (НДС) в размере 20% от стоимости товара и считать, что дело сделано (хотя есть список категорий товаров с уменьшенной ставкой НДС или вообще не облагаемых НДС). Если хотите, подумайте о том, чтобы вынести свой бизнес в оффшорную зону и вообще не платить налогов (Amazon сделал это, так почему бы и вам не последовать его примеру). Во многих странах (прежде всего в США) система налогообложения чрезвычайно запутана, и часто продавец может подсчитать необходимые налоги только при отправке товара покупателю в пределах одного города, поскольку у других городов есть свои законы.

В этой книге мы реализуем только простейшую схему учета налогов. В таблице в базе данных будет храниться информация о величинах налогов, которые нужно применять к товарам, и выбор величины будет определяться тем, куда нужно доставить товар. Мы будем считать, что ко всем товарам применяется одна схема налогообложения. Разумеется, на самом деле это не так, но по крайней мере мы будем вычислять величину налога и указывать ее в счетах. Позже вы сможете реализовать более подробную схему.

Стоимость доставки

Определить стоимость доставки несколько проще, чем величину налога, хотя, безусловно, и этот механизм можно сделать весьма изощренным. Поскольку рассылка купленных товаров для электронного магазина принципиально ничем не отличается от аналогичной процедуры в обычном магазине, предлагающем доставку товаров почтой, все нужные механизмы давно существуют и хорошо отработаны. Возможно, какие-то тонкости будут отличаться, но мы не будем рассматривать организацию службы доставки заказов.

Возможно, у заказчика проекта уже будет налажено взаимодействие с компанией, обеспечивающей доставку заказов. В этом случае лучше всего — не ломать существующий механизм, а адаптировать к нему проект сайта. Но если вы создаете

механизмы "с нуля", учтите, что существует несколько вариантов, из которых можно сделать выбор.

Самый простой из них — вообще забыть о стоимости доставки. Это оптимальный вариант, если стоимости доставки просто нет (например, в магазине, торгующем электронными книгами или музыкой для загрузки через Интернет). Можно также включить стоимость доставки в стоимость товаров или ввести фиксированную стоимость доставки независимо от пункта назначения. Однако такой подход, скорее всего, приведет к тому, что одни покупатели будут переплачивать, а другие — недоплачивать, и это не прибавит популярности вашей компании.

Другая крайность — точно учитывать размеры и вес всех товаров и вычислять стоимость доставки. Этот подход несколько проще, чем вам может показаться. Многие крупные компании, занимающиеся доставкой грузов (например, FedEx), предоставляют API для решения таких задач. В некоторых случаях можно предложить разные варианты доставки с разными ценами (например, на завтра, в течение трех дней или в течение недели). Однако здесь многое зависит от возможностей вашей службы доставки, и вдаваться в подробности мы не будем.

В этой книге мы ограничимся простейшим механизмом учета стоимости доставки. Каждому региону, в который мы можем отправлять заказанные товары, мы назначим свою стоимость, предложив несколько вариантов доставки. Стоимость варианта, который выберет посетитель, будет добавлена к стоимости заказа. Именно для этого мы в главе 16 добавили в базу данных таблицу `shipping_region`.

Реализация учета налогов и стоимости доставки

Как вы, вероятно, уже поняли, для учета налогов и стоимости доставки в проект придется внести некоторые изменения. Нужно будет добавить в базу данных две новые таблицы, `tax` и `shipping`, и изменить таблицу `orders`. Кроме того, понадобится создать несколько новых хранимых процедур и изменить ряд уже существующих. Соответственно, нужно будет модифицировать код уровня логики приложения, чтобы он мог работать с изменившимися процедурами, а на уровне представления — предоставить посетителям возможность выбирать вариант доставки.

Вообще говоря, нужно очистить старые таблицы `orders` и `order_details`, поскольку в них хранятся заказы с неуказанными налогами и стоимостью доставки.

Итак, начнем.

Изменение уровня данных

В следующем упражнении мы добавим в базу данных две новые таблицы, модифицируем таблицу `orders`, добавим новые хранимые процедуры и модифицируем несколько уже существовавших.

Упражнение: модификации в базе данных

1. Загрузите phpMyAdmin, откройте базу данных `tshirtshop` и перейдите на страницу нового запроса SQL.
2. Выполните следующий оператор SQL, чтобы добавить в базу данных таблицу `shipping`.

```
-- Создаем таблицу shipping
CREATE TABLE `shipping` (
  `shipping_id`        INT NOT        NULL AUTO_INCREMENT,
  `shipping_type`      VARCHAR(100)   NOT NULL,
  `shipping_cost`      NUMERIC(10, 2) NOT NULL,
```

```
  `shipping_region_id` INT            NOT NULL,
  PRIMARY KEY (`shipping_id`),
  KEY `idx_shipping_shipping_region_id` (`shipping_region_id`)
);
```

3. Выполните следующий оператор SQL, чтобы заполнить таблицу `shipping` данными.

```
-- Заполняем таблицу shipping данными
INSERT INTO `shipping` (`shipping_id`, `shipping_type`,
                        `shipping_cost`, `shipping_region_id`) VALUES
       (1, 'Next Day Delivery ($20)', 20.00, 2),
       (2, '3-4 Days ($10)',          10.00, 2),
       (3, '7 Days ($5)',              5.00, 2),
       (4, 'By air (7 days, $25)',    25.00, 3),
       (5, 'By sea (28 days, $10)',   10.00, 3),
       (6, 'By air (10 days, $35)',   35.00, 4),
       (7, 'By sea (28 days, $30)',   30.00, 4);
```

4. Выполните следующий оператор SQL, чтобы добавить в базу данных таблицу `tax`.

```
-- Создаем таблицу tax
CREATE TABLE `tax` (
  `tax_id`         INT            NOT NULL  AUTO_INCREMENT,
  `tax_type`       VARCHAR(100)   NOT NULL,
  `tax_percentage` NUMERIC(10, 2) NOT NULL,
  PRIMARY KEY (`tax_id`)
);
```

5. Выполните следующий оператор SQL, чтобы заполнить таблицу `tax` данными.

```
-- Заполняем таблицу tax данными
INSERT INTO `tax` (`tax_id`, `tax_type`, `tax_percentage`) VALUES
       (1, 'Sales Tax at 8.5%', 8.50),
       (2, 'No Tax',            0.00);
```

6. Выполните следующие операторы SQL, чтобы добавить в таблицу `orders` новый столбец `shipping_id` и создать в ней индекс.

```
-- Добавляем в таблицу orders новый столбец shipping_id
ALTER TABLE `orders` ADD COLUMN `shipping_id` INT;

-- Добавляем в таблицу orders новый индекс
CREATE INDEX `idx_orders_shipping_id` ON `orders` (`shipping_id`);
```

7. Выполните следующие операторы SQL, чтобы добавить в таблицу `orders` новый столбец `tax_id` и создать в ней индекс.

```
-- Добавляем в таблицу orders новый столбец tax_id
ALTER TABLE orders ADD COLUMN tax_id INT;

-- Добавляем в таблицу orders новый индекс
CREATE INDEX `idx_orders_tax_id` ON `orders` (`tax_id`);
```

8. Выполните следующие операторы SQL, чтобы удалить старую версию хранимой процедуры `shopping_cart_create_order` и создать новую (прежде чем выполнять эти операторы, не забудьте задать символы `$$` в качестве разделителя).

```
-- Удаляем хранимую процедуру shopping_cart_create_order
DROP PROCEDURE shopping_cart_create_order$$

-- Создаем хранимую процедуру shopping_cart_create_order
CREATE PROCEDURE shopping_cart_create_order(IN inCartId CHAR(32),
  IN inCustomerId INT, IN inShippingId INT, IN inTaxId INT)
BEGIN
  DECLARE orderId INT;
```

```
  -- Вставляем в таблицу orders новую запись и получаем идентификатор заказа
  INSERT INTO orders (created_on, customer_id, shipping_id, tax_id) VALUES
         (NOW(), inCustomerId, inShippingId, inTaxId);
  -- Получаем новый идентификатор заказа
  SELECT LAST_INSERT_ID() INTO orderId;

  -- Вставляем сведения о заказе в таблицу order_detail
  INSERT INTO order_detail (order_id, product_id, attributes,
                            product_name, quantity, unit_cost)
  SELECT      orderId, p.product_id, sc.attributes, p.name, sc.quantity,
              COALESCE(NULLIF(p.discounted_price, 0), p.price) AS unit_cost
  FROM        shopping_cart sc
  INNER JOIN  product p
                ON sc.product_id = p.product_id
  WHERE       sc.cart_id = inCartId AND sc.buy_now;

  -- Сохраняем общую стоимость заказа
  UPDATE orders
  SET total_amount = (SELECT SUM(unit_cost * quantity)
                      FROM   order_detail
                      WHERE  order_id = orderId)
  WHERE order_id = orderId;

  -- Очищаем корзину
  CALL shopping_cart_empty(inCartId);

  -- Возвращаем идентификатор заказа
  SELECT orderId;
END$$
```

9. Выполните следующие операторы SQL, чтобы удалить старую версию хранимой процедуры `orders_get_order_info` и создать новую (прежде чем выполнять эти операторы, не забудьте задать символы `$$` в качестве разделителя).

```
-- Удаляем хранимую процедуру orders_get_order_info
DROP PROCEDURE orders_get_order_info$$

-- Создаем хранимую процедуру orders_get_order_info
CREATE PROCEDURE orders_get_order_info(IN inOrderId INT)
BEGIN
  SELECT      o.order_id, o.total_amount, o.created_on, o.shipped_on,
              o.status, o.comments, o.customer_id, o.auth_code,
              o.reference, o.shipping_id, s.shipping_type, s.shipping_cost,
              o.tax_id, t.tax_type, t.tax_percentage
  FROM        orders o
  INNER JOIN tax t
                ON t.tax_id = o.tax_id
  INNER JOIN shipping s
                ON s.shipping_id = o.shipping_id
  WHERE       o.order_id = inOrderId;
END$$
```

10. Выполните следующий оператор SQL, чтобы создать хранимую процедуру `orders_get_shipping_info` (прежде чем выполнять этот оператор, не забудьте задать символы `$$` в качестве разделителя).

```
-- Создаем хранимую процедуру orders_get_shipping_info
CREATE PROCEDURE orders_get_shipping_info(IN inShippingRegionId INT)
BEGIN
SELECT shipping_id, shipping_type, shipping_cost, shipping_region_id
FROM shipping
```

```
WHERE shipping_region_id = inShippingRegionId;
END$$
```

Изменение уровня логики приложения

Чтобы уровень логики приложения мог работать с изменившимся уровнем данных, нужно внести несколько изменений в файлы `business/shopping_cart.php` и `business/orders.php`.

Упражнение: модификации уровня логики приложения

1. Замените старый метод `CreateOrder()` в классе `ShoppingCart` (файл `business/shopping_cart.php`) таким.

```
// Создаем новый заказ
public static function CreateOrder($customerId, $shippingId, $taxId)
{
  // Составляем SQL-запрос
  $sql = 'CALL shopping_cart_create_order(:cart_id, :customer_id,
                :shipping_id, :tax_id)';

  // Создаем массив параметров
  $params = array (':cart_id' => self::GetCartId(),
                   ':customer_id' => $customerId,
                   ':shipping_id' => $shippingId,
                   ':tax_id' => $taxId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetOne($sql, $params);
}
```

2. Добавьте метод `GetShippingInfo()` в класс `Orders` в файле `business/orders.php`.

```
// Возвращает сведения о доставке для выбранного региона
public static function GetShippingInfo($shippingRegionId)
{
  // Составляем SQL-запрос
  $sql = 'CALL orders_get_shipping_info(:shipping_region_id)';

  // Создаем массив параметров
  $params = array (':shipping_region_id' => $shippingRegionId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetAll($sql, $params);
}
```

Изменение уровня представления

И наконец необходимо модифицировать уровень представления. Собственно говоря, все, что нужно здесь изменить, — это фрагменты страниц оплаты и администрирования заказов.

Упражнение: модификация уровня представления

1. Модифицируйте код в файле `presentation/templates/checkout.tpl` так, как показано ниже.

```
  Shipping region: {$obj->mShippingRegion}
</p>
{/if}
{if $obj->mNoCreditCard!= 'yes' && $obj->mNoShippingAddress != 'yes'}
<p>
  Shipping type:
  <select name="shipping">
  {section name=i loop=$obj->mShippingInfo}
    <option value="{$obj->mShippingInfo[i].shipping_id}">
      {$obj->mShippingInfo[i].shipping_type}
    </option>
  {/section}
  </select>
</p>
{/if}
<input type="submit" name="place_order" value="Place Order"
  {$obj->mOrderButtonVisible} /> |
<a href="{$obj->mLinkToCart}">Edit Shopping Cart</a> |
```

2. Добавьте новый элемент в определение класса `CheckoutInfo` в файле `presentation/checkout_info.php`.

```
public $mLinkToCart;
public $mLinkToContinueShopping;
public $mShippingInfo;
```

3. Модифицируйте метод `init()` класса `CheckoutInfo` в файле `presentation/checkout_info.php`:

```
if(isset ($_POST['place_order']))
{
  $this->mCustomerData = Customer::Get();
  $tax_id = '';

  switch ($this->mCustomerData['shipping_region_id'])
  {
    case 2:
      $tax_id = 1;

    break;
  default:
    $tax_id = 2;
}

// Получаем идентификатор заказа
$order_id = ShoppingCart::CreateOrder(
              $this->mCustomerData['customer_id'],
              (int)$_POST['shipping'], $tax_id);

// Составляем запрос для PayPal
$redirect =
  PAYPAL_URL . '&item_name=TShirtShop Order ' .
  urlencode('#') . $order_id .
  '&item_number=' . $order_id .
  '&amount=' . $this->mTotalAmount .
  '&currency_code=' . PAYPAL_CURRENCY_CODE .
  '&return=' . PAYPAL_RETURN_URL .
  '&cancel_return=' . PAYPAL_CANCEL_RETURN_URL;
```

4. В конец того же метода добавьте код, выделенный полужирным шрифтом в следующем листинге.

```
      foreach ($shipping_regions as $item)
        if ($item['shipping_region_id'] ==
            $this->mCustomerData['shipping_region_id'])
          $this->mShippingRegion = $item['shipping_region'];

      if ($this->mNoCreditCard == 'no' && $this->mNoShippingAddress == 'no')
      {
        $this->mShippingInfo = Orders::GetShippingInfo(
                               $this->mCustomerData['shipping_region_id']);
      }
    }
  }
}
?>
```

5. Добавьте в файл `index.php` ссылку на файл класса `orders`, как показано ниже.

```
require_once BUSINESS_DIR . 'secure_card.php';
require_once BUSINESS_DIR . 'customer.php';
require_once BUSINESS_DIR . 'orders.php';
```

6. Добавьте два новых элемента в определение класса `AdminOrderDetails` в файле `presentation/admin_order_details.php`.

```
public $mLinkToOrdersAdmin;
public $mCustomerInfo;
public $mTotalCost;
public $mTax = 0.0;
```

7. Добавьте в код метода `init()` класса `AdminOrderDetails` строки, выделенные полужирным шрифтом в следующем листинге.

```
$this->mOrderInfo = Orders::GetOrderInfo($this->mOrderId);
$this->mOrderDetails = Orders::GetOrderDetails($this->mOrderId);
$this->mCustomerInfo = Customer::Get($this->mOrderInfo['customer_id']);
$this->mTotalCost = $this->mOrderInfo['total_amount'];

if ($this->mOrderInfo['tax_percentage'] !== 0.0)
  $this->mTax = round((float)$this->mTotalCost *
                      (float)$this->mOrderInfo['tax_percentage'], 2)
                     / 100.00;
$this->mTotalCost += $this->mOrderInfo['shipping_cost'];
$this->mTotalCost += $this->mTax;

// Форматируем значения
$this->mTotalCost = number_format($this->mTotalCost, 2, '.', '');
$this->mTax = number_format($this->mTax, 2, '.', '');
```

8. Модифицируйте шаблон `presentation/templates/admin_order_details.tpl` так, как показано ниже:

```
<input type="hidden" name="Page" value="OrderDetails" />
<input type="hidden" name="OrderId"
 value="{$obj->mOrderInfo.order_id}" />
<table class="borderless-table">
<tr>
  <td class="bold-text">Total Amount: </td>
  <td class="price">
    ${$obj->mOrderInfo.total_amount}
  </td>
</tr>
<tr>
  <td class="bold-text">Tax: </td>
```

```
  <td class="price">{$obj->mOrderInfo.tax_type} ${$obj->mTax}</td>
</tr>
<tr>
  <td class="bold-text">Shipping: </td>
  <td class="price">{$obj->mOrderInfo.shipping_type}</td>
</tr>
<tr>
  <td class="bold-text">Date Created: </td>
  <td>
    {$obj->mOrderInfo.created_on|date_format:"%Y-%m-%d %T"}
  </td>
</tr>
```

Как это работает: учет стоимости доставки и налогов

Это упражнение — одна из самых важных частей данной главы. Именно в нем мы добавляем в проект код, реализующий подсчет налогов и транспортных расходов. Если в дальнейшем вы захотите реализовать более сложную систему подсчета, вам, скорее всего, понадобится модифицировать именно этот код. Уровень данных и уровень логики приложения в данном случае выполняют более общие задачи, вряд ли вам понадобится что-то в них менять (хотя и этот вариант исключить нельзя).

Прежде чем тестировать систему подсчета налогов и стоимости доставки, проверьте, остались ли в базе данных старые заказы (воспользуйтесь для этого страницей администрирования заказов). Они должны были сохраниться в целости, поскольку мы не изменяли данные в базе данных.

Попробуйте создать новый заказ, предпочтительно для региона United States/Canada (так как в данный момент наша программа вычисляет налоги только для этого региона). Заметьте, что на странице оплаты следует выбрать способ доставки.

Разместив заказ, просмотрите его в базе данных. Он должен быть аналогичным показанному на рис. 17.3.

В этой главе мы рассмотрели почти все, что необходимо для учета налогов и транспортных расходов. Прежде всего посетитель должен указать свой регион и адрес доставки. Не указав эту информацию, посетители не могут размещать заказы, поскольку не могут выбрать способ доставки. Когда пользователь размещает заказ, выбранный регион сохраняется в записи о заказе в таблице `orders`. Кроме того, сохраняется вычисленная сумма налогов, но на данный момент она подсчитывается по очень простому алгоритму, и ее вычисление не требует никаких действий от пользователя.

Дальнейшее развитие проекта

Далее можно двигаться в нескольких направлениях. Вероятно, прежде всего стоит добавить на сайт систему администрирования для вычисления налогов и транспортных расходов. Мы не реализовали ее до сих пор, поскольку это довольно тривиальная задача по сравнению с тем, чего мы уже достигли в этой книге. К тому же для ее реализации понадобится слишком много данных, которые сложно получить: сведения о ставках налогов в разных государствах и частях государств, о стоимости перевозки товаров и пр.

Также можно реализовать подключение к оперативным службам вычисления транспортных расходов и налогов (это весьма привлекательный вариант). Например, компания FedEx предлагает API для сетевого доступа к своим услугам, похожий на API для работы с кредитными картами, который мы рассмотрим немного позже. Для того чтобы воспользоваться этим API, в базу данных понадобится добавить сведения о размере и весе каждого товара, но никаких принципиальных сложностей здесь нет.

Резюме

В этой главе мы предоставили посетителям возможность размещать заказы с помощью механизма, созданного в главе 16. Большую часть изменений, сделанных в проекте в этой главе, мы используем в следующей главе, когда будем работать над конвейером обработки заказов. Кроме того, мы добавили на сайт возможность быстрого просмотра заказов, хотя полноценный механизм их администрирования будет создан позже.

Мы также реализовали простую систему вычисления стоимости доставки и налогов. Она далеко не идеальна, но проста и вполне работоспособна. А что самое важное — мы можем легко модифицировать эту систему, сделав ее более сложной и универсальной.

Начиная со следующей главы, мы займемся системой обработки заказов.

Глава 18

Реализация конвейера обработки заказов: часть I

Реализация конвейера обработки заказов — это наш первый шаг в создании профессиональной системы менеджмента заказов. В этой и следующей главах мы создадим полный конвейер для обработки заказов, в который будут входить авторизация кредитных карт, проверка запасов товаров, отправка заказов, уведомление покупателей об этом по электронной почте и т.д. Тонкости работы с кредитными картами мы рассмотрим в главе 20, но из этой главы вы узнаете, что именно понадобится реализовать.

Конвейер обработки заказов — чрезвычайно полезный механизм для сайта электронного магазина. Он позволит отслеживать состояние каждого заказа и получать информацию, которая пригодится, если с каким-либо заказом что-то будет не так. Весь конвейер мы создадим, не используя товаров сторонних разработчиков, тем самым заметно уменьшив свои затраты.

Большую часть этой главы мы будем обсуждать конвейер и создавать его компоненты, но также придется внести некоторые изменения в базу данных, которую использует наш сайт. Однако код из этой главы не намного сложнее того, который вам пришлось писать в предыдущих главах. Основная сложность будет заключаться именно в составлении проекта конвейера. По окончании проектирования будут созданы следующие компоненты.

- Механизм обновления статуса заказа
- Механизм отправки сведений для авторизации кредитных карт
- Механизм задания даты отправки заказа
- Механизм отправки писем покупателям и поставщикам товаров
- Механизм просмотра подробной информации о заказе

К концу этой главы покупатели смогут помещать свои заказы в начало конвейера, а мы сможем отслеживать, как эти заказы проходят через стадии конвейера. Хотя рабочей системы работы с кредитными картами еще не будет, в остальном конвейер будет вполне работоспособен, включая страницу администрирования, которая позволит поставщикам подтверждать наличие товаров и отправку заказов. Но для начала давайте разберемся, что именно нужно создать.

Что такое конвейер обработки заказов

Любая коммерческая транзакция — покупка в обычном магазине, на рынке или в Интернете — состоит из нескольких операций, которые должны быть выполнены, прежде чем транзакцию можно будет считать завершенной. Например, нельзя просто забрать товар из магазина, не заплатив, и сказать, что он куплен, — оплата является необходимым условием покупки. Чтобы транзакция считалась завершенной, должны быть успешно выполнены все операции, из которых она состоит. Например, если кредитная карта покупателя заблокирована, осуществить оплату не удастся, и транзакция не будет считаться завершенной.

Последовательность операций, из которых состоит транзакция, часто представляется в виде конвейера. Заказы поступают на первую стадию конвейера и продвигаются по стадиям соответственно выполнению необходимых операций. Каждая стадия конвейера отвечает за одну операцию, и, если какая-то операция завершается неудачей, заказ "застревает" на конвейере, и потребуется вмешательство людей, чтобы устранить проблему или вообще отменить заказ.

На рис. 18.1 показан конвейер обработки заказов, подходящий для обычного магазина.

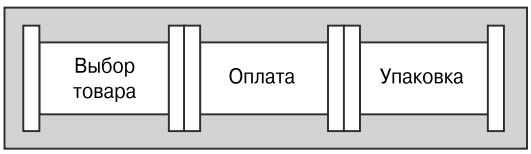


Рис. 18.1. Простой конвейер для обычного магазина

Последний этап — упаковка — может быть необязательным, но может быть и очень важным, например, если мы продаем сувениры, которые нужно красиво упаковывать. На этапе оплаты может использоваться несколько разных механизмов, поскольку покупатель может платить наличными, кредитной картой, чеками и т.д.

В электронных магазинах конвейер будет несколько длиннее, но ничего принципиально нового в нем не появится.

Проектирование конвейера обработки заказов

В приложении TShirtShop структура конвейера обработки заказов будет такой, как показано на рис. 18.2.

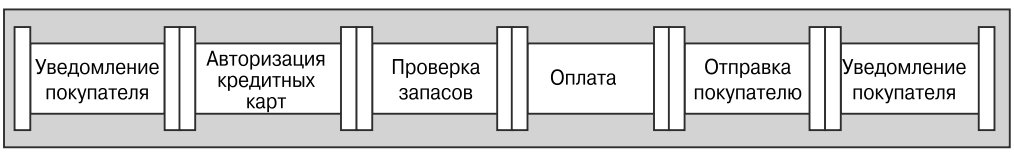


Рис. 18.2. Конвейер для TShirtShop

Вот задачи, решаемые на каждой ступени конвейера.

- *Уведомление покупателя.* Сайт будет отправлять покупателям по электронной почте сообщения о том, что заказы приняты к исполнению и заказанные товары отправлены по указанному адресу.

- *Авторизация кредитных карт.* Проверяется доступность кредитных карт, используемых для оплаты, и на соответствующих счетах резервируются необходимые суммы (хотя сама оплата на данном этапе конвейера еще не производится).
- *Проверка запасов.* Поставщику отправляется по электронной почте сообщение со списком заказанных товаров. Обработка заказа продолжится, когда поставщик подтвердит доступность этих товаров.
- *Оплата.* Транзакции с кредитными картами завершаются списанием зарезервированных ранее сумм.
- *Отправка покупателю.* Поставщику отправляется по электронной почте сообщение о том, что заказанные товары оплачены покупателем. Обработка заказа продолжается после того, как поставщик подтвердит отправку товаров.
- *Уведомление покупателя.* Покупателю по электронной почте отправляется письмо с уведомлением об отправке товаров по указанному адресу.

ЗАМЕЧАНИЕ

С точки зрения реализации, как вы вскоре убедитесь, каждый этап делится на более мелкие стадии, например проверка запасов и отправка состоят из отправки сообщений и ожидания ответов.

По мере продвижения заказа по конвейеру прохождение этапов фиксируется в таблице базы данных, которая называется `audit`. Записи из этой таблицы можно просмотреть, чтобы выяснить, что происходило с заказом, — это очень удобный способ выяснения, в чем заключаются проблемы, если таковые возникают. Кроме того, у каждой записи в таблице `orders` будет определенный статус, указывающий, на каком этапе конвейера в данный момент пребывает соответствующий заказ.

С точки зрения кода каждому этапу конвейера соответствует свой класс. В этих классах будут выполняться все необходимые операции, включая изменение статуса записей в таблице `orders`. Кроме того, нам понадобится координирующий класс, который будет определять, к какому классу нужно обращаться для обработки каждого заказа. К координирующему классу мы будем обращаться при помещении заказа на конвейер, при подтверждении наличия нужных товаров в запасах и при подтверждении отправки товаров.

Чтобы упростить задачу, мы также определим стандартный интерфейс, который будет поддерживаться всеми классами конвейера. Это позволит координирующему классу обращаться к каждой ступени по стандартному протоколу. Кроме того, мы определим в координирующем классе несколько вспомогательных свойств и методов, которыми будут пользоваться классы этапов конвейера. Например, идентификатор заказа должен быть доступен всем этапам конвейера, поэтому мы будем хранить его в координирующем классе, чтобы избежать дублирования кода.

А теперь перейдем к деталям. Мы создадим несколько файлов в папке `business` и разместим в этих файлах новые классы. Затем мы добавим ссылки на эти файлы в проект TShirtShop. Вот классы, которые мы создадим.

- `OrderProcessing` (координирующий класс для обработки заказов)
- `IPipelineSection` (определение интерфейса для этапов конвейера)
- `PsInitialNotification`, `PsCheckFunds`, `PsCheckStock`, `PsStockOk`, `PsTakePayment`, `PsShipGoods`, `PsShipOk`, `PsFinalNotification` (классы, непосредственно образующие конвейер; мы создадим их в главе 19, а пока что вместо них будет использоваться класс `PsDummy`)

Процесс продвижения заказа по конвейеру под управлением координирующего класса изображен на рис. 18.3.

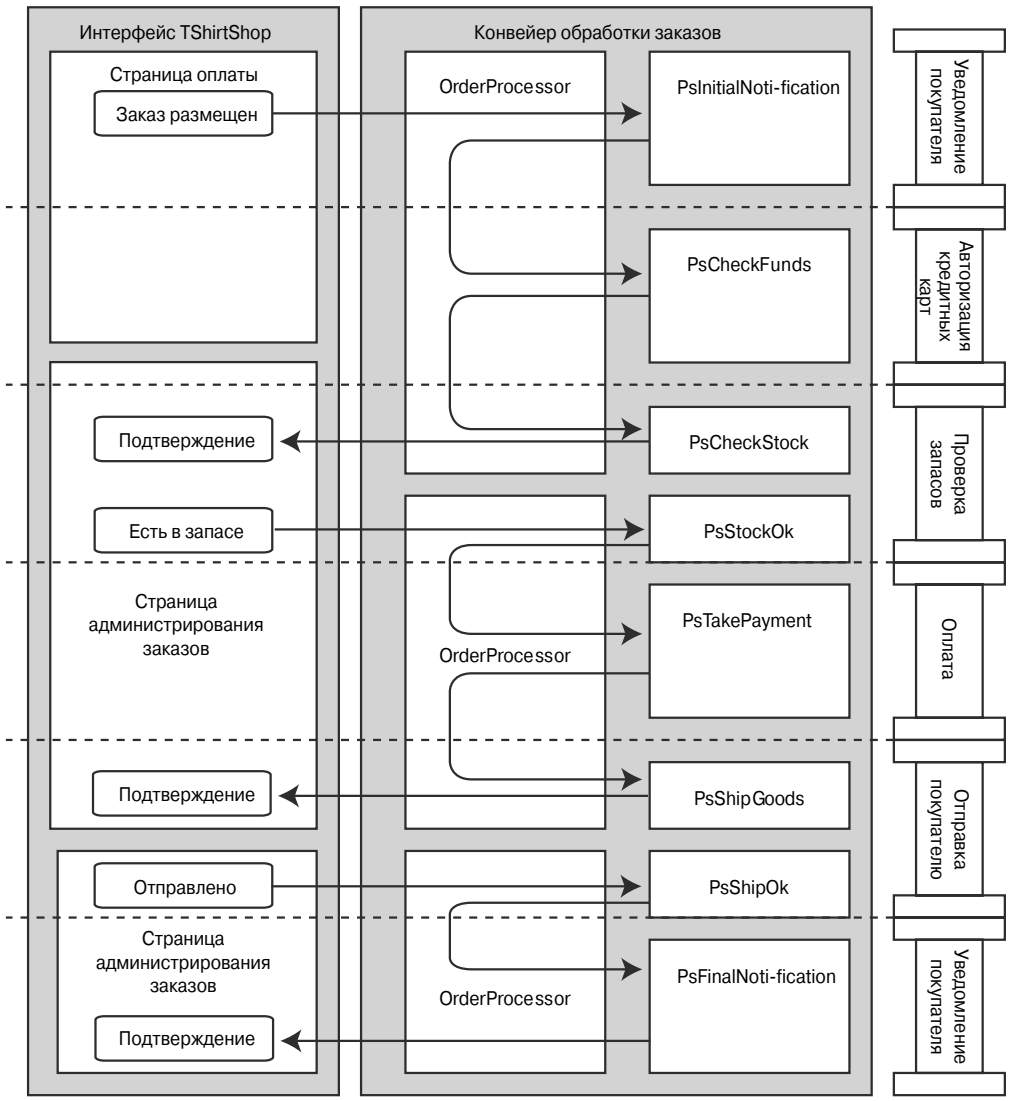


Рис. 18.3. Продвижение заказа по конвейеру TShirtShop

Как видно из рис. 18.3, в обработке заказа есть три ключевых момента.

- Размещение заказа покупателем
- Подтверждение поставщиком наличия товаров в запасе
- Подтверждение поставщиком отправки товаров

Между первым и вторым ключевыми моментами заказ проходит первую часть конвейера.

1. Когда покупатель размещает заказ, код из файла `presentation/checkout.php` создает запись о заказе в базе данных и вызывает класс `OrderProcessing`, чтобы начать обработку заказа.
2. Класс `OrderProcessing` определяет, что заказ только что создан, и обращается к классу `PsInitialNotification`.
3. Класс `PsInitialNotification` отправляет покупателю по электронной почте сообщение с подтверждением приема заказа и передает заказ следующему этапу конвейера, а затем сообщает классу `OrderProcessing`, что можно продолжать обработку заказа.
4. Класс `OrderProcessing` проверяет статус заказа и обращается к классу `PsCheckFunds`.
5. Класс `PsCheckFunds` проверяет, есть ли на кредитной карте покупателя достаточная сумма для оплаты заказа, и сохраняет сведения, необходимые для завершения транзакции, если такая сумма есть (хотя пока что никакие деньги с карты не снимаются). Если все проходит успешно, класс `OrderProcessing` получает уведомление о том, что можно продолжать обработку заказа.
6. Класс `OrderProcessing` проверяет статус заказа и обращается к классу `PsCheckStock`.
7. Класс `PsCheckStock` отправляет покупателю по электронной почте сообщение со списком заказанных товаров и просьбой подтвердить их наличие через систему администрирования заказов. Затем этот класс обновляет статус заказа.
8. Класс `OrderProcessing` завершает первую часть обработки заказа.

Вторая часть конвейера работает так.

1. Когда поставщик подтверждает наличие требуемых товаров на странице администрирования заказов, код из файла `presentation/admin_order_details.php` вызывает класс `OrderProcessor`, чтобы продолжить обработку заказа.
2. Получив от поставщика подтверждение наличия товаров, класс `OrderProcessor` вызывает класс `PsStockOk`.
3. Класс `PsStockOk` меняет статус заказа и разрешает классу `OrderProcessor` продолжать обработку заказа.
4. Класс `OrderProcessor` обнаруживает изменение статуса заказа и обращается к классу `PsTakePayment`.
5. Класс `PsTakePayment` использует сведения о транзакции, ранее сохраненные классом `PsCheckFunds`, чтобы завершить эту транзакцию, сняв со счета деньги для оплаты заказа. Затем класс `PsTakePayment` меняет статус заказа и обращается к классу `OrderProcessor` для продолжения обработки заказа.
6. Класс `OrderProcessor` обнаруживает изменение статуса заказа и обращается к классу `PsShipGoods`.
7. Класс `PsCheckStock` отправляет поставщику по электронной почте сообщение со списком заказанных товаров и просьбой доставить их по указанному адресу. Затем этот класс обновляет статус заказа.
8. Класс `OrderProcessing` завершает вторую часть обработки заказа.

Третья часть конвейера работает так.

1. Когда поставщик подтверждает отправку товаров, код в файле `presentation/admin_order_details.php` вызывает класс `OrderProcessor` для продолжения обработки заказа.

2. Класс `OrderProcessor` обнаруживает изменение статуса заказа и обращается к классу `PsShipOk`.
3. Класс `PsShipOk` сохраняет в базе данных дату отправки товаров покупателю, меняет статус заказа и вызывает класс `OrderProcessor` для продолжения обработки заказа.
4. Класс `OrderProcessor` обнаруживает изменение статуса заказа и обращается к классу `PsFinalNotification`.
5. Класс `PsFinalNotification` отправляет по электронной почте сообщение об отправке товаров покупателю и меняет статус заказа.
6. Класс `OrderProcessor` завершает обработку заказа.

Если на каком-то этапе обработки заказа возникают проблемы, например на кредитной карте нет достаточной суммы для оплаты заказа, администратору сайта по электронной почте отправляется сообщение. У администратора будет вся необходимая информация, чтобы выяснить причину сбоя, связаться с покупателем и при необходимости отменить или модифицировать заказ.

В целом в коде обработки заказов нет ничего особенно сложного — просто этот код довольно объемен.

Закладываем основы

Прежде чем приступить непосредственно к созданию компонентов конвейера, нужно внести несколько изменений в базу данных `tshirtshop` и само веб-приложение.

При обработке заказа одна из основных функций конвейера — сохранять информацию, которая может понадобиться при аудите. Она будет храниться в специальной таблице `audit` в базе данных. Мы создадим эту таблицу в следующем упражнении.

Чтобы реализовать описанную выше функциональность, также понадобится добавить в базу данных новую хранимую процедуру `orders_create_audit`, которая будет добавлять записи в таблицу `audit`.

Кроме того, мы создадим класс `OrderProcessor`, отвечающий за продвижение заказа по конвейеру. Код этого класса довольно объемен, но мы сможем сначала создать простую версию, а затем постепенно добавлять нужную функциональность. Для начала создадим версию класса `OrderProcessor` с такой функциональностью.

- Возможность динамического выбора части конвейера с помощью интерфейса `IPipelineSection`
- Добавление основных данных для аудита
- Предоставление доступа к сведениям о текущем заказе
- Предоставление покупателю доступа к его заказу
- Предоставление доступа к механизму уведомления администраторов
- Отправка администраторам сообщений об ошибках при обработке заказов

ИНТЕРФЕЙСЫ В PHP

В этой главе вы впервые в данной книге встретитесь с интерфейсами PHP. Интерфейсы — это распространенный элемент современных объектно-ориентированных языков программирования. Интерфейс представляет собой набор методов, которые должны быть реализованы в классе, реализующем данный интерфейс.

Когда класс реализует какой-нибудь интерфейс, в этом классе должны быть реализованы все методы этого интерфейса. Таким образом, наличие у класса определенного интерфейса гарантирует, что у этого класса есть определенный набор методов. Например, в следующем упражнении мы создадим интерфейс `IPipelineSection` с единственным методом `Process()`.

```
   interface IPipelineSection
{
  public function Process($processor);
}
```

Этот интерфейс мы реализуем во всех классах, реализующих этапы конвейера (эти классы мы создадим в следующей главе). Чтобы реализовать интерфейс в каждом из классов, достаточно будет реализовать в них метод `Process()`. И, зная, что эти классы реализуют интерфейс `IPipelineSection`, мы сможем спокойно вызывать метод `Process()`, зная, что такой метод в них есть.

Объекты класса интерфейса создавать нельзя, поскольку у его методов нет собственных реализаций, — есть только объявления (реализации этих методов содержатся в классах, в которых реализуется интерфейс).

Классы реализуют интерфейс с помощью оператора `implements`. В классе можно реализовывать несколько интерфейсов, но в этих интерфейсах не должно быть одинаковых методов во избежание конфликтов.

Более подробно об интерфейсах в PHP вы можете узнать по адресу `http://php.net/interfaces`.

Мы создадим сначала один огромный этап конвейера — `PsDummy`, — чтобы протестировать функциональность. Этот этап мы будем использовать в данной главе вместо настоящих классов этапов конвейера, которые создадим в следующей главе.

Упражнение: создание каркаса конвейера обработки заказов

1. С помощью phpMyAdmin подключитесь к базе данных tshirtshop и откройте страницу нового SQL-запроса.
2. Выполните следующий оператор SQL, чтобы создать таблицу `audit`.

```
-- Создает таблицу audit
CREATE TABLE `audit` (
  `audit_id`   INT      NOT NULL AUTO_INCREMENT,
  `order_id`   INT      NOT NULL,
  `created_on` DATETIME NOT NULL,
  `message`    TEXT     NOT NULL,
  `code`       INT      NOT NULL,
  PRIMARY KEY (`audit_id`),
  KEY `idx_audit_order_id` (`order_id`)
);
```

3. Выполните следующий оператор SQL, чтобы создать хранимую процедуру `orders_create_audit` (не забудьте предварительно задать символы-разделители `$$`).

```
-- Создает хранимую процедуру orders_create_audit
CREATE PROCEDURE orders_create_audit(IN inOrderId INT,
  IN inMessage TEXT, IN inCode INT)
BEGIN
  INSERT INTO audit (order_id, created_on, message, code)
         VALUES (inOrderId, NOW(), inMessage, inCode);
END$$
```

4. Перейдите к коду веб-приложения. Добавьте следующий метод в класс `Orders` в файле `business/orders.php`.

```
// Создаем запись в таблице audit
public static function CreateAudit($orderId, $message, $code)
{
  // Составляем SQL-запрос
  $sql = 'CALL orders_create_audit(:order_id, :message, :code)';

  // Создаем массив параметров
  $params = array (':order_id' => $orderId,
                   ':message' => $message,
                   ':code' => $code);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}
```

5. Добавьте в папку `business` новый файл `order_processor.php` и поместите в него следующий код.

```
<?php
/* Основной класс конвейера, используется для получения информации
   о заказах, вызова классов этапов конвейера, аудита
   заказов и прочего */
class OrderProcessor
{
  public $mOrderInfo;
  public $mOrderDetailsInfo;
  public $mCustomerInfo;
  public $mContinueNow;

  private $_mCurrentPipelineSection;
  private $_mOrderProcessStage;

  // Конструктор класса
  public function __construct($orderId)
  {
    // Получаем заказ
    $this->mOrderInfo = Orders::GetOrderInfo($orderId);

    if (empty ($this->mOrderInfo['shipping_id']))
      $this->mOrderInfo['shipping_id'] = -1;

    if (empty ($this->mOrderInfo['tax_id']))
      $this->mOrderInfo['tax_id'] = -1;

    // Получаем подробные сведения о заказе
    $this->mOrderDetailsInfo = Orders::GetOrderDetails($orderId);

    // Получаем идентификатор покупателя, разместившего заказ
    $this->mCustomerInfo = Customer::Get($this->mOrderInfo['customer_id']);
    $credit_card = new SecureCard();
    $credit_card->LoadEncryptedDataAndDecrypt(
    $this->mCustomerInfo['credit_card']);
    $this->mCustomerInfo['credit_card'] = $credit_card;
  }

  /* Метод Process() вызывается из файлов presentation/checkout_info.php и
     presentation/admin_orders.php для обработки заказа */
  public function Process()
  {
    // Настраиваем обработку заказа
    $this->mContinueNow = true;
```

```
    // Фиксируем в журнале начало обработки
    $this->CreateAudit('Order Processor started.', 10000);

    // Вызываем этап конвейера
    try
    {
      while ($this->mContinueNow)
      {
        $this->mContinueNow = false;
        $this->_GetCurrentPipelineSection();
        $this->_mCurrentPipelineSection->Process($this);
      }
    }
    catch(Exception $e)
    {
      $this->MailAdmin('Order Processing error occurred.',
                       'Exception: "' . $e->getMessage() . '" on ' .
                       $e->getFile() . ' line ' . $e->getLine(),
                       $this->_mOrderProcessStage);
      $this->CreateAudit('Order Processing error occurred.', 10002);

      throw new Exception('Error occurred, order aborted. ' .
                          'Details mailed to administrator.');
    }

    $this->CreateAudit('Order Processor finished.', 10001);
  }

  // Добавляет сообщение аудита
  public function CreateAudit($message, $code)
  {
    Orders::CreateAudit($this->mOrderInfo['order_id'], $message, $code);
  }

  // Составляет сообщение электронной почты
  public function MailAdmin($subject, $message, $sourceStage)
  {
    $to = ADMIN_EMAIL;
    $headers = 'From: ' . ORDER_PROCESSOR_EMAIL . "\r\n";
    $body = 'Message: ' . $message . "\n" .
            'Source: ' . $sourceStage . "\n" .
            'Order ID: ' . $this->mOrderInfo['order_id'];

    $result = mail($to, $subject, $body, $headers);

    if ($result === false)
    {
      throw new Exception ('Failed sending this mail to administrator:' .
                           "\n" . $body);
    }
  }

  // Возвращает текущий этап конвейера
  private function _GetCurrentPipelineSection()
  {
    $this->_mOrderProcessStage = 100;
    $this->_mCurrentPipelineSection = new PsDummy();
  }
}
?>
```

6. Добавьте в папку business новый файл i_pipeline_section.php и поместите в него код интерфейса Ipipeline.

```
<?php
interface IPipelineSection
{
  public function Process($processor);
}
?>
```

7. Добавьте в папку business новый файл ps_dummy.php и поместите в него приведенный ниже код. Это код класса PsDummy, который мы будем пока использовать вместо настоящих этапов конвейера, которые создадим в следующей главе.

```
<?php
class PsDummy implements IPipelineSection
{
  public function Process($processor)
  {
    $processor->CreateAudit('PsDoNothing started.', 99999);

    $processor->CreateAudit('Customer: ' .
      $processor->mCustomerInfo['name'], 99999);

    $processor->CreateAudit('Order subtotal: ' .
      $processor->mOrderInfo['total_amount'], 99999);

    $processor->MailAdmin('Test.', 'Test mail from PsDummy.', 99999);

    $processor->CreateAudit('PsDoNothing finished', 99999);
  }
}
?>
```

8. Добавьте в файл include/config.php следующий код, указав в нем свой адрес электронной почты.

```
// Определения констант для работы системы уведомлений
define('ADMIN_EMAIL', 'Admin@example.com');
define('CUSTOMER_SERVICE_EMAIL', 'CustomerService@example.com');
define('ORDER_PROCESSOR_EMAIL', 'OrderProcessor@example.com');
define('SUPPLIER_EMAIL', 'Supplier@example.com');
```

ЗАМЕЧАНИЕ

Значения констант ADMIN_EMAIL и SUPPLIER_EMAIL должны быть реальными адресами — по этим адресам будут поступать сообщения, которые вам нужно будет просмотреть. В константах CUSTOMER_SERVICE_EMAIL и ORDER_PROCESSOR_EMAIL указывать реальные адреса необязательно, поскольку с них должны отправляться сообщения, — просматривать сообщения, приходящие на них, мы не будем.

9. Добавьте строки, выделенные полужирным шрифтом в следующем листинге, в файл admin.php.

```
...
require_once BUSINESS_DIR . 'customer.php';
require_once BUSINESS_DIR . 'i_pipeline_section.php';
require_once BUSINESS_DIR . 'ps_dummy.php';
require_once BUSINESS_DIR . 'order_processor.php';
```

10. Вставьте строку, выделенную полужирным шрифтом в следующем листинге, в файл `presentation/templates/admin_order_details.tpl`.

```
  <input type="submit" name="submitCancel" value="Cancel"
    {if ! $obj->mEditEnabled} disabled="disabled" {/if} />
  <input type="submit" name="submitProcessOrder" value="Process Order" />
</p>
<h3>Order contains these products:</h3>
```

11. Модифицируйте метод `init()` в файле `presentation/admin_order_details.php` так, как показано ниже.

```
public function init()
{
  if (isset ($_GET['submitUpdate']))
  {
    Orders::UpdateOrder($this->mOrderId, $_GET['status'],
      $_GET['comments'], $_GET['authCode'], $_GET['reference']);
  }

  if (isset ($_GET['submitProcessOrder']))
  {
    $processor = new OrderProcessor($this->mOrderId);
    $processor->Process();
  }

  $this->mOrderInfo = Orders::GetOrderInfo($this->mOrderId);
  $this->mOrderDetails = Orders::GetOrderDetails($this->mOrderId);
```

12. Загрузите страницу администрирования заказов в браузере и просмотрите сведения о каком-нибудь заказе. На странице сведений щелкните на кнопке Process Order (рис. 18.4).

ЗАМЕЧАНИЕ

Если вы хотите получать письма от веб-приложения на адреса на локальном компьютере (т.е. адреса вида `your_name@localhost`), запустите на этом компьютере SMTP-сервер.

В дистрибутивах Linux Red Hat (или Fedora) запуск SMTP-сервера выполняется командой

```
service sendmail start
```

На компьютерах под управлением Windows сервер электронной почты входит в состав IIS, и убедиться в том, что он запущен, можно с помощью IIS Management Console.

13. Загляните в почтовый ящик, адрес которого вы указали в константах настройки веб-приложения. В ящике должно быть письмо с текстом `Test mail from PsDummy`.

14. Просмотрите таблицу `audit` в базе данных с помощью phpMyAdmin. В ней должны присутствовать примерно такие записи, как на рис. 18.5.

Как это работает: каркас функциональности обработки заказов

Записи в таблицу `audit` будут добавлять класс `OrderProcessor` и классы этапов конвейера. В каждой записи указывается, успешно ли прошла операция, которую документирует эта запись. С помощью таких записей можно при необходимости проследить всю последовательность обработки заказа.

Столбец `code` позволяет задавать идентификаторы, которые можно ассоциировать с конкретными проблемами при обработке заказов. Теоретически можно создать в базе данных дополнительную таблицу, которая будет возвращать сообщения по заданным кодам, но в этом нет особой нужды, как вы увидите немного позже: сами коды достаточно просты и понятны для человека.

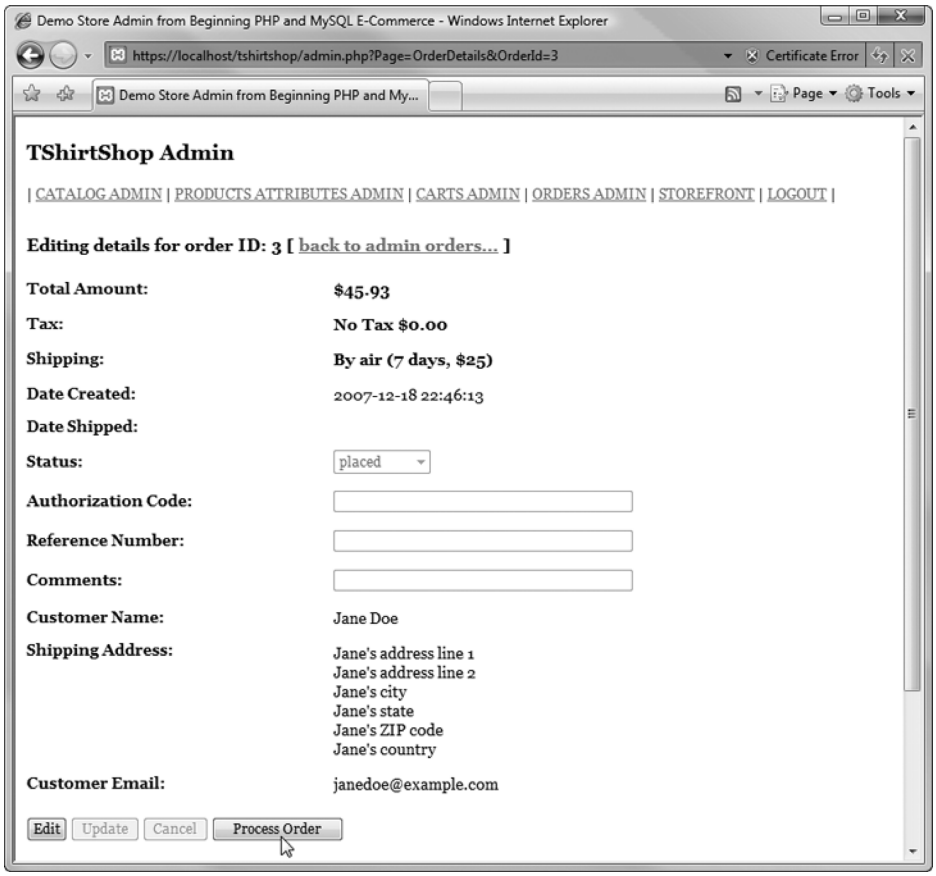


Рис. 18.4. Страница сведений о заказе с кнопкой Process Order

←T→	audit_id ▲	order_id	created_on	message	code
	15	3	2007-12-18 22:47:41	Order Processor started.	10000
	16	3	2007-12-18 22:47:41	PsDoNothing started.	99999
	17	3	2007-12-18 22:47:41	Customer: Jane Doe	99999
	18	3	2007-12-18 22:47:41	Order subtotal: 45.93	99999
	19	3	2007-12-18 22:47:41	PsDoNothing finished	99999
	20	3	2007-12-18 22:47:41	Order Processor finished.	10001

Рис. 18.5. Записи в таблице `audit`, сгенерированные классом `PsDummy`

В демонстрационных целях мы задали фиктивные адреса электронной почты для поставщика и администратора. Последний также выступает в роли покупателя, создавая заказы для тестирования конвейера. Это необходимо, чтобы проверить работу приложения до его запуска в эксплуатацию.

А теперь давайте посмотрим, как работает класс `OrderProcessor`. Большая часть кода этого класса находится в методе `Process()`, который теперь вызывается из кода в файле `presentation/admin_order_details.php` для обработки заказов.

```
public function Process()
{
  // Настраиваем обработку заказа
  $this->mContinueNow = true;
```

Далее мы вызываем метод `CreateAudit()`, чтобы добавить в таблицу audit запись о начале обработки заказа:

```
// Фиксируем в журнале начало обработки
$this->CreateAudit('Order Processor started.', 10000);
```

ЗАМЕЧАНИЕ

Код 10000 будет сохранен в записи в таблице `audit`. Смысл кодов мы рассмотрим немного позже.

После этого начинается непосредственно обработка заказов. Перед началом любого этапа обработки код проверяет состояние флага `$mContinueNow`. Это позволяет или запустить следующий этап конвейера сразу после завершения предыдущего (если предыдущий этап установил `$mContinueNow` равным `true`), или приостановить обработку (если `$mContinueNow` равен `false`). Такой механизм необходим, поскольку для начала некоторых этапов программа должна дождаться определенных действий людей, например подтверждения того, что нужные товары есть в запасе или что на кредитной карте есть сумма, достаточная для оплаты заказа.

Какой этап конвейера запустить, определяет метод `_GetCurrentPipelineSection()`, который возвращает класс этапа конвейера (мы создадим эти классы в следующей главе). Нужный этап должен выбираться по статусу заказа, но пока что метод `_GetCurrentPipelineSection()` просто возвращает класс `PsDummy`. В следующей главе мы модифицируем этот метод для работы с реальными классами этапов.

```
// Возвращает текущий этап конвейера
private function _GetCurrentPipelineSection()
{
  $this->_mOrderProcessStage = 100;
  $this->_mCurrentPipelineSection = new PsDummy();
}
```

В методе `Process()` вызов метода `_GetCurrentPipelineSection()` помещен в блок `try`.

```
// Вызываем этап конвейера
try
{
  while ($this->mContinueNow)
  {
    $this->mContinueNow = false;
    $this->_GetCurrentPipelineSection();
    $this->_mCurrentPipelineSection->Process($this);
  }
}
```

Заметьте, что в цикле `while` флаг `$mContinueNow` устанавливается равным `false`, — по умолчанию конвейер останавливается по окончании каждого этапа. Однако вызов метода `Process()` класса текущего этапа конвейера (этот метод принимает в качестве параметра экземпляр класса `OrderProcessor` и, таким образом, получает доступ к `$mContinueNow`) может установить этот флаг равным `true`, разрешив вызов следующего этапа конвейера без вмешательства человека.

Также обратите внимание на то, что в приведенном выше фрагменте кода вызов метода `Process()` выполняется для неизвестного объекта. Такой подход разрешен, если у этого объекта есть доступный метод `Process()`. Чтобы гарантировать наличие этого метода, мы создали интерфейс.

Все классы этапов конвейера поддерживают простой интерфейс `IPipelineSection`, определение которого выглядит так.

```
<?php
interface IPipelineSection
{
  public function Process($processor);
}
?>
```

Все классы конвейера используют метод `Process()` для выполнения необходимых операций с заказами. Этот метод в качестве аргумента получает ссылку на экземпляр класса `OrderProcessor`, чтобы получить доступ к `public`-свойствам и методам этого класса.

Последняя часть метода `Process()` в классе `OrderProcessor` выполняет обработку исключений. В ней мы перехватываем любые исключения, которые могут генерироваться классами этапов конвейера, и сообщаем об этих исключениях администратору по электронной почте, вызывая метод `MailAdmin()`. Кроме того, мы добавляем записи в таблицу `audit` и генерируем новые исключения, которые могут перехватываться в коде, использующем класс `OrderProcessor`.

```
catch(Exception $e)
{
  $this->MailAdmin('Order Processing error occurred.',
                   'Exception: "' . $e->getMessage() . '" on ' .
                   $e->getFile() . ' line ' . $e->getLine(),
                   $this->_mOrderProcessStage);

  $this->CreateAudit('Order Processing error occurred.', 10002);

  throw new Exception('Error occurred, order aborted. ' .
                      'Details mailed to administrator.');
}
```

Независимо от того, успешно ли завершился этап конвейера, последняя запись, добавляемая в таблицу `audit`, сообщает об окончании обработки.

```
  $this->CreateAudit('Order Processor finished.', 10001);
}
```

А теперь рассмотрим метод `MailAdmin()`, который просто составляет сообщение электронной почты с заданными параметрами.

```
// Составляет сообщение электронной почты
public function MailAdmin($subject, $message, $sourceStage)
{
  $to = ADMIN_EMAIL;
  $headers = 'From: ' . ORDER_PROCESSOR_EMAIL . "\r\n";
  $body = 'Message: ' . $message . "\n" .
          'Source: ' . $sourceStage . "\n" .
          'Order ID: ' . $this->mOrderInfo['order_id'];

  $result = mail($to, $subject, $body, $headers);

  if ($result === false)
  {
    throw new Exception ('Failed sending this mail to administrator:' .
                         "\n" . $body);
  }
}
```

Метод `CreateAudit()` также не содержит ничего интересного — он вызывает метод уровня логики приложения `Orders::CreateAudit()`, показанный выше.

```
// Добавляет сообщение аудита
public function CreateAudit($message, $code)
{
  Orders::CreateAudit($this->mOrderInfo['order_id'], $message, $code);
}
```

Теперь мы можем подробнее рассмотреть коды, которые записываются в таблицу `audit`. Во всех случаях они являются пятиразрядными числами. Первая цифра указывает на автора записи: цифра 1 означает, что запись создана классом `OrderProcessor`, а цифра 2 указывает на классы этапов конвейера. Следующие две цифры указывают на этап конвейера, добавивший запись (они соответствуют статусу заказа в момент добавления запи-

си в таблицу). Последние две цифры — это порядковый номер типа сообщения. Например, до сих пор мы встречались с тремя кодами.

- `10000` — запущена обработка заказа
- `10001` — завершена обработка заказа
- `10002` — при обработке заказа произошла ошибка

Позже мы встретимся с кодами, которые начинаются с 2, когда начнем реализовывать классы настоящих этапов конвейера. Надеемся, вы согласитесь, что такая система кодов достаточно понятна и в то же время универсальна, хотя вы, разумеется, можете при желании использовать свои коды. Последнее замечание: коды, заканчивающиеся цифрами 00 и 01, обозначают начало и конец работы этапов конвейера, а все остальные — дополнительные сообщения аудита.

Класс `PsDummy`, используемый в коде как образец этапа конвейера, содержит лишь несколько проверок.

```
<?php
class PsDummy implements IPipelineSection
{
  public function Process($processor)
  {
    $processor->CreateAudit('PsDoNothing started.', 99999);
    $processor->CreateAudit('Customer: ' .
      $processor->mCustomerInfo['name'], 99999);

     $processor->CreateAudit('Order subtotal: ' .
       $processor->mOrderInfo['total_amount'], 99999);

    $processor->MailAdmin('Test.', 'Test mail from PsDummy.', 99999);

    $processor->CreateAudit('PsDoNothing finished', 99999);
  }
}
?>
```

Этот код использует методы `CreateAudit()` и `MailAdmin()` класса `OrderProcessor` для генерации записей аудита и сообщений электронной почты, показывающих, что конвейер работает. Обратите внимание на код сообщения `99999`: он используется, потому что `PsDummy` не является настоящим этапом конвейера.

В целом, весь код, который мы написали в этом упражнении, просто воплощает проектные решения, принятые нами ранее. Это отличный пример того, как правильные решения позволяют легко и быстро написать работающий код.

Обновление кода обработки заказов

Нужно добавить еще несколько деталей к классу `OrderProcessor`, и именно этим мы займемся в нескольких следующих упражнениях. В них мы реализуем механизмы, упомянутые в начале главы:

- Обновление статуса заказа
- Задание сведений для аутентификации кредитной карты
- Задание даты отправки заказа
- Отправка сообщений электронной почты поставщикам и покупателям
- Получение подробных сведений о заказах и адреса доставки

Сначала мы напишем код, который позволит обновлять статус заказа. Каждый этап конвейера должен менять статус заказа, чтобы этот заказ мог принять следующий этап. Вместо простого увеличения числа, обозначающего статус заказа,

мы реализуем несколько более сложный механизм. Это может пригодиться, если в дальнейшем мы захотим реализовать ветвящийся конвейер. Нам понадобится создать в базе данных хранимую процедуру `orders_update_status`, а на уровне логики приложения — метод `UpdateOrderStatus()` в классе `Orders` (файл `business/orders.php`).

Упражнение: обновление статуса заказа

1. С помощью phpMyAdmin создайте в базе данных `tshirtshop` хранимую процедуру `orders_update_status`, используя следующий оператор SQL (не забудьте предварительно задать символы `$$` в качестве разделителя).

```
-- Создает хранимую процедуру orders_update_status
CREATE PROCEDURE orders_update_status(IN inOrderId INT, IN inStatus INT)
BEGIN
  UPDATE orders SET status = inStatus WHERE order_id = inOrderId;
END$$
```

2. Добавьте метод `UpdateOrderStatus()` в класс `Orders` в файле `business/orders.php`.

```
// Обновляем статус заказа, находящегося на конвейере
public static function UpdateOrderStatus($orderId, $status)
{
  // Составляем SQL-запрос
  $sql = 'CALL orders_update_status(:order_id, :status)';

  // Создаем массив параметров
  $params = array (':order_id' => $orderId, ':status' => $status);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}
```

3. Метод в классе `OrderProcessor` (в файле `business/order_processor.php`), вызывающий только что созданный метод уровня логики приложения, также называется `UpdateOrderStatus()`. Добавьте его в класс `OrderProcessor`.

```
// Задаем статус заказа
public function UpdateOrderStatus($status)
{
  Orders::UpdateOrderStatus($this->mOrderInfo['order_id'], $status);
  $this->mOrderInfo['status'] = $status;
}
```

Упражнение: задание сведений для аутентификации кредитной карты

1. С помощью phpMyAdmin создайте в базе данных `tshirtshop` хранимую процедуру `orders_set_auth_code`, используя следующий оператор SQL (не забудьте предварительно задать символы `$$` в качестве разделителя).

```
-- Создает хранимую процедуру orders_set_auth_code
CREATE PROCEDURE orders_set_auth_code(IN inOrderId INT,
  IN inAuthCode VARCHAR(50), IN inReference VARCHAR(50))
BEGIN
  UPDATE orders
  SET    auth_code = inAuthCode, reference = inReference
```

```
  WHERE  order_id = inOrderId;
END$$
```

2. Добавьте метод `SetOrderAuthCodeAndReference()` в класс `Orders` в файле `business/orders.php`:

```
// Задаем код авторизации заказа
public static function SetOrderAuthCodeAndReference ($orderId, $authCode,
                                                      $reference)
{
  // Составляем SQL-запрос
  $sql = 'CALL orders_set_auth_code(:order_id, :auth_code, :reference)';

  // Создаем массив параметров
  $params = array (':order_id' => $orderId,
                   ':auth_code' => $authCode,
                   ':reference' => $reference);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}
```

3. Для сохранения необходимых значений в базе данных предназначен метод `SetOrderAuthCodeAndReference()`. Добавьте его в класс `OrderProcessor`.

```
// Задаем код авторизации и ссылки на заказ
public function SetAuthCodeAndReference($authCode, $reference)
{
  Orders::SetOrderAuthCodeAndReference($this->mOrderInfo['order_id'],
                                       $authCode, $reference);

  $this->mOrderInfo['auth_code'] = $authCode;
  $this->mOrderInfo['reference'] = $reference;
}
```

Этот код также задает значения соответствующих элементов в массиве `$mOrderInfo` на случай, если они понадобятся до того, как класс `OrderProcessor` завершит работу. В такой ситуации нет особого смысла повторно извлекать эти значения из базы данных — мы все равно уже их знаем.

В следующей главе, занимаясь кредитными картами, мы будем задавать значения для полей `auth_code` и `reference` в таблице `orders`.

Упражнение: задание даты отправки товаров

1. Отправляя заказанные товары, мы должны записать дату их отправки в базу данных (это может быть просто текущая дата). Добавьте в базу данных хранимую процедуру `orders_set_date_shipped` (не забудьте предварительно задать символы `$$` в качестве разделителя).

```
-- Создает хранимую процедуру orders_set_date_shipped
CREATE PROCEDURE orders_set_date_shipped(IN inOrderId INT)
BEGIN
  UPDATE orders SET shipped_on = NOW() WHERE order_id = inOrderId;
END$$
```

2. Добавьте в уровень данных новый метод, который будет вызывать только что созданную хранимую процедуру. Это метод `SetDateShipped()` класса `Orders` в файле `business/orders.php`. Вот его код.

```
// Задаем дату отправки заказанных товаров
public static function SetDateShipped($orderId)
{
  // Составляем SQL-запрос
  $sql = 'CALL orders_set_date_shipped(:order_id)';

  // Создаем массив параметров
  $params = array (':order_id' => $orderId);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}
```

3. Добавьте следующий метод в класс `OrderProcessor` в файле `business/order_processor.php`.

```
// Задаем дату отправки заказа
public function SetDateShipped()
{
  Orders::SetDateShipped($this->mOrderInfo['order_id']);

  $this->mOrderInfo['shipped_on'] = date('Y-m-d');
}
```

Упражнение: отправка сообщений электронной почты поставщикам и покупателям

1. Нам понадобятся два метода для отправки сообщений электронной почты один — для поставщиков и один — для покупателей. Добавьте в класс `OrderProcessor` метод `MailCustomer()`.

```
// Отправляем сообщение электронной почты покупателю
public function MailCustomer($subject, $body)
{
  $to = $this->mCustomerInfo['email'];
  $headers = 'From: ' . CUSTOMER_SERVICE_EMAIL . "\r\n";
  $result = mail($to, $subject, $body, $headers);

  if ($result === false)
  {
    throw new Exception ('Unable to send e-mail to customer.');
  }
}
```

2. Добавьте в класс `OrderProcessor` метод `MailSupplier()`.

```
// Отправляем сообщение электронной почты поставщику
public function MailSupplier($subject, $body)
{
  $to = SUPPLIER_EMAIL;
  $headers = 'From: ' . ORDER_PROCESSOR_EMAIL . "\r\n";
  $result = mail($to, $subject, $body, $headers);

  if ($result === false)
  {
    throw new Exception ('Unable to send e-mail to supplier.');
  }
}
```

Упражнение: получение сведений о заказе и адреса покупателя

1. Нам понадобится возможность получать сведения о заказе и адрес покупателя в виде строк. Адрес нам будет предоставлять метод `GetCustomerAddressString()` в классе `OrderProcessor`. Добавьте этот метод в файл `business/order_processor.php`.

```
// Возвращает адрес покупателя в виде строки
public function GetCustomerAddressAsString()
{
  $new_line = "\n";

  $address_details = $this->mCustomerInfo['name'] . $new_line .
                     $this->mCustomerInfo['address_1'] . $new_line;

  if (!empty ($this->mOrderInfo['address_2']))
    $address_details .= $this->mCustomerInfo['address_2'] . $new_line;

  $address_details .= $this->mCustomerInfo['city'] . $new_line .
                      $this->mCustomerInfo['region'] . $new_line .
                      $this->mCustomerInfo['postal_code'] . $new_line .
                      $this->mCustomerInfo['country'];

  return $address_details;
}
```

2. Добавьте в класс `OrderProcessor` метод `GetOrderAsString()`.

```
// Возвращает строку со сведениями о заказе
public function GetOrderAsString($withCustomerDetails = true)
{
  $total_cost = 0.00;
  $order_details = '';
  $new_line = "\n";

  if ($withCustomerDetails)
  {
    $order_details = 'Customer address:' . $new_line .
                     $this->GetCustomerAddressAsString() .
                     $new_line . $new_line;

    $order_details .= 'Customer credit card:' . $new_line .
                      $this->mCustomerInfo['credit_card']->CardNumberX .
                      $new_line . $new_line;
  }

  foreach ($this->mOrderDetailsInfo as $order_detail)
  {
    $order_details .= $order_detail['quantity'] . ' ' .
                      $order_detail['product_name'] . '(' .
                      $order_detail['attributes'] . ') $' .
                      $order_detail['unit_cost'] . ' each, total cost $' .
                      number_format($order_detail['subtotal'],
                                    2, '.', '') . $new_line;

    $total_cost += $order_detail['subtotal'];
  }

  // Добавляем стоимость доставки
  if ($this->mOrderInfo['shipping_id'] != -1)
  {
```

```
    $order_details .= 'Shipping: ' . $this->mOrderInfo['shipping_type'] .
                      $new_line;

    $total_cost += $this->mOrderInfo['shipping_cost'];
  }

  // Добавляем налоги
  if ($this->mOrderInfo['tax_id'] != -1 &&
      $this->mOrderInfo['tax_percentage'] != 0.00)
  {
    $tax_amount = round((float)$total_cost *
                  (float)$this->mOrderInfo['tax_percentage'], 2)
                  / 100.00;
    $order_details .= 'Tax: ' . $this->mOrderInfo['tax_type'] . ', $' .
                      number_format($tax_amount, 2, '.', '') .
                      $new_line;

    $total_cost += $tax_amount;
  }

  $order_details .= $new_line . 'Total order cost: $' .
                    number_format($total_cost, 2, '.', '');

  return $order_details;
}
```

Как это работает: модификации класса OrderProcessor

Мы добавили в классы `Orders` и `OrderProcessor` несколько новых методов и создали необходимые для работы этих методов хранимые процедуры. Вся эта инфраструктура необходима для работы конвейера обработки заказов, классы которого мы создадим в следующей главе.

Резюме

В этой главе мы выбрали структуру конвейера обработки заказов для нашего проекта и начали ее реализовывать. Большую часть кода конвейера мы напишем в следующей главе.

Мы уже выполнили часть работы по созданию конвейера, в частности — сделали следующее.

- Внесли в код проекта изменения, необходимые для работы конвейера
- Создали инфраструктуру для поддержки конвейера
- Добавили в базу данных возможность сохранения данных для аудита конвейера

В следующей главе мы закончим реализацию конвейера обработки заказов.

Глава 19

Реализация конвейера обработки заказов: часть II

В предыдущей главе мы реализовали базовую функциональность класса `OrderProcessor`, отвечающего за перемещение заказа между этапами конвейера. Для демонстрации работы конвейера мы создали класс фиктивного этапа конвейера — `PsDummy`, а настоящие этапы конвейера нам еще предстоит создать.

В этой главе мы займемся именно созданием классов, реализующих этапы конвейера. Это позволит конвейеру полностью выполнять обработку заказов, хотя функциональность для работы с кредитными картами мы добавим только в следующей главе.

Кроме того, мы займемся администрированием заказов — нужно будет адаптировать страницы их администрирования для работы с конвейером.

Реализация этапов конвейера

В предыдущей главе мы создали класс `OrderProcessor`, но он не вызывает классы настоящих этапов конвейера. Вместо этого в нем использовался класс фиктивного этапа — `PsDummy`. В этой главе мы позволим классу `OrderProcessor` обращаться к классам настоящих этапов, выбирая нужный по статусу заказа.

Рассмотрим поочередно код каждого из классов этапов. Всего мы создадим восемь классов этапов, после чего конвейер будет готов, за исключением механизма авторизации для кредитных карт, которым мы займемся в главе 20. Вот имена классов, которые мы создадим.

- `PsInitialNotification`
- `PsCheckFunds`
- `PsCheckStock`
- `PsStockOk`
- `PsTakePayment`
- `PsShipGoods`
- `PsShipOk`
- `PsFinalNotification`

Мы будем обсуждать классы по мере их создания — обычных упражнений для создания этих классов не будет. Прежде чем двигаться дальше, напомним, что весь исходный код доступен на сайте издательства Apress (`http://www.apress.com`).

Класс `PsInitialNotification`

Это класс первого этапа конвейера, отвечающий за отправку покупателю сообщения электронной почты. Такое сообщение подтверждает факт размещения заказа. Создайте в папке `business` файл `ps_initial_notification.php` и начинайте добавлять в него фрагменты кода, приведенные ниже. Начало кода класса выглядит вполне стандартно.

```
<?php
class PsInitialNotification implements IPipelineSection
{
  private $_mProcessor;

  public function Process($processor)
  {
    // Задаем ссылку на переменную
    $this->_mProcessor = $processor;

    // Аудит
    $processor->CreateAudit('PsInitialNotification started.', 20000);
```

Класс реализует интерфейс `IPipelineSection`, а также содержит `private`-элемент для хранения ссылки на объект класса `OrderProcessor`, вызвавший `PsInitialNotification`, и метод `Process()`. Метод `Process()` начинает работу с получения ссылки на объект `OrderProcessor`. Это будут делать и методы `Process()` некоторых других классов этапов, поскольку такая ссылка понадобится для работы данных классов. Затем метод добавляет запись в таблицу `audit` в базе данных, используя код, описанный в главе 18 (начальная цифра 2 означает, что запись сделана классом этапа конвейера, а последние цифры 00 — что это запись о начале работы этапа).

Остальной код в методе `Process()` занимается непосредственно составлением и отправкой сообщения. Для составления сообщения ему понадобится информация о покупателе, которую мы легко можем получить. Кроме того, мы используем для составления текста сообщения специальный метод, который рассмотрим немного позже.

```
// Отправляем покупателю сообщение по электронной почте
$processor->MailCustomer(STORE_NAME . ' order received.',
                         $this->GetMailBody());
```

После отправки сообщения мы добавляем в таблицу `audit` еще одну запись и сообщаем классу `OrderProcessor`, что можно запускать следующий этап конвейера.

```
// Аудит
$processor->CreateAudit('Notification e-mail sent to customer.', 20002);
// Обновляем статус заказа
$processor->UpdateOrderStatus(1);
// Продолжаем обработку
$processor->mContinueNow = true;
```

Если все идет по плану, метод `Process()` завершает работу добавлением еще одной записи в таблицу `audit`.

```
  // Аудит
  $processor->CreateAudit('PsInitialNotification finished.', 20001);
}
```

Метод `GetMailBody()` генерирует текст сообщения, отправляемого покупателю по электронной почте. Этот текст содержит информацию о заказе, включая его номер и дату отправки. Добавьте этот метод в класс `PsInitialNotification`.

```
private function GetMailBody()
{
  $body = 'Thank you for your order! ' .
          'The products you have ordered are as follows:';
  $body.= "\n\n";

  $body.= $this->_mProcessor->GetOrderAsString(false);
  $body.= "\n\n";

  $body.= 'Your order will be shipped to:';
  $body.= "\n\n";

  $body.= $this->_mProcessor->GetCustomerAddressAsString();
  $body.= "\n\n";

  $body.= 'Order reference number: ';
  $body.= $this->_mProcessor->mOrderInfo['order_id'];
  $body.= "\n\n";

  $body.= 'You will receive a confirmation e-mail when this order ' .
          'has been dispatched. Thank you for shopping at ' .
          STORE_NAME . '!';

    return $body;
  }
}
?>
```

Следующий этап конвейера — `PsCheckFunds`.

Класс `PsCheckFunds`

Этот этап конвейера должен проверить наличие на кредитной карте покупателя суммы, достаточной для оплаты заказа. Пока что мы создадим заместитель этого класса и будем считать, что сумма всегда доступна. Версию класса `PsCheckFunds`, которая действительно будет проверять наличие денег, мы создадим в следующей главе.

Создайте в папке `business` файл `ps_check_funds.php`. В нем и будет размещаться класс `PsCheckFunds`. Основной метод этого класса — `Process()` — начинается почти так же, как и в классе `PsInitialNotification`.

```
<?php
class PsCheckFunds implements IPipelineSection
{
  public function Process($processor)
  {
    // Аудит
    $processor->CreateAudit('PsCheckFunds started.', 20100);
```

Хотя на самом деле сейчас этот метод не будет ничего проверять, мы зададим коды авторизации и ссылки для транзакции, чтобы класс `OrderProcessor` мог нормально работать.

```
/* Пока что считаем, что у покупателя всегда хватает денег, и
   задаем код авторизации и ссылку */
$processor->SetAuthCodeAndReference('DummyAuthCode',
                                    'DummyReference');
```

Кроме того, метод создает записи для аудита и содержит код, обеспечивающий продолжение обработки заказа.

```
    // Аудит
    $processor->CreateAudit('Funds available for purchase.', 20102);

    // Обновляем статус заказа
    $processor->UpdateOrderStatus(2);

    // Продолжаем обработку
    $processor->mContinueNow = true;

    // Аудит
    $processor->CreateAudit('PsCheckFunds finished.', 20101);
  }
}
?>
```

За этим этапом конвейера следует этап класса `PsCheckStock`.

Класс `PsCheckStock`

Этот этап конвейера отправляет по электронной почте поставщику сообщение с просьбой проверить наличие заказанных товаров в запасе. Создайте в папке `business` файл `ps_check_stock.php` и поместите в него следующий код.

```
<?php
class PsCheckStock implements IPipelineSection
{
  private $_mProcessor;

  public function Process($processor)
  {
    // Задаем ссылку на объект OrderProcessor
    $this->_mProcessor = $processor;

    // Аудит
    $processor->CreateAudit('PsCheckStock started.', 20200);
```

Сообщение электронной почты составляется так же, как и в классе `PsInitialNotification`, — с помощью специального метода для генерации текста.

```
// Отправляем сообщение поставщику
  $processor->MailSupplier(STORE_NAME . ' stock check.',
                           $this->GetMailBody());
```

Как и в предыдущих классах, метод `Process()` завершается генерацией записей для аудита и обновлением статуса заказа, хотя на этот раз код не приказывает классу `OrderProcessor` немедленно продолжать обработку заказа.

```
  // Аудит
  $processor->CreateAudit('Notification email sent to supplier.', 20202);

  // Обновляем статус заказа
  $processor->UpdateOrderStatus(3);

  // Аудит
  $processor->CreateAudit('PsCheckStock finished.', 20201);
}
```

Метод, генерирующий сообщение электронной почты, несложен: он просто перечисляет поочередно каждый товар из заказа и просит поставщика подтвердить

через страницу администрирования заказов в TShirtShop наличие всех этих товаров. Страницу администрирования заказов нужно будет модифицировать, чтобы у поставщика была возможность подтвердить наличие товаров (этим мы займемся немного позже).

```
private function GetMailBody()
{
    $body = 'The following goods have been ordered:';
    $body .= "\n\n";

    $body .= $this->_mProcessor->GetOrderAsString(false);
    $body .= "\n\n";

    $body .= 'Please check availability and confirm via ' .
             Link::ToAdmin();

    $body .= "\n\n";
    $body .= 'Order reference number: ';

    $body .= $this->_mProcessor->mOrderInfo['order_id'];
    return $body;
  }
}
?>
```

Когда этот этап конвейера завершает работу, обработка заказа приостанавливается. Позже, когда поставщик подтвердит доступность всех товаров из заказа, обработку этого заказа продолжит этап класса `PsStockOk`.

Класс `PsStockOk`

Этот этап конвейера просто подтверждает, что у поставщика есть в запасе все товары из заказа, и передает заказ на следующий этап. Метод `Process()` класса `PsStockOk` вызывается для заказов, для которых наличие товаров подтвердил поставщик. Создайте в папке `business` файл `ps_stock_ok.php` и поместите в него следующий код.

```
<?php
class PsStockOk implements IPipelineSection
{
  public function Process($processor)
  {
    // Аудит
    $processor->CreateAudit('PsStockOk started.', 20300);

    /* Этот метод вызывается, когда поставщик подтверждает
       доступность товаров, и здесь не нужно делать ничего, кроме
       обновления статуса заказа и аудита */
    $processor->CreateAudit('Stock confirmed by supplier.', 20302);

    // Обновляем статус заказа
    $processor->UpdateOrderStatus(4);

    // Продолжаем обработку
    $processor->mContinueNow = true;

    // Аудит
    $processor->CreateAudit('PsStockOk finished.', 20301);
  }
}
?>
```

Когда этот этап конвейера завершается, обработку заказа продолжает этап класса PsTakePayment.

Класс PsTakePayment

Этот этап конвейера должен завершить транзакцию, начатую на этапе PsCheckFunds. Как и для этапа PsCheckFunds, для этого мы создадим пока что только заместитель. Создайте в папке business файл ps_take_payment.php и поместите в него следующий код.

```
<?php
class PsTakePayment implements IPipelineSection
{
  public function Process($processor)
  {
    // Аудит
    $processor->CreateAudit('PsTakePayment started.', 20400);

    // Пока что считаем, что оплата успешно выполнена

    // Аудит
    $processor->CreateAudit
      ('Funds deducted from customer credit card account.',
       20402);

    // Обновляем статус заказа
    $processor->UpdateOrderStatus(5);

    // Продолжаем обработку
    $processor->mContinueNow = true;

    // Аудит
    $processor->CreateAudit('PsTakePayment finished.', 20401);
  }
}
?>
```

Когда этот этап конвейера завершается, обработку заказа продолжает этап класса PsShipOk.

Класс PsShipOk

Этот этап очень похож на PsStockOk, но на нем нужно выполнить немного больше операций. Поскольку мы знаем, что товары отправлены, мы сохраняем дату отправки в таблице orders. Технической необходимости в этом нет, поскольку все равно в каждой записи в таблице audit указываются дата и время ее создания. Но сохранение даты отправки в таблице orders упрощает последующий просмотр информации о заказах. Создайте в папке business файл ps_ship_ok.php и поместите в него следующий код.

```
<?php
class PsShipOk implements IPipelineSection
{
  public function Process($processor)
  {
    // Аудит
    $processor->CreateAudit('PsShipOk started.', 20600);

    // Задаем дату отправки заказа
```

```
    $processor->SetDateShipped();

    // Аудит
    $processor->CreateAudit('Order dispatched by supplier.', 20602);

    // Обновляем статус заказа
    $processor->UpdateOrderStatus(7);

    // Продолжаем обработку
    $processor->mContinueNow = true;

    // Аудит
    $processor->CreateAudit('PsShipOk finished.', 20601);
  }
}
?>
```

Когда этот этап конвейера завершается, обработку заказа продолжает этап класса `PsFinalNotification`.

Класс `PsFinalNotification`

Последний этап конвейера очень похож на первый, поскольку он отправляет покупателю сообщение по электронной почте. Это сообщение подтверждает отправку заказа. Создайте в папке `business` файл `ps_final_notification.php` и поместите в него следующий код.

```
<?php
class PsFinalNotification implements IPipelineSection
{
  private $_mProcessor;

  public function Process($processor)
  {
    // Задаем ссылку на объект OrderProcessor
    $this->_mProcessor = $processor;

    // Аудит
    $processor->CreateAudit('PsFinalNotification started.', 20700);

    // Отправляем покупателю сообщение по электронной почте
    $processor->MailCustomer(STORE_NAME . ' order dispatched.',
                             $this->GetMailBody());

    // Аудит
    $processor->CreateAudit('Dispatch e-mail send to customer.', 20702);

    // Обновляем статус заказа
    $processor->UpdateOrderStatus(8);

    // Аудит
    $processor->CreateAudit('PsFinalNotification finished.', 20701);
  }
```

Для генерации текста сообщения используется метод `GetMailBody()`, код которого довольно тривиален.

```
private function GetMailBody()
{
    $body = 'Your order has now been dispatched! ' .
            'The following products have been shipped:';
```

```
    $body .= "\n\n";

    $body .= $this->_mProcessor->GetOrderAsString(false);
    $body .= "\n\n";

    $body .= 'Your order has been shipped to:';
    $body .= "\n\n";

    $body .= $this->_mProcessor->GetCustomerAddressAsString();
    $body .= "\n\n";

    $body .= 'Order reference number: ';
    $body .= $this->_mProcessor->mOrderInfo['order_id'];
    $body .= "\n\n";

    $body .= 'Thank you for shopping at ' . STORE_NAME . '!';

    return $body;
  }
}
?>
```

Когда данный этап конвейера завершает работу, статус заказа получает значение 8, что означает завершение обработки этого заказа. Дальнейшие попытки обработать этот заказ с помощью класса `OrderProcessor` будут приводить к генерации исключений.

Проверка работы конвейера

А теперь давайте проведем простой тест, чтобы убедиться, что конвейер работает так, как мы ожидали.

Упражнение: проверка работы конвейера

1. Добавьте константу `STORE_NAME` в файл `include/config.php`.

```
// Название магазина
define('STORE_NAME', 'TShirtShop');
```

2. Добавьте ссылки на новые файлы классов в файл `admin.php` (также можете удалить ссылку на файл `ps_dummy.php`, который больше не нужен в программе).

```
require_once BUSINESS_DIR . 'order_processor.php';
require_once BUSINESS_DIR . 'ps_initial_notification.php';
require_once BUSINESS_DIR . 'ps_check_funds.php';
require_once BUSINESS_DIR . 'ps_check_stock.php';
require_once BUSINESS_DIR . 'ps_stock_ok.php';
require_once BUSINESS_DIR . 'ps_take_payment.php';
require_once BUSINESS_DIR . 'ps_ship_goods.php';
require_once BUSINESS_DIR . 'ps_ship_ok.php';
require_once BUSINESS_DIR . 'ps_final_notification.php';
```

3. Добавьте ссылки на новые файлы в файл `index.php`.

```
require_once BUSINESS_DIR . 'order_processor.php';
require_once BUSINESS_DIR . 'ps_initial_notification.php';
require_once BUSINESS_DIR . 'ps_check_funds.php';
require_once BUSINESS_DIR . 'ps_check_stock.php';
```

4. Модифицируйте метод `_GetCurrentPipelineSection()` класса `OrderProcessor` (в файле `business/order_processor.php`) так, как показано ниже:

```
private function _GetCurrentPipelineSection()
{
  switch($this->mOrderInfo['status'])
  {
    case 0:
      $this->_mOrderProcessStage = $this->mOrderInfo['status'];
      $this->_mCurrentPipelineSection = new PsInitialNotification();

      break;
    case 1:
      $this->_mOrderProcessStage = $this->mOrderInfo['status'];
      $this->_mCurrentPipelineSection = new PsCheckFunds();

      break;
    case 2:
      $this->_mOrderProcessStage = $this->mOrderInfo['status'];
      $this->_mCurrentPipelineSection = new PsCheckStock();

      break;
    case 3:
      $this->_mOrderProcessStage = $this->mOrderInfo['status'];
      $this->_mCurrentPipelineSection = new PsStockOk();

      break;
    case 4:
      $this->_mOrderProcessStage = $this->mOrderInfo['status'];
      $this->_mCurrentPipelineSection = new PsTakePayment();

      break;
    case 5:
      $this->_mOrderProcessStage = $this->mOrderInfo['status'];
      $this->_mCurrentPipelineSection = new PsShipGoods();

      break;
    case 6:
      $this->_mOrderProcessStage = $this->mOrderInfo['status'];
      $this->_mCurrentPipelineSection = new PsShipOk();

      break;
    case 7:
      $this->_mOrderProcessStage = $this->mOrderInfo['status'];
      $this->_mCurrentPipelineSection = new PsFinalNotification();

      break;
    case 8:
      $this->_mOrderProcessStage = 100;
      throw new Exception('Order already been completed.');

      break;
    default:
      $this->_mOrderProcessStage = 100;
      throw new Exception('Unknown pipeline section requested.');
  }
}
```

5. Откройте файл business/orders.php и замените в нем массив $mOrdersStatus Options в классе Orders новым вариантом для корректной обработки статуса заказов. Заметьте, что это повлияет на старые заказы, которые использовали другие коды статуса.

```
public static $mOrderStatusOptions = array (
                'Order placed, notifying customer',    // 0
                'Awaiting confirmation of funds',      // 1
                'Notifying supplier-stock check',      // 2
                'Awaiting stock confirmation',         // 3
                'Awaiting credit card payment',        // 4
                'Notifying supplier-shipping',         // 5
                'Awaiting shipment confirmation',      // 6
                'Sending final notification',          // 7
                'Order completed',                     // 8
                'Order canceled');                     // 9
```

6. Откройте файл presentation/admin_order_details.php и добавьте в определение класса AdminOrderDetails новый элемент, выделенный полужирным шрифтом ниже:

```
public $mTax = 0.0;
public $mOrderProcessMessage;
```

7. Модифицируйте код метода init() в классе AdminOrderDetails (в файле presentation/admin_order_details.php) так, как показано ниже. Модифицированный код будет выполнять необходимые операции, когда пользователь щелкнет на кнопке Process.

```
if (isset ($_GET['submitProcessOrder']))
{
  $processor = new OrderProcessor($this->mOrderId);

  try
  {
    $processor->Process();
    $this->mOrderProcessMessage = 'Order processed, status now: ' .
                                  $processor->mOrderInfo['status'];
  }
  catch (Exception $e)
  {
    $this->mOrderProcessMessage = 'Processing error, status now: ' .
    $processor->mOrderInfo['status'];
  }
}
```

8. Откройте файл presentation/templates/admin_order_details.tpl и добавьте в него код, выделенный полужирным шрифтом в листинге ниже:

```
{* admin_order_details.tpl *}
{load_presentation_object filename="admin_order_details" assign="obj"}
<form method="get" action="{$obj->mLinkToAdmin}">
  <h3>
    Editing details for order ID:
    {$obj->mOrderInfo.order_id} [
    <a href="{$obj->mLinkToOrdersAdmin}">back to admin orders...</a> ]
  </h3>
  {if $obj->mOrderProcessMessage}
  <p><strong>{$obj->mOrderProcessMessage}</strong></p>
  {/if}
  <input type="hidden" name="Page" value="OrderDetails" />
```

9. Откройте приложение TShirtShop в браузере, создайте новый заказ и откройте его на странице администрирования заказов; затем щелкните на кнопке Process Order. Вы должны получить сообщение электронной почты с уведомлением о размещении заказа (примерно такое, как на рис. 19.1).

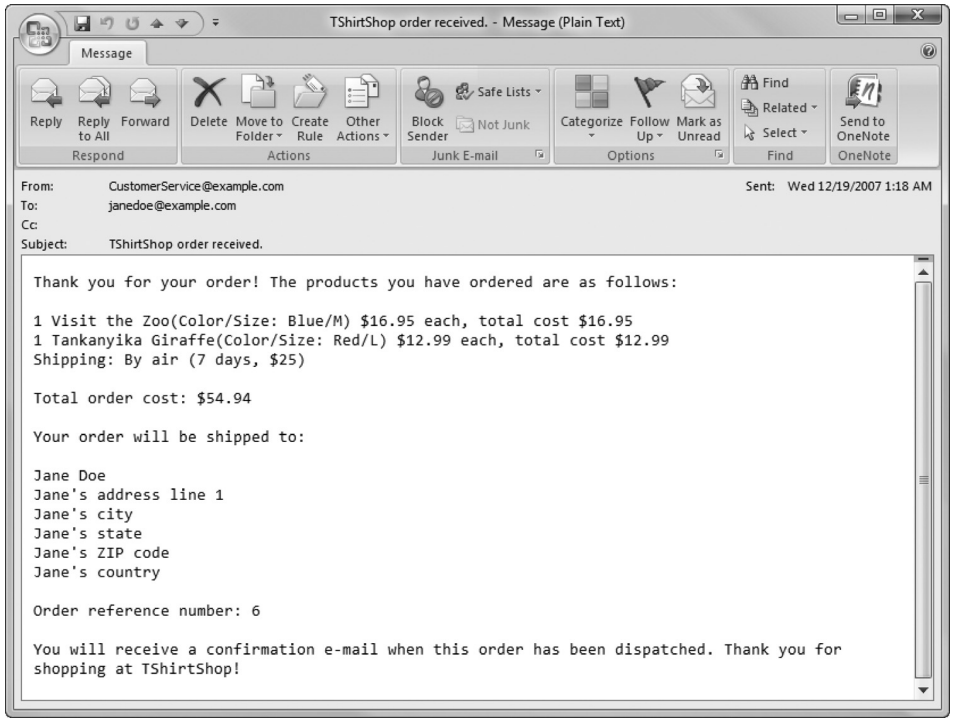


Рис. 19.1. Сообщение электронной почты о размещении заказа

10. Кроме того, на адрес поставщика должно также прийти сообщение электронной почты с просьбой подтвердить наличие заказанных товаров (рис. 19.2).

11. Продолжите обработку заказа, еще раз щелкнув на кнопке Process Order на странице администрирования заказов. Это инициирует работу второго этапа конвейера. На адрес поставщика должно также прийти сообщение электронной почты — на этот раз с просьбой отправить заказанные товары по указанному адресу (рис. 19.3).

12. Продолжите обработку заказа, еще раз щелкнув на кнопке Process Order на странице администрирования заказов. При этом будет запущена последняя часть конвейера.

13. На адрес электронной почты покупателя должно прийти сообщение об отправке заказанных товаров (рис. 19.4).

14. Просмотрите записи в таблице `audit` об обработке заказа; они должны располагаться в такой последовательности, как на рис. 19.5.

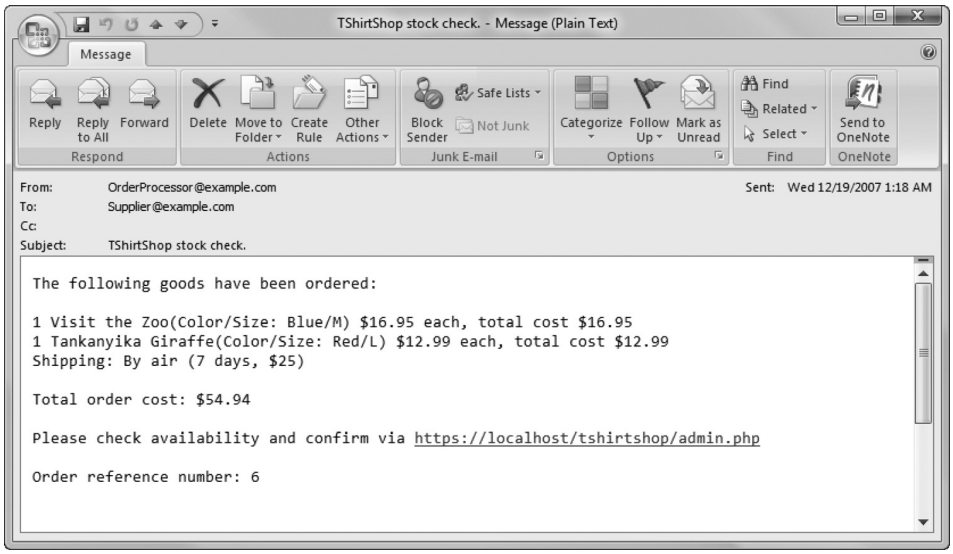


Рис. 19.2. Сообщение электронной почты поставщику

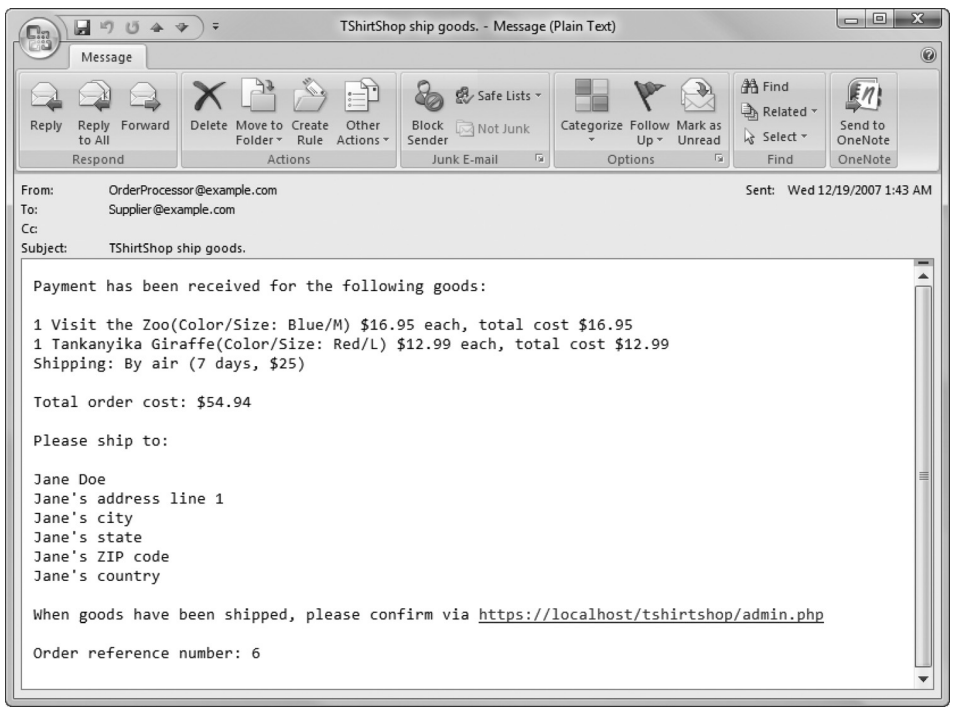


Рис. 19.3. Сообщение с просьбой выполнить отправку

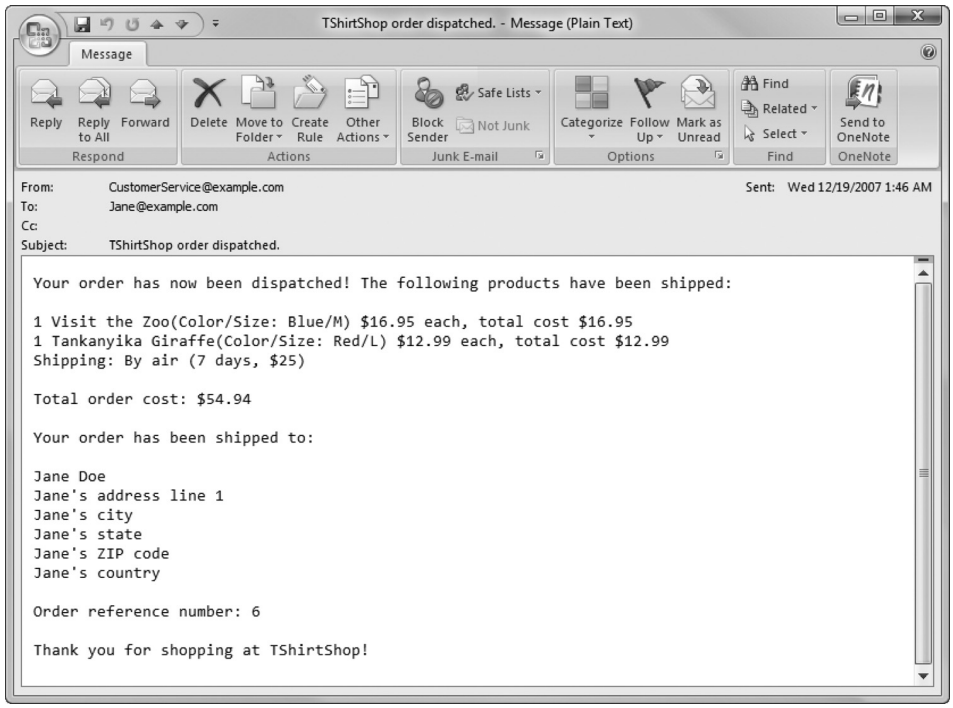


Рис. 19.4. Сообщение покупателю об отправке заказанных товаров

Как это работает: конвейер обработки заказов

Мы уже рассматривали каждый этап конвейера, поэтому осталось рассмотреть только новый код в классе `OrderProcessor`. Мы модифицировали код метода `_GetCurrentPipelineSection()`, который отвечает за выбор нужного этапа конвейера.

Новый код представляет собой просто оператор `switch`, присваивающий требуемое значение переменной `$_mCurrentPipelineSection`.

```
private function _GetCurrentPipelineSection()
{
  switch($this->mOrderInfo['status'])
  {
    case 0:
      $this->_mOrderProcessStage = $this->mOrderInfo['status'];
      $this->_mCurrentPipelineSection = new PsInitialNotification();

      break;
    case 1:
      $this->_mOrderProcessStage = $this->mOrderInfo['status'];
      $this->_mCurrentPipelineSection = new PsCheckFunds();

      break;
    case 2:
      $this->_mOrderProcessStage = $this->mOrderInfo['status'];
      $this->_mCurrentPipelineSection = new PsCheckStock();
```

audit_id	order_id	created_on	message	code
34	6	2007-12-19 01:18:20	Order Processor started.	10000
35	6	2007-12-19 01:18:20	PsInitialNotification started.	20000
36	6	2007-12-19 01:18:20	Notification e-mail sent to customer.	20002
37	6	2007-12-19 01:18:20	PsInitialNotification finished.	20001
38	6	2007-12-19 01:18:20	PsCheckFunds started.	20100
39	6	2007-12-19 01:18:20	Funds available for purchase.	20102
40	6	2007-12-19 01:18:20	PsCheckFunds finished.	20101
41	6	2007-12-19 01:18:20	PsCheckStock started.	20200
42	6	2007-12-19 01:18:20	Notification email sent to supplier.	20202
43	6	2007-12-19 01:18:20	PsCheckStock finished.	20201
44	6	2007-12-19 01:18:20	Order Processor finished.	10001
45	6	2007-12-19 01:42:50	Order Processor started.	10000
46	6	2007-12-19 01:42:50	PsStockOk started.	20300
47	6	2007-12-19 01:42:50	Stock confirmed by supplier.	20302
48	6	2007-12-19 01:42:50	PsStockOk finished.	20301
49	6	2007-12-19 01:42:50	PsTakePayment started.	20400
50	6	2007-12-19 01:42:50	Funds deducted from customer credit card account.	20402
51	6	2007-12-19 01:42:50	PsTakePayment finished.	20401
52	6	2007-12-19 01:42:50	PsShipGoods started.	20500
53	6	2007-12-19 01:42:50	Ship goods e-mail sent to supplier.	20502
54	6	2007-12-19 01:42:50	PsShipGoods finished.	20501
55	6	2007-12-19 01:42:50	Order Processor finished.	10001
56	6	2007-12-19 01:46:29	Order Processor started.	10000
57	6	2007-12-19 01:46:29	PsShipOk started.	20600
58	6	2007-12-19 01:46:29	Order dispatched by supplier.	20602
59	6	2007-12-19 01:46:29	PsShipOk finished.	20601
60	6	2007-12-19 01:46:29	PsFinalNotification started.	20700
61	6	2007-12-19 01:46:29	Dispatch e-mail send to customer.	20702
62	6	2007-12-19 01:46:29	PsFinalNotification finished.	20701
63	6	2007-12-19 01:46:29	Order Processor finished.	10001

Рис. 19.5. Записи в таблице `audit` об обработке заказа

```
break;
    case 3:
      $this->_mOrderProcessStage = $this->mOrderInfo['status'];
      $this->_mCurrentPipelineSection = new PsStockOk();

      break;
    case 4:
      $this->_mOrderProcessStage = $this->mOrderInfo['status'];
      $this->_mCurrentPipelineSection = new PsTakePayment();

      break;
    case 5:
      $this->_mOrderProcessStage = $this->mOrderInfo['status'];
      $this->_mCurrentPipelineSection = new PsShipGoods();

      break;
    case 6:
      $this->_mOrderProcessStage = $this->mOrderInfo['status'];
```

```
        $this->_mCurrentPipelineSection = new PsShipOk();

        break;
      case 7:
        $this->_mOrderProcessStage = $this->mOrderInfo['status'];
        $this->_mCurrentPipelineSection = new PsFinalNotification();

        break;
      case 8:
        $this->_mOrderProcessStage = 100;
        throw new Exception('Order already been completed.');

        break;
      default:
        $this->_mOrderProcessStage = 100;
        throw new Exception('Unknown pipeline section requested.');
    }
}
```

Если обработка выбранного заказа уже завершена или его статус неизвестен, код генерирует исключение.

Проверить, работает ли код генерации исключения, очень просто. Достаточно еще раз щелкнуть на кнопке Process Order для заказа, обработка которого уже завершена (т.е. статус заказа равен 8). Код в обработчике исключения отправит администратору сайта по электронной почте сообщение об ошибке, примерно такое, как на рис. 19.6.

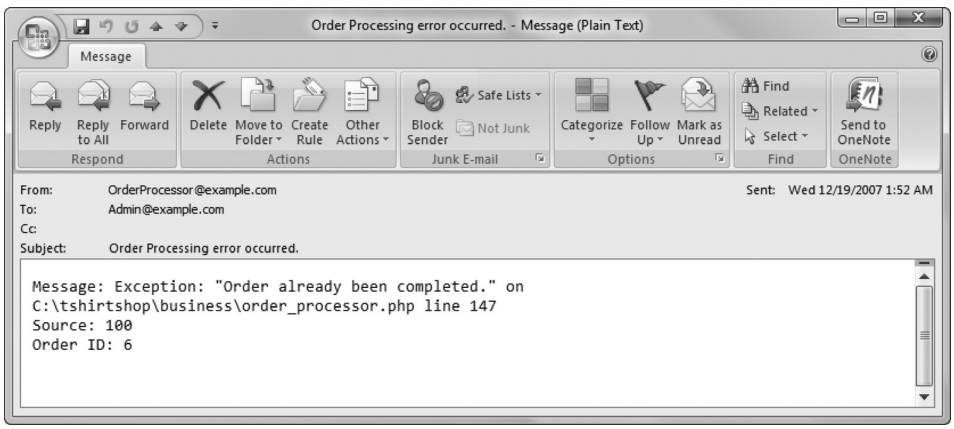


Рис. 19.6. Сообщение об ошибке при обработке заказа

Этого сообщения должно быть достаточно, чтобы администратор смог понять, что случилось с заказом.

Обновление страницы оплаты

В предыдущем упражнении нам пришлось трижды вызвать метод `OrderProcessor::Process()` с помощью кнопки Process Order, чтобы заказ был полностью обработан. В реально работающем магазине этого делать не придется: метод будет вызван один раз со страницы оплаты (файл `presentation/checkout_info.php`), когда покупатель разместит заказ, и еще два раза — поставщиком со страницы администрирования заказа (файл `presentation/admin_order_details.php`). Нам понадобится внести соответствующие изменения в эти файлы.

Выполните следующее упражнение, чтобы страница checkout_info.php смогла взаимодействовать с конвейером обработки заказов.

Упражнение: обновление страницы оплаты

1. Модифицируйте код метода init() класса CheckoutInfo в файле presentation/checkout_info.php: удалите код, выполняющий перенаправление на PayPal, и вставьте вместо него код, выделенный полужирным шрифтом в следующем листинге.

```
    // Создаем заказ и получаем его идентификатор
    $order_id = ShoppingCart::CreateOrder(
                  $this->mCustomerData['customer_id'],
                  (int)$_POST['shipping'], $tax_id);

    // Если получилось, открываем страницу с сообщением о заказе
    $redirect_to = Link::ToOrderDone();

    // Создаем объект класса OrderProcessor
    $processor = new OrderProcessor($order_id);

    try
    {
      $processor->Process();
    }
    catch (Exception $e)
    {
      // Если возникла ошибка, открываем страницу с сообщением о ней
      $redirect_to = Link::ToOrderError();
    }

    // Перенаправление на страницу с результатами обработки заказа
    header('Location: ' . $redirect_to);

    exit();
  }
...
```

2. Создайте в папке presentation/templates файл order_done.tpl и поместите в него такой код.

```
{* order_done.tpl *}
<h3>Thank you for your order!</h3>
<p class="description">
  A confirmation of your order will be sent to your registered email address.
</p>
```

3. Создайте в папке presentation/templates файл order_error.tpl и поместите в него такой код.

```
{* order_error.tpl *}
<h3>An error has occurred during the processing of your order.</h3>
<p class="description">
  If you have an inquiry regarding this message please email
  <a href="mailto:{$smarty.const.CUSTOMER_SERVICE_EMAIL}">
  {$smarty.const.CUSTOMER_SERVICE_EMAIL}</a>
</p>
```

4. Добавьте в класс Link (в файле presentation/link.php) следующие методы.

```
// Создает ссылку на страницу с сообщением
// о размещении заказа
```

```
public static function ToOrderDone()
{
  return self::Build('order-done/');
}

// Создает ссылку на страницу с сообщением об
// ошибке при обработке заказа
public static function ToOrderError()
{
  return self::Build('order-error/');
}
```

5. Откройте файл `.htaccess` в корневой папке проекта и добавьте в него строки, выделенные полужирным шрифтом в следующем листинге.

```
  RewriteRule ^checkout/?$ index.php?Checkout [L]

  # Переписываем ссылку для страницы сообщения о заказе
  RewriteRule ^order-done/?$ index.php?OrderDone [L]

  # Переписываем ссылку для страницы сообщения об ошибке
  RewriteRule ^order-error/?$ index.php?OrderError [L]
</IfModule>
```

6. Модифицируйте метод `CheckRequest()` в классе `Link` (файл `presentation/link.php`), вставив в него две строки, выделенные полужирным шрифтом в следующем листинге.

```
public static function CheckRequest()
{
  $proper_url = '';

  if (isset ($_GET['Search']) || isset($_GET['SearchResults']) ||
      isset ($_GET['CartAction']) || isset ($_GET['AjaxRequest']) ||
      isset ($_POST['Login']) || isset ($_GET['Logout']) ||
      isset ($_GET['RegisterCustomer']) ||
      isset ($_GET['AddressDetails']) ||
      isset ($_GET['CreditCardDetails']) ||
      isset ($_GET['AccountDetails']) || isset ($_GET['Checkout']) ||
      isset ($_GET['OrderDone']) || isset ($_GET['OrderError']))
  {
    return ;
  }
```

7. Откройте файл `presentation/store_front.php` и модифицируйте содержащийся в нем метод `init()` класса `StoreFront` так, как показано ниже. Модифицированный код будет загружать шаблон `order_done.tpl` или `order_error.tpl` в зависимости от того, успешно ли обработан заказ.

```
if (isset ($_GET['Checkout']))
{
  if (Customer::IsAuthenticated())
    $this->mContentsCell = 'checkout_info.tpl';
  else
    $this->mContentsCell = 'checkout_not_logged.tpl';
    $this->mHideBoxes = true;
  }

  if (isset($_GET['OrderDone']))
    $this->mContentsCell = 'order_done.tpl';
  elseif (isset($_GET['OrderError']))
    $this->mContentsCell = 'order_error.tpl';
```

```
  // Загружаем заголовок страницы
  $this->mPageTitle = $this->_GetPageTitle();
}
```

Теперь можно размещать заказы в электронном магазине, но при их обработке программа будет приостанавливать работу, чтобы получить подтверждение наличия товаров у поставщика. Сейчас мы займемся механизмом, который позволит поставщикам и администраторам инициировать дальнейшую обработку заказов.

Обновление страницы администрирования заказов

Основная задача этой страницы — позволять поставщикам и администраторам просматривать список обрабатываемых заказов и инициировать продолжение их обработки. Для этого код страницы просто вызывает метод `OrderProcessor::Process()`, как описано выше.

Такую страницу можно реализовать по-разному. Вообще говоря, ее можно сделать отдельным приложением, например, если ваши поставщики работают с магазином через локальную сеть, или оформить ее в виде веб-службы.

Чтобы упростить задачу, создадим общую страницу для поставщиков и администраторов. Это решение, вероятно, не идеально, поскольку вряд ли нужно знакомить поставщиков со всеми сведениями о заказах. Но в нашем случае это уменьшит объем кода, который нам придется написать. Кроме того, мы не будем создавать отдельный механизм защиты для этой страницы и сочтем, что люди, у которых есть право редактировать каталог, могут работать с заказами. В реальном магазине, вероятно, стоило бы разделить учетные записи по ролям, разрешая одной роли работать с каталогом, а другой — обрабатывать заказы, но в нашем проекте мы этого делать не будем.

Реализация уровня данных

Добавим в базу данных новую хранимую процедуру (`orders_get_audit_trail`) и обновим уже существующую (`orders_update_order`) для работы с новыми кодами статуса заказа.

Запустите phpMyAdmin, подключитесь к базе данных `tshirtshop` и откройте страницу нового SQL-запроса. Затем выполните следующие операторы SQL, чтобы создать новую хранимую процедуру `orders_update_order` (не забывайте задавать символы `$$` в качестве разделителя перед выполнением операторов).

```
-- Удаляем хранимую процедуру orders_update_order
DROP PROCEDURE orders_update_order$$

-- Создаем новую хранимую процедуру orders_update_order
CREATE PROCEDURE orders_update_order(IN inOrderId INT, IN inStatus INT,
  IN inComments VARCHAR(255), IN inAuthCode VARCHAR(50),
  IN inReference VARCHAR(50))
BEGIN
  DECLARE currentDateShipped DATETIME;

  SELECT shipped_on
  FROM   orders
```

```
  WHERE  order_id = inOrderId
  INTO   currentDateShipped;

  UPDATE orders
  SET    status = inStatus, comments = inComments,
         auth_code = inAuthCode, reference = inReference
  WHERE  order_id = inOrderId;

  IF inStatus < 7 AND currentDateShipped IS NOT NULL THEN
    UPDATE orders SET shipped_on = NULL WHERE order_id = inOrderId;
  ELSEIF inStatus > 6 AND currentDateShipped IS NULL THEN
    UPDATE orders SET shipped_on = NOW() WHERE order_id = inOrderId;
  END IF;
END$$
```

Выполните следующий оператор SQL, чтобы создать хранимую процедуру `orders_get_audit_trail`.

```
-- Создаем хранимую процедуру orders_get_audit_trail
CREATE PROCEDURE orders_get_audit_trail(IN inOrderId INT)
BEGIN
  SELECT audit_id, order_id, created_on, message, code
  FROM audit
  WHERE order_id = inOrderId;
END$$
```

Реализация уровня логики приложения

Чтобы воспользоваться только что созданной хранимой процедурой, нашему приложению понадобится новый метод уровня логики приложения. Добавьте этот метод, `GetAuditTrail()`, в класс `Orders` в файле `business/orders.php`.

```
// Возвращаем записи аудита об обработке заданного заказа
public static function GetAuditTrail($orderId)
{
  // Составляем SQL-запрос
  $sql = 'CALL orders_get_audit_trail(:order_id)';

  // Создаем массив параметров
  $params = array (':order_id' => $orderId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetAll($sql, $params);
}
```

Реализация уровня представления

Обновим компонентный шаблон `admin_order_details`, отображающий сведения о заказе. Ранее этот шаблон также позволял тестировать процесс обработки заказа, но сейчас мы удалим эту возможность. Вместо нее создадим механизм для продолжения обработки заказов, которые остались на этапах подтверждения наличия товаров и подтверждения отправки.

Кроме того, теперь вся информация для аудита заказа будет отображаться в специальной новой таблице. Новый вариант страницы администрирования заказов будет выглядеть примерно так, как показано на рис. 19.7.

В целом новую страницу можно разбить на три раздела:

- В первом разделе кнопка Process будет заменена кнопкой подтверждения для поставщиков.

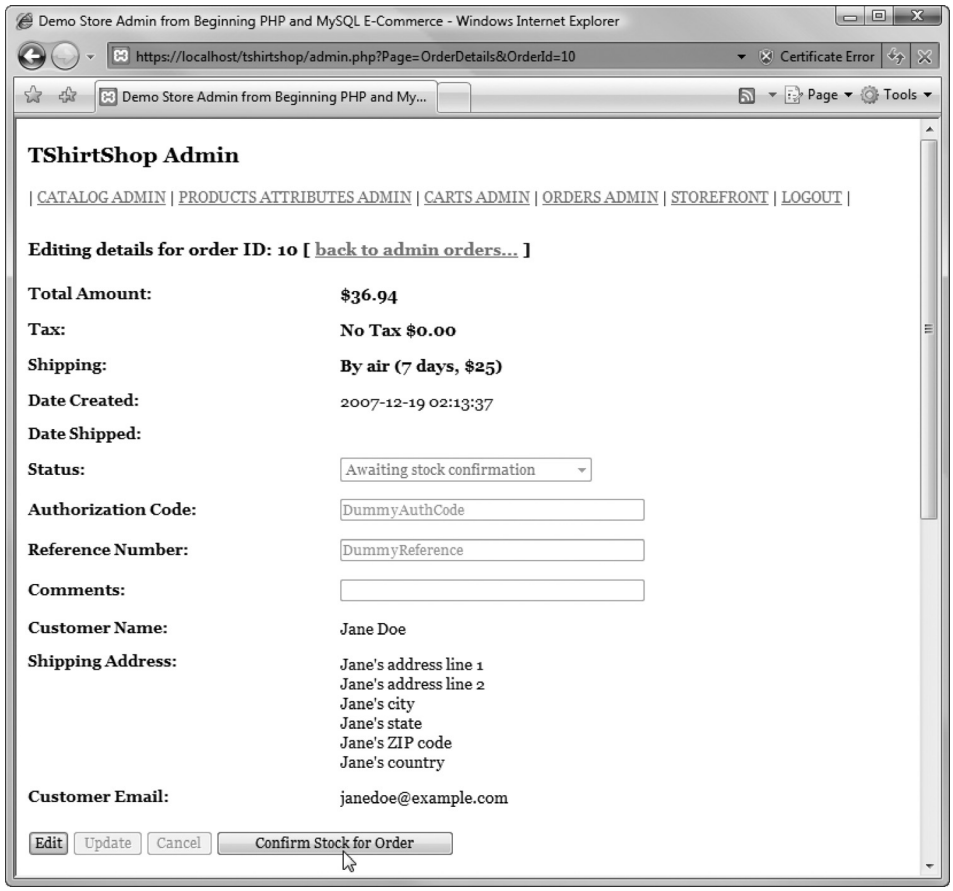


Рис. 19.7. Новая страница администрирования заказов

- Во втором разделе будет находиться таблица с названиями товаров из заказа.
- В третьем разделе будут отображаться записи для аудита обработки заказа.

Всю новую функциональность мы реализуем в следующем упражнении.

Упражнение: обновление страницы администрирования заказов

1. Удалите следующие строки из файла `presentation/templates/admin_order_details.tpl`.

```
{if $obj->mOrderProcessMessage}
<p><strong>{$obj->mOrderProcessMessage}</strong></p>
{/if}
```

2. В том же файле замените код, генерирующий кнопку Process Order, кодом, который в следующем листинге выделен полужирным шрифтом.

```
<input type="submit" name="submitCancel" value="Cancel"
  {if ! $obj->mEditEnabled} disabled="disabled" {/if} />
{if $obj->mProcessButtonText}
<input type="submit" name="submitProcessOrder"
  value="{$obj->mProcessButtonText}" />
```

```
  {/if}
</p>
<h3>Order contains these products:</h3>
<table class="tss-table">
```

3. В тот же файл вставьте код, выделенный полужирным шрифтом в следующем листинге.

```
{section name=i loop=$obj->mOrderDetails}
  <tr>
    <td>{$obj->mOrderDetails[i].product_id}</td>
    <td>
      {$obj->mOrderDetails[i].product_name}
      ({$obj->mOrderDetails[i].attributes})
    </td>
      <td>{$obj->mOrderDetails[i].quantity}</td>
      <td>${$obj->mOrderDetails[i].unit_cost}</td>
      <td>${$obj->mOrderDetails[i].subtotal}</td>
    </tr>
  {/section}
  </table>
  <h3>Order audit trail:</h3>
  <table class="tss-table">
    <tr>
      <th>Audit ID</th>
      <th>Created On</th>
      <th>Code</th>
      <th>Message</th>
    </tr>
  {section name=j loop=$obj->mAuditTrail}
    <tr>
      <td>{$obj->mAuditTrail[j].audit_id}</td>
      <td>{$obj->mAuditTrail[j].created_on}</td>
      <td>{$obj->mAuditTrail[j].code}</td>
      <td>{$obj->mAuditTrail[j].message}</td>
    </tr>
  {/section}
  </table>
</form>
```

4. Откройте файл `presentation/admin_order_details.php` и удалите из класса `AdminOrderDetails` строку с определением элемента `$mOrderProcessMessage`, показанную ниже:

```
public $mOrderProcessMessage;
```

5. В тот же класс добавьте определения двух новых элементов.

```
public $mProcessButtonText;
public $mAuditTrail;
```

6. Внесите в метод `init()` класса `AdminOrderDetails` изменения, выделенные полужирным шрифтом в следующем листинге.

```
public function init()
{
  if (isset ($_GET['submitUpdate']))
  {
    Orders::UpdateOrder($this->mOrderId, $_GET['status'],
      $_GET['comments'], $_GET['authCode'], $_GET['reference']);
  }

  if (isset ($_GET['submitProcessOrder']))
  {
```

```
    $processor = new OrderProcessor($this->mOrderId);
    $processor->Process();
  }

  $this->mOrderInfo = Orders::GetOrderInfo($this->mOrderId);
  $this->mOrderDetails = Orders::GetOrderDetails($this->mOrderId);
  $this->mCustomerInfo = Customer::Get($this->mOrderInfo['customer_id']);
  $this->mTotalCost = $this->mOrderInfo['total_amount'];
  $this->mAuditTrail = Orders::GetAuditTrail($this->mOrderId);

  if ($this->mOrderInfo['tax_percentage'] !== 0.0)
    $this->mTax = round((float)$this->mTotalCost *
                        (float)$this->mOrderInfo['tax_percentage'], 2)
                       / 100.00;
  $this->mTotalCost += $this->mOrderInfo['shipping_cost'];
  $this->mTotalCost += $this->mTax;

  // Форматируем значения
  $this->mTotalCost = number_format($this->mTotalCost, 2, '.', '');
  $this->mTax = number_format($this->mTax, 2, '.', '');

  if ($this->mOrderInfo['status'] == 3)
    $this->mProcessButtonText = 'Confirm Stock for Order';
  elseif ($this->mOrderInfo['status'] == 6)
    $this->mProcessButtonText = 'Confirm Shipment for Order';

  // Значение, определяющее, активен ли режим редактирования
  if (isset ($_GET['submitEdit']))
    $this->mEditEnabled = true;
  else
    $this->mEditEnabled = false;
}
```

7. Откройте TShirtShop в браузере, разместите новый заказ и перейдите на страницу администрирования заказов, чтобы протестировать внесенные изменения. Разместив новый заказ, вы должны получить подтверждение (примерно такое, как на рис. 19.8). После этого администратор сайта сможет подтвердить наличие заказанных товаров в запасе и их отправку, как на рис. 19.7. Кроме того, на странице сведений о заказе теперь должны отображаться записи аудита заказа, как на рис. 19.9.

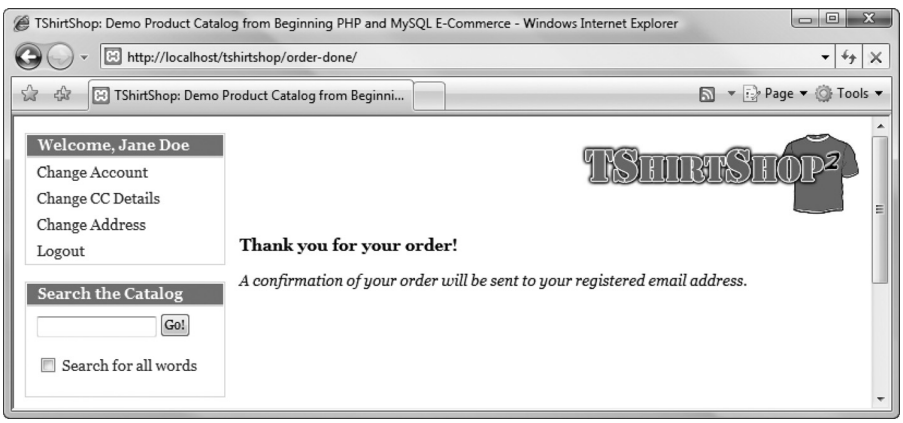


Рис. 19.8. Подтверждение размещения заказа

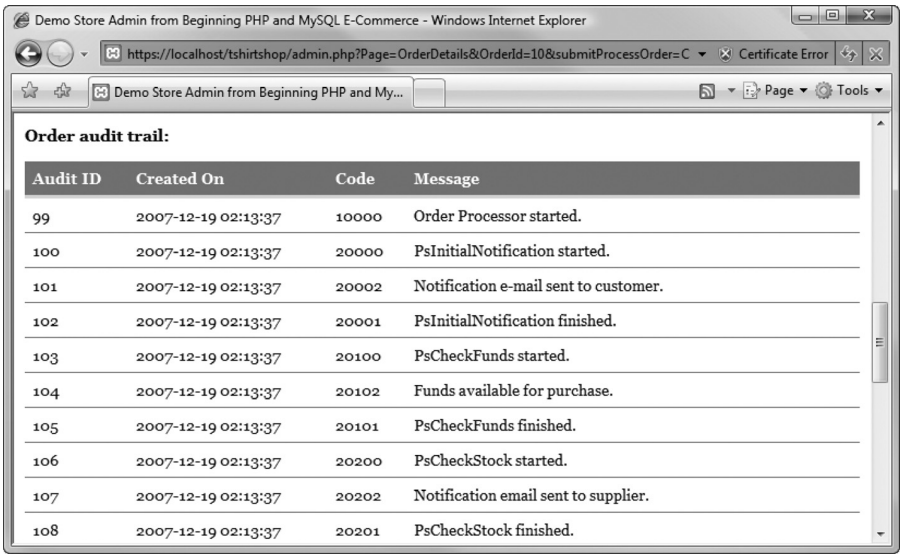


Рис. 19.9. Записи аудита обработки заказа

Как это работает: администрирование заказа

Метод `init()` класса `AdminOrderDetails` запускает следующий этап конвейера после щелчка на кнопке `Process`. Будет ли эта кнопка отображаться на странице, определяется значением элемента `$mProcessButtonText`. Это значение равно Confirm Stock, если текущий этап конвейера — третий (ожидание подтверждения наличия запасов), или Confirm Shipment, если текущий этап — шестой (ожидание подтверждения отправки заказа). Если текущий этап — не третий или шестой, значит, обработка заказа не требует наличия этой кнопки на странице и кнопка отображаться не будет. Проверить, как обрабатывался заказ, администратор всегда сможет по записям аудита.

Все, что осталось сделать теперь, — убедиться, что все работает как полагается. Чтобы сделать это, разместите заказ и с помощью страницы администрирования проследите, как выполняется его обработка. Сначала необходимо подтвердить наличие у поставщика заказанных товаров, как на рис. 19.7.

Щелкните на кнопке Confirm Stock for Order, и обработка заказа продолжится. На месте кнопки, на которой вы щелкнули, появится кнопка Confirm Shipment, а в таблицу с записями аудита добавятся новые записи о завершенных этапах.

Щелкните на кнопке Confirm Shipment, и обработка заказа будет закончена, о чем будут свидетельствовать новые записи в таблице аудита.

Резюме

В этой главе мы совершили огромный шаг к завершению работы над проектом TShirtShop. Теперь у нас есть полностью работающий конвейер обработки заказов с поддержкой аудита.

Если быть более точными, то в этой главе мы сделали следующее.

- Создали классы этапов конвейера
- Внесли в проект изменения, обеспечившие работу этих классов

- Добавили в базу данных механизмы для доступа к данным аудита и для хранения дополнительных данных в таблице `orders`

Все, что осталось сделать для завершения работы над проектом TShirtShop, — добавить в него механизм для работы с кредитными картами. Именно этим мы и займемся в следующей главе.

Глава 20

Работа с кредитными картами

Последний элемент, который нам нужно добавить в проект TShirtShop, прежде чем он будет готов, — это механизм для работы с кредитными картами. Из этой главы вы узнаете, как встроить такой механизм в конвейер, созданный в предыдущих двух главах.

Для начала рассмотрим несколько возможных вариантов работы с кредитными картами, а также организации, предоставляющие необходимые для этого услуги. Сначала в качестве примеров мы рассмотрим две организации, DataCash и Authorize.net, затем обсудим специфику их API, обеспечивающих выполнение операций с кредитными картами, после чего создадим библиотеку классов, которая поможет использовать эти API в нашем приложении.

И наконец мы интегрируем эту библиотеку в TShirtShop и задействуем ее в конвейере обработки заказов.

Основы транзакций с кредитными картами

Для всех транзакций в банках и других финансовых организациях используются защищенные сети, основанные на протоколе X.25 вместо используемых в Интернете протоколов TCP/IP. Нам не нужно разбираться в тонкостях X.25 — достаточно знать, что он *не* совместим с TCP/IP. Соответственно, сети X.25 существуют независимо от Интернета, и хотя к ним можно получить прямой доступ, особого смысла в этом нет: это не только требует выполнения бюрократических процедур, но и довольно сложная задача с технической точки зрения. Прежде всего нужно доказать владельцам сети, к которой мы захотим подключиться, что мы не злоумышленники. Разумеется, владельцы сети не выдают разрешения на подключение кому попало — ваша компания должна соответствовать целому ряду требований, чтобы получить разрешение.

Альтернатива прямому доступу — доступ через шлюз провайдера. Это позволит нам выполнить свою часть транзакции с кредитной картой через Интернет (с помощью защищенного протокола), а шлюз обеспечит выполнение оставшейся части транзакции. Хотя такой подход, скорее всего, не будет бесплатным, он все равно обойдется намного дешевле, чем организация собственного подключения к сети X.25. Вероятно, он будет даже дешевле, чем использование сторонней службы оплаты, поскольку нам потребуется только минимум функциональности (у нас уже есть свой конвейер для обработки заказов). Поэтому нам не понадобится, например,

функциональность аудита заказов, которую предоставляют компании наподобие PayPal, — мы уже реализовали эту функциональность сами.

Работа со шлюзами оплаты

Чтобы работать с провайдером шлюза, нужен счет в банке, который сотрудничает с этим провайдером. Такой счет можно открыть в большинстве банков; банк выдаст идентификатор, который вы сможете использовать, чтобы заключить договор с провайдером.

Открыть счет и получить идентификатор несложно. Найти провайдера, который согласится с вами работать на приемлемых условиях, — на порядок сложнее. Существуют сотни провайдеров, которые с радостью предоставят вам свои услуги в обмен на солидную долю в ваших доходах. Чтобы найти таких провайдеров, попробуйте задать в Google строку поиска `credit card gateway`. К сожалению, на сайтах, которые вы найдете таким способом, чаще всего есть только реклама и раздел, в котором вы сможете заполнить анкету. Если провайдеру понравится то, что вы напишете в анкете, он с вами свяжется для более серьезного диалога.

В целом условия, предлагаемые большинством провайдеров, примерно одинаковы. Различия между ними в том, с какими банками они работают (среди них должен быть банк, в котором вы открыли счет), поддерживаемые валюты и, разумеется, цены.

В табл. 20.1 перечислено несколько провайдеров. В этой главе мы будем работать с двумя из них — DataCash и Authorize.net.

Таблица 20.1. Провайдеры шлюзов

Провайдеры в США	URL	Провайдеры в Великобритании	URL
Authorize.net	`http://www.authorize.net`	Arcot	`http://www.arcot.com`
First Data	`http://www.firstdata.com`	WorldPay	`http://www.worldpay.com`
Cardservice	`http://cardservice.com`	DataCash	`http://www.datacash.com`
ICVerify	`http://www.icverify.com`		

DataCash и Authorize.net

В этой главе мы продемонстрируем работу с провайдерами шлюзов для кредитных карт на примерах двух организаций — DataCash и Authorize.net.

DataCash — это организация, в основном работающая с клиентами из Великобритании. Для работы с ней вам понадобится счет в одном из британских банков. Но пока что можете об этом не волноваться — DataCash позволяет легко создать тестовую учетную запись, для которой банковский счет не нужен вообще.

Authorize.net, как указано на сайте этой организации, `http://www.authorize.net`, "позволяет выполнять транзакции с кредитными картами где угодно и когда угодно". Другими словами, Authorize.net также предоставляет все нужные нам возможности для работы с кредитными картами в приложении TShirtShop.

Важно помнить, что методики, которые мы рассмотрим в этой главе, с минимальными изменениями будут работать с любым провайдером шлюзов кредитных карт. Детали могут слегка отличаться, но общие принципы будут одинаковыми.

Как вы убедитесь позже в этой главе, Authorize.net и DataCash позволяют выполнять тестовые транзакции с помощью специальных номеров кредитных карт (такие номера выдаются этими организациями для специальных тестовых целей). Транзакции с этими номерами можно принимать и отменять, и при этом не будут

выполняться никакие реальные финансовые операции. Это именно то, что требуется нам для работы с TShirtShop, — вряд ли вы согласились бы использовать в тестах собственную кредитную карту.

Основы транзакций с кредитными картами

Независимо от того, с каким провайдером мы будем работать, основы транзакций останутся неизменными. Прежде всего, запомните, что транзакции, которые мы будем выполнять, относятся к категории CNP (Card Not Present — без предъявления карты). Это значит, что данные транзакции не требуют предъявления кредитной карты и сверки подписи на ней. Вероятно, вы уже участвовали в таких транзакциях в роли клиента, покупая что-нибудь через Интернет, хотя, возможно, и не сталкивались с аббревиатурой CNP. Ну, теперь вы знаете, что она означает.

Провайдеры часто предлагают дополнительные услуги по работе с кредитными картами — проверку адреса владельца карты, защиту от мошенничества и еще много чего. Каждая из подобных услуг добавляет еще один уровень сложности в работе с кредитными картами, так что мы не будем использовать эти возможности. Не забывайте, что в этой главе описываются только основы работы с кредитными картами. Если вы захотите реализовать более развитой механизм для работы с ними, это ваше право. Чем больше будет оборот вашего магазина, тем больше вероятность, что он станет мишенью для атак злоумышленников, а большинство дополнительных услуг провайдеров как раз и предназначены для защиты от этих атак. Если вам нужна более подробная информация, служба поддержки провайдера с радостью ее вам предоставит.

Нас интересуют несколько типов операций с кредитными картами.

- *Авторизация.* Проверка наличия необходимой суммы на счету и вычитание этой суммы.
- *Преавторизация.* Проверка наличия необходимой суммы на счету и ее резервирование; вычитание суммы при этом не производится.
- *Завершение транзакции.* Завершение преавторизованной транзакции, вычитание ранее зарезервированной суммы.
- *Возврат денег.* Отмена ранее завершенной транзакции или просто перечисление денег на счет.

Детали транзакций могут различаться у разных провайдеров, но общий смысл операций останется тем же.

В этой главе мы будем использовать операции преавторизации и завершения транзакции, т.е. не будем снимать деньги с карты до отправки товара покупателю. Такой подход хорошо совместим со структурой конвейера, который мы создали ранее в этой главе.

Работа с DataCash

Рассмотрев основы транзакций, давайте рассмотрим детали, специфичные для провайдера DataCash. Первое, что нам понадобится, — создать пробную учетную запись на сайте DataCash. Вот как это делается.

1. Откройте в браузере сайт `http://www.datacash.com/`.
2. На главной странице щелкните на ссылке Developers Area, а на открывшейся странице — на ссылке Get a Test Account.

3. Заполните открывшуюся форму.
4. По адресу электронной почты, который вы укажете, должно прийти письмо с именем учетной записи и паролем, а также дополнительной информацией, которая вам понадобится для работы с DataCash.

ЗАМЕЧАНИЕ

Получив тестовую учетную запись, загрузите руководство разработчика по работе с DataCash — `http://testserver.datacash.com/software/DevelopersGuide.pdf`. Это официальный документ, и информацию в нем можно считать достоверной.

DataCash предлагает API для PHP-приложений, состоящий из нескольких классов. Этот API мы и будем использовать, чтобы работать со службами DataCash. Классы, из которых состоит API, генерируют XML-сообщения и отправляют их на сервер DataCash. В этой главе мы будем создавать эти XML-сообщения непосредственно, чтобы лучше уяснить себе, как работает система.

При работе с DataCash придется уделить много внимания XML — мы будем составлять XML-сообщения, отправлять их DataCash и получать ответы на них также в виде XML-сообщений. Сейчас мы кратко рассмотрим XML-документы, с которыми нам придется работать, а также коммуникационный протокол, реализующий следующие этапы транзакций с кредитными картами.

- Запрос преаутентификации
- Ответ на запрос преаутентификации
- Запрос завершения транзакции
- Ответ на запрос завершения транзакции

Все эти этапы мы интегрируем в конвейер обработки заказов TShirtShop. Выполняя задания из этой главы, не забывайте при необходимости обращаться к официальному руководству разработчика — `http://testserver.datacash.com/software/DevelopersGuide.pdf`.

Запрос преаутентификации

Отправляя на сервер DataCash запрос преаутентификации, нужно включить в этот запрос следующие данные.

- Имя пользователя DataCash (также известное как имя клиента)
- Пароль DataCash
- Уникальный номер транзакции
- Сумма, которую нужно зарезервировать на счету
- Валюта, в которой выполняется транзакция (доллары США, фунты стерлингов и т.д.)
- Тип транзакции (код `pre` для преаутентификации и код `fulfill` для завершения)
- Номер кредитной карты
- Срок действия кредитной карты
- Дата выдачи кредитной карты (если эта информация доступна для используемой кредитной карты)
- Номер выпуска кредитной карты (если эта информация доступна для используемой кредитной карты)

Уникальный номер транзакции должен иметь длину от 6 до 12 десятичных разрядов и используется для обозначения транзакции в заказе. Поскольку мы не можем использовать в качестве номера транзакции короткое число, применить в этом качестве идентификатор заказа не удастся. Но мы можем, например, прибавить к номеру заказа определенное число и получить пригодный к использованию номер транзакции. Повторно использовать номер нельзя — нужно быть уверенным в том, что однажды выполненная транзакция не будет выполнена повторно, иначе, например, с карты покупателя может быть дважды списана сумма оплаты. Однако это требование означает, что, если транзакция не пройдет, нужно будет сгенерировать новый заказ и новый идентификатор транзакции. Впрочем, это не представляет особых проблем.

XML-запрос оформляется следующим образом (в листинге отправляемые данные выделены полужирным шрифтом).

```
<?xml version="1.0" encoding="UTF-8"?>
<Request>
  <Authentication>
    <password>Пароль</password>
    <client>Имя пользователя</client>
  </Authentication>
  <Transaction>
    <TxnDetails>
      <merchantreference>Уникальный номер транзакции</merchantreference>
      <amount currency='Валюта'>Сумма</amount>
    </TxnDetails>
    <CardTxn>
      <method>pre</method>
      <Card>
        <pan>Номер кредитной карты</pan>
        <expirydate>Срок действия кредитной карты</expirydate>
      </Card>
    </CardTxn>
  </Transaction>
</Request>
```

Ответ на запрос преаутентификации

В ответе на запрос преаутентификации содержатся следующие данные.

- Код состояния, указывающий на результат обработки запроса: `1` — если обработка прошла успешно или один из нескольких других кодов — при возникновении различных сбоев. Полный список этих кодов можно найти в документации к серверу DataCash (страница `http://testserver.datacash.com/software/returncodes.shtml`).
- Поле с описанием причин сбоя — это, по сути, просто текст на английском языке. Если результат равен `1`, это слово `ACCEPTED`.
- Код аутентификации и номер транзакции, которые будут использоваться на этапе завершения транзакции (мы рассмотрим его позже).
- Время, затраченное на обработку транзакции.
- Режим транзакции (для всех тестовых транзакций — `TEST`).
- Подтверждение типа использованной кредитной карты.
- Подтверждение страны, в которой была выдана использованная кредитная карта.

- Код авторизации, используемый банком (только в справочных целях).

В виде XML-документа все это представлено так.

```
<?xml version="1.0" encoding="utf-8"?>
<Response>
  <status>Код состояния</status>
  <reason>Причина</reason>
  <merchantreference>Код аутентификации</merchantreference>
  <datacash_reference>Номер</datacash_reference>
  <time>Время</time>
  <mode>TEST</mode>
  <CardTxn>
    <card_scheme>Тип карты</card_scheme>
    <country>Страна</country>
    <issuer>Банк, выдавший карту</issuer>
    <authcode>Код авторизации банка</authcode>
  </CardTxn>
</Response>
```

Запрос завершения

В запросе завершения нужно отправить следующую информацию.

- Имя пользователя DataCash (также известное как имя клиента)
- Пароль DataCash
- Уникальный номер транзакции
- Код аутентификации, полученный ранее
- Номер транзакции для завершения, полученный ранее

Можно включить в запрос и другую информацию, например подтверждение суммы, которую нужно списать с кредитной карты, но это необязательно.

XML-запрос завершения выглядит так.

```
<?xml version="1.0" encoding="UTF-8"?>
<Request>
  <Authentication>
    <password>Пароль</password>
    <client>Имя пользователя</client>
  </Authentication>
  <Transaction>
    <HistoricTxn>
      <reference>Номер</reference>
      <authcode>Код аутентификации</authcode>
      <method>fulfill</method>
    </HistoricTxn>
  </Transaction>
</Request>
```

Ответ на запрос завершения

В ответе на запрос подтверждения содержатся следующие данные.

- Код состояния, указывающий на результат обработки запроса: `1` — если обработка прошла успешно, или один из нескольких других кодов — при возникновении различных сбоев. Полный список этих кодов можно найти в документации к серверу DataCash (страница `http://testserver.datacash.com/software/returncodes.shtml`).

- Поле с описанием причин сбоя — это, по сути, просто текст на английском языке. Если результат равен 1, это слово FULFILLED OK.
- Коды для использования в транзакциях (два экземпляра).
- Время обработки транзакции.
- Режим транзакции (для тестовых учетных записей TEST).

В XML-формате ответ на запрос завершения выглядит так.

```
<?xml version="1.0" encoding="utf-8"?>
<Response>
  <status>Код состояния</status>
  <reason>Причина</reason>
  <merchantreference>Код</merchantreference>
  <datacash_reference>Код</datacash_reference>
  <time>Время</time>
  <mode>TEST</mode>
</Response>
```

Обмен XML-данными с DataCash

Поскольку XML-данные, которые нужно отправлять на сервер DataCash, имеют относительно простую структуру, мы будем сразу формировать их в виде строки. Для этого не понадобятся механизмы PHP 5 для работы с XML, но мы воспользуемся расширением PHP 5 SimpleXML для чтения XML.

Хотя возможности SimpleXML не столь обширны, как возможности DOMDocument, они позволяют легко работать с XML-данными, преобразуя их в структуру, элементы которой можно перебирать. Мы уже использовали SimpleXML в главе 6.

ЗАМЕЧАНИЕ

В коде, который будет взаимодействовать с DataCash, мы применим библиотеку cURL (http://curl.haxx.se/). Раскомментируйте следующую строку в файле php.ini, удалив точку с запятой в ее начале: extension=php_curl.dll. После этого перезапустите Apache.

Файл php.ini находится в папке C:\xampp\apache\bin под управлением Windows или в папке /opt/lampp/etc под управлением Linux.

Более подробная информация о библиотеке cURL, учебный курс по работе с ней, доступна по адресу http://www.zend.com/pecl/tutorials/curl.php. Официальная документация по поддержке cURL в PHP доступна по адресу http://www.php.net/curl.

Упражнение: взаимодействие с DataCash

1. Создайте в папке business файл datacash_request.php и поместите в него следующий код.

```
<?php
class DataCashRequest
{
  // URL сервера DataCash
  private $_mUrl;

  // Для хранения XML-документа, который нужно отправить на сервер DataCash
  private $_mXml;

  // Конструктор инициализирует объект URL сервера DataCash
```

```
public function __construct($url)
{
  // URL сервера Datacash
  $this->_mUrl = $url;
}

/* Составляем XML-структуру запроса преаутентификации для
   сервера DataCash */
public function MakeXmlPre($dataCashClient, $dataCashPassword,
                           $merchantReference, $amount, $currency,
                           $cardNumber, $expiryDate,
                           $startDate = '', $issueNumber = '')
{
  $this->_mXml =
    "<?xml version=\"1.0\" encoding=\"UTF-8\"\x3F>
     <Request>
       <Authentication>
         <password>$dataCashPassword</password>
         <client>$dataCashClient</client>
       </Authentication>
       <Transaction>
         <TxnDetails>
           <merchantreference>$merchantReference</merchantreference>
           <amount currency=\"$currency\">$amount</amount>
         </TxnDetails>
         <CardTxn>
           <method>pre</method>
           <Card>
             <pan>$cardNumber</pan>
             <expirydate>$expiryDate</expirydate>
             <startdate>$startDate</startdate>
             <issuenumber>$issueNumber</issuenumber>
           </Card>
         </CardTxn>
       </Transaction>
     </Request>";
}

// Составляем XML-структуру для запроса завершения на сервер DataCash
public function MakeXmlFulfill($dataCashClient, $dataCashPassword,
                               $authCode, $reference)
{
  $this->_mXml =
    "<?xml version=\"1.0\" encoding=\"UTF-8\"\x3F>
     <Request>
       <Authentication>
         <password>$dataCashPassword</password>
         <client>$dataCashClient</client>
       </Authentication>
       <Transaction>
         <HistoricTxn>
           <reference>$reference</reference>
           <authcode>$authCode</authcode>
           <method>fulfill</method>
         </HistoricTxn>
       </Transaction>
     </Request>";
}

// Получаем XML-текст
public function GetRequest()
```

```
  {
    return $this->_mXml;
  }

  // Отправляем POST-запрос HTTP на сервер DataCash с помощью cURL
  public function GetResponse()
  {
    // Инициализируем сеанс cURL
    $ch = curl_init();

    // Готовимся к POST-запросу HTTP
    curl_setopt($ch, CURLOPT_POST, 1);

    // Подготавливаем XML-документ к отправке с помощью POST
    curl_setopt($ch, CURLOPT_POSTFIELDS, $this->_mXml);

    // Задаем URL, по которому отправляется с помощью POST наша XML-
структура
    curl_setopt($ch, CURLOPT_URL, $this->_mUrl);

    /* Не проверяем имя сертификата SSL */
    curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);

    // Запрещаем cURL проверять сертификат
    curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);

    /* Мы хотим, чтобы cURL возвращала получаемые данные, не выводя их*/
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

    // Выполняем сеанс cURL
    $result = curl_exec($ch);

    // Закрываем сеанс cURL
    curl_close ($ch);

    // Возвращаем ответ
    return $result;
  }
}
?>
```

2. Добавьте определения констант, необходимых для работы с DataCash, в конец файла `include/config.php`.

```
// Определения констант для работы с DataCash
define('DATACASH_URL', 'https://testserver.datacash.com/Transaction');
define('DATACASH_CLIENT', 'номер вашей учетной записи');
define('DATACASH_PASSWORD', пароль вашей учетной записи');
```

ЗАМЕЧАНИЕ

Не забудьте вставить в определения констант данные из вашей учетной записи DataCash.

3. Создайте файл `test_datacash.php` в корневой папке проекта (папке `tshirtshop`) и поместите в него следующий код.

```
<?php
session_start();

if (empty ($_GET['step']))
{
```

```
  require_once 'include/config.php';
  require_once BUSINESS_DIR . 'datacash_request.php';

  $request = new DataCashRequest(DATACASH_URL);
  $request->MakeXmlPre(DATACASH_CLIENT, DATACASH_PASSWORD,
                       8880000 + rand(0, 10000), 49.99, 'GBP',
                       '3528000000000007', '11/09');

  $request_xml = $request->GetRequest();
  $_SESSION['pre_request'] = $request_xml;

  $response_xml = $request->GetResponse();
  $_SESSION['pre_response'] = $response_xml;

  $xml = simplexml_load_string($response_xml);
  $request->MakeXmlFulfill(DATACASH_CLIENT, DATACASH_PASSWORD,
                           $xml->merchantreference,
                           $xml->datacash_reference);

  $response_xml = $request->GetResponse();
  $_SESSION['fulfill_response'] = $response_xml;
}
else
{
  header('Content-type: text/xml');

  switch ($_GET['step'])
  {
    case 1:
      print $_SESSION['pre_request'];
      break;
    case 2:
      print $_SESSION['pre_response'];
      break;
    case 3:
      print $_SESSION['fulfill_response'];
      break;
    }
    exit();
  }
  ?>
  <frameset cols="33%, 33%, 33%">
    <frame src="test_datacash.php?step=1">
    <frame src="test_datacash.php?step=2">
    <frame src="test_datacash.php?step=3">
  </frameset>
```

4. Откройте файл `test_datacash.php` в окне браузера и посмотрите на результат. Если вы используете Opera, он должен быть похож на представленный на рис. 20.1, поскольку Opera отображает только содержимое XML-элементов. Другие браузеры должны отобразить корректно отформатированные XML-документы.
5. Откройте в браузере страницу `http://testserver.datacash.com/reporting2/` и просмотрите журнал транзакций своей учетной записи (учтите, что этот журнал обновляется с определенными интервалами, и на появление в нем новых записей может потребоваться некоторое время). Пример записей в журнале показан на рис. 20.2.

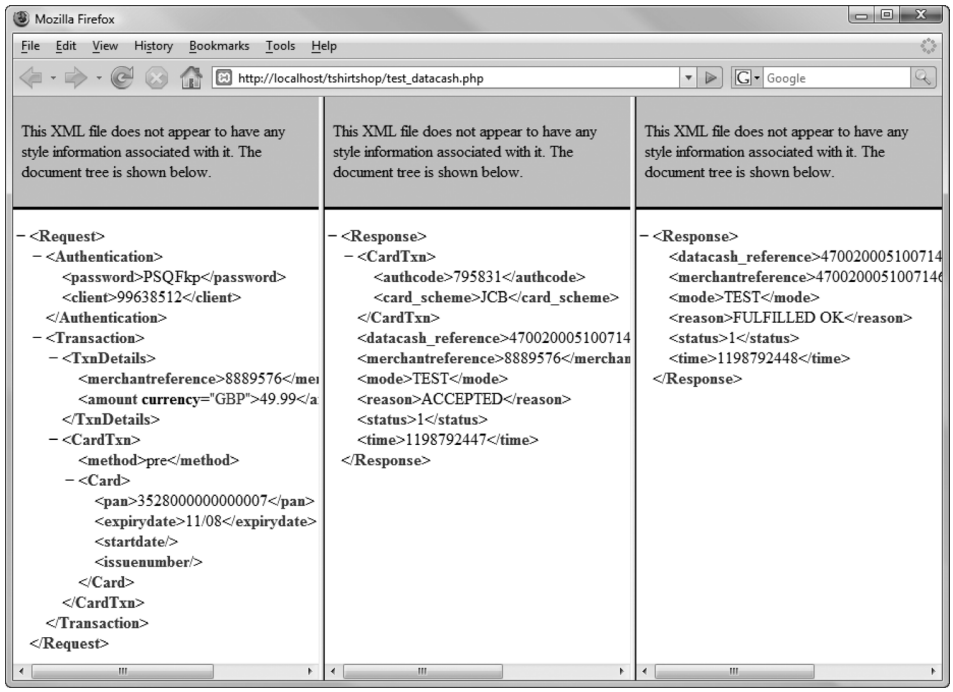


Рис. 20.1. Генерируемые XML-запросы для DataCash

Как это работает: код взаимодействия с DataCash

Класс `DataCashRequest` довольно прост. Сначала конструктор задает HTTPS-адрес, по которому нужно отправлять запросы.

```
// Конструктор инициализирует объект URL сервера DataCash
public function __construct($url)
{
  // URL сервера Datacash
  $this->_mUrl = $url;
}
```

Чтобы отправить запрос преаутентификации, нужно вызвать метод `MakeXmlPre()` для формирования XML-структуры. Некоторые элементы в этой структуре необязательны (например, `startdate` или `issuenumber`, которые получат значения автоматически, если мы их не зададим, — просмотрите код метода `MakeXmlPre()`), но большинство элементов необходимо.

ЗАМЕЧАНИЕ

Если вы хотите узнать, какие элементы обязательны, а какие — нет для каждого из типов запросов, обратитесь к документации по XML API, доступной на сайте DataCash.

Следующий тип запроса, который нам нужно отправлять, — запрос завершения транзакции. Для этого типа запросов XML-структуру готовит метод `MakeXmlFulfill()`.

Кроме этих двух методов, в классе есть метод `GetRequest()`, который возвращает последний XML-документ, сгенерированный методами `MakeXmlPre()` и `MakeXmlFulfill()`.

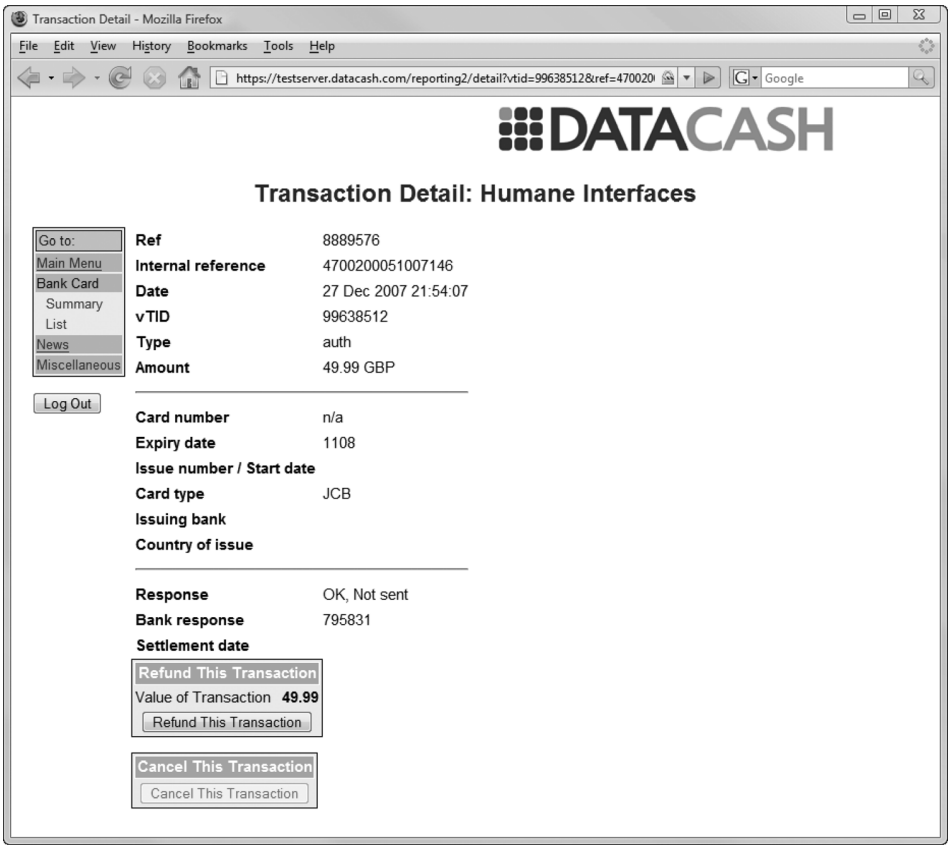


Рис. 20.2. Записи в журнале транзакций DataCash

```
// Получаем XML-текст
public function GetRequest()
{
  return $this->_mXml;
}
```

И наконец метод `GetResponse()` непосредственно отправляет сгенерированный запрос на сервер и возвращает полученный с сервера ответ. Рассмотрим этот метод подробнее.

Сначала он инициализирует сеанс cURL и задает использование метода POST для отправки данных.

```
// Отправляем POST-запрос HTTP на сервер DataCash с помощью cURL
public function GetResponse()
{
  // Инициализируем сеанс cURL
  $ch = curl_init();

  // Готовимся к POST-запросу HTTP
  curl_setopt($ch, CURLOPT_POST, 1);

  // Подготавливаем XML-документ к отправке с помощью POST
  curl_setopt($ch, CURLOPT_POSTFIELDS, $this->_mXml);

  // Задаем URL, по которому отправляется с помощью POST XML-структура
  curl_setopt($ch, CURLOPT_URL, $this->_mUrl);
```

```
  /* Не проверяем имя сертификата SSL */
  curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);

  // Запрещаем cURL проверять сертификат
  curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
```

Чтобы вернуть получаемый с сервера ответ в переменную PHP, устанавливаем параметр `CURLOPT_RETURNTRANSFER` равным 1, а затем отправляем запрос и закрываем сеанс cURL.

```
  /* Мы хотим, чтобы cURL возвращала получаемые данные, не выводя их*/
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  // Выполняем сеанс cURL
  $result = curl_exec($ch);

  // Закрываем сеанс cURL
  curl_close ($ch);

  // Возвращаем ответ
  return $result;
}
```

Файл `test_datacash.php` работает следующим образом: когда мы открываем его в браузере, скрипт отправляет запрос преаутентификации и запрос завершения, а затем сохраняет XML-тексты запросов и ответов на них в сеансовой переменной.

```
session_start();

if (empty ($_GET['step']))
{
  require_once 'include/config.php';
  require_once BUSINESS_DIR . 'datacash_request.php';

  $request = new DataCashRequest(DATACASH_URL);
  $request->MakeXmlPre(DATACASH_CLIENT, DATACASH_PASSWORD,
                       8880000 + rand(0, 10000), 49.99, 'GBP',
                       '3528000000000007', '11/09');

  $request_xml = $request->GetRequest();
  $_SESSION['pre_request'] = $request_xml;

  $response_xml = $request->GetResponse();
  $_SESSION['pre_response'] = $response_xml;

  $xml = simplexml_load_string($response_xml);
  $request->MakeXmlFulfill(DATACASH_CLIENT, DATACASH_PASSWORD,
                           $xml->merchantreference,
                           $xml->datacash_reference);

  $response_xml = $request->GetResponse();
  $_SESSION['fulfill_response'] = $response_xml;
}
```

Страница `test_datacash.php` будет загружаться трижды, поскольку мы хотим заполнить данными три фрейма.

```
<frameset cols="33%, 33%, 33%">
  <frame src="test_datacash.php?step=1">
  <frame src="test_datacash.php?step=2">
  <frame src="test_datacash.php?step=3">
</frameset>
```

В зависимости от значения переменной step код отображает в фрейме ранее сохраненные в сеансовой переменной XML-данные. Если это значение равно 1, отображается XML-текст запроса преаутентификации, если 2 — текст ответа на этот запрос, если 3 — текст запроса завершения.

```
{
  header('Content-type: text/xml');

  switch ($_GET['step'])
  {
    case 1:
      print $_SESSION['pre_request'];
      break;
    case 2:
      print $_SESSION['pre_response'];
      break;
    case 3:
      print $_SESSION['fulfill_response'];
      break;
    }
    exit();
  }
?>
```

Интеграция DataCash в TShirtShop

Итак, мы создали класс, выполняющий транзакции с кредитными картами. Все, что осталось сделать, — интегрировать его функциональность в конвейер обработки заказов, который мы создали в предыдущих главах. Чтобы полностью интегрировать функциональность для работы с DataCash в TShirtShop, нужно обновить классы `PsCheckFunds` и `PsTakePayments`.

Необходимо модифицировать классы этапов конвейера, выполняющих операции с кредитными картами. Вся инфраструктура для хранения кодов аутентификации у нас уже есть — она доступна в методе `OrderProcessor::SetOrderAuthCodeAndReference()`.

Упражнение: интеграция DataCash в TShirtShop

1. Замените старый код в файле `business/ps_check_funds.php` кодом, который будет работать с DataCash.

```
<?php
class PsCheckFunds implements IPipelineSection
{
  public function Process($processor)
  {
    // Аудит
    $processor->CreateAudit('PsCheckFunds started.', 20100);

    $order_total_cost = $processor->mOrderInfo['total_amount'];
    $order_total_cost += $processor->mOrderInfo['shipping_cost'];
    $order_total_cost +=
      round((float)$order_total_cost *
            (float)$processor->mOrderInfo['tax_percentage'], 2) / 100.00;

    $request = new DataCashRequest(DATACASH_URL);
    $request->MakeXmlPre(DATACASH_CLIENT, DATACASH_PASSWORD,
      $processor->mOrderInfo['order_id'] + 1000006,
      $order_total_cost, 'GBP',
      $processor->mCustomerInfo['credit_card']->CardNumber,
      $processor->mCustomerInfo['credit_card']->ExpiryDate,
      $processor->mCustomerInfo['credit_card']->IssueDate,
```

```
      $processor->mCustomerInfo['credit_card']->IssueNumber);

    $responseXml = $request->GetResponse();
    $xml = simplexml_load_string($responseXml);

    if ($xml->status == 1)
    {
      $processor->SetAuthCodeAndReference(
        $xml->merchantreference, $xml->datacash_reference);

      // Аудит
      $processor->CreateAudit('Funds available for purchase.', 20102);

      // Обновление статуса заказа
      $processor->UpdateOrderStatus(2);

      // Продолжаем обработку
      $processor->mContinueNow = true;
    }
    else
    {
      // Аудит
      $processor->CreateAudit('Funds not available for purchase.', 20103);

      throw new Exception('Credit card check funds failed for order ' .
                          $processor->mOrderInfo['order_id'] . "\n\n" .
                          'Data exchanged:' . "\n" .
                          $request->GetResponse() . "\n" . $responseXml);
    }

    // Аудит
    $processor->CreateAudit('PsCheckFunds finished.', 20101);
  }
}
?>
```

2. Замените код в файле `business/ps_take_payment.php` следующим.

```
<?php
class PsTakePayment implements IPipelineSection
{
  public function Process($processor)
  {
    // Аудит
    $processor->CreateAudit('PsTakePayment started.', 20400);

    $request = new DataCashRequest(DATACASH_URL);
    $request->MakeXmlFulFill(DATACASH_CLIENT, DATACASH_PASSWORD,
                             $processor->mOrderInfo['auth_code'],
                             $processor->mOrderInfo['reference']);

    $responseXml = $request->GetResponse();
    $xml = simplexml_load_string($responseXml);

    if ($xml->status == 1)
    {
      // Аудит
      $processor->CreateAudit(
        'Funds deducted from customer credit card account.', 20402);

      // Обновляем статус заказа
      $processor->UpdateOrderStatus(5);

      // Продолжаем обработку
```

```
      $processor->mContinueNow = true;
    }
    else
    {
      // Аудит
      $processor->CreateAudit('Could not deduct funds from credit card.',
                              20403);

      throw new Exception('Credit card take payment failed for order ' .
                          $processor->mOrderInfo['order_id'] . "\n\n" .
                          'Data exchanged:' . "\n" .
                          $request->GetResponse() . "\n" . $responseXml);
    }

    // Аудит
    $processor->CreateAudit('PsTakePayment finished.', 20401);
  }
}
?>
```

3. Добавьте ссылку на файл `business/datacash_request.php` в файл `index.php`, как показано ниже.

```
require_once BUSINESS_DIR . 'ps_check_funds.php';
require_once BUSINESS_DIR . 'ps_check_stock.php';
require_once BUSINESS_DIR . 'datacash_request.php';
```

4. Добавьте ссылку на файл `business/datacash_request.php` в файл `admin.php`, как показано ниже.

```
require_once BUSINESS_DIR . 'ps_ship_ok.php';
require_once BUSINESS_DIR . 'ps_final_notification.php';
require_once BUSINESS_DIR . 'datacash_request.php';
```

Тестирование интеграции DataCash в TShirtShop

Интегрировав поддержку DataCash в приложение, следует провести хотя бы простейшие тесты. Для этого можно создать тестового пользователя с параметрами кредитной карты, которые нам предоставит DataCash. Как уже говорилось, DataCash предоставляет эти параметры специально для тестирования. Пример результатов тестирования показан в табл. 20.2. Полный список доступен в разделе Developers Area веб-сайта DataCash, на странице Magic Card Numbers.

Таблица 20.2. Тестовые номера кредитных карт DataCash

Тип карты	Номер карты	Возвращаемое значение	Описание	Пример сообщения
Switch	4936000000000000001	1	Авторизуется со случайным кодом авторизации	AUTH CODE ??????
	4936000000000000019	7	Отклоняет транзакцию	DECLINED
	6333000000000005	1	Авторизуется со случайным кодом авторизации	AUTH CODE ??????
	6333000000000013	7	Отклоняет транзакцию	DECLINED
	6333000000123450	1	Авторизуется со случайным кодом авторизации	AUTH CODE ??????
Visa	4242424242424242	7	Отклоняет транзакцию	DECLINED

Окончание табл. 20.2

Тип карты	Номер карты	Возвращаемое значение	Описание	Пример сообщения
	4444333322221111	1	Авторизуется со случайным кодом авторизации	AUTH CODE ??????
	4546389010000131	1	Авторизуется со случайным кодом авторизации	AUTH CODE ??????

Теперь можете поэкспериментировать с нашим приложением, размещая заказы, проверяя сообщения электронной почты, отправляемые сайтом, просматривая журналы ошибок и т.д.

Запуск в рабочем режиме

Чтобы задействовать вместо тестовой учетной записи DataCash рабочую, достаточно заменить параметры учетной записи в файле `include/config.php`. Открыв счет в банке, сообщите об этом DataCash, чтобы получить рабочую учетную запись. Вам понадобятся имя этой учетной записи и пароль к ней. Кроме того, необходимо поменять URL сервера DataCash, которому вы отправляете данные, — это должен быть рабочий сервер, а не тестовый. Внеся изменения в код приложения, можно переместить его на рабочую Интернет-площадку.

Работа с Authorize.net

Чтобы получить тестовую учетную запись Authorize.net, обратитесь по адресу `http://developer.authorize.net/testaccount/`. Основная страница с информацией для разработчиков расположена по адресу `http://developer.authorize.net/`.

Обмен данными с сервером Authorize.net организован не так, как в DataCash. Вместо XML-файлов здесь используются строки, состоящие из пар "имя–значение", разделенных амперсандами (&). В целом этот синтаксис похож на используемый в URL.

Authorize.net возвращает результаты транзакций в виде строк, содержащих возвращаемые значения (без имен), разделенные символом, который можно задать, отправляя запрос. В наших примерах мы будем использовать в качестве разделителя вертикальную черту (|). Возвращаемые значения следуют в определенном порядке, и их смысл определяется именно их местами в строке ответа.

ЗАМЕЧАНИЕ

Полную документацию по API Authorize.net можно найти в руководстве *Advanced Integration Method (AIM) Implementation Guide: Card-Not-Present Transactions* по адресу `http://www.authorize.net/support/AIM_guide.pdf`. Дополнительную информацию можно получить из библиотеки документов, доступной по адресу `http://www.authorize.net/resources/documentlibrary/`.

По умолчанию используются транзакции типа `AUTH_CAPTURE`, в которых мы запрашиваем и списываем деньги с кредитной карты одной операцией. Для TShirtShop мы применим два других типа транзакций — `AUTH_ONLY`, которые проверяют наличие необходимой суммы, и `PRIOR_AUTH_CAPTURE`, которые списывают деньги с карты после авторизации с помощью транзакции `AUTH_ONLY`. Транзакции `AUTH_ONLY` мы будем использовать в классе `PsCheckFunds`, а `PRIOR_AUTH_CAPTURE` — в классе `PsTakePayment`.

Чтобы выполнить транзакцию AUTH_ONLY, сначала создадим массив с необходимыми данными.

```
// Авторизация
$transaction =
  array ('x_invoice_num' => '99999',          // Номер квитанции
         'x_amount' => '45.99',               // Сумма
         'x_card_num' => '4007000000027',     // Номер кредитной карты
         'x_exp_date' => '1209',              // Срок действия
         'x_method' => 'CC',                  // Метод оплаты
         'x_type' => 'AUTH_ONLY');            // Тип транзакции
```

Для транзакций PRIOR_AUTH_CAPTURE не нужно указывать всю эту информацию снова — достаточно указать идентификатор транзакции, который возвращается в ответе на запрос AUTH_ONLY.

```
// Получение
$transaction =
  array ('x_ref_trans_id' => $ref_trans_id,   // Идентификатор транзакции
         'x_method' => 'CC',                  // Метод оплаты
         'x_type' => 'PRIOR_AUTH_CAPTURE');   // Тип транзакции
```

Мы будем преобразовывать эти массивы в строки из пар "имя–значение" и передавать эти строки серверу Authorize.net. Ответы будут поступать в виде строк, значения в которых разделяются задаваемым символом. Позже, на рис. 20.3, вы увидите пример ответа на запрос AUTH_ONLY (в левой части окна) и пример ответа на запрос PRIOR_AUTH_CAPTURE (в правой части).

Прежде чем вносить какие-либо изменения в TShirtShop, напишем простой тест для этих транзакций. Выполните следующее упражнение, чтобы написать тестовое приложение для Authorize.net.

Упражнение: создание теста для Authorize.net

1. Создайте в папке business файл authorize_net_request.php и поместите в него следующий код.

```
<?php
class AuthorizeNetRequest
{
  // URL сервера Authorize.net
  private $_mUrl;

  // Для хранения текущего запроса, который нужно
  // отправить на Authorize.net
  private $_mRequest;

  // Конструктор инициализирует класс URL сервера Authorize.net
  public function __construct($url)
  {
    // URL Authorize.net
    $this->_mUrl = $url;
  }

  public function SetRequest($request)
  {
    $this->_mRequest = '';

    $request_init =
      array ('x_login' => AUTHORIZE_NET_LOGIN_ID,
             'x_tran_key' => AUTHORIZE_NET_TRANSACTION_KEY,
             'x_version' => '3.1',
```

```
            'x_test_request' => AUTHORIZE_NET_TEST_REQUEST,
            'x_delim_data' => 'TRUE',
            'x_delim_char' => '|',
            'x_relay_response' => 'FALSE');

    $request = array_merge($request_init, $request);

    foreach($request as $key => $value )
      $this->_mRequest .= $key . '=' . urlencode($value) . '&';
  }

  // Отправляем HTTP POST-запрос на сервер
  // Authorize.net с помощью cURL
  public function GetResponse()
  {
    // Инициализируем сеанс cURL
    $ch = curl_init();

    // Подготавливаем HTTP POST-запрос
    curl_setopt($ch, CURLOPT_POST, 1);

    // Подготавливаем запрос к отправке
    curl_setopt($ch, CURLOPT_POSTFIELDS, rtrim($this->_mRequest, '& '));

    // Задаем URL для отправки запроса
    curl_setopt($ch, CURLOPT_URL, $this->_mUrl);

    /* Не проверять сертификат SSL */
    curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
    curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);

    /* Мы хотим, чтобы cURL прямо возвращала результат,
       не выводя его */
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

    // Выполняем сеанс cURL
    $result = curl_exec($ch);

    // Закрываем сеанс cURL
    curl_close ($ch);

    // Возвращаем ответ
    return $result;
  }
}
?>
```

2. Добавьте следующие строки в конец файла `include/config.php`, подставив в них сведения из своей учетной записи Authorize.net.

```
// Определения констант для Authorize.net
define('AUTHORIZE_NET_URL',
'https://test.authorize.net/gateway/transact.dll');
define('AUTHORIZE_NET_LOGIN_ID', '[Имя вашей учетной записи]');
define('AUTHORIZE_NET_TRANSACTION_KEY', '[Ваш ключ транзакций]');
define('AUTHORIZE_NET_TEST_REQUEST', 'FALSE');
```

3. Добавьте в корневую папку TShirtShop файл `test_authorize_net.php` с таким содержимым.

```
<?php
session_start();

if (empty ($_GET['step']))
{
```

```
  require_once 'include/config.php';
  require_once BUSINESS_DIR . 'authorize_net_request.php';

  $request = new AuthorizeNetRequest(AUTHORIZE_NET_URL);

  // Авторизация
  $transaction =
    array ('x_invoice_num' => '99999',         // Номер квитанции
           'x_amount' => '45.99',              // Сумма
           'x_card_num' => '4007000000027',    // Номер кредитной карты
           'x_exp_date' => '1209',             // Срок действия
           'x_method' => 'CC',                 // Метод оплаты
           'x_type' => 'AUTH_ONLY');           // Тип транзакции

  $request->SetRequest($transaction);
  $auth_only_response = $request->GetResponse();

  $_SESSION['auth_only_response'] = $auth_only_response;

  $auth_only_response = explode('|', $auth_only_response);

  // Считываем ID транзакции, который понадобится для оплаты
  $ref_trans_id = $auth_only_response[6];

  // Оплачиваем
  $transaction =
    array ('x_ref_trans_id' => $ref_trans_id, // Идентификатор транзакции
           'x_method' => 'CC',                // Метод оплаты
           'x_type' => 'PRIOR_AUTH_CAPTURE'); // Тип транзакции

  $request->SetRequest($transaction);
  $prior_auth_capture_response = $request->GetResponse();

  $_SESSION['prior_auth_capture_response'] = $prior_auth_capture_response;
}
else
{
  switch ($_GET['step'])
  {
    case 1:
      print $_SESSION['auth_only_response'];

      break;
    case 2:
      print $_SESSION['prior_auth_capture_response'];

      break;
  }

  exit();
}
?>
<frameset cols="50%, 50%">
  <frame src="test_authorize_net.php?step=1">
  <frame src="test_authorize_net.php?step=2">
</frameset>
```

4. Откройте страницу `test_authorize_net.php` в браузере. Результат должен быть таким, как на рис. 20.3.
5. Откройте сайт Authorize.net и перейдите в раздел Merchant Interface (`https://test.authorize.net`). Результаты только что выполненных транзакций можно просмотреть в разделе Unsettled Transactions на вкладке Search. Пример таких результатов показан на рис. 20.4.

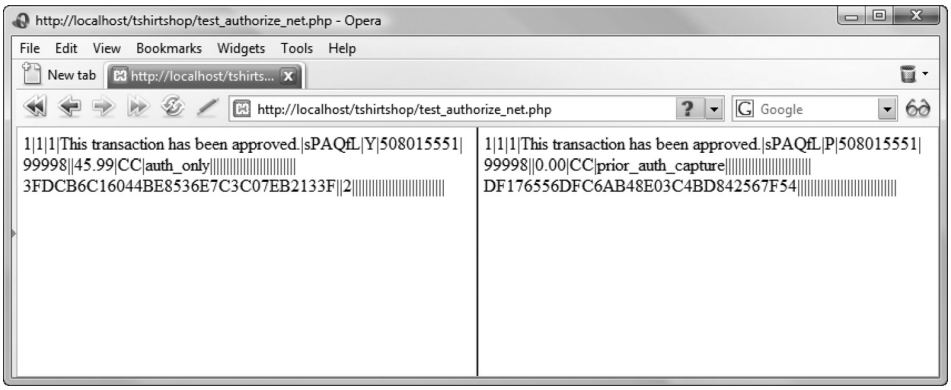


Рис. 20.3. Ответы Authorize.net на запросы

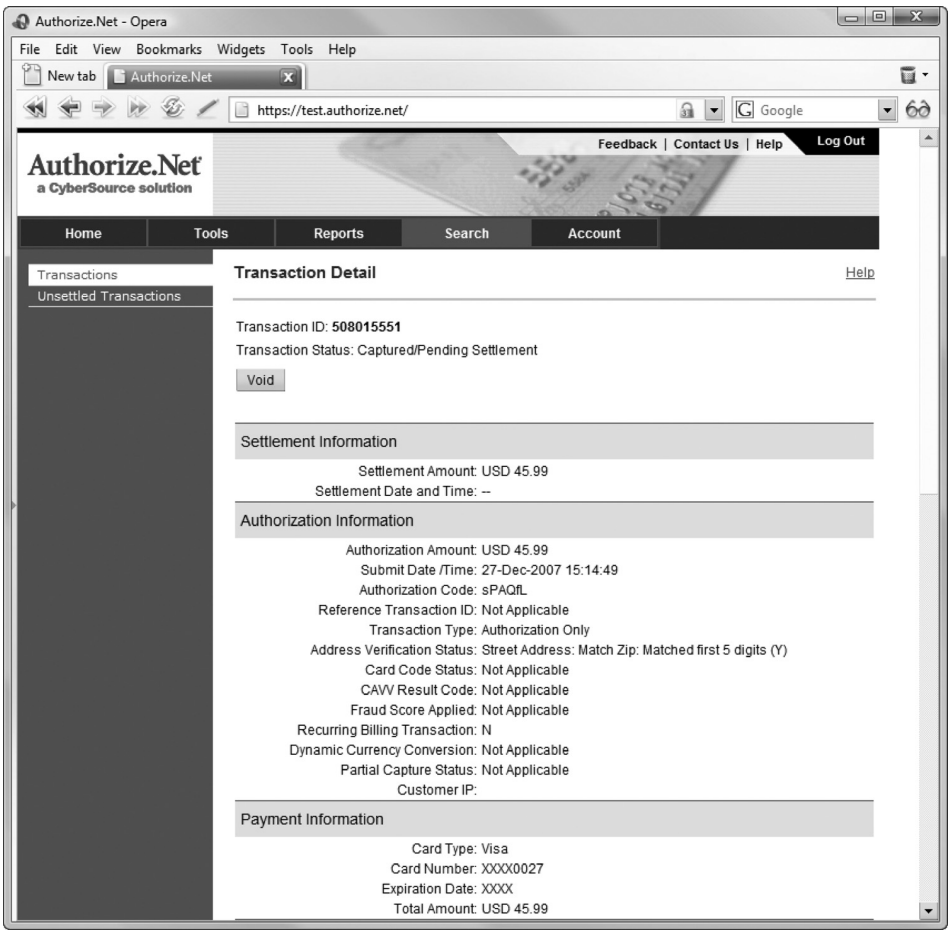


Рис. 20.4. Результаты тестовых транзакций на сайте Authorize.net

Как это работает: транзакции Authorize.net

Львиную долю работы выполняет класс `AuthorizeNetRequest`, в котором есть два основных метода: `SetRequest()` для задания параметров транзакций и `GetResponse()` для отправки запросов и получения ответов на них. В приведенном ниже фрагменте кода представлены вызовы этих методов.

```
// Авторизация
$transaction =
  array ('x_invoice_num' => '99999',          // Номер квитанции
         'x_amount' => '45.99',               // Сумма
         'x_card_num' => '4007000000027',     // Номер кредитной карты
         'x_exp_date' => '1209',              // Срок действия
         'x_method' => 'CC',                  // Метод оплаты
         'x_type' => 'AUTH_ONLY');            // Тип транзакции

$request->SetRequest($transaction);
$auth_only_response = $request->GetResponse();
```

ЗАМЕЧАНИЕ

Данные о кредитной карте, использованные в приведенном выше фрагменте кода, предоставлены сайтом Authorize.net в тестовых целях. Полный список тестовых параметров карт можно найти в руководстве *AIM Implementation Guide*.

В качестве параметра мы передаем методу `SetRequest()` массив с параметрами транзакции. Метод объединяет этот массив с другим массивом, содержащим сведения об учетной записи Authorize.net.

```
public function SetRequest($request)
{
  $this->_mRequest = '';

  $request_init =
    array ('x_login' => AUTHORIZE_NET_LOGIN_ID,
           'x_tran_key' => AUTHORIZE_NET_TRANSACTION_KEY,
           'x_version' => '3.1',
           'x_test_request' => AUTHORIZE_NET_TEST_REQUEST,
           'x_delim_data' => 'TRUE',
           'x_delim_char' => '|',
           'x_relay_response' => 'FALSE');

  $request = array_merge($request_init, $request);
```

Данные из массива собираются в строку в виде последовательности пар “имя–значение” с помощью функции `urlencode()`.

```
  foreach($request as $key => $value )
    $this->_mRequest .= $key . '=' . urlencode($value) . '&';
}
```

Метод `GetResponse()` класса `AuthorizeNetRequest` непосредственно отправляет запрос и принимает ответ на него с помощью библиотеки cURL.

```
  public function GetResponse()
  {

  ...

    // Выполняем сеанс cURL
    $result = curl_exec($ch);

    // Закрываем сеанс cURL
    curl_close ($ch);
```

```
    // Возвращаем ответ
    return $result;
  }
}
?>
```

Если метод `GetResponse()` вызывается для выполнения транзакции `AUTH_ONLY`, в ответе будет содержаться идентификатор транзакции. Если авторизация прошла успешно, этот идентификатор можно использовать в транзакции `PRIOR_AUTH_CAPTURE`, непосредственно снимающей деньги с карты.

Как уже упоминалось, ответ на запрос представляет собой строку, значения в которой разделены заданным символом, в нашем случае — вертикальной чертой (`|`). Чтобы считывать эти значения из строки, преобразуем ее в массив с помощью функции PHP `explode()` (описание этой функции доступно по адресу `http://www.php.net/manual/en/function.explode.php`).

```
$auth_only_response = $request->GetResponse();

$_SESSION['auth_only_response'] = $auth_only_response;

$auth_only_response = explode('|', $auth_only_response);
```

После выполнения этого фрагмента кода в переменной `$auth_only_response` будет содержаться массив, элементами которого являются значения из строки. Нас интересует седьмой элемент этого массива, в котором согласно документации Authorize.net содержится идентификатор транзакции (ответы Authorize.net на транзакции описаны в документе, доступном по адресу `http://www.authorize.net/support/AIM_guide.pdf`).

```
// Считываем ID транзакции, который понадобится для оплаты
$ref_trans_id = $auth_only_response[6];
```

ЗАМЕЧАНИЕ

Элементы массива в переменной `$auth_only_response`, созданного функцией `explode()`, индексируются от нуля, поэтому индекс седьмого элемента равен 6.

Код, снимающий деньги с карты с помощью идентификатора транзакции, не содержит ничего особо интересного. Поскольку транзакция уже авторизована, нужно указать только идентификатор транзакции, полученный после авторизации.

```
// Оплачиваем
$transaction =
  array ('x_ref_trans_id' => $ref_trans_id,    // Идентификатор транзакции
         'x_method' => 'CC',                   // Метод оплаты
         'x_type' => 'PRIOR_AUTH_CAPTURE');    // Тип транзакции

$request->SetRequest($transaction);
$prior_auth_capture_response = $request->GetResponse();
```

Интеграция Authorize.net и TShirtShop

Как и в случае с DataCash, для интеграции поддержки Authorize.net в TShirtShop нужно модифицировать классы `PsCheckFunds` и `PsTakePayment`.

Не забывайте, что вы можете использовать файлы с исходным кодом, доступные в разделе Source Code/Download сайта издательства Apress (`http://www.apress.com/`), если не хотите набирать код сами.

Мы должны будем модифицировать классы этапов конвейера, выполняющих операции с кредитными картами. Вся инфраструктура для хранения кодов аутентификации у нас уже есть — она доступна в методе `OrderProcessor::SetOrderAuthCodeAndReference()`.

Упражнение: интеграция Authorize.net и TShirtShop

1. Модифицируйте код в файле `business/ps_check_funds.php` для работы с Authorize.net. Если вы уже модифицировали этот код для работы с Datacash, можете создать резервную копию файла перед модификацией.

```
<?php
class PsCheckFunds implements IPipelineSection
{
  public function Process($processor)
  {
    // Аудит
    $processor->CreateAudit('PsCheckFunds started.', 20100);

    $order_total_cost = $processor->mOrderInfo['total_amount'];
    $order_total_cost += $processor->mOrderInfo['shipping_cost'];
    $order_total_cost +=
      round((float)$order_total_cost *
            (float)$this->mOrderInfo['tax_percentage'], 2) / 100.00;

    $exp_date = str_replace('/', '',
      $processor->mCustomerInfo['credit_card']->ExpiryDate);

    $transaction =
      array (
        'x_invoice_num' => $processor->mOrderInfo['order_id'],
        'x_amount' => $order_total_cost, // Amount to charge
        'x_card_num' => $processor->mCustomerInfo['credit_card']->CardNumber,
        'x_exp_date' => $exp_date, // Expiry (MMYY)
        'x_method' => 'CC',
        'x_type' => 'AUTH_ONLY');

    // Обрабатываем транзакцию
    $request = new AuthorizeNetRequest(AUTHORIZE_NET_URL);
    $request->SetRequest($transaction);

    $response = $request->GetResponse();

    $response = explode('|', $response);

    if ($response[0] == 1)
    {
      $processor->SetAuthCodeAndReference($response[4], $response[6]);

      // Аудит
      $processor->CreateAudit('Funds available for purchase.', 20102);

      // Обновляем статус заказа
      $processor->UpdateOrderStatus(2);

      // Продолжаем обработку
      $processor->mContinueNow = true;
    }
    else
    {
      // Аудит
      $processor->CreateAudit('Funds not available for purchase.', 20103);

      throw new Exception('Credit card check funds failed for order ' .
                          $processor->mOrderInfo['order_id'] . ".\n\n" .
```

```
                        'Data exchanged:' . "\n" .
                        var_export($transaction, true) . "\n" .
                        var_export($response, true));
    }

    // Аудит
    $processor->CreateAudit('PsCheckFunds finished.', 20101);
  }
}
?>
```

2. Модифицируйте файл `business/ps_take_payment.php` так, как показано ниже.

```
<?php
class PsTakePayment implements IPipelineSection
{
  public function Process($processor)
  {
    // Аудит
    $processor->CreateAudit('PsTakePayment started.', 20400);

    $transaction =
      array ('x_ref_trans_id' => $processor->mOrderInfo['reference'],
             'x_method' => 'CC',
             'x_type' => 'PRIOR_AUTH_CAPTURE');

    // Обрабатываем транзакцию
    $request = new AuthorizeNetRequest(AUTHORIZE_NET_URL);

    $request->SetRequest($transaction);
    $response = $request->GetResponse();

    $response = explode('|', $response);

    if ($response[0] == 1)
    {
      // Аудит
      $processor->CreateAudit(
        'Funds deducted from customer credit card account.', 20402);

      // Обновляем статус заказа
      $processor->UpdateOrderStatus(5);

      // Продолжаем обработку
      $processor->mContinueNow = true;

      // Аудит
      $processor->CreateAudit('PsTakePayment finished.', 20401);
    }
    else
    {
      // Аудит
      $processor->CreateAudit(
        'Error taking funds from customer credit card.', 20403);

      throw new Exception('Credit card take payment failed for order ' .
                          $processor->mOrderInfo['order_id'] . ".\n\n" .
                          'Data exchanged:' . "\n" .
                          var_export($transaction, true) . "\n" .
                          var_export($response, true));
    }
  }
}
?>
```

3. Добавьте в файл `index.php` ссылку на файл `business/authorize_net_request.php`.

```
require_once BUSINESS_DIR . 'ps_check_funds.php';
require_once BUSINESS_DIR . 'ps_check_stock.php';
require_once BUSINESS_DIR . 'authorize_net_request.php';
```

4. Добавьте в файл `admin.php` ссылку на файл `business/authorize_net_request.php`.

```
require_once BUSINESS_DIR . 'ps_ship_ok.php';
require_once BUSINESS_DIR . 'ps_final_notification.php';
require_once BUSINESS_DIR . 'authorize_net_request.php';
```

Тестирование интеграции Authorize.net в TShirtShop

Все, что осталось сделать теперь, — протестировать только что добавленную функциональность. Получите тестовые параметры кредитных карт из руководства *AIM Integration Guide* и выполните несколько тестовых транзакций.

Резюме

В этой главе мы практически закончили разработку нашего приложения, добавив в него поддержку кредитных карт. Все, что осталось сделать для запуска электронного магазина в работу, — открыть счет в банке, наладить связи с поставщиками и заполнить базу данных товарами. Да, это непростые задачи, но они не относятся к задачам программирования.

Мы разобрались, как работать с провайдерами шлюзов для кредитных карт, на примере DataCash и Authorize.net, и интегрировали поддержку этих провайдеров в наше приложение.

Глава 21

Отзывы о товарах

К данному моменту у нас есть полностью рабочий сайт электронного магазина. Но это не значит, что мы ничего больше не можем добавить к нему, чтобы сделать его более удобным и привлекательным для посетителей.

Добавив на сайт систему отзывов о товарах, мы повысим вероятность того, что посетители будут возвращаться на наш сайт — чтобы оставить отзыв о купленном товаре или чтобы почитать отзывы других посетителей о товаре, который их интересует.

Кроме того, система отзывов поможет нам узнать, чего хотят посетители, и за счет этого улучшить систему рекомендаций или даже внести изменения в веб-сайт или структуру каталога товаров.

Чтобы упростить работу с отзывами и для посетителей, и для нас, мы добавим на сайт список для хранения отзывов и форму для оставления новых отзывов прямо на страницу со сведениями о товаре. Возможность оставить отзыв будет только у зарегистрированных посетителей, поскольку мы решили, что анонимные отзывы неприемлемы (впрочем, вы легко можете это изменить). Мы напишем код для этого нового механизма как обычно — начав с уровня данных и закончив на уровне интерфейса.

Планирование механизма отзывов

В механизме для работы с отзывами нет ничего особенно сложного. Он должен соответствовать таким требованиям:

- Список уже существующих отзывов о товаре и элементы управления для оставления новых отзывов должны отображаться под рекомендациями товаров на странице сведений о товаре, как показано на рис. 21.1.
- Если о товаре еще нет отзывов, мы должны предложить посетителю оставить первый отзыв, как показано на рис. 21.2.
- Отзывы могут оставлять только зарегистрированные посетители.

Мы будем хранить отзывы в специальной таблице `review` в базе данных. Для работы с этой таблицей мы создадим две хранимые процедуры — `catalog_get_product_reviews` и `catalog_create_product_review`. Весь остальной код будет, по сути, только оболочкой, предоставляющей посетителям сайта доступ к этой функциональности.

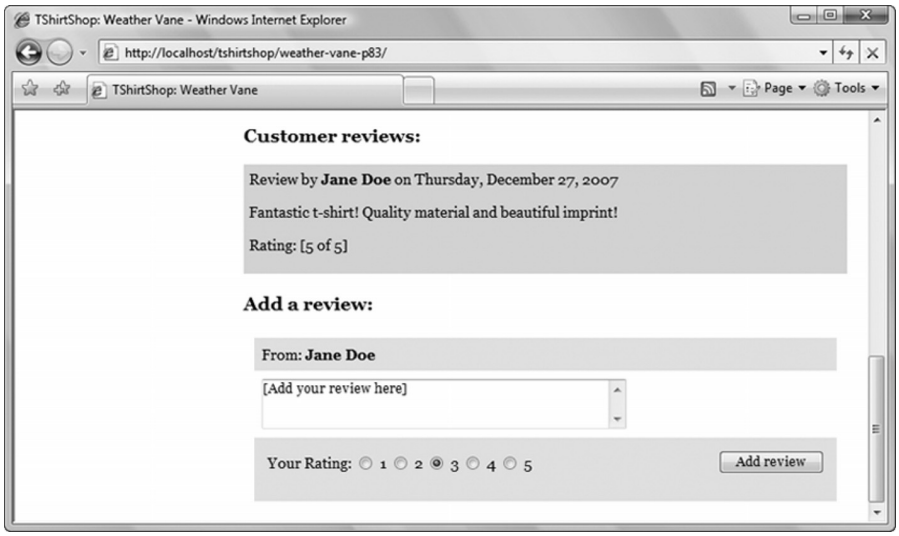


Рис. 21.1. Страница сведений о товаре с отзывами о нем

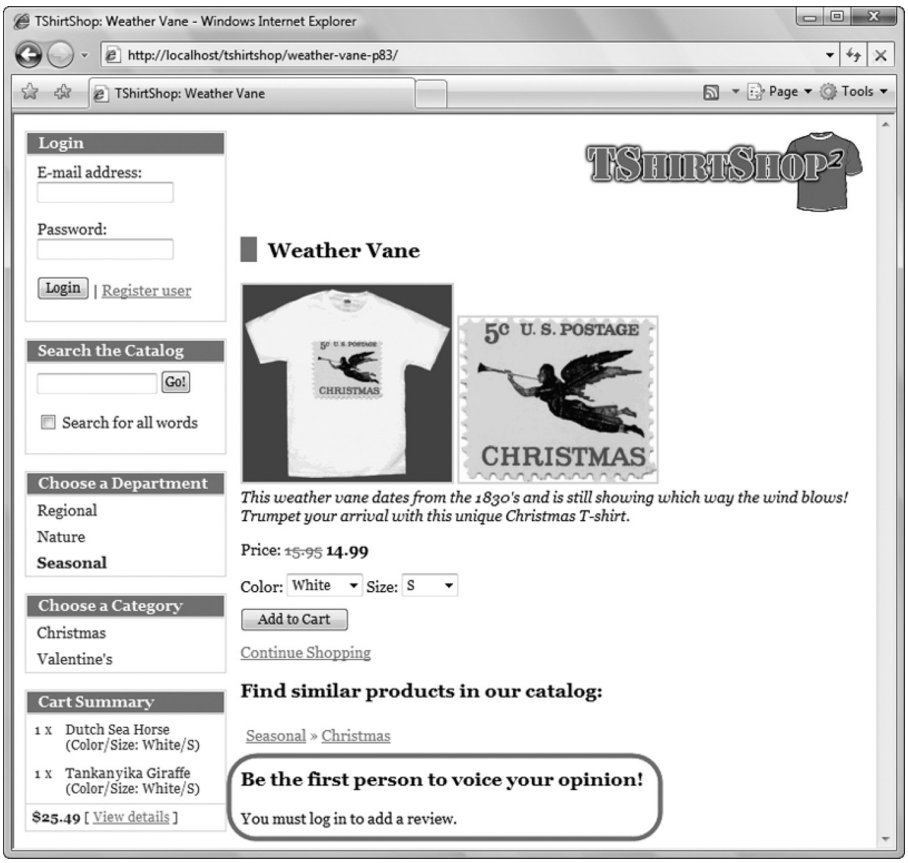


Рис. 21.2. Приглашение оставить первый отзыв о товаре

Реализация отзывов о товарах

Выполните шаги из упражнения ниже, чтобы создать таблицу в базе данных, хранимые процедуры, код уровня логики приложения и уровня представления для механизма отзывов о товарах.

Упражнение: реализация отзывов о товарах

1. Запустите phpMyAdmin, откройте базу данных `tshirtshop` и перейдите на страницу создания SQL-запроса. Затем выполните приведенный ниже оператор SQL, чтобы создать в базе данных таблицу `review`:

```
-- Создаем таблицу review
CREATE TABLE `review` (
  `review_id`   INT      NOT NULL AUTO_INCREMENT,
  `customer_id` INT      NOT NULL,
  `product_id`  INT      NOT NULL,
  `review`      TEXT     NOT NULL,
  `rating`      SMALLINT NOT NULL,
  `created_on`  DATETIME NOT NULL,
  PRIMARY KEY (`review_id`),
  KEY `idx_review_customer_id` (`customer_id`),
  KEY `idx_review_product_id` (`product_id`)
);
```

2. Выполните следующий оператор SQL, чтобы создать хранимую процедуру `catalog_get_product_reviews` (не забудьте предварительно задать символы `$$` в качестве разделителя). Эта хранимая процедура извлекает из базы данных отзывы о товаре с заданным идентификатором. Кроме того, нам понадобится извлекать из базы данных имя автора отзыва, поэтому мы используем соединение с таблицей `customer`.

```
-- Создаем хранимую процедуру catalog_get_product_reviews
CREATE PROCEDURE catalog_get_product_reviews(IN inProductId INT)
BEGIN
  SELECT     c.name, r.review, r.rating, r.created_on
  FROM       review r
  INNER JOIN customer c
               ON c.customer_id = r.customer_id
  WHERE      r.product_id = inProductId
  ORDER BY   r.created_on DESC;
END$$
```

3. Выполните следующий оператор SQL, чтобы создать хранимую процедуру `catalog_create_product_review` (не забудьте предварительно задать символы `$$` в качестве разделителя). Эта хранимая процедура будет вызываться, когда зарегистрированный посетитель добавит отзыв о товаре.

```
-- Создаем хранимую процедуру catalog_create_product_review
CREATE PROCEDURE catalog_create_product_review(IN inCustomerId INT,
  IN inProductId INT, IN inReview TEXT, IN inRating SMALLINT)
BEGIN
  INSERT INTO review (customer_id, product_id, review, rating, created_on)
         VALUES (inCustomerId, inProductId, inReview, inRating, NOW());
END$$
```

4. Добавьте соответствующие этим хранимым процедурам методы в класс `Catalog` в файле `business/catalog.php`:

```
// Возвращает отзывы о заданном товаре
public static function GetProductReviews($productId)
{
  // Составляем SQL-запрос
  $sql = 'CALL catalog_get_product_reviews(:product_id)';

  // Создаем массив параметров
  $params = array (':product_id' => $productId);

  // Выполняем запрос и возвращаем результаты
  return DatabaseHandler::GetAll($sql, $params);
}

// Добавляет отзыв в базу данных
public static function CreateProductReview($customer_id, $productId,
                                           $review, $rating)
{
  // Составляем SQL-запрос
  $sql ='CALL catalog_create_product_review(:customer_id, :product_id,
                                            :review, :rating)';

  // Создаем массив параметров
  $params = array (':customer_id' => $customer_id,
                   ':product_id' => $productId,
                   ':review' => $review,
                   ':rating' => $rating);

  // Выполняем запрос
  DatabaseHandler::Execute($sql, $params);
}
```

5. Пользовательский интерфейс в данном случае состоит из компонентного шаблона, который будет использоваться на странице сведений о товаре. Сначала создайте файл `presentation/templates/reviews.tpl` и добавьте в него следующий код:

```
{* reviews.tpl *}
{load_presentation_object filename="reviews" assign="obj"}
{if $obj->mTotalReviews != 0}
<h2>Customer reviews:</h2>
<ul class="reviews-list">
  {section name=i loop=$obj->mReviews}
  <li>
    <p>
      Review by <strong>{$obj->mReviews[i].name}</strong> on
      {$obj->mReviews[i].created_on|date_format:"%A, %B %e, %Y"}
    </p>
    <p>{$obj->mReviews[i].review}</p>
    <p>Rating: [{$obj->mReviews[i].rating} of 5]</p>
  </li>
  {/section}
</ul>
{else}
<h2>Be the first person to voice your opinion!</h2>
{/if}
{if $obj->mEnableAddProductReviewForm}
{* add review form *}
<h2>Add a review:</h2>
<form method="post" action="{$obj->mLinkToProduct}">
<table class="review-table">
  <tr>
```

```
    <td class="add-review">From: <strong>{$obj->mReviewerName}</strong></td>
  </tr>
  <tr>
    <td>
      <textarea name="review"
       rows="3" cols="65">[Add your review here]</textarea>
    </td>
  </tr>
  <tr>
   <td class="add-review">
     <table class="review-table">
       <tr>
         <td>
           Your Rating:
           <input type="radio" name="rating" value="1" /> 1
           <input type="radio" name="rating" value="2" /> 2
           <input type="radio" name="rating" value="3" checked="checked" /> 3
           <input type="radio" name="rating" value="4" /> 4
           <input type="radio" name="rating" value="5" /> 5
         </td>
            <td align="right">
              <input type="submit" name="AddProductReview"
                     value="Add review" />
            </td>
          </tr>
        </table>
      </td>
    </tr>
  </table>
</form>
{else}
<p>You must log in to add a review.</p>
{/if}
```

6. Создайте файл `presentation/reviews.php` и добавьте в него следующий код:

```
<?php
// Класс, обеспечивающий поддержку отзывов на товары
class Reviews
{
  public $mProductId;
  public $mReviews;
  public $mTotalReviews;
  public $mReviewerName;
  public $mEnableAddProductReviewForm = false;
  public $mLinkToProduct;
  public function __construct()
  {
    if (isset ($_GET['ProductId']))
      $this->mProductId = (int)$_GET['ProductId'];
    else
      trigger_error('ProductId not set', E_USER_ERROR);

    $this->mLinkToProduct = Link::ToProduct($this->mProductId);
  }

  public function init()
  {
    // Если пользователь прошел аутентификацию...
    if (Customer::IsAuthenticated())
    {
      // Проверяем, хочет ли пользователь добавить отзыв
```

```
      if (isset($_POST['AddProductReview']))
        Catalog::CreateProductReview(Customer::GetCurrentCustomerId(),
                                     $this->mProductId, $_POST['review'],
                                     $_POST['rating']);

      // Отображаем форму "add review"
      $this->mEnableAddProductReviewForm = true;

      // Получаем имя пользователя (reviewer's)
      $customer_data = Customer::Get();
      $this->mReviewerName = $customer_data['name'];
    }

    // Получаем отзывы об этом товаре
    $this->mReviews = Catalog::GetProductReviews($this->mProductId);

    // Получаем количество отзывов
    $this->mTotalReviews = count($this->mReviews);
  }
}
?>
```

7. Откройте файл `presentation/templates/product.tpl` и добавьте в конец этого файла следующую строку:

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

8. Добавьте в конец файла `tshirtshop.css` следующие стили:

```
.reviews-list li {
  background: #ccddff;
  border-bottom: #fff solid 3px;
  display: block;
  padding: 5px;
}

.review-table {
  width: 100%;
}

.review-table tr td {
  border: none;
}

.add-review {
  background: #e6e6e6
  padding: 5px;
}
```

9. Загрузите сайт TShirtShop в браузер и откройте страницу со сведениями о какомнибудь товаре. Если вы не проходили аутентификацию на сайте и об этом товаре нет отзывов, страница должна выглядеть примерно так, как на рис. 21.2. Если о товаре есть один отзыв, страница должна быть похожа на рис. 21.1.

Как это работает: компонентный шаблон `reviews`

Компонентный шаблон `reviews` отвечает и за отображение существующих отзывов, и за добавление новых. Первая часть кода в файле `reviews.tpl` определяет, есть ли отзывы о выбранном товаре. Если нет, на вебстранице отображается короткое сообщение, предлагающее оставить первый отзыв:

```
{if $obj->mTotalReviews != 0}
<h2>Customer reviews:</h2>
```

```
[a list with reviews]
{else}
<h2>Be the first person to voice your opinion!</h2>
{/if}
```

Вторая часть шаблона отображает форму для ввода отзывов или сообщение о необходимости пройти аутентификацию, чтобы получить возможность оставить отзыв:

```
{if $obj->mEnableAddProductReviewForm}
{* add review form *}
<h2>Add a review:<h2>
[add review form]
{else}
<p>You must log in to add a review.</p>
{/if}
```

Код уровня логики приложения и уровня данных тривиален и не содержит ничего заслуживающего внимания.

Резюме

Да, это действительно было просто. Возможно, вы захотите улучшить механизм, созданный нами в этой главе (например, чтобы пользователи могли редактировать отзывы, оставленные ими ранее), но базовый вариант готов и работает.

В следующей, последней главе книги, мы рассмотрим продажу товаров из внешнего источника (в качестве примера выбран Amazon.com) с помощью XML веб-служб.

Глава 22

Работа с веб-службами Amazon.com

Ранее в этой книге мы уже добавляли в наш проект возможности использования внешних механизмов — PayPal, DataCash и Authorize.net. В этой главе мы рассмотрим еще один вариант интеграции механизмов из внешних источников, на этот раз — с помощью веб-служб. Научившись работать с веб-службами, вы получите важное преимущество перед конкурентами. В этой главе вы:

- узнаете, что такое веб-службы;
- узнаете, как использовать веб-службы Amazon E-Commerce;
- научитесь использовать веб-службы Amazon, чтобы продавать футболки Amazon на сайте TShirtShop.

Если вы хотите более подробно изучить работу с веб-службами с помощью PHP, попробуйте найти книгу *Pro PHP XML and Web Services* (автор — Robert Richards, издательство — Apress, 2006). В ней есть примеры работы с веб-службами Amazon, Google, Yahoo и eBay.

Введение в веб-службы

Веб-служба (web-service) — это элемент функциональности, доступный через Интернет с помощью стандартных протоколов (прежде всего, с помощью HTTP). Этот элемент обменивается с клиентом сообщениями, которые обычно представляются в виде XML-документов (такой протокол называется SOAP (Simple Object Access Protocol — простой протокол доступа к объектам)). Также распространен протокол REST (REpresentational State Transfer — передача представления состояний). И REST, и SOAP используют HTTP для передачи данных через Интернет.

REST использует специальным образом составленные URL с парами "имя–значение" для вызова определенных методов на сервере. Этот протокол считается самым простым методом взаимодействия с веб-службами. Чтобы использовать REST, достаточно отправить на сервер запрос HTTP `GET` и получить ответ на него в виде XML-документа.

SOAP — это основанный на XML стандарт кодирования для обмена данными между веб-службами и клиентами. В развитии SOAP участвуют несколько организаций, в том числе Microsoft, IBM и Sun.

Основное преимущество веб-служб состоит в том, что использующий их клиент может работать на любой платформе. Если для обмена информацией применяются

стандартные протоколы, например SOAP и HTTP, клиентом может быть сотовый телефон с приложением на Java, а сервер может работать под управлением Solaris и использовать какую-нибудь экзотическую СУБД.

Такого рода возможности всегда привлекательны для разработчиков, и мы рекомендуем вам купить специальную книгу о веб-службах, чтобы изучить их подробнее. Список общедоступных веб-служб можно найти по адресу `http://www.xmethods.net/` (зайдите по этому адресу, чтобы составить себе представление о возможностях веб-служб).

В этой главе мы интегрируем в наш проект веб-службы Amazon (Amazon Web Services — AWS), чтобы продавать товары Amazon на своем веб-сайте.

У нас уже есть веб-сайт по продаже футболок. Мы можем продвинуться немного дальше и предложить покупателям другие товары, имеющие отношение к футболкам. Бесплатно? Разумеется, нет; мы будем отображать на своем сайте товары `Amazon.com`, и оплата за них будет проходить также через `Amazon.com`, но в конце концов на ваш банковский счет будет переведена небольшая комиссия за каждый проданный товар. Неплохо, правда?

Из этой главы вы узнаете, как с помощью AWS добавить на свой сайт специальный раздел Amazon T-Shirts, показанный на рис. 22.1. Этот раздел будет работать не так, как остальные разделы сайта, — например, оплата за предложенные в нем товары будет осуществляться через Amazon.com, а не через наш механизм получения оплаты. Мы рассмотрим только небольшую часть возможностей AWS, так что, если вы действительно хотите серьезно зарабатывать с помощью `Amazon.com`, вам многое придется узнать самостоятельно.



Рис. 22.1. Страница раздела Amazon.com на нашем веб-сайте

Оставшаяся часть главы поделена на две части. Первая содержит описание методов работы с AWS; вторая посвящена интеграции AWS в наш веб-сайт.

ПОДСКАЗКА

Код в этой главе не связан с остальной частью сайта, поэтому нужно начинать интегрировать веб-службы `Amazon.com` в код после освоения первых четырех глав книги (как только у вас появляется работающий каталог товаров). Разумеется, с небольшими изменениями этот код можно адаптировать и к версиям сайта из последующих глав.

Доступ к веб-службам Amazon.com

Большинство провайдеров веб-служб (в том числе `Amazon.com`) используют протокол SOAP или REST (или оба сразу). В этой главе мы рассмотрим взаимодействие с веб-службами AWS с помощью как SOAP, так и REST (результаты будут одинаковыми в обоих случаях).

AWS — это общее обозначение всех веб-служб, предоставляемых сайтом `Amazon.com`. Из всего набора нас интересует Amazon Associates Web Service (A2S, ранее известная как Amazon E-Commerce Service, или ECS). Эта веб-служба позволяет отправлять запросы к каталогу товаров Amazon.com и получать информацию о товарах из этого каталога. Мы воспользуемся ею, чтобы получать данные о футболках из каталога `Amazon.com` и отображать их на страницах нашего сайта.

Обращаясь к веб-службам Amazon, в том числе к A2S, можно оформлять запросы с помощью REST или составлять сообщения SOAP. Веб-служба будет возвращать ответы на запросы в виде XML-документов.

В этой главе мы воспользуемся лишь небольшой частью возможностей A2S — на полное их рассмотрение ушла бы не одна толстая книга. Кроме того, учтите, что в примерах этой главы мы будем работать с американским отделением Amazon, но с помощью одной и той же учетной записи в AWS можно работать с любым отделением Amazon — `Amazon.fr`, `Amazon.ca`, `Amazon.de`, `Amazon.co.jp` или `Amazon.co.uk`.

Создание учетной записи Amazon.com Web Services

Официальный сайт AWS доступен по адресу `http://www.amazon.com/webservices`. Самую свежую документацию по веб-службам AWS можно найти по адресу `http://developer.amazonwebservices.com/connect/` (лучше будет, если вы добавите этот адрес в адресную книгу своего браузера, поскольку вам, вероятно, придется часто ее посещать).

Прежде чем двигаться дальше, нужно создать учетную запись AWS. Для этого понадобится *ключ доступа (access key ID)*, обозначающий вашу учетную запись в системе AWS. Если такого ключа у вас нет, можете получить его по адресу `http://www.amazon.com/gp/aws/registration/registration-form.html`. Ключ доступа представляет собой 20-символьную строку из букв и цифр.

ЗАМЕЧАНИЕ

До 11 октября 2005 года Amazon.com использовал вместо ключей доступа так называемые идентификаторы подписки (subscription ID). По назначению они аналогичны ключам доступа, и если у вас есть идентификатор подписки, вы можете продолжать его использовать. Новым пользователям Amazon.com выдает только ключи доступа.

Ключ доступа позволяет использовать веб-службы Amazon и Alexa (Alexa — это служба, принадлежащая Amazon), как показано на рис. 22.2. Ключ доступа должен быть известен только вам, но он не является особо ценной информацией — вы всегда можете бесплатно получить новый, если забудете (или разгласите) старый.

Для платных служб и для служб, работающих с конфиденциальными данными, Amazon использует другой ключ — ключ секретного доступа (secret access key), который вы тоже получаете при регистрации. В этой главе мы не будем использовать ключ секретного доступа — для работы с A2S достаточно обычного ключа доступа.

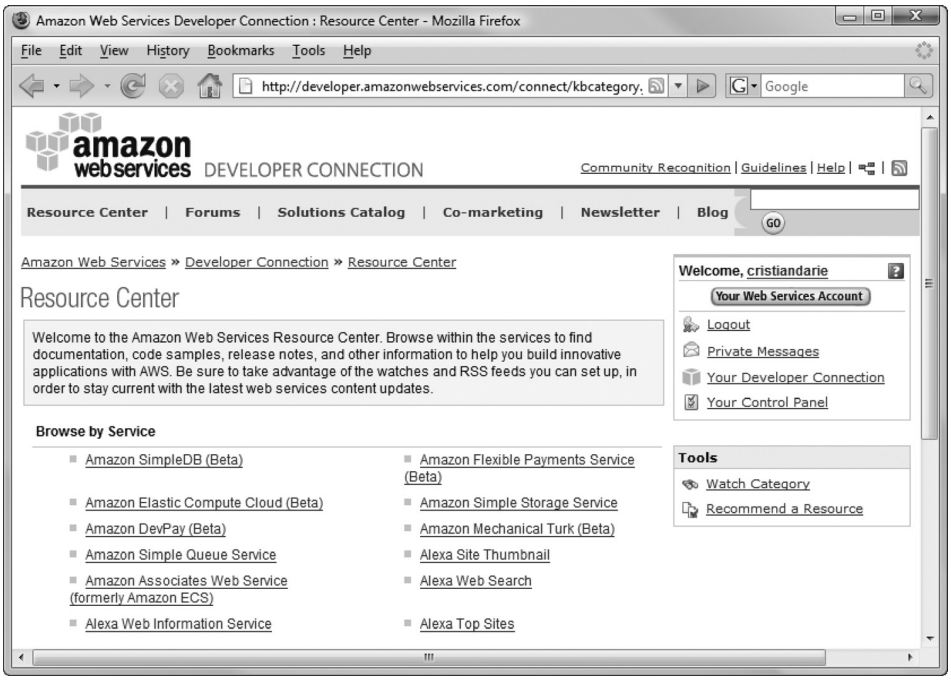


Рис. 22.2. Веб-службы Amazon.com

Идентификатор сообщества Amazon

Ключ доступа, который у вас уже есть, позволяет получать данные от AWS. Их можно отображать в разделе Amazon T-Shirt нашего электронного магазина, показанном на рис. 22.1.

Чтобы получать от Amazon комиссионные за товары, проданные через ваш сайт, вам понадобится идентификатор сообщества (associate ID). Он используется в ссылках Buy from Amazon, которые будут отображаться в разделе Amazon T-Shirt вашего сайта, и именно по этому идентификатору Amazon будет определять, что продажа прошла именно через ваш сайт. Поэтому, прежде чем двигаться дальше, получите идентификатор по адресу `http://associates.amazon.com/gp/associates/apply/main.html`.

Заметьте, что ключ доступа и идентификатор сообщества — это два никак не связанных объекта, у каждого из которых свое предназначение. Идентификатор сообщества используется в ссылках на сайт Amazon.com в электронном магазине. Это позволяет Amazon определять, с какого из сайтов сообщества посетители пере-

ходят на его сайт. Идентификатор сообщества не является секретным, поскольку любой посетитель вашего сайта может увидеть его в составе URL. Чтобы продавать товары Amazon.com через свой сайт, вам достаточно получить идентификатор содружества — ключ доступа понадобится только для доступа к веб-службам Amazon.com.

В TShirtShop мы будем подключаться к веб-службам Amazon.com (и использовать ключ доступа) для выполнения операций поиска в каталоге и получения информации о товарах, отображаемых на страницах сайта в разделе Amazon T-Shirts.

Доступ к Amazon.com E-Commerce Service с помощью REST

Для доступа к веб-службам с помощью протокола REST используются URL определенной структуры. Попробуйте открыть в браузере такой URL (в книге он разбит на несколько строк, но в браузере его нужно вводить одной строкой, кроме того, не забудьте подставить в URL свой ключ доступа).

```
http://webservices.amazon.com/onca/xml?Service
=AWSECommerceService&AWSAccessKeyId=[Ваш ключ доступа]
&Operation=ItemLookup
&IdType=ASIN
&ItemId=1590598644
```

Браузер должен отобразить XML-структуру с информацией о книге. На рис. 22.3 показана эта информация в окне Firefox, который аккуратно отображает дерево узлов XML (изящное отображение данных в TShirtShop мы обсудим позже, а пока что давайте сосредоточимся на самих данных, возвращаемых в ответ на запрос).

Неплохо, правда? Все, что нужно, чтобы воспользоваться веб-службой, поддерживающей REST, — это правильно составить URL. У каждого товара в базе данных Amazon.com есть уникальный идентификатор — ASIN (Amazon.com Standard Item Number — стандартный номер предмета Amazon.com). Для книг, например, в качестве идентификатора используется номер ISBN, поэтому у книги с рис. 22.3 идентификатор равен 1590598644.

URL, которым мы только что воспользовались, сообщил веб-службе следующее: наш ключ доступа, тип операции (поиск в каталоге) и идентификатор искомого товара (ASIN 1590598644).

Среди данных, полученных нами в ответ на запрос, нет некоторых интересующих нас сведений о книге, например цены, ссылки на фотографию с обложки или отзывов покупателей. Мы можем точно указать веб-службе, какие данные хотим получить, с помощью *групп ответов* (response groups). Группы ответов — это наборы стандартных элементов данных, которые веб-служба может включать в ответ на запрос.

ЗАМЕЧАНИЕ

На момент написания этой книги AWS предлагала более 35 групп ответов. В этой книге мы рассмотрим только группы, которые будем использовать в TShirtShop; полный список групп можно найти в документации к AWS.

Давайте рассмотрим пример использования групп ответов. В конец ссылки, которую мы только что использовали, добавьте такую строку.

```
&ResponseGroup=Request,SalesRank,Small,Images,
VariationSummary
```

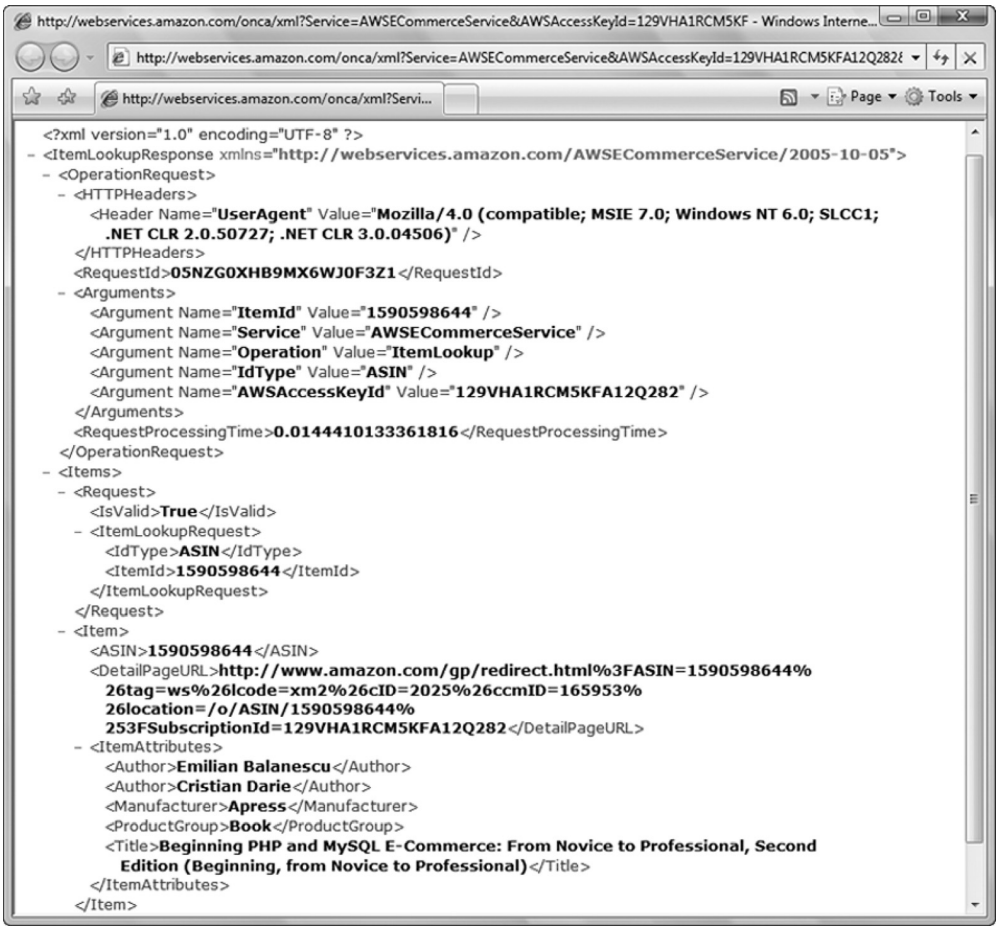


Рис. 22.3. XML-ответ веб-службы Amazon.com на запрос REST

Вся ссылка должна выглядеть так.

```
http://webservices.amazon.com/onca/xml?Service
=AWSECommerceService&AWSAccessKeyId=[Ваш ключ доступа]
&Operation=ItemLookup
&IdType=ASIN
&ItemId=1590598644
&ResponseGroup=Request,SalesRank,Small,Images,
VariationSummary
```

Добавленный фрагмент сообщит веб-службе, что нас интересуют дополнительные сведения о книге. Ответ на этот запрос будет таким, как на рис. 22.4.

В новый запрос мы включили пять групп ответов: `Request`, `SalesRank`, `Small`, `Images`, `VariationSummary`. Дополнительная информация о группах ответов приведена в документации к AWS. Эту документацию можно загрузить в виде файлов PDF или просмотреть в оперативном режиме по адресу `http://docs.amazonwebservices.com/AWSECommerceService/`.



Рис. 22.4. Ответ веб-службы на запрос с заданными группами ответов

В документации к AWS описание групп ответов содержится в разделе `API Reference, Response Groups`. Далее представлено краткое описание групп, использованных в приведенном выше запросе.

- Группа ответов `Request` используется по умолчанию в любом ответе на запрос и возвращает набор пар "имя–значение", входивших в запрос.
- Группа ответов `SalesRank` возвращает информацию о текущем рейтинге продаж запрошенного товара.
- Группа ответов `Small` возвращает общую информацию о запрошенном товаре (ASIN, название товара, URL и т.д.). Эта группа используется по умолчанию в ответах на запросы типа `ItemLookup` (поиск товара), в том числе на запрос, который составили мы.
- Группа ответов `Images` содержит адреса трех изображений (маленького, среднего и большого) для каждого из товаров в ответе на запрос.
- Группа ответов `VariationSummary` возвращает подробную информацию о ценах (наименьшая цена, наибольшая цена, наименьшая и наибольшая цены продаж) для каждого из товаров в ответе на запрос.

Давайте посмотрим, как можно отправить REST-запрос из кода на PHP. Чтобы заполнить данными будущий отдел Amazon T-Shirts, нужно выполнить поиск в разделе Apparel сайта Amazon.com с ключевыми словами `postal t-shirt`. Самый прямолинейный способ сделать это — воспользоваться функцией PHP `file_get_contents()`, как показано в приведенном ниже примере сценария.

Чтобы протестировать использование REST в веб-приложении, создайте файл `test_rest.php` в папке `tshirtshop` и поместите в него такой код.

```
<?php
// Сообщаем браузеру, что он должен получить XML-документ.
header('Content-type: text/xml');

/* Не забудьте указать в запросе ваш ключ доступа */
$url = 'http://webservices.amazon.com/onca/xml?Service=AWSECommerceService' .
       '&AWSAccessKeyId=[Ваш ключ доступа]' .
       '&Operation=ItemSearch' .
       '&Keywords=postal+t-shirt' .
       '&SearchIndex=Apparel' .
       '&ResponseGroup=Request,Medium,VariationSummary';

echo file_get_contents($url);
?>
```

ЗАМЕЧАНИЕ

На некоторых платформах PHP этот код по умолчанию не работает. Чтобы он стал работать, нужно изменить в файле `php.ini` следующую настройку.

```
allow_url_fopen = On
```

В качестве альтернативы можно включить в файл `include/config.php` строку вида

```
ini_set('allow_url_fopen', 'On');
```

У этого варианта есть преимущество — он повлияет только на работу нашего приложения, а не всего веб-сервера сразу. Кроме того, этот вариант сделает приложение работающим и при переносе на другой веб-сервер с тем же ограничением.

Открыв в браузере страницу с адресом `http://localhost/tshirtshop/test_rest.php`, вы увидите XML-данные о футболках, предлагаемых Amazon.com (рис. 22.5).

Дополнительные примеры ссылок, используемых во время работы с веб-службами с помощью REST, можно найти в документации к AWS, в разделе *API Reference*.

Доступ к Amazon.com E-Commerce Service с помощью SOAP

Для доступа к веб-службам с помощью протокола SOAP необходимо использовать специальный API. В приведенном ниже примере выполняется та же операция поиска, которую мы только что выполнили с помощью REST. Для этого в примере используются объекты `AWSECommerceService`, `ItemSearch` и `ItemSearchRequest` из API Amazon.com.

ЗАМЕЧАНИЕ

Для доступа к веб-службам Amazon.com с помощью SOAP мы будем использовать расширение PHP SOAP. Убедитесь, что в файле `php.ini` раскомментирована ссылка на файл `php_soap.dll`. Документацию к PHP SOAP можно найти по адресу `http://www.php.net/soap/`.

Чтобы протестировать доступ к веб-службам с помощью SOAP, создайте в папке `tshirtshop` файл `test_soap.php` и поместите в него такой код.

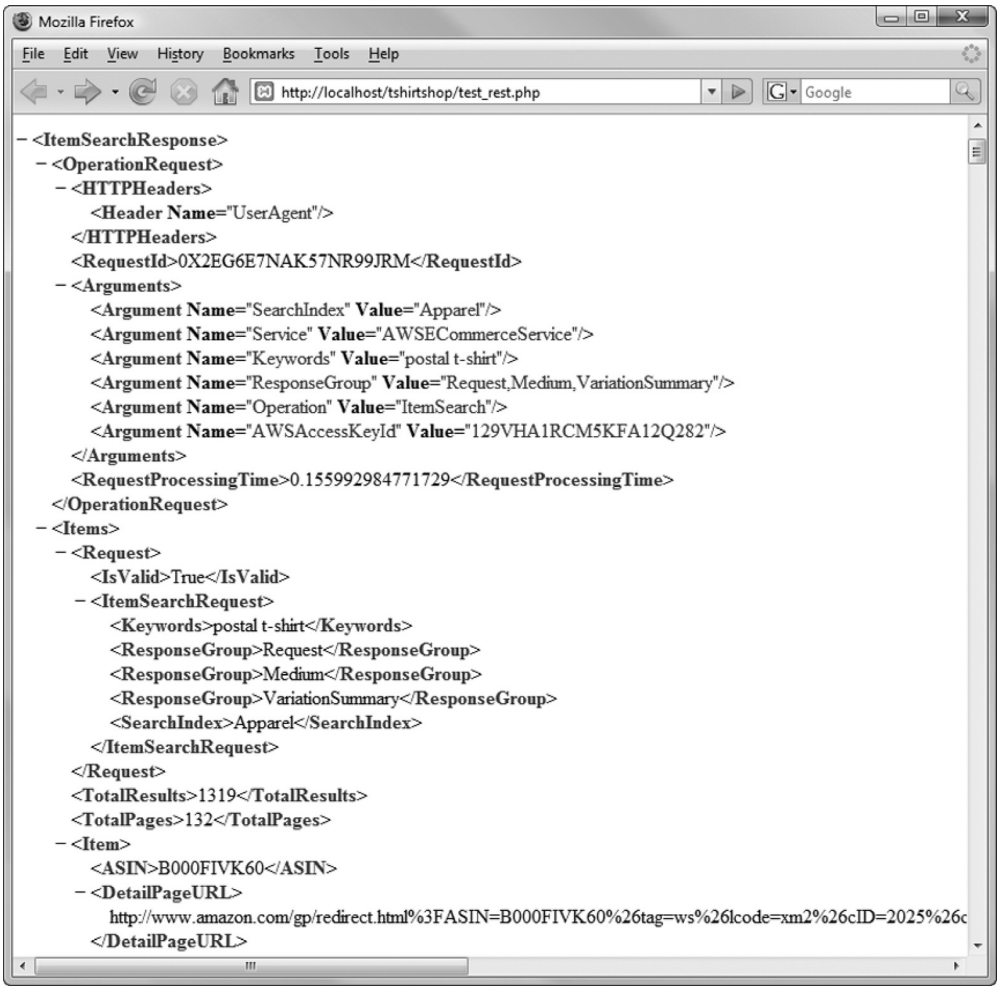


Рис. 22.5. Ответ Amazon.com на REST-запрос

```
<?php
try
{
// Инициализация объекта клиента SOAP
$client = new SoapClient(
'http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl');
/* Не забудьте заменить строку'[Ваш ключ доступа]' в следующем коде
 ключом доступа */
  $request = array ('Service' => 'AWSECommerceService',
                    'AWSAccessKeyId' => '[Ваш ключ доступа]',
                    'Request' => array ('Operation' => 'ItemSearchRequest',
                                        'Keywords' => 'postal+t-shirt',
                                        'SearchIndex' => 'Apparel',
                                        'ResponseGroup' =>
                                          array ('Request',
                                                 'Medium',
                                                 'VariationSummary')));
```

```
  $result = $client->ItemSearch($request);
  echo '<pre>';
  print_r($result);
  echo '</pre>';
}
catch (SoapFault $fault)
{
  trigger_error('SOAP Fault: (faultcode: ' . $fault->faultcode . ', ' .
                'faultstring: ' . $fault->faultstring . ')', E_USER_ERROR);
}
?>
```

Весь код для работы с SOAP помещен в блок `try`. Если запрос SOAP не удается выполнить, генерируется исключение типа `SoapFault`, которое мы преобразуем в ошибку с помощью функции `trigger_error`. За более подробной информацией об исключениях в SOAP обращайтесь по адресу `http://www.php.net/manual/en/function.is-soap-fault.php`.

Результат запроса SOAP — это объект, содержащий запрошенные данные. Если вы загрузите файл `test_soap.php` в браузер, в окне отобразится текст, который довольно трудно понять (он предназначен для обработки программами, а не для просмотра пользователем).

Код нашего примера начинается с создания объекта клиента SOAP для веб-службы Amazon.com.

```
// Инициализация объекта клиента SOAP
$client = new SoapClient(
'http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl')
```

Файл с расширением `.wsdl`, на который ссылается этот код, содержит описание всех функций, доступных на сервере SOAP Amazon.com, и типов их параметров. Создав объект клиента, мы можем вызывать эти функции, например, так.

```
$result = $client->ItemSearch($request);
```

Можно получить такой же результат, вызвав метод веб-службы неявно, с помощью функции `__soapCall()` (см. `http://www.php.net/manual/en/function.soap-soapclient-soap-call.php`), например, так.

```
$result->__soapCall('ItemSearch', array ($request));
```

Этот вызов метода просит веб-службу выполнить операцию `ItemSearch` по ключевым словам `postal+t-shirt` в магазине `Apparel`. Весь запрос помещен в блок try-catch, который перехватывает исключения в случае их возникновения и генерирует сообщения об ошибках. Класс исключений `SoapFault`, предназначенный для работы с SOAP, описан в документации по адресу `http://www.php.net/manual/en/function.is-soap-fault.php`. Открыв наш пример в окне браузера, вы получите примерно такой результат (рис. 22.6).

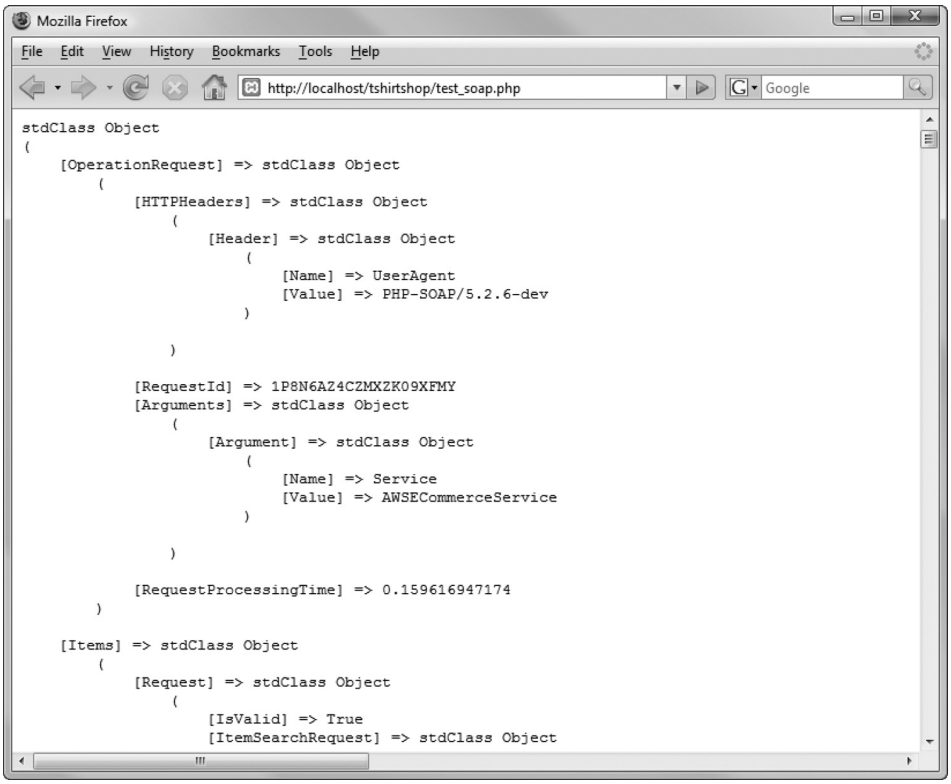


Рис. 22.6. Ответ Amazon.com на SOAP-запрос

Интеграция A2S и TShirtShop

В этом разделе наша задача — отображать на сайте электронного магазина предложения книг от Amazon.com, посвященные футболкам. Эти книги будут выбираться по запросу с ключевыми словами `postal+t-shirt`. Для книг мы создадим на сайте специальный отдел, в котором будем отображать названия книг, сведения об авторах, цены и изображения обложек. Для каждой книги будет также отображаться ссылка Buy from Amazon, которая позволит посетителю купить книгу у Amazon.com. Если вы будете использовать учетную запись сообщества Amazon, то за каждую покупку книги, сделанную через ваш сайт, вы будете получать небольшую сумму в качестве комиссии. Выполнив упражнения из этой главы, вы должны получить примерно такой сайт, какой показан на рис. 22.1.

Приведенная ниже ссылка содержит REST-запрос к Amazon.com для поиска книг по ключевым словам `postal+t-shirt`. Этот запрос возвращает первые десять результатов, выбранных по максимальным рейтингам продаж.

```
http://webservices.amazon.com/onca/xml?Service=AWSECommerceService
&Operation=ItemSearch
&AWSAccessKeyId=[Ваш ключ доступа]
&Keywords=postal+t-shirt
&SearchIndex=Apparel
&ResponseGroup=Request%2CMedium%2CVariationSummary&Sort=salesrank
```

Из возвращаемых запросом товаров мы будем отображать только те, которые доступны для покупки и снабжены изображениями обложек.

Реализация логики уровня приложения

В уровень логики приложения мы добавим код для работы с системой A2S.

Упражнение: изменение уровня логики приложения для работы с A2S

1. Добавьте в файл `include/config.php` следующий код.

```
// Amazon E-Commerce Service
// define('AMAZON_METHOD', 'REST');
define('AMAZON_METHOD', 'SOAP');
define('AMAZON_WSDL',
'http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl');
define('AMAZON_REST_BASE_URL',
'http://webservices.amazon.com/onca/xml?Service=AWSECommerceService');

// Задаем ключ доступа Amazon
define('AMAZON_ACCESS_KEY_ID', '[Your Access Key ID]');

// Задаем идентификатор сообщества Amazon
define('AMAZON_ASSOCIATES_ID', '[Your Amazon associate ID]');

// Задаем параметры запроса
define('AMAZON_SEARCH_KEYWORDS', 'postal t-shirt');
define('AMAZON_SEARCH_NODE', 'Apparel');
define('AMAZON_RESPONSE_GROUPS', 'Request,Medium,VariationSummary');
```

2. Создайте в папке `business` файл `amazon.php` и поместите в него приведенный ниже код. Единственный public-метод в этом коде, который будет вызываться из уровня представления, — метод `GetProducts()`. Все остальные методы предназначены только для вызова из `GetProducts()`.

```
<?php
// Класс для работы с A2S
class Amazon
{
  public function Amazon()
  {
  }

  // Возвращает товары Amazon для передачи уровню представления
  public function GetProducts()
  {
    // Используем SOAP для получения данных
    if (AMAZON_METHOD == 'SOAP')
      $result = $this->_GetDataWithSoap();
    // Используем REST для получения данных
    else
      $result = $this->_GetDataWithRest();

    // Инициализируем объект Array
    $results = array ();

    // Результаты форматирования
    $results = $this->_DataFormat($result);

    // Возвращаем результаты
```

```
  return $results;
}

// Работаем с A2S с помощью REST
private function _GetDataWithRest()
{
  $params = array ('Operation' => 'ItemSearch',
                   'SubscriptionId' => AMAZON_ACCESS_KEY_ID,
                   'Keywords' => AMAZON_SEARCH_KEYWORDS,
                   'SearchIndex' => AMAZON_SEARCH_NODE,
                   'ResponseGroup' => AMAZON_RESPONSE_GROUPS,
                   'Sort' => 'salesrank');

  $query_string = '&';
  foreach ($params as $key => $value)
    $query_string .= $key . '=' . urlencode($value) . '&';

  $amazon_url = AMAZON_REST_BASE_URL . $query_string;

  // Получаем XML-ответ с помощью REST
  $amazon_xml = file_get_contents($amazon_url);

  // Обрабатываем XML и возвращаем результат
  return simplexml_load_string($amazon_xml);
}

// Работаем с A2S с помощью SOAP
private function _GetDataWithSoap()
{
  try
  {
    $client = new SoapClient(AMAZON_WSDL);

    /* Создаем массив для входных параметров, которые нужно
    передать удаленно вызываемой процедуре*/
    $request = array ('SubscriptionId' => AMAZON_ACCESS_KEY_ID,
                      'Request' => array ('Operation' =>
                                          'ItemSearchRequest',
                                            'Keywords' =>
                                          AMAZON_SEARCH_KEYWORDS,
                                            'SearchIndex' =>
                                          AMAZON_SEARCH_NODE,
                                            'ResponseGroup' =>
                                          AMAZON_RESPONSE_GROUPS,
                                            'Sort' => 'salesrank'));

    // Вызываем метод
    $result = $client->ItemSearch($request);

    return $result;
  }
  catch (SoapFault $fault)
  {
    trigger_error('SOAP Fault: (faultcode: ' . $fault->faultcode . ', ' .
                  'faultstring: ' . $fault->faultstring . ')',
                  E_USER_ERROR);
  }
}

/* Подставляет надпись "image not available" для товаров без изображений
   и сохраняет результаты в массив, с которым проще работать коду уровня
```

```
    представления */
  private function _DataFormat($result)
  {
    /* Переменная k - это индекс массива $new_result, в котором
       будут содержаться товары Amazon, отображаемые на сайте TShirtShop */
    $k = 0;
    $new_result = array ();
    /* Анализируем все товары, сведения о которых получены от A2S, и
       сохраняем их в массив $new_result */
    for ($i = 0; $i < count($result->Items->Item); $i++)
    {
      // Делаем временную копию данных о товаре
      $temp = $result->Items->Item[$i];
      /* Задаем файл images/not_available.jpg в качестве
         изображения товара,если URL изображения пустой */
      if (property_exists($temp, 'MediumImage') &
          ((string) $temp->MediumImage->URL) != '')
        $new_result[$k]['image'] = (string) $temp->MediumImage->URL;
      else
        $new_result[$k]['image'] = 'images/not_available.jpg';
      // Сохраняем идентификатор, марку, название и цену товара
      // в массив $new_result array
      $new_result[$k]['asin'] = (string) $temp->ASIN;
      $new_result[$k]['brand'] = (string) $temp->ItemAttributes->Brand;
      $new_result[$k]['item_name'] = (string) $temp->ItemAttributes->Title;
      if (property_exists($temp, 'VariationSummary') &&
          property_exists($temp->VariationSummary, 'LowestPrice'))
      {
        $new_result[$k]['price'] =
          (string) $temp->VariationSummary->LowestPrice->FormattedPrice;

        $highest_price = $new_result[$k]['price'];

        if (property_exists($temp->VariationSummary, 'HighestPrice'))
          $highest_price =
            (string) $temp->VariationSummary->HighestPrice->FormattedPrice;

        if ($highest_price !== $new_result[$k]['price'])
          $new_result[$k]['price'] .= ' - ' . $highest_price;
      }
      else
        $new_result[$k]['price'] = '';

      $k++;
    }

    return $new_result;
  }
}
?>
```

Как это работает: взаимодействие с AWS

Единственный `public`-метод в коде, который мы только что написали, — `GetProducts()`. Он выполняет все действия, необходимые для получения от AWS сведений о товарах. Его код довольно прямолинеен, но он вызывает несколько других методов для выполнения вспомогательных операций. Сначала он определяет, нужно ли использовать SOAP или REST, проверяя значение константы из файла `include/config.php`.

```
define('AMAZON_METHOD', 'SOAP');
```

Константа AMAZON_METHOD, определенная в файле include/config.php, указывает метод, который будет использоваться для работы с AWS. Значение этой константы (REST или SOAP) определяет, какой метод будет вызываться, _GetDataWithRest() или _GetDataWithSoap(). Но результаты должны быть одинаковыми при использовании любого метода.

```
// Используем SOAP для получения данных
if (AMAZON_METHOD == 'SOAP')
  $result = $this->_GetDataWithSoap();
// Используем REST для получения данных
else
  $result = $this->_GetDataWithRest();
```

Методы _GetDataWithRest() и _GetDataWithSoap() возвращают список товаров в виде объекта. После этого мы с помощью метода _DataFormat() обрабатываем список и преобразуем его в ассоциативный массив. Кроме того, метод _DataFormat() помещает рисунок с надписью image unavailable в описания товаров, для которых нет изображений.

```
  // Инициализируем объект Array
  $results = array ();

  // Результаты форматирования
  $results = $this->_DataFormat($result);

  // Возвращаем результаты
  return $results;
}
```

Методы _GetDataWithRest() и _GetDataWithSoap(), непосредственно отправляющие запрос к Amazon и получающие ответ, также не слишком сложны. Метод _GetDataWithRest() получает данные с помощью REST. Сначала он составляет строку запроса для Amazon из отдельных параметров.

```
// Работаем с A2S с помощью REST
private function _GetDataWithRest()
{
  $params = array ('Operation' => 'ItemSearch',
                   'SubscriptionId' => AMAZON_ACCESS_KEY_ID,
                   'Keywords' => AMAZON_SEARCH_KEYWORDS,
                   'SearchIndex' => AMAZON_SEARCH_NODE,
                   'ResponseGroup' => AMAZON_RESPONSE_GROUPS,
                   'Sort' => 'salesrank');

  $query_string = '&';
  foreach ($params as $key => $value)
    $query_string .= $key . '=' . urlencode($value) . '&';
```

Полностью URL запроса составляется из базового URL, который хранится в константе в файле include/config.php, и строки параметров, которую генерирует приведенный выше код. Готовый URL запроса помещается в переменную $amazon_url.

```
$amazon_url = AMAZON_REST_BASE_URL . $query_string;
```

С помощью функции file_get_contents() мы отправляем простой запрос GET на сервер Amazon. Это, по сути, то же самое, что ввод URL в поле адреса в браузере.

```
    // Получаем XML-ответ с помощью REST
    $amazon_xml = file_get_contents($amazon_url);
```

В переменной $amazon_xml будут XML-данные, возвращенные веб-службой в ответ на запрос. Чтобы их обработать, воспользуемся функцией simplexml_load_string(), которая возвращает объект класса SimpleXMLElement. В этом объекте будут содержаться данные из исходного XML-текста. Более подробно эта функция описана в документации по адресу http://www.php.net/manual/en/function.simplexml-load-string.php.

```
    // Обрабатываем XML и возвращаем результат
    return simplexml_load_string($amazon_xml);
```

Метод `_GetDataWithSoap()` работает почти так же, как `_GetDataWithRest()`, но использует SOAP для выполнения операции `ItemSearch`. Как работает код этого метода, мы уже рассматривали ранее в этой главе, в примере, посвященном SOAP.

Реализация уровня представления

Давайте создадим компонентный шаблон, который будет отображать полученные от веб-службы данные. Кроме того, нам понадобится модифицировать компонентный шаблон `departments_list`, включив в него новый отдел.

Упражнение: отображение товаров Amazon.com в TShirtShop

1. Добавьте в папку `presentation/templates` новый файл `amazon_products_list.tpl` и поместите в него такой код.

```
{* amazon_products_list.tpl *}
{load_presentation_object filename="amazon_products_list" assign="obj"}
<h1>{$obj->mDepartmentName}</h1>
<p class="description">{$obj->mDepartmentDescription}</p>
<table class="product-list">
  <tbody>
{section name=k loop=$obj->mProducts}
  {if $smarty.section.k.index % 2 == 0}
    <tr>
  {/if}
    <td valign="top">
      <h3 class="product-title">
        <a href="{$obj->mProducts[k].link_to_product}">
          {$obj->mProducts[k].item_name}
        </a>
        <br />
        by {$obj->mProducts[k].brand}
      </h3>
      <p>
        {if $obj->mProducts[k].image neq ""}
        <a href="{$obj->mProducts[k].link_to_product}">
          <img src="{$obj->mProducts[k].image}"
           alt="{$obj->mProducts[k].item_name}" width="120" />
        </a>
        {/if}
      </p>
        <p class="attributes">
          {if $obj->mProducts[k].price neq ""}
          Price: <font class="price">{$obj->mProducts[k].price}</font>
          {/if}
        </p>
        <p class="section">
          <a target="_blank" href="{$obj->mProducts[k].link_to_product}">
           Buy From Amazon
          </a>
        </p>
      </td>
  {if $smarty.section.k.index % 2 != 0 && !$smarty.section.k.first ||
      $smarty.section.k.last}
    </tr>
```

```
  {/if}
{/section}
  </tbody>
</table>
```

2. Создайте в папке `presentation` файл `amazon_products_list.php` и поместите в него такой код.

```
<?php
// Класс для работы с данными, полученными от AWS
class AmazonProductsList
{
  // Public-переменные, доступные в шаблоне Smarty
  public $mProducts;
  public $mDepartmentName;
  public $mDepartmentDescription;

  // Конструктор
  public function __construct()
  {
    $this->mDepartmentName = AMAZON_DEPARTMENT_TITLE;
    $this->mDepartmentDescription = AMAZON_DEPARTMENT_DESCRIPTION;
  }

  public function init()
  {
    $amazon = new Amazon();
    $this->mProducts = $amazon->GetProducts();

    for ($i = 0;$i < count($this->mProducts); $i++)
      $this->mProducts[$i]['link_to_product'] =
        'http://www.amazon.com/exec/obidos/ASIN/' .
        $this->mProducts[$i]['asin'] . '/ref=nosim/' .
AMAZON_ASSOCIATES_ID;
  }
}
?>
```

3. Добавьте следующие строки в конец файла `include/config.php`.

```
// Параметры отображения отдела Amazon.com
define('AMAZON_DEPARTMENT_TITLE', 'Amazon T-Shirts');
define('AMAZON_DEPARTMENT_DESCRIPTION',
       'Browse these wonderful t-shirts that Amazon.com offers');
```

4. Вставьте в файл `presentation/templates/departments_list.tpl` код, выделенный полужирным шрифтом в листинге ниже, чтобы отображать отдел Amazon T-Shirts.

```
    <li>
      {* Генерируем ссылку для нового отдела в списке *}
      <a {$selected} href="{$obj->mDepartments[i].link_to_department}">
        {$obj->mDepartments[i].name}
      </a>
    </li>
  {/section}
    {assign var=selected value=""}
    {if $obj->mAmazonSelected}
      {assign var=selected value="class=\"selected\""}
    {/if}
    <li>
      <a {$selected} href="{$obj->mLinkToAmazonDepartment}">
        {$obj->mAmazonDepartmentName}
```

```
      </a>
    </li>
  </ul>
</div>
```

5. Вставьте в файл `presentation/departments_list.php` код, выделенный полужирным шрифтом в следующем листинге.

```
<?php
// Отвечает за список отделов
class DepartmentsList
{
  /* Public-переменные, доступные в шаблоне Smarty departments_list.tpl */
  public $mSelectedDepartment = 0;
  public $mDepartments;
  public $mAmazonSelected = false;
  public $mAmazonDepartmentName;
  public $mLinkToAmazonDepartment;
  // Конструктор считывает строку запроса как параметр
  public function __construct()
  {
    /* Если в строке запроса есть DepartmentId, мы посещаем отдел */
    if (isset ($_GET['DepartmentId']))
      $this->mSelectedDepartment = (int)$_GET['DepartmentId'];
    elseif (isset($_GET['ProductId']) &&
            isset($_SESSION['link_to_continue_shopping']))
    {
      $continue_shopping =
        Link::QueryStringToArray($_SESSION['link_to_continue_shopping']);
      if (array_key_exists('DepartmentId', $continue_shopping))
        $this->mSelectedDepartment =
          (int)$continue_shopping['DepartmentId'];
    }

    // Задаем название отдела Amazon и создаем ссылку на него
    $this->mAmazonDepartmentName = AMAZON_DEPARTMENT_TITLE;
    $this->mLinkToAmazonDepartment = Link::ToAmazonDepartment();

    // Проверяем, выбран ли отдел Amazon
    if ((isset ($_GET['DepartmentId'])) &&
        ((string) $_GET['DepartmentId'] == 'Amazon'))
      $this->mAmazonSelected = true;
  }
...
```

6. Добавьте в класс `Link` в файле `presentation/link.php` следующий метод.

```
// Создает ссылку на отдел Amazon T-Shirts
public static function ToAmazonDepartment()
{
  return self::Build('amazon-t-shirts/');
}
```

7. В том же классе модифицируйте метод `CheckRequest()`, чтобы он ничего не делал, если мы обращаемся к отделу Amazon T-Shirts.

```
...
    elseif (isset ($_GET['DepartmentId']))
    {
      if ((string) $_GET['DepartmentId'] == 'Amazon') return;

      if (isset ($_GET['Page']))
        $proper_url = self::ToDepartment($_GET['DepartmentId'],
```

```
                                          $_GET['Page']);
      else
        $proper_url = self::ToDepartment($_GET['DepartmentId']);
}
...
```

8. Откройте файл `.htaccess` из корневой папки проекта и добавьте в него правило для переписывания ссылок на отдел Amazon T-Shirts.

```
...
  # Переписываем страницы сообщений об ошибках в заказах
  RewriteRule ^order-error/?$ index.php?OrderError [L]

  # Переписываем страницы отдела Amazon t-shirts
  RewriteRule ^amazon-t-shirts/?$ index.php?DepartmentId=Amazon [L]
</IfModule>
```

9. Вставьте ссылку на новый шаблон в файл `include/index.php`, над строкой, вызывающей метод проверки ссылок.

```
require_once BUSINESS_DIR . 'amazon.php';

// Коррекция URL
Link::CheckRequest();

// Загрузка файла шаблона Smarty
$application = new Application();
```

10. Откройте файл `presentation/store_front.php` и модифицируйте метод `init()` в классе `StoreFront`, чтобы он мог загружать страницу отдела Amazon T-Shirts по запросу:

```
...
    // Загружаем подробные сведения об отделе на страницу отдела
    if (isset ($_GET['DepartmentId']))
    {
      if ((string) $_GET['DepartmentId'] == 'Amazon')
        $this->mContentsCell = 'amazon_products_list.tpl';
      else
      {
        $this->mContentsCell = 'department.tpl';
        $this->mCategoriesCell = 'categories_list.tpl';
      }
    }
...
```

11. Откройте страницу TShirtShop в браузере и попытайтесь перейти в отдел Amazon T-Shirts. Страница отдела должна выглядеть примерно так, как показано на рис. 22.1.

Как это работает: отображение товаров Amazon.com в TShirtShop

В этом упражнении мы просто реализовали отображение товаров Amazon.com в нашем приложении, используя методы, рассмотренные в первых главах книги. Новая функциональность не содержит ничего сложного, и подробно рассматривать весь новый код мы не будем.

Чтобы изменить метод доступа к веб-службам, раскомментируйте нужную строку в файле `include/config.php`.

```
// Amazon E-Commerce Service
define('AMAZON_METHOD', 'REST');
//define('AMAZON_METHOD', 'SOAP');
```

Если пользователь щелкнет на какой-нибудь из ссылок `Buy From Amazon`, сайт Amazon отнесет эту покупку на счет вашего идентификатора сообщества (этот идентификатор указан в коде и передается в ссылках). В методе `init()` класса `AmazonProductsList` для получения данных о товарах вызывается метод `GetProducts` класса `Amazon`. Полученные данные используются для генерации ссылок на товары на сайте Amazon.com.

```
public function init()
{
  $amazon = new Amazon();
  $this->mProducts = $amazon->GetProducts();

  for ($i = 0;$i < count($this->mProducts); $i++)
    $this->mProducts[$i]['link'] =
      'http://www.amazon.com/exec/obidos/ASIN/' .
      $this->mProducts[$i]['asin'] .
      '/ref=nosim/' . AMAZON_ASSOCIATES_ID;
}
```

Но не забывайте, что Amazon.com позволяет генерировать ссылки на свои товары множеством способов. Зайдя в раздел `Associates`, вы увидите различные варианты ссылок, которые можете интегрировать в свой веб-сайт.

Пожалуй, самая интересная возможность, предлагаемая Amazon, — это возможность создавать специальные корзины покупателей с помощью PHP и API Amazon.com. Если вы действительно заинтересованы в интеграции своего электронного магазина с Amazon, вам стоит изучить эту возможность.

Резюме

В этой главе вы научились работать с веб-службами Amazon.com E-Commerce Service с помощью REST и SOAP. Вы сможете использовать методы, рассмотренные в настоящей главе, и для доступа к другой функциональности этих веб-служб.

Поздравляем! На этом разработка сайтов электронных магазинов с помощью PHP и MySQL завершена. Теперь вы знаете, как создавать такие сайты (возможно, даже более мощные и интересные, чем тот, который мы создали в этой книге).

В главах 1–11 мы рассмотрели первый этап разработки — создание каркаса сайта и реализацию основных функций. К концу главы 11 сайт мог принимать и обрабатывать заказы покупателей.

Если более подробно, в главах 1–3 мы обсуждали электронные магазины в целом и проект электронного магазина, который хотели создать, — TshirtShop. Мы перечислили программы, которые для этого потребовались, установили их и настроили. Кроме того, мы обсудили трехуровневую архитектуру.

В главах 4–6 мы создали сначала базу данных `tshirtshop` на сервере MySQL с помощью phpMyAdmin, а затем — каталог товаров на основе этой базы данных. Вы узнали, как создавать сохраненные процедуры в MySQL, вызывать их из логики уровня приложения и отображать данные из базы данных с помощью объектов представления и шаблонов Smarty.

В главах 7 и 8 мы создали код для поиска товаров в базе данных через каталог товаров на сайте `tshirtshop`, а также оптимизировали наш сайт для взаимодействия с поисковыми системами. Эти возможности необходимы для успеха электронного магазина.

В главе 9 мы обсудили получение платежей от покупателей с помощью PayPal. Обработка платежей — необходимый элемент электронного магазина, и в начале разработки проще всего воспользоваться службой обработки платежей, которая выполнит вместо нас большую часть работы.

В главах 10 и 11 мы создали механизмы администрирования сайта TShirtShop, которые позволяют легко редактировать сведения об отделах и категориях, а также описания и цены товаров. Кроме того, механизмы администрирования позволяют удалять товары из каталога или добавлять в каталог новые товары и выполнять другие служебные операции.

Главы 12–15 посвящены второй стадии создания электронного магазина, в которой мы заменили корзину покупателя, предоставленную PayPal, корзиной собственной разработки. Кроме того, мы добавили поддержку AJAX и динамических рекомендаций товаров, сделав сайт более привлекательным для посетителей. В конце этой стадии сайт тоже полностью готов к работе и может принимать и обрабатывать заказы.

Глава 12 посвящена созданию собственной корзины покупателя и ее применению на сайте вместо корзины PayPal.

В главе 13 мы рассмотрели применение AJAX в каталоге товаров и корзине покупателя для повышения отзывчивости сайта. Улучшения не были радикальными, но привередливые покупатели наверняка заметили, что сайт стал реагировать на их действия быстрее. Кроме того, умение пользоваться AJAX очень пригодится вам при разработке веб-проектов.

В главе 14 мы создали механизм обработки заказов, взаимодействующий с корзиной покупателя, созданной в главе 12.

В главе 15 мы рассмотрели анализ заказов покупателей и использование полученной информации для выдачи динамических рекомендаций товаров.

Главы 16–22 посвящены третьей стадии создания электронного магазина, на которой мы добавили в проект вспомогательные возможности. К концу главы 22 наш сайт работал с кредитными картами и получил полностью рабочий конвейер обработки заказов, поддержку отзывов о товарах и возможность работать с веб-службами Amazon.com.

В главе 16 был создан механизм поддержки учетных записей посетителей. Этот механизм позволил хранить сведения о заказах каждого отдельного посетителя, а также другие необходимые сведения о нем — адрес доставки, информацию о кредитной карте и т.д. Для обеспечения безопасного хранения информации на сервере использовались средства шифрования.

В главах 17–19 мы обновили базу данных для отслеживания обработки каждого заказа на конвейере и начали хранить информацию о посетителях и их заказах в базе данных.

В главе 20 рассмотрены механизмы работы с кредитными картами на примере двух компаний, предоставляющих такие услуги, DataCash и Authorize.net.

В главе 21 была реализована поддержка отзывов посетителей о товарах — такая возможность есть в большинстве современных электронных магазинов.

И наконец, в главе 22 мы рассмотрели интеграцию нашего электронного магазина с веб-службами Amazon.com для получения дополнительного дохода.

Поздравляем вас с успешным завершением проекта TShirtShop! Мы надеемся, что вам понравилась эта книга, и желаем вам успехов в других веб-проектах!

Предметный указатель

.

.htaccess, файл, 196; 486; 493

_

_GetCurrentPipelineSection(), метод класса OrderProcessor, 551
_IsSensitivePage(), метод класса StoreFront, 497

3

3DES, *452*

4

404.php, файл, 219

5

500.php, файл, 216

A

ACID, *227*
AddDepartment(), метод класса AdminDepartment, 285
admin.php, файл, 272
admin_attribute_values.php, файл, 302
admin_attribute_values.tpl, файл, 301
admin_attributes.php, файл, 300
admin_attributes.tpl, файл, 299
admin_carts.tpl, файл, 373
admin_categories.php, файл, 290
admin_categories.tpl, файл, 288
admin_departments.php, файл, 282
admin_departments.tpl, файл, 281
admin_login.php, файл, 275
admin_login.tpl, файл, 275
admin_menu.php, файл, 276
admin_menu.tpl, файл, 276
admin_order_details.php, файл, 428
admin_order_details.tpl, файл, 427
admin_orders.php, файл, 422
admin_orders.tpl, файл, 421
admin_product_details.php, файл, 320
admin_product_details.tpl, файл, 317
admin_products.php, файл, 311
admin_products.tpl, файл, 310
AdminDepartment, класс
  AddDepartment(), метод, 285
  DeleteDepartment(), метод, 285
  GetDepartmentsWithDescriptions(), метод, 285
  UpdateDepartment(), метод, 285
AdminOrderDetails, класс, 428
AdminOrders, класс, 422
AES, *452*
AJAX, *28*; *377*
ajax.js, файл, 392
Alexa, *604*
ALTER PROCEDURE, оператор, 98
Amazon Web Services, *602*
Amazon, класс, 612
Amazon.com, *29*
amazon.php, файл, 612
amazon_products_list.php, файл, 617
amazon_products_list.tpl, файл, 616
application.php, файл, 114
AssignAttributeValueToProduct(), метод класса Catalog, 329
AssignProductToCategory(), метод класса Catalog, 328
attribute, таблица, 183
attribute_value, таблица, 183
audit, таблица, 529
AUTH_ONLY, транзакции, 584
Authorize.net, *29*; *568*
authorize_net_request.php, файл, 584

AuthorizeNetRequest, класс, 584
AWS, *602*

B

blank.tpl, файл, 157
Build(), метод класса Link, 276

C

Call stack, *103*
cart_details.php, файл, 364
cart_details.tpl, файл, 367
cart_summary.php, файл, 362
cart_summary.tpl, файл, 363
Catalog, класс
  AssignAttributeValueToProduct(), метод, 329
  AssignProductToCategory(), метод, 328
  CreateProductReview(), метод, 595
  DeleteProduct(), метод, 327
  GetAttributesNotAssignedToProduct(), метод, 329
  GetCategories(), метод, 328
  GetCategoriesForProduct(), метод, 328
  GetCategoriesInDepartment(), метод, 151
  GetCategoryDetails(), метод, 151
  GetDepartmentDetails(), метод, 150
  GetDepartments(), метод, 111
  GetProductDetails(), метод, 155
  GetProductLocations(), метод, 155
  GetProductReviews(), метод, 595
  GetProductsInCategory(), метод, 153
  GetProductsOnCatalog(), метод, 154
  GetProductsOnDepartment(), метод, 154
  HowManyPages(), метод, 151
  MoveProductToCategory(), метод, 329
  RemoveProduct(), метод, 327
  RemoveProductAttributeValue(), метод, 329
  Search(), метод, 233
  SetImage(), метод, 330
  SetImage2(), метод, 330
  SetProductDisplayOptions(), метод, 328
  SetThumbnail(), метод, 330
  UpdateProduct(), метод, 327
catalog.php, файл, 109
catalog_add_attribute, хранимая процедура, 307
catalog_add_category, хранимая процедура, 293
catalog_add_department, хранимая процедура, 287
catalog_assign_attribute_to_product, хранимая процедура, 333
catalog_assign_product_to_category, хранимая процедура, 332
catalog_count_products_in_category, хранимая процедура, 143
catalog_count_products_on_catalog, хранимая процедура, 147
catalog_count_products_on_department, хранимая процедура, 145
catalog_count_search_result, хранимая процедура, 231
catalog_create_product_review, хранимая процедура, 595
catalog_delete_attribute, хранимая процедура, 307
catalog_delete_category, хранимая процедура, 293
catalog_delete_department, хранимая процедура, 287
catalog_delete_product, хранимая процедура, 331; 370
catalog_get_attributes, хранимая процедура, 307
catalog_get_attributes_not_assigned_to_product, хранимая процедура, 333
catalog_get_categories, хранимая процедура, 332
catalog_get_categories_for_product, хранимая процедура, 332
catalog_get_categories_list, хранимая процедура, 142
catalog_get_category_details, хранимая процедура, 143
catalog_get_category_name, хранимая процедура, 203
catalog_get_department_categories, хранимая процедура, 293
catalog_get_department_details, хранимая процедура, 141
catalog_get_department_name, хранимая процедура, 203
catalog_get_departments, хранимая процедура, 287
catalog_get_departments_list, хранимая процедура, 98

catalog_get_product_attributes, хранимая процедура, 184
catalog_get_product_details, хранимая процедура, 147
catalog_get_product_info, хранимая процедура, 332
catalog_get_product_locations, хранимая процедура, 148
catalog_get_product_name, хранимая процедура, 203
catalog_get_product_reviews, хранимая процедура, 595
catalog_get_products_in_category, хранимая процедура, 143
catalog_get_products_on_catalog, хранимая процедура, 147
catalog_get_products_on_department, хранимая процедура, 145
catalog_get_recommendations, хранимая процедура, 437
catalog_move_product_to_category, хранимая процедура, 333
catalog_remove_product_attribute, хранимая процедура, 333
catalog_remove_product_from_category, хранимая процедура, 331
catalog_search, хранимая процедура, 232
catalog_set_image, хранимая процедура, 334
catalog_set_image2, хранимая процедура, 334
catalog_set_product_display_option, хранимая процедура, 332
catalog_set_thumbnail, хранимая процедура, 334
catalog_update_attribute, хранимая процедура, 307
catalog_update_category, хранимая процедура, 293
catalog_update_department, хранимая процедура, 287
catalog_update_product, хранимая процедура, 331
catch, блок, 102
categories_list.php, файл, 162
CategoriesList, класс, 162
category, таблица, 131
checkout_info.php, файл, 490
checkout_info.tpl, файл, 490
checkout_not_logged.tpl, файл, 492
CheckRequest(), метод класса Link, 207; 255; 397
CleanUrlText(), метод класса Link, 204
Close(), метод класса DatabaseHandler, 106
CNP, транзакции, 569
config.php, файл, 120; 254
Cookie-файлы, *348*
CreateAudit(), метод класса Orders, 529
CreateOrder(), метод класса ShoppingCart, 411; 501; 517
CreateProductReview(), метод класса Catalog, 595
cURL, *573*
Customer, класс, 469
customer, таблица, 447; 500
customer.php, файл, 469
customer_add, хранимая процедура, 467
customer_address.php, файл, 480
customer_address.tpl, файл, 479
customer_credit_card.php, файл, 484
customer_credit_card.tpl, файл, 483
customer_details.php, файл, 477
customer_details.tpl, файл, 475
customer_get_customer, хранимая процедура, 467
customer_get_login_info, хранимая процедура, 467
customer_get_shipping_regions, хранимая процедура, 468
customer_logged.php, файл, 474
customer_logged.tpl, файл, 474
customer_login.php, файл, 473
customer_login.tpl, файл, 472
customer_update_account, хранимая процедура, 467
customer_update_address, хранимая процедура, 468
customer_update_credit_card, хранимая процедура, 468
Customers, класс GetCustomersList(), метод, 506
customers_get_customers_list, хранимая процедура, 505

D

database_handler.php, файл, 105
DatabaseHandler, класс, 105
  Close(), метод, 106
  Execute(), метод, 105; 106

GetAll(), метод, 105; 107
GetHandler(), метод, 105
GetOne(), метод, 105; 108
GetRow(), метод, 105
DataCash, *29*; *568*
datacash_request.php, файл, 573
DataCashRequest, класс, 573
DB 2, *125*
Decrypt(), метод класса SymmetricCrypt, 455
DELETE, оператор, 97
DeleteDepartment(), метод класса AdminDepartment, 285
DeleteProduct(), метод класса Catalog, 327
Department, класс, 157
department, таблица
создание, 91
department.tpl, файл, 157
departments_list.php, файл, 115
departments_list.tpl, файл, 114
DES, *452*
Document Object Model, *379*
DOM, *379*
DROP PROCEDURE, оператор, 98
DSA, *451*

E

Encrypt(), метод класса SymmetricCrypt, 455
ErrorHandler, класс, 69
Handler(), метод, 69
Exception handling, *102*
Execute(), метод
класса DatabaseHandler, 105; 106

F

Field, *85*
FULLTEXT-индек, *226*
function.load_presentation_object.php, файл, 115

G

GetAll(), метод
класса DatabaseHandler, 105; 107
GetAttributesNotAssignedToProduct(), метод класса Catalog, 329
GetAuditTrail(), метод класса Orders, 561
GetByCustomerId(), метод класса Orders, 506
GetCategories(), метод класса Catalog, 328
GetCategoriesForProduct(), метод класса Catalog, 328
GetCategoriesInDepartment(), метод класса Catalog, 151
GetCategoryDetails(), метод класса Catalog, 151
GetCustomerAddressString(), метод класса OrderProcessor, 541
GetCustomersList(), метод класса Customers, 506
GetDepartmentDetails(), метод класса Catalog, 150
GetDepartments(), метод класса Catalog, 111
GetDepartmentsWithDescriptions(), метод класса AdminDepartment, 285
GetHandler(), метод
класса DatabaseHandler, 105
GetMailBody(), метод класса PsInitialNotification, 545
GetOne(), метод
класса DatabaseHandler, 105; 108
GetOrderAsString(), метод класса OrderProcessor, 541
GetOrderShortDetails(), метод класса Orders, 506
GetProductLocations(), метод класса Catalog, 155
GetProductReviews(), метод класса Catalog, 595
GetProductsDetail(), метод класса Catalog, 155
GetProductsInCategory(), метод класса Catalog, 153
GetProductsOnCatalog(), метод класса Catalog, 154
GetProductsOnDepartment(), метод класса Catalog, 154
GetRow(), метод
класса DatabaseHandler, 105; 107
GetShippingInfo(), метод класса Orders, 517
GnuPG, *451*

H

Handler(), метод, 69
HEAP, *131*

HowManyPages(), метод класса Catalog, 151
HTTP, коды состояния, 206

I

i_pipeline_section.php, файл, 532
Images, группа ответов AWS, 607
index.php, файл, 109; 116
InnoDB, *131*; *227*
INSERT, оператор, 95

J

JavaScript, *378*

L

Link, класс
  Build(), метод, 276
  CheckRequest(), метод, 207; 255; 397
  CleanUrlText(), метод, 204
  QueryStringToArray(), метод, 176
  ToAddProduct(), метод, 256
  ToAdmin(), метод, 276
  ToAttributesAdmin(), метод, 305
  ToAttributeValuesAdmin(), метод, 305
  ToCart(), метод, 368
  ToCart(), метод класса Link, 360
  ToCartsAdmin(), метод, 373
  ToCategory(), метод, 162; 168
  ToCategoryProductsAdmin(), метод, 313
  ToCheckout(), метод, 492
  ToDepartment(), метод, 168
  ToDepartmentCategoriesAdmin(), метод, 292
  ToDepartmentsAdmin(), метод, 284
  ToIndex(), метод, 172
  ToOrderDetailsAdmin(), метод, 418
  ToOrdersAdmin(), метод, 418
  ToProductAdmin(), метод, 324
  ToSearch(), метод, 238
  ToSearchResults(), метод, 238
link.php, файл, 120; 255; 276

M

MailCustomer(), метод класса OrderProcessor, 540
MailSupplier(), метод класса OrderProcessor, 540
mcrypt, библиотека, 455
MD5, *448*
mod_rewrite, модуль Apache, 196
MoveProductToCategory(), метод класса Catalog, 329
MyISAM, *130*; *227*
MySQL, *125*

O

Object detection, *382*
open(), метод объекта XMLHttpRequest, 383
Oracle, *125*
order_detail, таблица, 408
order_done.tpl, файл, 558
order_error.tpl, файл, 558
order_processor.php, файл, 530
OrderProcessor, класс, 530
  _GetCurrentPipelineSection(), метод, 551
  GetCustomerAddressString(), метод, 541
  GetOrderAsString(), метод, 541
  MailCustomer(), метод, 540
  MailSupplier(), метод, 540
  SetDateShipped(), метод, 540
  SetOrderAuthCodeAndReference(), метод, 539
  UpdateOrderStatus(), метод, 538
Orders, класс, 420
  CreateAudit(), метод, 529
  GetAuditTrail(), метод, 561
  GetByCustomerId(), метод, 506
  GetOrderShortDetails(), метод, 506
  GetShippingInfo(), метод, 517
  SetDateShipped(), метод, 539
  SetOrderAuthCodeAndReference(), метод, 539
  Update(), метод, 506
  UpdateOrderStatus(), метод, 538
orders, таблица, 407; 500
orders.php, файл, 420
orders_create_audit, хранимая процедура, 529
orders_get_audit_trail, хранимая процедура, 561
orders_get_most_recent_orders, хранимая процедура, 419; 503
orders_get_order_details, хранимая процедура, 425

orders_get_order_info, хранимая процедура, 425; 504; 516
orders_get_order_short_detail, хранимая процедура, 505
orders_get_orders_between_dates, хранимая процедура, 420; 504
orders_get_orders_by_customer_id, хранимая процедура, 505
orders_get_orders_by_status, хранимая процедура, 420; 504
orders_get_shipping_info, хранимая процедура, 516
orders_set_auth_code, хранимая процедура, 538
orders_set_date_shipped, хранимая процедура, 539
orders_update_order, хранимая процедура, 425; 504; 560
orders_update_status, хранимая процедура, 538

P

password_hasher.php, файл, 449
PasswordHasher, класс, 449
PayPal, *28*; *247*
  документация, 249
  недостатки, *29*
  учетная запись, создание, 249
PDO, 43
  методы, 104
  подготовленные операторы, 104; 145
  создание объекта, 101
Persistent connections, *100*
PGP, *451*
PHP, *27*
  catch, блок, 102
  try, блок, 102
  исключения, 102
  обработка ошибок, 68; 102
PHP Data Objects, 43
PostgreSQL, *125*
Prepared statement, *145*
PRIOR_AUTH_ONLY, транзакции, 584
Product, класс, 174
product.tpl, файл, 188; 256
product_attribute, таблица, 184
products, таблица, 133
products.tpl, файл, 174
products_list.php, файл, 167; 188; 254
products_list.tpl, файл, 166; 187; 254
ProductsList, класс, 167
ps_check_funds.php, файл, 545; 580; 590
ps_check_stock.php, файл, 546
ps_dummy.php, файл, 532
ps_final_notification.php, файл, 549
ps_initial_notification.php, файл, 544
ps_ship_ok.php, файл, 548
ps_stock_ok.php, файл, 547
ps_take_payment.php, файл, 548; 581; 591
PsCheckFunds, класс, 545; 580; 590
PsCheckStock, класс, 546
PsDummy, класс, 532
PsFinalNotification, класс, 549
PsInitialNotification, класс, 544
  GetMailBody(), метод, 545
PsShipOk, класс, 548
PsStockOk, класс, 547
PsTakePayment, класс, 548; 581; 591

Q

QueryStringToArray(), метод класса Link, 176

R

RC2, *452*
readyState, свойство объекта XMLHttpRequest, 385
Record, *85*
RemoveProduct(), метод класса Catalog, 327
RemoveProductAttributeValue(), метод класса Catalog, 329
Request, группа ответов AWS, 607
REST, *601*
review, таблица, 595
reviews.php, файл, 597
reviews.tpl, файл, 596
Rijndael, *452*
RSA, *451*

S

SalesRank, группа ответов AWS, 607
Search Engine Optimization, *193*
Search(), метод класса Catalog, 233
search_box.php, файл, 237
search_box.tpl, файл, 237
search_results.tpl, файл, 241

Secure Socket Layer, *29*; *267*
secure_card.php, файл, 456
SecureCard, класс, 456
SELECT, оператор, 94
send(), метод объекта XMLHttpRequest, 383
SEO, *193*
set_error_handler(), функция, 69
SetDateShipped(), метод класса OrderProcessor, 540
SetDateShipped(), метод класса Orders, 539
SetImage(), метод класса Catalog, 330
SetImage2(), метод класса Catalog, 330
SetOrderAuthCodeAndReference(), метод класса OrderProcessor, 539
SetOrderAuthCodeAndReference(), метод класса Orders, 539
SetProductDisplayOptions(), метод класса Catalog, 328
SetThumbnail(), метод класса Catalog, 330
SHA1, *448*
shipping, таблица, 514
shipping_region, таблица, 447
shopping_cart, таблица, 348
shopping_cart.php, файл, 353
shopping_cart_add_product, хранимая процедура, 350
shopping_cart_create_order, хранимая процедура, 410; 501; 515
shopping_cart_empty, хранимая процедура, 409
shopping_cart_get_products, хранимая процедура, 351
shopping_cart_get_recommendations, хранимая процедура, 438
shopping_cart_get_saved_products, хранимая процедура, 352
shopping_cart_get_total_amount, хранимая процедура, 352
shopping_cart_move_product_to_cart, хранимая процедура, 352
shopping_cart_remove_product, хранимая процедура, 351
shopping_cart_save_product_for_later, хранимая процедура, 352
shopping_cart_update, хранимая процедура, 350
ShoppingCart, класс, 353; 372
  CreateOrder(), метод, 411; 501; 517
SimpleXML, *573*
Small, группа ответов AWS, 607
Smarty
  объекты представления, 112
  подключаемые файлы, 112
  установка, 56
SOAP, *601*
SQL, *27*; *75*
SQL Server, *125*
SQL, язык, 94
SSL, *29*; *267*
store_admin.php, файл, 273
store_admin.tpl, файл, 273
store_front.php
  файл, 210
store_front.php, файл, 120; 257
store_front.tpl, файл, 116; 121
Stored procedure, *93*
StoreFront, класс, 210
  _IsSensitivePage(), метод, 497
symmetric_crypt.php, файл, 453
SymmetricCrypt, класс, 453
  Decrypt(), метод, 455
  Encrypt(), метод, 455

T

tax, таблица, 515
ToAddProduct(), метод класса Link, 256
ToAdmin(), метод класса Link, 276
ToAttributesAdmin(), метод класса Link, 305
ToAttributeValuesAdmin(), метод класса Link, 305
ToCart(), метод класса Link, 368
ToCartsAdmin(), метод класса Link, 373
ToCategory(), метод класса Link, 162; 168
ToCategoryProductsAdmin(), метод класса Link, 313
ToCheckout(), метод класса Link, 492
ToDepartment(), метод класса Link, 168
ToDepartmentCategoriesAdmin(), метод класса Link, 292
ToDepartmentsAdmin(), метод класса Link, 284
ToIndex(), метод класса Link, 172
ToOrderDetailsAdmin(), метод класса Link, 418
ToOrdersAdmin(), метод класса Link, 418
ToProductAdmin(), метод класса Link, 324
ToSearch(), метод класса Link, 238
ToSearchResults(), метод класса Link, 238

try, блок, 102
TShirtShop
  входная страница, 61
  задание псевдонима, 54
  псевдоним, задание, 54
  создание базы данных, 75
  создание каркаса сайта, 58
tshirtshop.css, файл, 113

U

Update(), метод класса Orders, 506
UPDATE, оператор, 96
UpdateDepartment(), метод класса AdminDepartment, 285
UpdateOrderStatus(), метод класса OrderProcessor, 538
UpdateOrderStatus(), метод класса Orders, 538
UpdateProduct(), метод класса Catalog, 327
URL с ключевыми словами, *195*

V

VariationSummary, группа ответов AWS, 607
VeriSign Payflow Pro, *29*

W

Web 2.0, *378*
Web-service, *601*

X

X.25, протокол, 567
XAMPP
  установка, 50
XML, *29*
XMLHttpRequest, объект
  open(), метод, 383
  readyState, свойство, 385
  send(), метод, 383
  методы и свойства, 382
  создание, 380

А

Автоинкремент, 90
Авторизация, 267; *569*
Архитектура “клиент/сервер”, 39
Асимметричное шифрование, 451
Аутентификация, 267

Б

Базы данных
  автоинкремент, 90
  внешние ключи, 129
  декартово произведение, 185
  индексы, 90
  ограничения, 88
  отношения, 127
  первичные ключи, 86
  реляционные, *85*
  соединения таблиц, 137
  столбцы, 88
  типы данных, 88
Библиотека
  mcrypt, *452*
Бизнес-уровень, 35

В

Веб-сайт
  MySQL, 43
  PHP, 40
  Smarty, 41
  архитектура “клиент/сервер”, 39
  бизнес-уровень, 35
  входная страница, 61
  выбор технологий, 40
  гибкая архитектура, 35
  задание псевдонима, 54
  компонентные шаблоны, 41
  масштабируемость, 35
  надежность, 35
  обработка ошибок, 68; 102
  оформление кода, 46
  псевдоним, задание, 54
  создание базы данных, 75
  создание каркаса сайта, 58
  трехуровневая архитектура, 35
  уровень данных, 36
  уровень представления, 35
Веб-службы, *601*
Внешние ключи, в базах данных, 129

Г

Группы ответов, *605*
Группы ответов, в AWS, 605

Д

Декартово произведение, 185
Дизайн, *34*
Динамический URL, *195*

З

Завершение транзакции, *569*
Закрытые ключи, 451
Записи, *85*
Запросы
  GET, *384*
  POST, *384*
Защита каналов связи, 267

И

Индексы, в базах данных, 90
Интернет-магазин, *23*
Интерфейсы, в PHP, 528
Исключения, *102*; 381
  в PHP, 102
Исходный код
  оформление, 46
  редакторы, 50

К

Класс
  PDO, *101*
  ShoppingCart, 353
Ключевые слова
  catch, *102*
  throw, *103*
  try, *102*
Коды состояния HTTP, 206
Компонентные шаблоны, 41; 60
Конвейер
  обработки заказов, *524*
  обработки заказов, структура, 524
Конструктор класса, *72*
Корзина покупателя, *346*

М

Модули Apache
  mod_rewrite, *194*

Н

Налоги, *512*
Начальный вектор шифрования, *452*

О

Обнаружение объектов, *382*
Обработка исключений, *102*
Объекты
  document, *379*
  Exception, *103*
  XMLHttpRequest, *378*
  представления Smarty, 112
Ограничение, *86*
Ограничения, в базах данных, 88
Открытые ключи, 451
Отношения, *127*
Отношения, в базах данных, 127
Оформление кода, 46
Ошибки
  PHP, *102*

П

Панель управления, *264*
Первичные ключи, 86
Переписывание URL, 195
Платежные компании, 248
Подготовленный оператор, *145*
Подключаемые файлы Smarty, 112
Поиск
  критерии, *223*
Полнотекстовый поиск
  в MySQL, *226*
Поля, *85*
Постоянные соединения, *100*
Преавторизация, *569*
Протоколы
  HTTPS, *267*

Р

Регулярные выражения, *198*
Редакторы исходного кода, 50
Реляционная база данных, *85*

РСУБД, *125*

С

Связующая таблица, в базах данных, 129
Сертификат безопасности, 267
  получение, 270
Симметричное шифрование, 451
Соединения таблиц, в базах данных, 137
Стек вызовов, *103*
Стоимость доставки, *513*
Столбцы, в базах данных, 88

Т

Таблица
  shopping_cart, *348*
  данных, *85*
  соединения, в базах данных, 129
Типы данных, в базах данных, 88
Транзакция, *524*
Триггеры, *130*

У

Увеличение продаж, *431*
Упаковка, *524*
Уровень
  данных, 36
  представления, 35

Ф

Фабрика ссылок, 119
Файл
  .htaccess, 196; 486; 493
  404.php, 219
  500.php, 216
  admin.php, 272
  admin_attribute_values.php, 302
  admin_attribute_values.tpl, 301
  admin_attributes.php, 300
  admin_attributes.tpl, 299
  admin_carts.tpl, 373
  admin_categories.php, 290
  admin_categories.tpl, 288
  admin_departments.php, 282
  admin_departments.tpl, 281
  admin_login.php, 275
  admin_login.tpl, 275
  admin_menu.php, 276
  admin_menu.tpl, 276
  admin_order_details.php, 428
  admin_order_details.tpl, 427
  admin_orders.php, 422
  admin_orders.tpl, 421
  admin_product_details.php, 320
  admin_product_details.tpl, 317
  admin_products.php, 311
  admin_products.tpl, 310
  ajax.js, 392
  application.php, 64; 114
  blank.tpl, 157
  cart_details.php, 364
  cart_summary.php, 362
  cart_summary.tpl, 363; 367
  catalog.php, 109
  categories_list.php, 162
  checkout_info.php, 490
  checkout_info.tpl, 490
  checkout_not_logged.tpl, 492
  config.php, 64; 69; 120; 254
  customer.php, 469
  customer_address.php, 480
  customer_address.tpl, 479
  customer_credit_card.php, 484
  customer_credit_card.tpl, 483
  customer_details.php, 477
  customer_details.tpl, 475
  customer_logged.php, 474
  customer_logged.tpl, 474
  customer_login.php, 473
  customer_login.tpl, 472
  database_handler.php, 105
  department.tpl, 157
  departments_list.php, 115
  departments_list.tpl, 114
  error_handler.php, 70
  function.load_presentation_object.php, 115
  index.php, 61; 72; 109; 116
  link.php, 120; 255; 276
  orders.php, 420
  password_hasher.php, 449
  product.tpl, 188; 256
  products.tpl, 174
  products_list.php, 167; 188; 254
  products_list.tpl, 166; 187; 254
  search_box.php, 237
  search_box.tpl, 237
  search_results.tpl, 241

secure_card.php, 456
shopping_cart.php, 353
site.conf, 61
store_admin.php, 273
store_admin.tpl, 273
store_front.php, 120; 210; 257
store_front.tpl, 61; 62; 116; 121
symmetric_crypt.php, 453
tshirtshop.css, 63; 113
Функции
urlencode(), *251*

Х

Хеширование, 271; *448*
Хеш-код, *446*
Хранимая процедура, *93*; 97
catalog_ count_products_on_catalog, 147
catalog_count_products_in_category, 143
catalog_count_products_on_department, 145
catalog_get_categories_list, 142
catalog_get_category_details, 143
catalog_get_department_details, 141
catalog_get_departments_list, 98
catalog_get_product_details, 147
catalog_get_product_locations, 148
catalog_get_products_in_category, 143
catalog_get_products_on_catalog, 147
catalog_get_products_on_department, 145
создание, *141*

Ш

Шаблоны
admin_product, *324*
products_list, *224*
reviews, *598*
search_box, *239*
Шлюзы оплаты, 568

Э

Электронный магазин, *23*
выбор технологий, 40
задачи, 24
преимущества, 24
риски и угрозы, 26

PHP: объекты, шаблоны и методики программирования, 2-е издание

Мэтт Зандстра



www.williamspublishing.com

В этой книге изложены методики объектно-ориентированного программирования и проектирования с точки зрения программиста на PHP.

Книга начинается с обзора объектно-ориентированных возможностей PHP, в который включены важные темы, наподобие определения класса, создания объектов, наследовании, инкапсуляции методов и свойств. Вы изучите также и дополнительные темы, такие как статические методы и свойства, абстрактные классы, обработка исключений, клонирование объектов и много другое.

Следующая часть книги посвящена шаблонам проектирования, которые органически дополняют тему ООП. В ней описываются концепции шаблонов проектирования и показаны способы реализации нескольких важных шаблонов в приложениях на PHP.

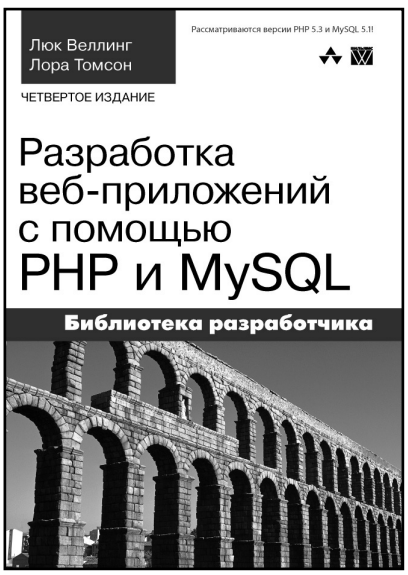
В последней части книги описывается несколько важных утилит, облегчающих процесс создания технической документации, управления работой групп программистов, тестирования кода и развертывания PHP-приложений: Phing, PHPUnit2, PHPDocumentor, PEAR и CVS.

ISBN 978-5-8459-1586-3

в продаже

РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЙ С ПОМОЩЬЮ PHP И MYSQL
БИБЛИОТЕКА РАЗРАБОТЧИКА
ЧЕТВЕРТОЕ ИЗДАНИЕ

Люк Веллинг
Лора Томсон



www.williamspublishing.com

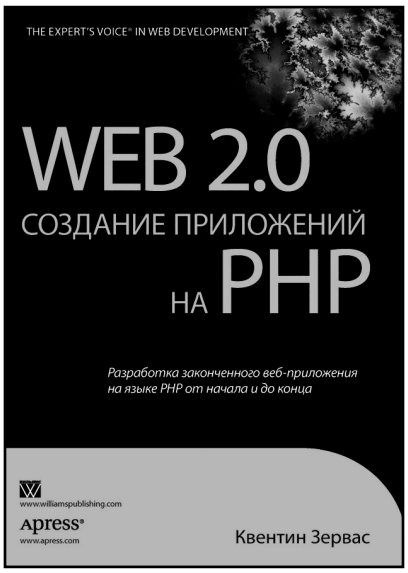
Эта книга предназначена для тех, кто знаком с основами HTML и ранее разрабатывал программы на современных языках программирования, но, возможно, не занимался программированием для веб или не использовал реляционные базы данных. В ней подробно описано применение последних версий PHP и MySQL для построения крупных коммерческих веб-сайтов. Основное внимание в книге уделено реальным приложениям. Здесь рассматриваются как простые интерактивные системы приема заказов, так и различные аспекты электронных систем продажи и безопасности во взаимосвязи с созданием реального веб-сайта. Подробно описаны все стадии разработки типовых проектов на PHP и MySQL, в числе которых служба веб-почты, приложение поддержки веб-форумов и электронный книжный магазин. Книга ориентирована на профессиональных разработчиков, но будет полезной и начинающим.

ISBN 978-5-8459-1574-0

в продаже

WEB 2.0 СОЗДАНИЕ ПРИЛОЖЕНИЙ НА PHP

Квентин Зервас



www.williamspublishing.com

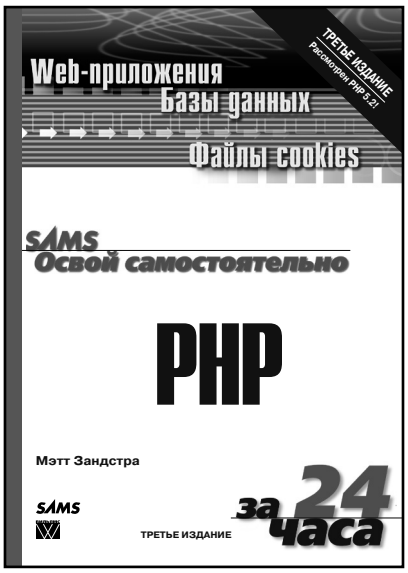
Книга посвящена пошаговой разработке законченного практического веб-приложения, следующего стандартам Web 2.0 — многопользовательской системы блогов с возможностью размещения фотографий и географических карт. Разработка ведется на языке PHP с использованием библиотеки Zend Framework и системы Smarty Template Engine. Для реализации стандартных функций веб-сайтов и визуальных эффектов привлекаются библиотеки JavaScript-кода Prototype и Scriptaculous. Рассмотрено создание базы данных MySQL или PostgreSQL для обслуживания приложения, а также применение микроформатов и технологии Ajax.

Книга предназначена для студентов и специалистов в области веб-программирования сайтов с динамическим контентом.

ISBN 978-5-8459-1590-0 **в продаже**

ОСВОЙ САМОСТОЯТЕЛЬНО PHP ЗА 24 ЧАСА
ТРЕТЬЕ ИЗДАНИЕ

Мэтт Зандстра



www.williamspublishing.com

Эта великолепная книга предназначена для начинающих программистов, не имеющих практического опыта в программировании на PHP. Автор очень доходчиво на живых примерах объясняет основы программирования на PHP, а затем переходит к более сложным темам, таким как объектно-ориентированное проектирование и программирование, работа с базами данных, XML, PEAR, использование регулярных выражений. PHP разработан с учетом высокой степени интеграции с базами данных. Поэтому ряд примеров в этой книге написан для работы с SQL-совместимыми базами данных. В конце книги рассмотрен пример работоспособного приложения, которое можно положить в основу любого Web-приложения. Коды примеров приведены на Web-сайте Издательского дома “Вильямс”.

ISBN 978-5-8459-1393-7 **в продаже**

ПОИСКОВАЯ ОПТИМИЗАЦИЯ НА PHP ДЛЯ ПРОФЕССИОНАЛОВ
РУКОВОДСТВО РАЗРАБОТЧИКА ПО SEO

Джейми Сирович
Кристиан Дари



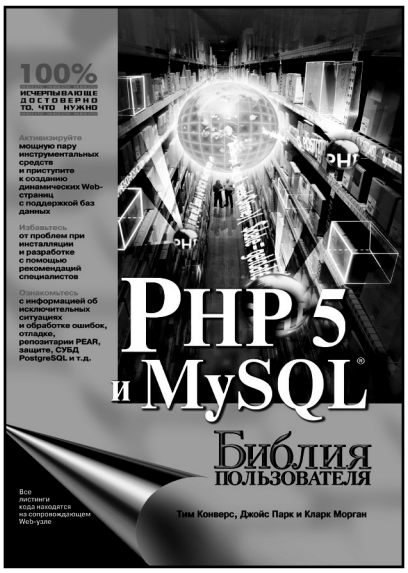
www.dialektika.com

Книга двух специалистов в области маркетинга и разработки для Web посвящена технологиям создания и обслуживания Web-сайтов, которые оптимизированы для поисковых машин. Подробно рассматриваются такие вопросы, как облегчение индексации сайта, средства для продвижения сайта, роль архитектуры сайта в поисковой оптимизации, способы достижения хороших поисковых рангов и многие другие. Читатели ознакомятся с методами переадресации, Web-каналами и социальными закладками, приманиванием ссылок, созданием блогов, оценкой оптимальности сайта с точки зрения поисковых механизмов. Кроме того, приводится пример построения электронного магазина с учетом аспектов поисковой оптимизации. Книга рассчитана на программистов и менеджеров по маркетингу разной квалификации, а также будет полезна для студентов и преподавателей дисциплин, связанных с разработкой и бизнес-аналитикой для Web.

ISBN 978-5-8459-1376-0 **в продаже**

PHP 5 И MYSQL
БИБЛИЯ ПОЛЬЗОВАТЕЛЯ

Тим Конверс,
Джойс Парк
и Кларк Морган



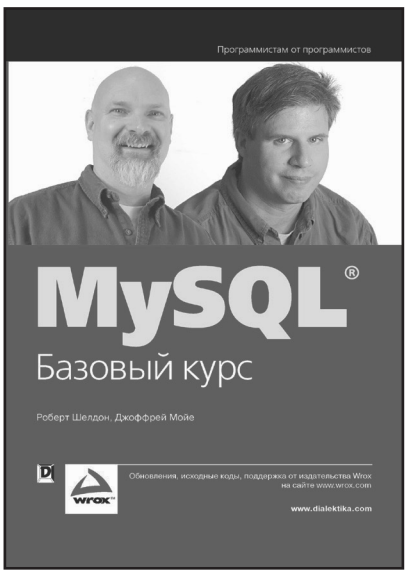
www.dialektika.com

В книге приведены исчерпывающие сведения по созданию динамических Web-узлов на основе программных средств с открытым исходным кодом: языка PHP, сервера Apache и СУБД MySQL, а также показано, как обеспечить бесперебойную эксплуатацию таких узлов под управлением операционных систем Windows или Linux. Многочисленные сценарии и готовые программы, представленные в книге, подробно описаны, тщательно прокомментированы и составляют основу практически значимых приложений. Книга дополняет электронную документацию, содержит все необходимые справочные данные и рассчитана на широкий круг читателей.

ISBN 5-8459-1022-6 **в продаже**

MYSQL®
БАЗОВЫЙ КУРС

Роберт Шелдон,
Джоффрей Мойе



www.williamspublishing.com

Книга опытных специалистов в области реляционных баз данных и разработки приложений для доступа к данным из Web предлагает полное введение как в саму СУБД MySQL, так и в основополагающие принципы работы с данными, хранящимися в этой СУБД, на популярных языках программирования для Web-приложений, среди которых PHP, Java и ASP.NET/C#. Книгу отличает простой и доступный стиль изложения, изобилие примеров и множество рекомендаций по эффективному применению MySQL. Подробно рассматриваются такие вопросы, как инсталляция и настройка MySQL, проектирование и создание баз данных, настройка таблиц различных типов, введение в язык SQL, экспорт и импорт данных, организация и управление транзакциями. Немало внимания уделяется администрированию MySQL, тонкой настройке производительности, а также резервному копированию и восстановлению баз данных.

ISBN 978-5-8459-1167-4

в продаже

ПРОГРАММИРОВАНИЕ БАЗ ДАННЫХ MICROSOFT® SQL SERVER® 2008
БАЗОВЫЙ КУРС

Роберт Виейра



www.dialektika.com

В этой книге SQL Server 2008 рассматривается главным образом с точки зрения разработчика. В ней описаны основные особенности этой версии СУБД, позволяющие значительно улучшить совместимость компонентов и расширить набор средств, обеспечивающих взаимодействие с языком XML, инфраструктурой .NET, определяемыми пользователем типами данных, а также со многими другими дополнительными службами. Книга целиком посвящена изложению основных сведений о средствах разработки, необходимых каждому разработчику независимо от уровня его подготовки, и содержит описание современной версии рассматриваемого программного продукта, но включает все необходимые сведения о проблемах обеспечения обратной совместимости, которые могут повлиять на выбор наиболее подходящих способов проектирования и написания кода.

ISBN 978-5-8459-1612-9

в продаже