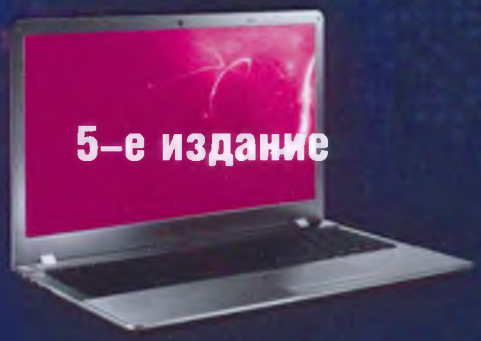
Кевин Янк

Простой способ создать сайт на основе базы данных

PHP и MySQL

от новичка к профессионалу

МИРОВОЙ КОМПЬЮТЕРНЫЙ БЕСТСЕЛЛЕР



ЭКСМО

Кевин Янк

PHP и MySQL

от новичка к профессионалу

МОСКВА 2013

ЭКСМО

УДК 004.45
ББК 32.973-018.2
Я 62

Kevin Yank

PHP & MySQL: Novice to Ninja, 5th. Edition Authorized Russian translation of the English edition of PHP & MySQL: Novice to Ninja, 5th. Edition (ISBN 9780987153081) © 2012 Sitepoint Pty. Ltd.
This translation is published and sold by permission of O'Reilly Media, Inc.,
which owns or controls all rights to publish and sell the same.

Янк К.

Я 62 PHP и MySQL. От новичка к профессионалу / Кевин Янк. — М. : Эксмо, 2013. — 384 с. — (Мировой компьютерный бестселлер).

ISBN 978-5-699-67363-6

Это практическое руководство станет незаменимым помощником для тех, кто желает создавать сайты на основе базы данных. В пятом издании этого бестселлера вы узнаете, как с помощью языка PHP, используемого в 20 миллионах сайтов по всему миру, создать собственную систему управления содержимым (CMS) на основе исключительно бесплатного программного обеспечения. Книга содержит практические советы по проектированию баз данных с использованием MySQL и охватывает самые последние технологии.

УДК 004.45
ББК 32.973-018.2

ISBN 978-5-699-67363-6

КРАТКОЕ ОГЛАВЛЕНИЕ

ОГЛАВЛЕНИЕ

Об авторе

Кевин Янк (Kevin Yank) занимается созданием сайтов уже более 15 лет. Он написал многочисленные книги, статьи, учебные курсы, электронные рассылки и подкасты по данной тематике.

Кевин стал первым автором, которому в 2001 году молодое сообщество SitePoint предложило сотрудничество. Дебютное издание называлось Build your own database driven web site using PHP & MySQL («Создайте собственный сайт на основе базы данных с помощью PHP и MySQL»). Впоследствии эта книга переиздавалась пять раз. Сейчас перед вами ее последняя версия, переведенная на русский язык. Кроме того, Кевин Янк — соавтор еще двух учебников (Simply JavaScript и Everything you know about CSS is wrong!), автор материалов для электронной рассылки SitePoint Tech Times и соведущий подкаста SitePoint Podcast.

Будучи главным инструктором проекта learnable.com, Кевин разработал популярные онлайн-курсы по изучению JavaScript, PHP и MySQL, а также HTML и CSS, выступил в роли консультанта при создании новых интернет-проектов для библиотеки Learnable.

Сейчас Кевин — технический директор компании Avalanche Technology Group, которая занимается выпуском электронных продуктов и распространением цифровых технологий на австралийском и мировом рынках. Он живет в Мельбурне (Австралия) вместе со своей подругой Джессикой, собакой, кошкой и двумя морскими свинками.

Кевин Янк стремится сделать так, чтобы веб-технологии стали понятными для всех.

О SitePoint

SitePoint специализируется на выпуске доступных и простых в изучении практических пособий, предназначенных для профессиональных веб-разработчиков. Посетив сайт http://www.sitepoint.com/[1], вы сможете поближе познакомиться с опубликованными книгами и статьями, получить доступ к блогам и электронным рассылкам, а также посетить форумы данного сообщества.

[1] Все указанные в книге сайты англоязычные. Издательство не несет ответственности за их содержимое и напоминает, что со времени написания книги сайты могли измениться или вовсе исчезнуть. — *Прим. ред.*

ВВЕДЕНИЕ

PHP и MySQL уже не те, что прежде.

Когда в 2001 году вышло в свет первое издание этой книги, читателей поразил тот факт, что для создания полноценного сайта вовсе не обязательно писать отдельный HTML-файл для каждой страницы. На фоне других языков программирования PHP выделялся главным образом своей простотой: любой желающий мог загрузить его, установить и начать изучение. Бесплатная база данных MySQL также предоставляла не менее простое решение задач, которые до сих пор были под силу только опытным программистам, имеющим в распоряжении корпоративный бюджет.

В то время PHP и MySQL были не просто особенными, они были настоящим чудом. С годами у них появилось множество быстроразвивающихся конкурентов. Теперь, когда любой человек, обладающий учетной записью на сайте WordPress (http://wordpress.com/), способен за полминуты создать полноценный блог, языкам программирования недостаточно быть легкими в изучении, а базам данных — только бесплатными.

Если вы взялись за чтение этой книги, скорее всего, вы жаждите чего-то большего, чем несколько щелчков кнопкой мыши, которые позволяют выполнить бесплатные веб-приложения. Возможно, вы даже подумываете о создании собственного, в конце концов, приложение WordPress написано именно с помощью PHP и MySQL. Так стоит ли ограничивать себя?

Чтобы выдержать конкуренцию и соответствовать нуждам все более требовательных проектов, PHP и MySQL пришлось серьезно измениться. Сейчас PHP — куда более сложный и мощный язык программирования по сравнению с версией 2001 года, база данных MySQL также предоставляет огромное количество новых возможностей. Изучение этих инструментов открывает перед вами широкие перспективы, которые были недоступны пользователям первых версий.

Однако не все так просто. Не секрет, что швейцарский армейский нож и обычный кухонный далеки друг от друга по своему устройству и безопасности использования. Точно так же заманчивые возможности и улучшения в PHP и MySQL привели к тому, что обе программы стало заметно сложнее освоить. Кроме того, PHP пришлось расстаться с некоторыми функциями, которые чрезмерно упрощали язык и позволяли неопытным программистам создавать сайты с зияющими дырами в безопасности. Хотя в 2001 году именно эти функции были конкурентным преимуществом PHP и делали его более дружественным по отношению к новичкам.

Конечно, теперь написать пособие о PHP и MySQL намного труднее. Однако именно это и добавляет ценность данной книге: чем труднее дорога, тем нужнее карта. Шаг за шагом изучая практический материал, вы узнаете, как создаются

сайты, основанные на базе данных. Если ваш хостинг поддерживает технологии PHP и MySQL — хорошо. Если нет — не переживайте: вы научитесь устанавливать их самостоятельно на компьютеры под управлением Windows, Mac OS X и Linux.

Издание станет незаменимым помощником для тех, кто находится в начале извилистого пути изучения PHP и MySQL. Итак, берите вашу карту и в путь!

Для кого предназначена эта книга

Книга ориентирована на веб-дизайнеров, уже имеющих некоторую базу знаний и желающих перейти в область серверного программирования. Она рассчитана на то, что вы знакомы с простым HTML-кодом, который приводится здесь без особых объяснений. Знания CSS (Cascading Style Sheets — каскадные таблицы стилей) или JavaScript от вас не требуется, но, если вы знакомы с последним, это значительно упростит изучение PHP, поскольку оба языка довольно похожи.

К концу чтения этой книги у вас должно появиться понимание того, как создаются сайты, основанные на базах данных. Следуя примерам, вы изучите основы языков PHP и SQL. Первый представляет собой серверный скриптовый язык, с помощью которого осуществляется доступ к базам. Второй — стандартный язык структурированных запросов для взаимодействия с реляционными базами данных, поддерживаемый самым популярным на сегодня движком баз данных — MySQL. Но что еще более важно, у вас будет все необходимое, чтобы начать собственный проект.

О чем эта книга

Издание состоит из 12 глав. Для получения исчерпывающих знаний читайте их по порядку от начала до конца. Если вам понадобится освежить в памяти конкретную тему, вы можете пропустить все лишнее.

Прежде чем приступить к созданию сайта на основе базы данных, сперва следует убедиться в том, что у вас есть все нужные для этого инструменты. В **главе 1** вы узнаете, где взять два необходимых программных пакета — скриптовый язык программирования PHP и систему управления базами данных MySQL. Вы также получите пошаговые инструкции по их установке на Windows, Linux и Mac OS X, узнаете, как проверить работоспособность PHP на своем веб-сервере.

В **главе 2** вы познакомитесь с базами данных в целом и с системой управления реляционными базами данных MySQL в частности. Если вы никогда прежде не имели дело с реляционными базами, эта глава подогреет ваш интерес к остальной части книги. Вы также создадите простую базу данных, которая пригодится в последующих главах.

С **главы 3** начинается самое интересное. Вы познакомитесь со скриптовым языком PHP. С его помощью создадите динамические веб-страницы, которые помогут получить посетителям вашего сайта самую последнюю информацию. Если у вас уже есть опыт программирования, вы можете бегло просмотреть эту главу — в ней объясняются основы языка. Новичкам рекомендуется прочитать главу целиком,

поскольку в ней описаны базовые концепции, которые широко используются в оставшейся части книги.

В **главе 4** вы создадите свою первую страницу, основанную на базе данных, изучите совместное использование технологий PHP и MySQL, которые были рассмотрены по отдельности в предыдущих главах. Познакомитесь с основными приемами использования PHP, направленными на извлечение содержимого базы данных и его отображение в Интернете в режиме реального времени. Вы также научитесь создавать с помощью PHP веб-формы, предназначенные для быстрого добавления в MySQL новых записей и редактирования уже имеющихся.

Ознакомившись с простейшими примерами баз данных, вы осознаете, что большинству сайтов необходимо хранить куда более сложные типы информации, чем те, с которыми вы успеете познакомиться. Очень многие веб-проекты останавливаются на полпути или переделываются заново из-за ошибок на раннем этапе проектирования структуры базы данных. В **главе 5** вы изучите основные принципы правильного проектирования, где особое внимание уделено нормализации содержимого базы данных. Если вы не совсем понимаете, что это значит, тогда вам определенно стоит ее прочитать.

Методики улучшения структурированности кода полезно использовать во всех проектах, за исключением разве что самых простых. Язык PHP — ваш основной помощник в этом деле. В **главе 6** мы раскроем некоторые из простейших приемов, которые помогут управлять вашим кодом и поддерживать его. Вы научитесь использовать подключаемые файлы, чтобы избежать многократного повторения одного и того же фрагмента кода на нескольких страницах сайта. Вы также узнаете, как создавать собственные функции и тем самым расширять базовые возможности PHP и упрощать скрипты.

Глава 7 во многом — кульминация книги и вознаграждение для тех веб-мастеров, которые устали обновлять сотни страниц при необходимости внести изменения в дизайн сайта. Вы напишите код для простой системы, которая позволит управлять содержимым базы данных и его категориями. В дальнейшем эта система пригодится при наполнении сайта: сделав несколько изменений, вы получите панель администрирования, с помощью которой пользователи смогут публиковать информацию на сайте, не имея ни малейшего представления об HTML.

В **главе 8** вы освоите несколько ловких приемов, позволяющих вывести содержимое базы данных не в виде обычного скучного текста, а с элементами форматирования, включая полужирное начертание и курсив.

Что такое сессии и как они связаны с куки — многострадальной технологией для хранения данных в Интернете? Почему постоянство данных настолько важно в современных системах электронной торговли и других веб-приложениях? На эти вопросы ответит **глава 9**. В ней вы узнаете, каким образом PHP поддерживает куки и сессии и как они взаимодействуют между собой, а затем примените полученные знания, чтобы создать простой интернет-магазин и систему контроля доступа для своего сайта.

Система MySQL — хорошее и простое решение для создания баз данных без излишеств. Тем не менее, если вы собираетесь хранить содержимое сайта с ее помощью, вам следует разобраться с некоторыми особенности системы. В **главе 10** вы научитесь создавать резервные копии для базы данных и управлять доступом

к ним. Помимо нескольких секретных приемов, например восстановление забытого пароля к MySQL, вы узнаете, как повысить производительность базы данных, когда ее скорость начинает падать, и как эффективно связать хранящуюся в ней информацию.

Как уже говорилось, в главе 5 вы ознакомитесь с процессом моделирования сложных связей между фрагментами информации в реляционной базе данных вроде MySQL. Теория — это хорошо, но чтобы применить эти концепции на практике, вам понадобится изучить еще несколько аспектов SQL. В **главе 11** рассмотрены расширенные возможности этого языка, освоив которые можно научиться профессионально управлять сложными данными.

Некоторые наиболее интересные веб-проекты, основанные на базах данных, подразумевают работу с бинарными данными. Самый показательный пример — интернет-сервис для хранения файлов. Тем не менее подобная функциональность пригодится и такой простой системе, как персональная фотогалерея: она позволит хранить и извлекать информацию (то есть изображения) налету. В **главе 12** вы узнаете, как ускорить работу вашего сайта, создавая статические копии динамических страниц с помощью PHP. Получив базовые навыки, вы возьметесь за разработку файлового хранилища и системы отображения, а также рассмотрите плюсы и минусы работы с бинарными файлами в MySQL.

Где искать помощь

Развитие PHP и MySQL не стоит на месте. Вполне вероятно, что к тому времени, когда вы дочитаете данную книгу, эти технологии немного изменятся и информация успеет слегка устареть. К счастью, у SitePoint есть внушительное сообщество PHP-разработчиков, готовых прийти на помощь, если у вас возникнут вопросы. На сайте SitePoint есть список найденных неточностей и опечаток из этой книги, где вы можете ознакомиться с последними исправлениями.

Форумы SitePoint

На форумах SitePoint (http://www.sitepoint.com/forums/) можно задавать любые вопросы, связанные с веб-разработкой, или помогать другим участникам решать возникшие проблемы. Именно так устроены любые дискуссии: кто-то спрашивает, кто-то отвечает, большинство делают и то и другое. От того, что вы делитесь своими знаниями, сообщество только выигрывает и становится сильнее. Форумы часто посещают интересные и опытные веб-дизайнеры и разработчики. Это отличный способ изучить что-то новое, быстро получить ответ на свой вопрос или просто хорошо провести время.

У SitePoint есть два отдельных форума: для PHP (http://www.sitepoint.com/forums/forumdisplay.php?34-PHP) и для баз данных MySQL (http://www.sitepoint.com/forums/forumdisplay.php?88-Databases-amp-MySQL).

Веб-страница книги

Веб-страница, посвященная книге, находится по адресу http://www.sitepoint.com/books/phpmysql5/ и содержит следующие разделы.

- **Архив с кодом.** По мере чтения вам будут встречаться отсылки на архив с кодом. Это доступные для загрузки файлы, которые содержат исходный код примеров, используемых в книге. Если хотите облегчить себе работу, перейдите по ссылке http://www.sitepoint.com/books/phpmysql5/code.php.
- **Обновления и список ошибок.** Идеальных книг не бывает. Практически в любом издании внимательные читатели могут найти ошибки. Данный раздел содержит самую свежую информацию об опечатках и неточностях в коде.

Электронные рассылки SitePoint

Помимо выпуска книг, подобных той, что вы держите в руках, сообщество SitePoint занимается подготовкой общедоступных электронных рассылок: SitePoint, PHPMaster, CloudSpring, RubySource, DesignFestival и BuildMobile. В них вы найдете последние новости, информацию о выпусках новых версий программных продуктов, тенденции, советы и методики, касающиеся всех аспектов веб-разработки. Подписаться на рассылки можно по адресу http://www.sitepoint.com/newsletter/.

Отзывы о книге

Если вам не удалось найти решение возникшей проблемы на форумах SitePoint или вы хотите связаться с сообществом по какой-либо другой причине, воспользуйтесь адресом books@sitepoint.com. Ваш вопрос не останется без внимания. Если команда поддержки не сможет на него ответить, она передаст его более компетентным людям. Особенно приветствуются советы по улучшению издания и сообщения о найденных ошибках.

Используемые условные обозначения

Чтобы выделить разные виды информации, в книге используются определенные стиль и разметка.

- Примеры кода отображаются с помощью моноширинного шрифта.

```
<h1>Чудесный осенний денек</h1>
<p>Стояла замечательная погода для прогулки по парку. Пели птицы,
ласково светило солнце.</p>
```

- Если код можно найти в архиве, в скобках перед листингом указывается имя соответствующего файла. Например, `example.css`. Если приводится только часть кода, после имени файла добавлено слово «фрагмент».
- Строки кода, дополняющие предыдущий листинг, выделены полужирным начертанием.

```
function animate() {
  new_variable = "Привет";
}
```

- Повторяющиеся участки кода заменяются символом «⋮».

```
function animate() {
  ⋮
  return new_variable;
}
```

В книге также используется несколько видов примечаний:

 — полезные подсказки;

 — отступления, связанные с обсуждаемой темой; дополнительная информация, которая не слишком важна;

 — важные заметки;

 — предупреждения, которые раскрывают неочевидные моменты, способные сбить вас с толку.

Глава 1
УСТАНОВКА PHP И MySQL

Эта книга поможет вам выйти за пределы статического мира, где веб-страницы создаются исключительно с помощью клиентских технологий, таких как HTML, CSS и JavaScript. Вместе мы погрузимся в мир сайтов, основанных на базах данных, исследуем ошеломляющий набор динамических инструментов, концепций и возможностей, которые они открывают.

Чтобы написать свой первый динамический сайт, вам понадобится собрать необходимые инструменты: загрузить и подготовить два обязательных программных пакета. Еще не догадываетесь, о чем идет речь? Вот подсказка: прочтите название этой книги. Конечно же, это PHP и MySQL!

Если вы уже создавали сайты с помощью HTML и CSS и, может быть, даже иногда применяли JavaScript, то, скорее всего, знаете, что файлы, из которых состоит сайт, следует загружать в определенное место. Это может быть платный хостинг, предоставляемое интернет-провайдером дисковое пространство или сервер вашей компании. Как только вы скопируете нужные файлы в одно из этих мест, специальная программа, называемая **веб-сервером**, сможет найти их, обработать и выдать по запросу любого браузера (Internet Explorer, Google Chrome, Safari или Firefox). Наиболее популярны веб-серверы Apache HTTP Server (Apache) и Internet Information Services (IIS). Скорее всего, вам знакомы эти названия.

PHP — скриптовый язык, работающий на стороне сервера и представляющий своеобразное дополнение к нему. Этот язык позволяет не просто отправлять браузерам точные копии запрашиваемых файлов, но и запускать небольшие программки для выполнения разных задач — **PHP-скрипты**. Например, вы можете извлечь свежую информацию из базы данных, сформировать на ее основе веб-страницу и отправить запросившему браузеру. Большая часть данной книги посвящена PHP-скриптам, которые выполняют именно такую работу. Загрузить и использовать PHP можно бесплатно.

Чтобы извлечь информацию с помощью PHP-скриптов из базы данных, для начала следует ее *создать*. Здесь нам пригодится **MySQL** — система управления реляционными базами данных (СУРБД). Ее предназначение и принцип работы мы обсудим позже. Если говорить коротко, эта программа способна систематизировать множество фрагментов информации, отслеживать между ними связь и эффективно ими управлять. Кроме того, MySQL позволяет получать доступ к хранимой информации с помощью скриптовых языков программирования, работающих на стороне сервера. Как и в случае с PHP, бесплатная версия програмы пригодна для выполнения большинства задач.

В главе 1 вы научитесь настраивать веб-сервер для работы с PHP и MySQL и получите пошаговые инструкции, актуальные для последних версий Windows и Mac OS X (инструкции по установке PHP и MySQL для пользователей Linux содержатся в приложении А).

Ваш собственный веб-сервер

На большинстве хостингов PHP и MySQL уже установлены, и это одна из причин, почему эти программные пакеты столь популярны. Если и ваш хостинг оснащен всем требуемым, можете облегченно вздохнуть: вам не придется покупать сервер с поддержкой правильных технологий, чтобы опубликовать свой первый сайт на основе базы данных. Однако необходимость в самостоятельной установке PHP и MySQL по-прежнему остается: перед тем как вы предъявите миру свой сайт, необходимо проверить его работу на собственном веб-сервере с поддержкой PHP и MySQL.

Когда вы разрабатываете статический сайт, чтобы оценить его вид, достаточно загрузить HTML-файлы с жесткого диска непосредственно в браузер. Поскольку браузер способен самостоятельно считывать и интерпретировать код в формате HTML, веб-сервер ему для этого не нужен. Если речь идет о динамических сайтах, написанных с использованием PHP и MySQL, браузеру понадобится помощь, так как он не способен понять содержимое PHP-скриптов. В этом случае веб-сервер, поддерживающий PHP, считывает содержащиеся в скриптах инструкции и *генерирует* понятный для браузера HTML-код.

Если вы работаете в компании с весьма любезными системными администраторами, вполне вероятно, что они разрешат вам воспользоваться уже настроенным внутренним веб-сервером. В этом случае вы можете спокойно заниматься безопасной разработкой собственного сайта и хранить все файлы на сетевом диске, который этим веб-сервером обслуживается. Когда будете готовы сделать сайт общедоступным, просто скопируйте файлы с сетевого диска на публичный веб-сервер.

Если вам на самом деле посчастливилось работать в таких условиях, вы можете пролистать большую часть этой главы. Вопросы, которые могут у вас возникнуть к специалистам, настроившим сервер для разработки, рассматриваются в разделе «Что нужно знать о веб-хостинге». Ознакомьтесь с ним перед тем, как начнете использовать предоставленные вам PHP и MySQL.

Установка в Windows

В этом разделе вы узнаете, как запустить веб-сервер с поддержкой PHP и MySQL на компьютере под управлением Windows XP, Windows Vista или Windows 7. Если вы используете другую операционную систему, данный раздел можно пропустить.

Самый простой способ настроить и запустить веб-сервер в Windows — воспользоваться бесплатным программным пакетом XAMPP. Это программа содержит встроенные копии Apache, PHP и MySQL. Давайте рассмотрим процесс ее установки.

САМОСТОЯТЕЛЬНАЯ УСТАНОВКА

В предыдущих редакциях данной книги рекомендовалось настраивать Apache, PHP и MySQL по отдельности, используя официальные установочные пакеты для каждой программы. Считалось, что так новички получат четкое представление о том, как данные компоненты сочетаются друг с другом. К сожалению, из-за этого многие проводили первые часы знакомства с PHP за чтением установочных инструкций, которые к тому же устаревали ввиду изменений в одном из программных пакетов. Кроме того, процесс установки мог затянуться надолго. Да и вообще, зачем мучить свой компьютер множеством отдельных, хотя и взаимосвязанных программных компонентов, которые затем сложно удалить?

Лучший способ изучить PHP и MySQL — начать их использовать немедленно. Чем более быстрым и беспроблемным окажется этап установки, тем лучше. Вот почему в этом издании рекомендуется работать с XAMPP.

Если вы любите все делать своими руками, являетесь технически подкованным пользователем или просто дочитали книгу до конца и теперь вам интересен профессиональный подход, приложение А содержит подробные инструкции по установке отдельных пакетов. Можете смело использовать их вместо пошагового руководства из данного раздела.

1. Загрузите последнюю версию XAMPP для Windows с сайта Apache Friends по адресу http://www.apachefriends.org/en/xampp-windows.html (на момент написания книги это XAMPP 1.7.7 для Windows размером 81 Мбайт). Чтобы увидеть ссылки для загрузки, прокрутите страницу вниз. Выберите рекомендуемый установщик, затем дважды щелкните на названии файла, чтобы начать процесс установки (рис. 1.1).

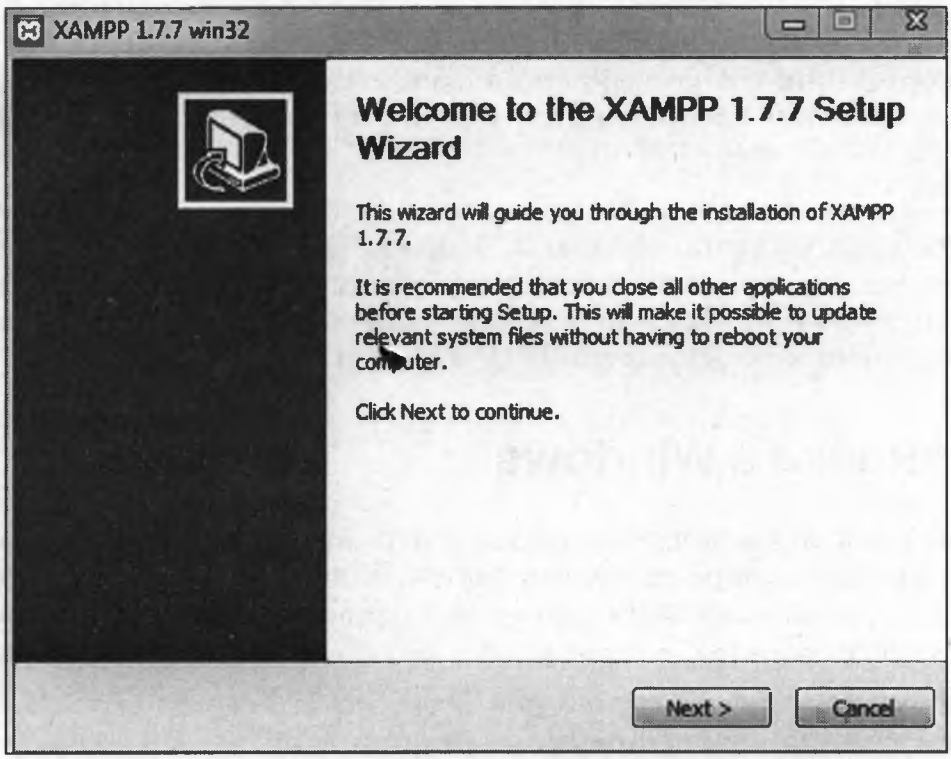


Рис. 1.1. Окно программы установки XAMPP

ПРЕДУПРЕЖДЕНИЕ О СИСТЕМЕ КОНТРОЛЯ УЧЕТНЫХ ЗАПИСЕЙ ПОЛЬЗОВАТЕЛЕЙ (UAC)

В зависимости от используемой вами версии Windows и конкретной системной конфигурации при установке XAMPP может появиться предупреждение (рис. 1.2).

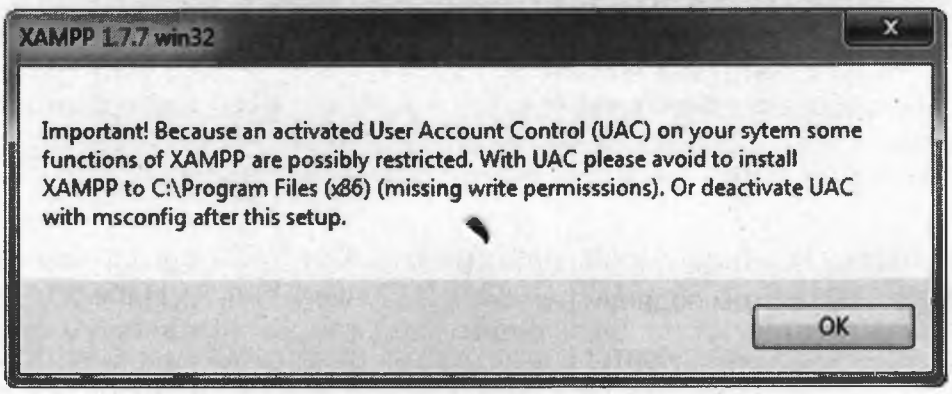


Рис. 1.2. Окно предупреждения о системе контроля учетных записей пользователей (UAC)

2. Хотя поначалу данное сообщение может вас насторожить, на самом деле в нем нет ничего страшного. Оно просто рекомендует не устанавливать XAMPP в каталог C:\Program Files, как это происходит в большинстве случаев, поскольку могут возникнуть проблемы, связанные с правами доступа к файлам. По умолчанию установщик использует путь C:\xampp.

3. На следующем этапе вам понадобится выбрать место, куда нужно установить XAMPP. Идеально использовать указанный по умолчанию каталог c:\xampp (рис. 1.3). Однако, если вы хотите произвести установку в другое место (например, на другой диск), можете выбрать иную директорию. Избегайте при этом стандартного каталога C:\Program Files (или аналогичного ему), поскольку в нем Windows ограничивает права доступа к файлам, необходимым для работы XAMPP.

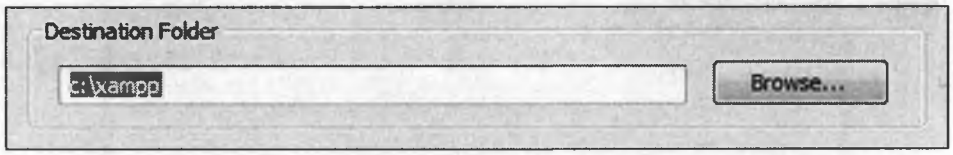


Рис. 1.3. Стандартный путь установки XAMPP

4. Установщик предложит вам на выбор несколько параметров. Возможно, на начальном этапе лучше оставить то, что указано по умолчанию (рис. 1.4). Если вы предпочитаете держать рабочий стол в чистоте, можете снять флажок с пункта Create a XAMPP desktop icon. Если вы хотите, чтобы серверы Apache и MySQL работали постоянно (без необходимости запускать их вручную каждый раз, когда вы хотите заняться разработкой), выберите пункты Install Apache as service и Install MySQL as service соответственно. Дальнейшие инструкции даны на основании того, что вы оставили выбор по умолчанию.

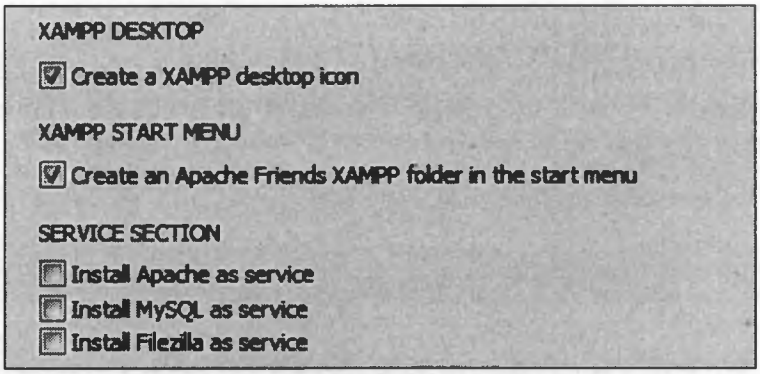


Рис. 1.4. Выбор параметров по умолчанию

5. Как только установщик завершит работу, вам будет предложено запустить панель управления XAMPP (XAMPP Control Panel). Нажмите кнопку No, чтобы отказаться от предложенного варианта и в дальнейшем рассмотреть, как запускается программа в обычных условиях. Установщик закроется.
6. На этом этапе лучше всего перезагрузить компьютер (даже несмотря на то что установщик XAMPP вас об этом не попросит). Как показывает практика и сообщения на форуме поддержки (http://www.apachefriends.org/f/viewtopic.php?f=16&t=48484), без перезагрузки выполнение следующих действий не даст результата.

Завершив установку и перезагрузив систему, вы можете запустить панель управления XAMPP. Для этого откройте меню Пуск и выполните команды Все программы ▶ Apache Friends ▶ XAMPP ▶ XAMPP Control Panel. В системном лотке Windows появится оранжевый значок XAMPP (по умолчанию он исчезнет через несколько секунд), и будет открыто приложение XAMPP Control Panel (рис. 1.5).

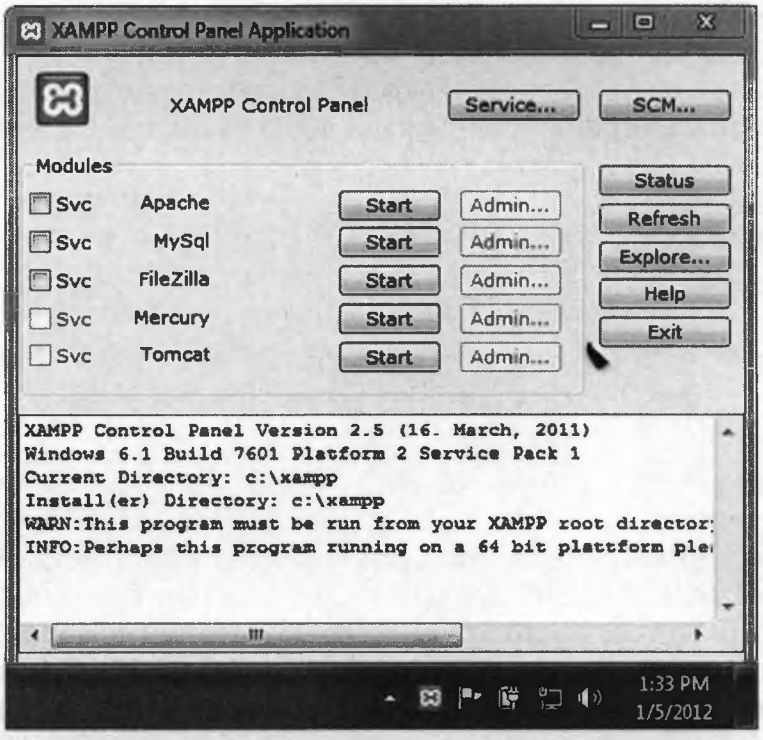


Рис. 1.5. Панель управления XAMPP

Нажмите кнопку Start напротив пунктов Apache и MySql в списке модулей, чтобы запустить соответствующие веб-серверы, встроенные в XAMPP. Рядом с каждым из них в списке должен появиться зеленый индикатор состояния с надписью Running.

В зависимости от используемой вами версии и конфигурации Windows вы, скорее всего, получите от брандмауэра Windows Firewall уведомление, касающееся каждого из веб-серверов (рис. 1.6). Это случится, как только серверы начнут отслеживать приходящие извне запросы.

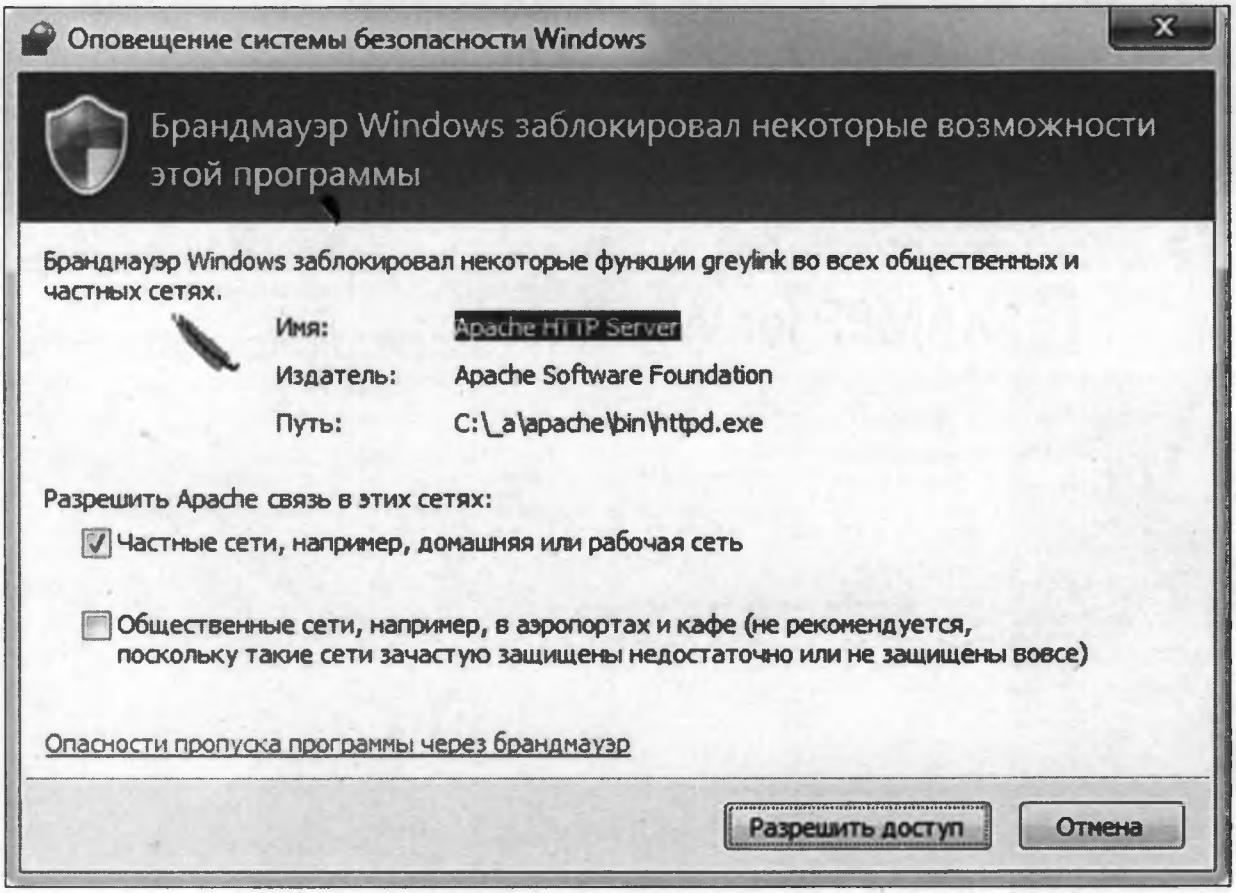


Рис. 1.6. Данное оповещение системы безопасности говорит о том, что ваш веб-сервер Apache работает

Нажмите кнопку Отмена, если хотите, чтобы никто, кроме вас, больше не имел доступ к вашим серверам. У вас останется возможность подключаться к веб-серверу с помощью браузера, запущенного на том же компьютере. Однако иногда иметь доступ к серверу другого компьютера из вашей сети может быть полезно (например, если вы захотите показать коллеге, какой потрясающий сайт вы создали). Для этого стоит выбрать пункт Частные сети, например, домашняя или рабочая сеть и нажать Разрешить доступ.

ПОЧЕМУ МОЙ СЕРВЕР НЕ ЗАПУСКАЕТСЯ?

Существует несколько причин, по которым серверы Apache или MySQL могли не запуститься. На данном этапе наиболее вероятно то, что на вашем компьютере уже запущен веб-сервер (например, IIS от Microsoft или еще одна копия Apache) либо сервер MySQL. Поищите другие установки Apache HTTP Server

или MySQL в меню Пуск в разделе Удаление программы Панели управления Windows, чтобы их закрыть или удалить. Если вы думаете, что у вас запущен веб-сервер IIS от Microsoft, попробуйте закрыть его, следуя официальным инструкциям (http://technet.microsoft.com/en-us/library/cc732317(WS.10).aspx). Еще одной причиной может быть установленная программа WampServer — аналог XAMPP.

Проблема так и не разрешилась? Обратитесь к разделу часто задаваемых вопросов по XAMPP для Windows (http://www.apachefriends.org/en/faq-xampp-windows.html#nostart), особенно если вы используете Skype (при определенных настройках сети он может конфликтовать с веб-серверами).

Добившись корректной работы обоих серверов, нажмите кнопку Admin рядом с надписью Apache. Запустите браузер и введите в нем адрес `http://localhost/xampp/`. В результате должна появиться панель администрирования XAMPP для Windows (рис. 1.7).

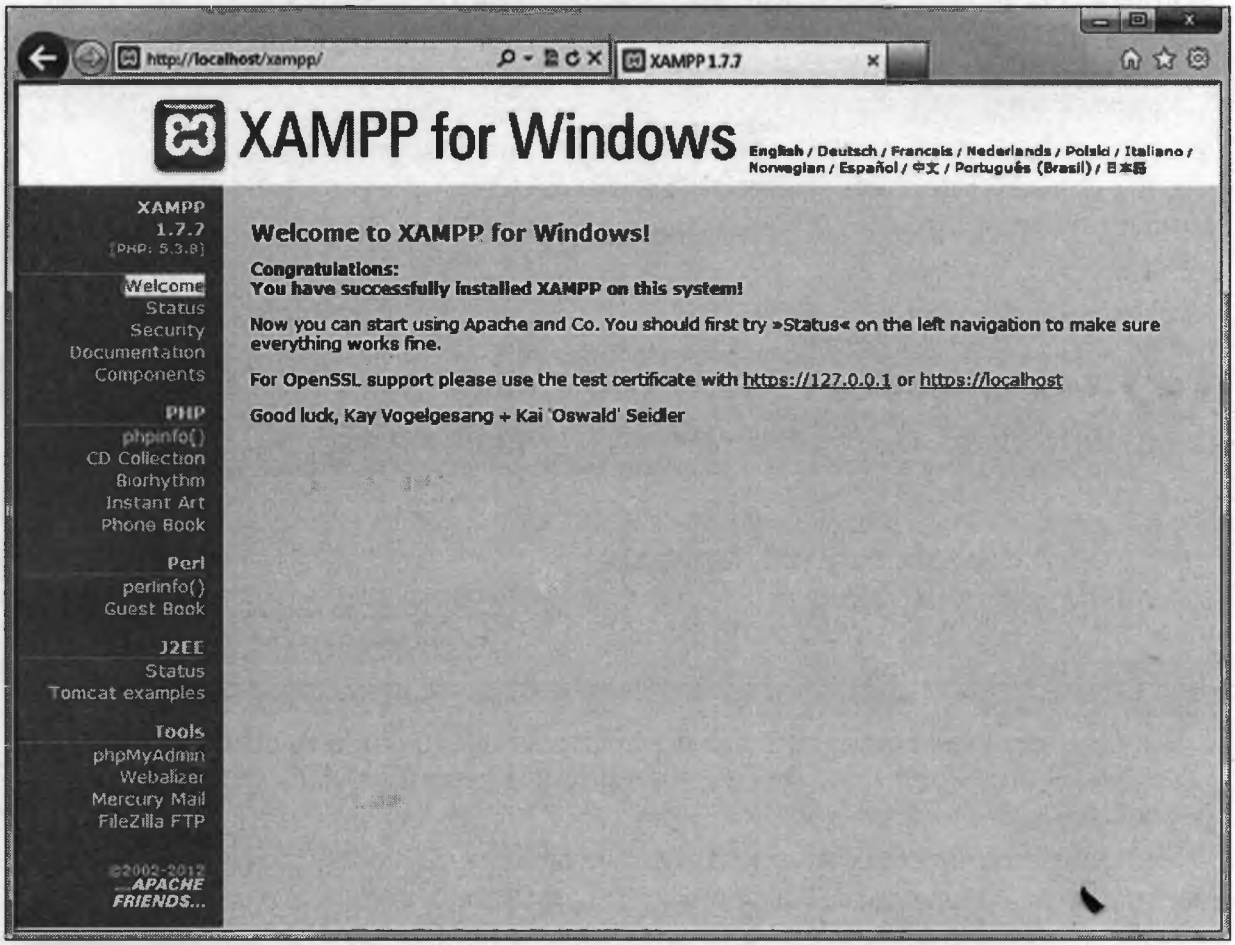


Рис. 1.7. Панель администрирования программы XAMPP

Если вы видите данную страницу, это значит, что ваш веб-сервер запустился и работает — ведь именно из него она и загружена. Обратите внимание, что URL в адресной строке вашего браузера начинается с http://localhost/ (некоторые современные браузеры скрывают протокол http://), где localhost — это специальное имя сервера, которое всегда указывает на ваш компьютер. В данной книге адреса всех веб-страниц, которые вы будете запрашивать у собственного веб-сервера, начинаются с http://localhost/.

Закончив работу с панелью управления XAMPP, закройте ее, нажав кнопку Exit. Вы также можете просто закрыть окно, оставив значок XAMPP в системном лотке Windows (если вы настроили его таким образом, что он оставался видимым). Когда вам снова понадобится панель управления XAMPP, вызовите ее, нажав этот значок.

ПАНЕЛЬ УПРАВЛЕНИЯ XAMPP НЕ УБИРАЕТ ЗА СОБОЙ

Когда вы закроете панель управления XAMPP, серверы Apache и MySQL продолжат работу в системе. Если ваш рабочий день завершен и вы больше не планируете писать код, прежде чем закрывать панель управления XAMPP, остановите каждый из серверов, нажав кнопку Stop. Освободившиеся ресурсы лучше потратить на игры в Facebook.

После того как вы установили на свой компьютер с системой Windows нужные серверы, необходимо указать **пароль администратора** для MySQL в XAMPP. Сервер MySQL должен знать, кто из пользователей авторизован, поскольку только они могут просматривать информацию и управлять ею в базах данных. При первой установке MySQL создает учетную запись с именем `root`, которая имеет доступ к большинству функций и не требует ввода пароля. Тем не менее первое, что вы должны сделать, — указать пароль для этой учетной записи, чтобы неавторизированные пользователи не могли выполнять никаких действий с вашими базами данных.

ЗАЧЕМ ЭТО НУЖНО?

К MySQL, как и к веб-серверу, можно получить доступ с любого компьютера в рамках одной сети. Если вы работаете на компьютере, где есть доступ к Интернету, то в зависимости от принятых мер безопасности к вашему серверу MySQL способен подключиться абсолютно любой человек. Необходимость использовать сложный пароль очевидна.

XAMPP помогает решить эту и другие проблемы, связанные с безопасностью ваших серверов. Убедитесь, что Apache и MySQL запущены, откройте браузер и введите адрес http://localhost/security/ (или используйте ссылку Security в меню панели администратора XAMPP). Открывшаяся страница отобразит список всех проблем с безопасностью, которые система XAMPP обнаружила в текущей конфигурации вашего сервера. Среди них должен значиться пункт The MySQL admin user root has NO password (Учетная запись администратора MySQL не защищена паролем). Прокрутите таблицу вниз и перейдите по ссылке, чтобы решить обнаруженные проблемы.

Появится форма, в первом разделе которой будет предложено установить пароль для учетной записи администратора MySQL. Введите запоминающуюся комбинацию. Оставьте значение поля PhpMyAdmin authentification без изменений (cookie) и выберите вариант с сохранением пароля в файл на случай, если вы его забудете (только учтите, что тогда пароль сможет найти любой, кто будет работать за вашим компьютером). Нажмите кнопку Password changing, чтобы внести сделанные изменения, затем остановите и заново запустите сервер MySQL, используя панель управления XAMPP.

Постарайтесь не забыть указанный пароль, иначе вам будет очень сложно его изменить (в главе 10 вы узнаете, как это делается). Если хотите, можете записать свой пароль администратора MySQL прямо в книге.

МОЙ ПАРОЛЬ АДМИНИСТРАТОРА MySQL (WINDOWS)

Пароль учетной записи администратора: _____ .

ЗАЩИТА ДИРЕКТОРИИ, В КОТОРУЮ БЫЛА УСТАНОВЛЕНА ПРОГРАММА XAMPP

В разделе безопасности вы увидите еще одно предупреждение о том, что ваши страницы доступны всем пользователям сети. Так оно и есть. Но стоит ли сильно переживать из-за того, что ваш коллега по работе или родственник случайно увидит сайт, который находится в процессе разработки? К тому же большинство домашних и офисных сетей настроены таким образом, что доступ извне к веб-серверу, запущенному на вашем компьютере, невозможен. Однако если вы хотите последовать совету XAMPP и установить имя пользователя и пароль для просмотра веб-страниц на вашем сервере, можете это сделать.

Установка в Mac OS X

В этом разделе вы узнаете, как запустить веб-сервер с поддержкой PHP и MySQL на компьютере под управлением Mac OS X версии 10.5 (Leopard). Если вы не используете Mac, данный раздел можно пропустить.

Mac OS X — единственная настольная операционная система, которая уже содержит Apache и PHP (а вместе с ними, к слову, и популярные языки для веб-программирования, такие как Ruby, Python и Perl). Однако чтобы все это заработало, необходимо уделить некоторое время настройке. Кроме того, вам понадобится сервер баз данных MySQL.

Лучше всего проигнорировать предустановленные программные компоненты и воспользоваться единым удобным пакетом MAMP (аббревиатура образована из первых букв слов Mac, Apache, MySQL и PHP). Этот бесплатный программный пакет включает в себя самые последние версии веб-сервера Apache, а также PHP и MySQL. Давайте рассмотрим процесс его установки.

САМОСТОЯТЕЛЬНАЯ УСТАНОВКА

В предыдущих редакциях этой книги рекомендовалось настраивать предустановленные в Mac OS X версии Apache и PHP и использовать официальный установочный пакет MySQL. Считалось, что так новички получат четкое представление о том, как данные компоненты сочетаются друг с другом. К сожалению, из-за этого многие проводили первые часы знакомства с PHP за чтением установочных инструкций, которые к тому же устаревали в виду изменений в одном из программных пакетов. Кроме того, процесс установки мог затянуться надолго. Да и вообще, зачем тратить время на настройку операционной системы, если можно оставить все как есть?

Лучший способ изучить PHP и MySQL — начать их использовать немедленно. Чем более быстрым и беспроблемным окажется этап установки, тем лучше. Вот почему в этом издании рекомендуется работать с MAMP.

На тот случай, если вы любите все делать своими руками, являетесь технически подкованным пользователем или просто дочитали книгу до конца и теперь вам интересен профессиональный подход, в приложении А содержатся подробные инструкции по установке отдельных пакетов. Можете использовать их вместо пошагового руководства из данного раздела.

1. Загрузите последнюю версию MAMP с официального сайта http://www.mamp.info (на момент написания книги это MAMP версии 2.0.5 размером 116 Мбайт). Будьте внимательны: вам нужна бесплатная версия MAMP, а не коммерческая MAMP PRO. Дважды щелкните на загруженном файле, чтобы распаковать установщик MAMP.pkg. Следующим двойным щелчком запустите программу установки (рис. 1.8).

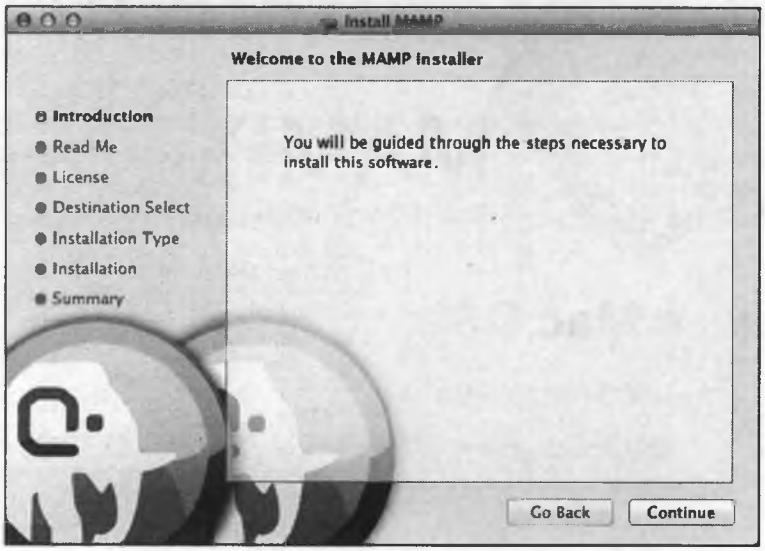


Рис. 1.8. Пакет MAMP

ОСТОРОЖНО

Следующий шаг довольно коварный. Перед тем как что-либо нажимать, прочтите внимательно информацию.

2. На этом этапе вас спросят, хотите ли вы выполнить стандартную установку. Нажмите вместо кнопки Install кнопку Customize — так вы сможете отказаться от версии MAMP PRO (которая незаметно устанавливается с расчетом на то, что в итоге вы ее купите). Это действительно важно, поскольку в случае установки MAMP PRO при каждом запуске бесплатной версии MAMP будет появляться предупреждение.

ПРОПУСТИЛИ ЭТОТ ШАГ?

Если вы все же позволили установить в своей системе MAMP PRO, это можно легко исправить. Откройте у себя каталог Applications, дважды щелкните на новой директории MAMP PRO и запустите файл MAMP PRO Uninstaller. В появившемся окне отметьте все пункты, после чего нажмите кнопку Uninstall. Закройте деинсталлятор.

Зайдите в папку MAMP в каталоге Applications и найдите значок MAMP. Дважды щелкните на нем, чтобы запустить панель управления. Появится окно программы MAMP (рис. 1.9), где два индикатора поменяют цвет с красного на зеленый, сигнализируя о запуске встроенных серверов Apache и MySQL. Затем MAMP откроет назначенный по умолчанию браузер и загрузит страницу с приветствием (рис. 1.10).

Если вы видите данную страницу, значит, ваш веб-сервер запустился и работает, так как именно из него она и загружена. Обратите внимание, что URL в адресной строке вашего браузера начинается с http://localhost:8888/ (некоторые современные браузеры скрывают протокол http://), где localhost — специальное имя сервера, которое всегда указывает на ваш компьютер, а 8888 — **номер порта**, который используется браузером для подключения к вашему компьютеру.

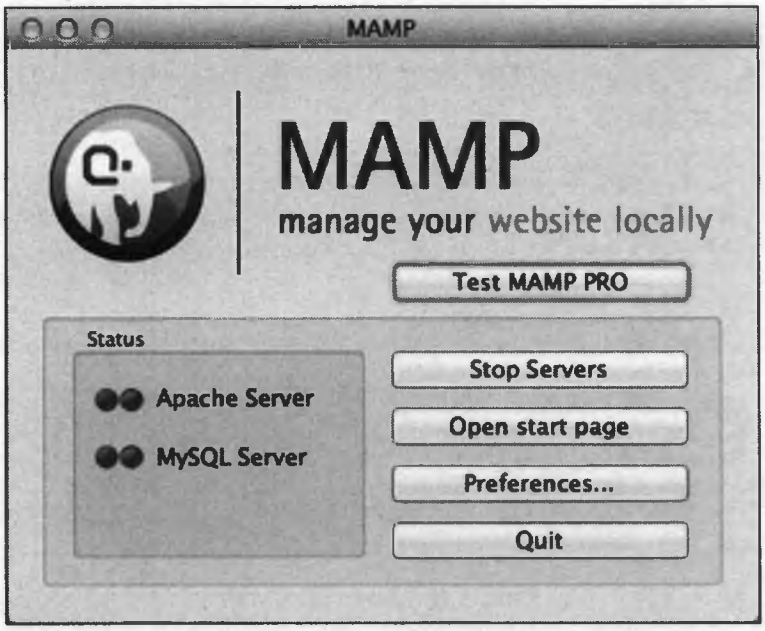


Рис. 1.9. Окно программы MAMP

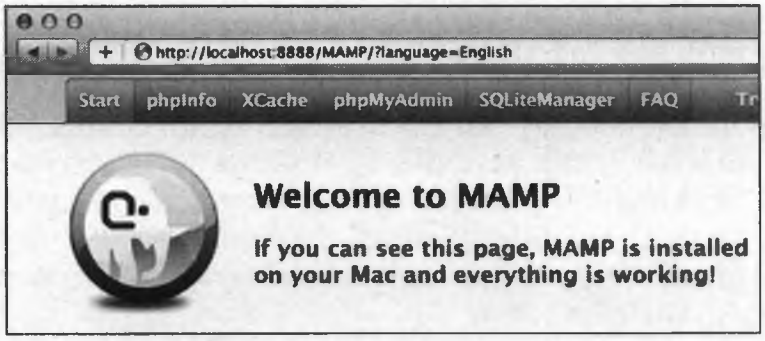


Рис. 1.10. Страница приветствия, подтверждающая конфигурацию и запуск Apache, PHP и MySQL

Каждый сервер, который запускается на компьютере, прослушивает порт с уникальным номером. Как правило, сайты размещаются на 80-м порту — он используется браузерами, когда порт не указан в адресе URL. По умолчанию программа MAMP сконфигурирована таким образом, что Apache прослушивает порт 8888, а MySQL – 8889. Благодаря этому она работает, даже если на вашем компьютере уже установлен веб-сервер, использующий 80-й порт, или сервер MySQL, привязанный к порту 3306 (стандартный для MySQL-сервера)[1]. Дальнейшие примеры

[1] Конечно, нельзя быть абсолютно уверенным в том, что другое приложение в вашей системе не попытается использовать порты 8888 или 8889. Если вы сомневаетесь, можете выбрать другой номер.

с кодом и инструкции даны исходя из того, что ваш веб-сервер работает на 80-м порту, а сервер MySQL — на порту 3306.

Проверьте, сможет ли MAMP работать, используя стандартные номера портов.

1. В окне MAMP нажмите кнопку **Stop Servers** и дождитесь, когда индикаторы станут красными.
2. Нажмите кнопку **Preferences** и перейдите на вкладку **Ports**.
3. Нажмите кнопку **Set to default Apache and MySQL ports**, чтобы назначить порт 80 для Apache и порт 3306 для MySQL. Нажмите **OK**.
4. Нажмите кнопку **Start Servers**. MAMP попросит ввести пароль, поскольку запуск сервера с использованием порта, закрепленного за интернет-службой (например, под номером 80), требует наличия администраторских полномочий.

Если оба индикатора стали зелеными, нажмите кнопку **Open start page** и убедитесь в том, что адрес страницы с приветствием на этот раз начинается с **http://localhost/** (без номера порта). Если все именно так, то вы хорошо поработали!

Если после выполнения первого шага один или оба индикатора остались зелеными или страница с приветствием не смогла как следует загрузиться, то, по всей видимости, произошел конфликт портов. Возможно, на компьютере запущены другие веб- или MySQL-серверы, которые используют один из этих портов или оба порта. Начните поиски в разделе **Sharing** системных настроек. Если параметр **Web Sharing** включен, то у вас работает веб-сервер Apache, встроенный в Mac OS X (как правило, на 80-м порту). Кроме того, попытайтесь закрыть разные приложения. Иногда Skype для Mac OS X не дает запуститься серверу MySQL из состава MAMP.

Если вам удалось запустить MAMP только с использованием стандартных портов (8888 и 8889), то можете ничего не менять. Просто везде, где в этой книге упоминается адрес, начинающийся с **http://localhost/**, добавляйте номер порта (**http://localhost:8888/**). Чуть позже, когда понадобится подключаться к MySQL, вы узнаете, как указать нестандартный порт.

Теперь осталось включить вывод ошибок, связанных с PHP, — это последнее изменение, которое нужно внести в стандартную конфигурацию MAMP. По умолчанию, когда вы допускаете серьезную ошибку в коде на языке PHP, сервер Apache из состава MAMP отображает пустую веб-страницу. В программе отчеты об ошибках выключены на тот случай, если вы захотите с ее помощью обслуживать публичный сайт, чтобы пользователи не видели никаких обескураживающих сообщений. Однако то, что хорошо для пользователей, не годится на этапе разработки сайта. Согласитесь, что пустое окно браузера совсем не помогает понять, что же было сделано не так. Полезнее увидеть в нем подробное сообщение об ошибке.

Чтобы включить отображение ошибок в PHP, откройте каталог **MAMP** в директории **Applications**. Перейдите дальше в **bin/php/** и найдите подкаталоги для всех версий PHP, которые поставляются вместе с MAMP. Скорее всего, MAMP использует самую последнюю из них, но вы можете это перепроверить в настройках. Итак, откройте соответствующий каталог (для нашей версии MAMP 2.0.5 это **php5.3.6**) и перейдите в поддиректорию **conf**. С помощью любого текстового редактора (например, TextEdit) откройте файл **php.ini** и взгляните на следующие строки.

```
; Print out errors (as a part of the output). For production web sites,
; you're strongly encouraged to turn this feature off, and use error
logging
; instead (see below). Keeping display_errors enabled on a production web
site
; may reveal security information to end users, such as file paths on
your Web
; server, your database schema or other information.
display_errors = Off
```

Замените в последней строке `Off` на `On` и сохраните файл. Теперь в окне MAMP нажмите поочередно кнопки **Stop Servers** и **Start Servers**, чтобы перезапустить Apache с новыми настройками. Вот и все — теперь PHP будет выводить полезные (хотя и слегка неприятные) сообщения об ошибках.

Когда закончите работу с программой MAMP (и ее встроенными серверами), закройте ее, нажав кнопку **Quit**. Вы сможете запустить ее снова, когда в очередной раз соберетесь поработать над сайтом с поддержкой баз данных.

Итак, вы установили на свой компьютер с системой Mac OS X нужные серверы. Теперь необходимо указать **пароль администратора** для MySQL.

Сервер MySQL должен знать, кто из пользователей авторизован, поскольку только они могут просматривать информацию и управлять ею в базах данных. При первой установке MySQL создает учетную запись с именем `root`, которая имеет доступ к большинству функций. Пароль у нее тоже `root` — не самая надежная защита. Поэтому первое, что вы должны сделать, — указать пароль для этой учетной записи, чтобы пресечь любые несанкционированные действия с вашими базами данных.

ЗАЧЕМ ЭТО НУЖНО?

К MySQL, как и к веб-серверу, можно получить доступ с любого компьютера в рамках одной сети. Если вы работаете на компьютере, где есть доступ к Интернету, то в зависимости от принятых мер безопасности к вашему серверу MySQL способен подключиться абсолютно любой человек. Необходимость использовать сложный пароль очевидна.

Чтобы установить пароль администратора для MySQL, сначала убедитесь, что у вас запущена программа MAMP и все ее серверы. Затем откройте приложение **Terminal**, которое входит в состав Mac OS X (его можно найти в разделе **Utilities** каталога **Applications**), и введите следующие команды, нажимая после каждой из них клавишу **Enter**.

- `cd /Applications/MAMP/Library/bin/`

  Позволит вам перейти в подкаталог **Library/bin/** внутри директории, где установлена MAMP. Здесь хранятся консольные утилиты.

- `./mysqladmin -u root -p password "`*`newpassword`*`"`

- Замените *`newpassword`* паролем, который вы хотите установить для учетной записи администратора MySQL. Когда вы нажмете **Enter**, вас попросят ввести текущий пароль `root`.

Закройте терминал.

Теперь, когда вы установили свой пароль, появилась новая проблема: программе MAMP необходим свободный доступ к серверу MySQL, чтобы она могла им управлять. Если на данном этапе вы нажмете кнопку Open start page, то увидите сообщение об ошибке: Error: Could not connect to MySQL server! Очевидно, стоит сообщить MAMP новый пароль для учетной записи администратора MySQL.

Чтобы все опять заработало, необходимо отредактировать несколько файлов в каталоге MAMP. Используйте для этого любой текстовый редактор, например TextEdit.

РЕДАКТИРОВАНИЕ PHP-СКРИПТОВ В MAC OS X В TextEdit

По умолчанию TextEdit открывает PHP-файл, как документ в формате HTML, пытаясь отобразить его в виде форматированного текста. Чтобы этого избежать, в диалоговом окне при открытии файла выберите пункт Ignore rich text commands.

- /Applications/MAMP/bin/mamp/index.php

  Найдите строку, которая выглядит следующим образом:

```
$link = @mysql_connect(':/Applications/MAMP/tmp/mysql/mysql.sock',
'root', 'root');
```

  Вместо второго значения `root` подставьте новый пароль администратора для MySQL (то есть *`newpassword`*).

- /Applications/MAMP/bin/phpMyAdmin/config.inc.php

  Это большой файл, поэтому лучше воспользоваться функцией поиска в текстовом редакторе, чтобы найти строки:

```
$cfg['Servers'][$i]['user'] = 'root'; // MySQL user
$cfg['Servers'][$i]['password'] = 'root'; // MySQL password (only
needed with 'config' auth_type)
```

  И снова вместо второго значения `root` подставьте новый пароль администратора для MySQL (то есть *`newpassword`*).

- /Applications/MAMP/bin/checkMysql.sh

  /Applications/MAMP/bin/quickCheckMysqlUpgrade.sh

  /Applications/MAMP/bin/repairMysql.sh

  /Applications/MAMP/bin/stopMysql.sh

  /Applications/MAMP/bin/upgradeMysql.sh

  Содержимое всех этих небольших файлов начинается примерно с таких строк (на примере checkMysql.sh):

```
# /bin/sh
/Applications/MAMP/Library/bin/mysqlcheck --all-databases --check
--check-upgrade -u root -proot --socket=/Applications/MAMP/tmp/mysql/
mysql.sock
```

Видите `-proot`? Буква `p` от слова password, а все остальное — это и есть пароль. Замените часть `root` новым паролем (у вас получится `-pnewpassword`).

Проделайте то же самое с каждым из этих пяти файлов.

После того как все эти изменения будут внесены и сохранены, вы получите правильно настроенный и защищенный от внешних вторжений сервер MySQL, а программа MAMP снова заработает.

Не забывайте ваш новый пароль, иначе его будет довольно сложно изменить (в главе 10 вы узнаете, как это делается). Можете записать свой пароль администратора MySQL прямо в книге.

МОЙ ПАРОЛЬ АДМИНИСТРАТОРА MySQL (MAC)

Пароль учетной записи администратора: _____ .

Установка в Linux

Люди, которые осознанно выбирают Linux в качестве операционной системы, технически грамотны настолько, что способны сами установить такое программное обеспечение, как Apache, PHP и MySQL. И скорее всего, у них есть собственное представление о том, *как* эта установка должна производиться. Поэтому нет смысла приводить здесь какие-то инструкции.

Если вы узнаете в этом описании себя, можете смело устанавливать самые последние версии Apache, PHP и MySQL, используя любой пакетный менеджер, который вам нравится. Темы, которые рассматриваются в данной книге, не настолько глубоки, чтобы тонкости настройки этих пакетов играли какую-то роль.

Те редкие пользователи Linux, которые нуждаются в некоторых советах по установке, могут обратиться к приложению А.

Что нужно знать о веб-хостинге

Пока вы разбираетесь с PHP и MySQL на своем компьютере, неплохо было бы собрать некоторую информацию о веб-хостинге. Она пригодится в дальнейшем, когда вы решите опубликовать сайт в Интернете.

Прежде всего нужно знать, как передаются файлы на ваш удаленный сервер. PHP-скрипты загружаются так же, как изображения, HTML- и CSS-файлы, из которых состоит статический веб-сайт. Если вам известно, как это происходит, то удаленный сервер можно лишний раз не тревожить. Если же вы начали работать с новым веб-хостингом, вам нужно знать, какие протоколы передачи данных он поддерживает (FTP или SFTP), а также имя пользователя и пароль, с помощью которых ваша программа будет производить подключение. Кроме того, вам понадобится информация о том, в какую директорию нужно копировать файлы, чтобы они были доступны для браузеров.

Уточните также некоторые детали, касающиеся сервера MySQL, предоставляемого вашим веб-хостингом: имя сервера, которое будет использоваться при под-

ключении (скорее всего, это localhost), а также логин и пароль для доступа к MySQL (они могут совпадать с учетной записью для (S)FTP). Наиболее вероятно, что веб-хостинг предоставит вам отдельную базу данных, чтобы вы не мешали другим пользователям, которые делят с вами общий MySQL-сервер. В таком случае вам придется получить имя этой базы данных.

Если вы собрали все необходимые данные о веб-хостинге, можете записать их прямо здесь.

ИНФОРМАЦИЯ О МОЕМ ХОСТИНГЕ

Протокол передачи данных (обведите нужный):	FTP SFTP
Имя (S)FTP-сервера:	_____
Имя пользователя для (S)FTP:	_____
Пароль для (S)FTP:	_____
Имя MySQL-сервера	_____
Имя пользователя для MySQL:	_____
Пароль для MySQL:	_____
Имя базы данных в MySQL:	_____

Ваш первый PHP-скрипт

С нашей стороны было бы нечестно сначала помочь вам установить все компоненты, а затем остановиться и вплоть до третьей главы не давать почувствовать, что же представляет собой PHP-скрипт. Следующий небольшой фрагмент кода должен подогреть ваш аппетит.

Откройте любой текстовый или HTML-редактор и создайте новый файл под названием **today.php** (`chapter1/today.php`). Наберите в нем следующий текст.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Сегодняшняя дата</title>
  </head>
  <body>
    <p>Сегодняшняя дата (согласно данному веб-серверу):
      <?php

      echo date('l, F jS Y.');

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

ЭТО БУКВА, А НЕ ЦИФРА

Вот самая важная строка в этом коде:

```
echo date('l, F jS Y.');
```

Обратите внимание на символ перед запятой. При наборе многие допускают ошибку в этом месте. Это не единица (1), а буква L в нижнем регистре (l).

РЕДАКТИРОВАНИЕ PHP-СКРИПТОВ В WINDOWS В БЛОКНОТЕ

Чтобы сохранить файл с расширением .php в программе Блокнот, в диалоговом окне Сохранить как в поле Тип файла выберите пункт Все файлы или заключите имя файла в кавычки. В противном случае программа сохранит ваш файл как today.php.txt, и скрипт не будет работать.

РЕДАКТИРОВАНИЕ PHP-СКРИПТОВ В MAC OS X В TextEdit

Будьте осторожны при использовании программы TextEdit для редактирования PHP-файлов. По умолчанию программа сохраняет их в формате Rich Text Format (RTF), добавляя к имени файла невидимое расширение .rtf. Чтобы сохранить новый PHP-файл, нужно преобразовать его в обычный текст, выбрав в меню TextEdit пункт Format ▶ Make Plain Text (⇧+⌘+T).

Еще один важный момент. Созданные PHP-файлы TextEdit открывает как документы в формате HTML, пытаясь отображать их в виде форматированного текста. Чтобы этого избежать, в диалоговом окне открытия файлов выберите пункт Ignore rich text commands.

ПОПРОБУЙТЕ БЕСПЛАТНУЮ ИНТЕГРИРОВАННУЮ СРЕДУ РАЗРАБОТКИ

Как вы уже могли понять из вышеприведенных рекомендаций, текстовые редакторы, которые поставляются с операционными системами, не совсем подходят для редактирования PHP-скриптов. Однако существуют и другие серьезные текстовые редакторы, а также интегрированные среды разработки (Integrated Development Environment, или IDE) с хорошей поддержкой языка PHP. Вот некоторые из них: NetBeans (http://www.netbeans.org/features/php/), Aptana (http://www.aptana.com/php), Komodo (http://www.activestate.com/komodo_edit/). Они работают в Windows, Mac OS X и Linux, к тому же их можно установить бесплатно.

Сохраните файл и переместите его в **корневую** директорию своего локального веб-сервера.

ГДЕ НАХОДИТСЯ КОРНЕВАЯ ДИРЕКТОРИЯ ВЕБ-СЕРВЕРА

Если вы устанавливали сервер Apache самостоятельно, то корневой директорией для него служит подкаталог htdocs внутри каталога, куда производилась установка: C:\Program Files\Apache Software Foundation\Apache2.2\htdocs для Windows или /usr/local/apache2/htdocs для Linux.

Для Apache из состава XAMPP корневая директория — это подкаталог htdocs внутри каталога, куда производилась установка. Чтобы его найти, выполните в меню Пуск цепочку команд Все программы ▶ Apache Friends ▶ XAMPP ▶ XAMPP htdocs folder.

Если вы используете сервер Apache, встроенный в Mac OS X, то его корневой каталог /Library/WebServer/Documents. Быстрый доступ к нему можно получить с помощью кнопки Open Computer Website Folder в разделе Web Sharing, который находится в меню Sharing системных настроек.

Для Apache из состава MAMP корневая директория — подкаталог htdocs внутри каталога, куда производилась установка (/Applications/MAMP/htdocs). При желании вы можете переместить корневую директорию в другое место, изменив настройки программы во вкладке Apache. Таким образом, указывая программе MAMP разные каталоги, вы получаете возможность легко переключаться между несколькими разрабатываемыми веб-проектами.

Откройте браузер и наберите в адресной строке http://localhost/today.php (или http://localhost:порт/today.php, если сервер Apache не использует стандартный порт 80), чтобы отобразить файл, который вы только что создали[1].

ВЫ ДОЛЖНЫ ВВЕСТИ АДРЕС URL

Раньше для предварительного просмотра веб-страниц вы загружали файлы напрямую с жесткого диска вашего компьютера, дважды щелкнув на имени файла или воспользовавшись в браузере пунктом меню Файл ▶ Открыть. Для PHP-файлов такой подход не годится. Как уже упоминалось ранее, для начала ваш веб-сервер должен сгенерировать код в формате HTML на основе PHP-скрипта и только затем отправить его в браузер. Чтобы браузер запросил файл у веб-сервера, достаточно набрать адрес URL (http://localhost/today.php).

На рис. 1.11 показано, как будет выглядеть веб-страница, сгенерированная вашим первым PHP-скриптом.

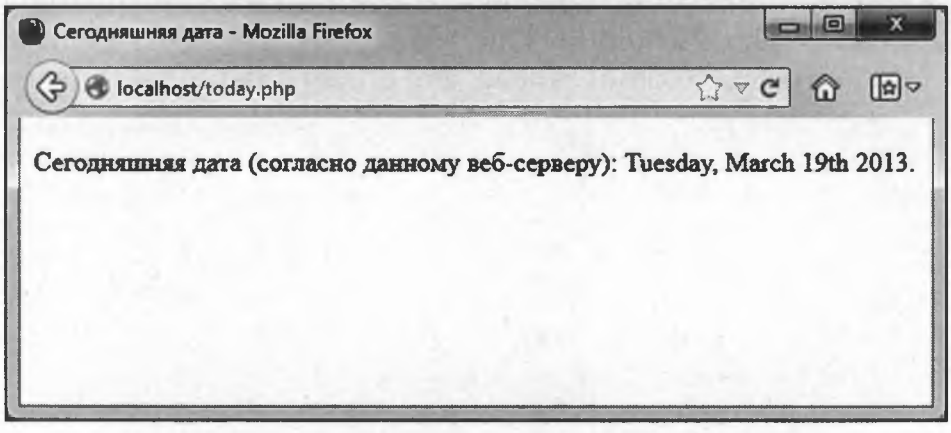


Рис. 1.11. Ваш первый PHP-скрипт в действии

Довольно просто, не так ли? Выбрав в браузере пункт меню Просмотр кода страницы, вы увидите обычный HTML-файл с датой внутри. Код на языке PHP (все, что находится между <?php и ?> в вышеприведенном примере) был интерпретирован веб-сервером и преобразован в обычный текст, а затем отправлен в браузер. Прелесть PHP и других скриптовых языков, работающих на стороне сервера, заключается в том, что браузеру не нужно о них ничего знать — обо всем позаботится веб-сервер.

[1] При установке Apache в Windows по умолчанию, скорее всего, будет использоваться порт 8080, при установке в составе программы MAMP — порт 8888.

Не переживайте, если вам не понятно, что означает набранный вами код. В главе 3 мы разберем, как именно он работает.

Если дата не отображается или ваш браузер предлагает загрузить PHP-файл вместо того, чтобы вывести его, значит, у вашего веб-сервера что-то не так с поддержкой PHP. Загляните в исходный код страницы, скорее всего, вы увидите там строки на языке PHP. Поскольку браузер не понимает этот язык, он воспринимает код `<?php ... ?>` как HTML-тег и игнорирует его. Еще раз убедитесь в том, что вы запросили файл у веб-сервера, а не открыли его на своем жестком диске (в адресной строке вашего браузера должен быть адрес URL, начинающийся с **http://localhost/**), и проверьте, поддерживает ли ваш веб-сервер PHP. Если вы следовали всем инструкциям, которые приводились ранее в этой главе, у вас все должно получиться.

Полный ящик инструментов и запачканные руки

Теперь у вас есть все необходимое: веб-сервер, поддерживающий PHP-скрипты, сервер баз данных MySQL и общее представление о том, как эти инструменты использовать. Вам даже пришлось немного потрудиться, чтобы написать и протестировать свой первый скрипт на языке PHP.

Если скрипт **today.php** у вас так и не заработал, загляните на форум SitePoint (**http://www.sitepoint.com/forums/**), мы с радостью поможем разобраться в возникшей проблеме.

В главе 2 вы познакомитесь с реляционными базами данных и начнете работать с MySQL. Кроме того, получите общее представление о структурированном языке запросов для баз данных (Structured Query Language, или SQL). Если вы никогда раньше не сталкивались с базами данных, это станет для вас настоящим откровением.

Глава 2
ЗНАКОМСТВО С MySQL

В главе 1 вы установили и настроили две программы: веб-сервер Apache с поддержкой PHP и сервер баз данных MySQL. И даже использование пакетов XAMPP или MAMP нисколько не должно помешать вам наслаждаться чувством удовлетворения от проделанной работы.

Как уже говорилось в предыдущей главе, PHP — скриптовый язык программирования, который работает на стороне сервера. С его помощью в HTML-код добавляются инструкции, выполняемые серверным программным обеспечением (как правило, Apache). Полученный результат возвращается в браузер, запросивший веб-страницу. На примере небольшого скрипта мы рассмотрели, как добавить на страницу текущую дату.

Это все хорошо, но самое интересное вас ждет впереди. В этой главе речь пойдет о MySQL. Вы узнаете, что такое базы данных и научитесь с ними работать, используя собственный сервер MySQL и структурированный язык запросов.

Введение в базы данных

Сервер баз данных (в нашем случае MySQL) — это программа, которая способна хранить большие объемы информации в структурированном виде и предоставлять доступ к ней с помощью таких языков программирования, как PHP. Например, с помощью PHP вы можете искать в базе данных шутки, которые хотите разместить на своем сайте.

В следующем примере вы создадите базу данных и заполните ее текстами шуток. Такой подход позволит решить две задачи одновременно. Во-первых, вместо того чтобы создавать для каждой шутки отдельную HTML-страницу, вы воспользуетесь одним PHP-скриптом, который найдет нужные тексты в базе данных и сгенерирует веб-страницу. Во-вторых, чтобы разместить шутку на сайте, достаточно будет добавить ее в базу данных — обо всем остальном позаботится код на языке PHP. Он автоматически извлечет новую шутку и выведет ее в числе прочих.

Давайте рассмотрим все более подробно и разберемся, каким образом в базе данных хранится информация. База данных включает одну или несколько **таблиц**, каждая из которых содержит список **элементов**, или **сущностей**. Исходя из этого, мы начнем работу с создания таблицы под названием joke (шутка), которая будет содержать тексты шуток. Любая таблица в базе данных состоит из одного или нескольких **столбцов**, или **полей**. Каждый такой столбец хранит определенный фрагмент информации. Наша таблица joke может состоять из двух столбцов: в первом будут храниться тексты с шутками, во втором — даты их добавления в базу данных.

Каждая такая шутка рассматриваться как строка, или запись. Описанные строки и столбцы формируют таблицу (рис. 2.1).

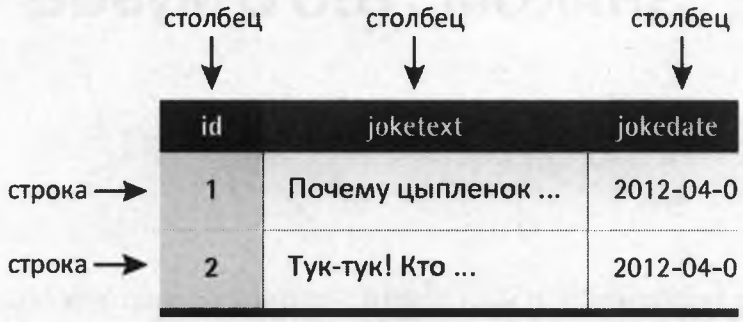


Рис. 2.1. Таблица базы данных, содержащая тексты шуток

Обратите внимание, что, помимо столбцов с текстами шуток (`joketext`) и датами их добавления (`jokedate`), есть еще один столбец `id`. Дело в том, что в таблицах базы данных каждая строка должна иметь однозначную идентификацию — это признак хорошего тона. Может оказаться так, что в одно и то же время в базу будут добавлены две одинаковые шутки, тогда при выводе отдельной шутки нельзя полагаться на столбцы `joketext` и `jokedate`. В столбце `id` каждой шутке присваивается уникальный номер, который помогает ссылаться на нее и отличать ее от других. Более подробно о проблемах проектирования баз данных рассказывается в главе 5.

Таким образом, на рис. 2.1 изображена таблица, состоящая из трех столбцов и двух строк, или записей. Каждая строка содержит три поля по числу столбцов: идентификатор шутки, ее текст и дату добавления в базу данных. Теперь, когда вы усвоили терминологию, приступим к использованию MySQL.

Использование приложения phpMyAdmin для выполнения SQL-запросов

Подобно тому как веб-сервер отвечает на пользовательские запросы из браузера, сервер баз данных MySQL отвечает на запросы **клиентских программ**. Чуть позже вы создадите собственное клиентское приложение для MySQL в виде PHP-скрипта, а пока воспользуемся клиентом phpMyAdmin, входящим в состав XAMPP и MAMP.

Замысловатое веб-приложение phpMyAdmin написано на языке PHP. Поскольку оно — часть XAMPP и MAMP, то предоставляется большинством коммерческих веб-хостингов с поддержкой PHP и MySQL. Разработчики используют его в качестве средства для управления базами данных сайта. Будучи столь же широко распространенным, как PHP и MySQL, приложение phpMyAdmin хорошо подходит начинающим программистам для изучения и использования.

У ВАС НЕТ phpMyAdmin?

Если вы решили следовать пошаговым инструкциям из приложения А вместо того, чтобы получить настроить веб-сервер с помощью пакетов, предлагаемых проектами XAMPP или MAMP, то у вас, скорее всего, нет установленной копии phpMyAdmin. В этом случае вы можете загрузить и установить phpMyAdmin с официального сайта http://www.phpmyadmin.net/. Там же находятся все необходимые инструкции.

Если вы используете XAMPP в Windows, то доступ к phpMyAdmin можно получить, нажав кнопку **Admin** напротив надписи **MySql** на панели управления XAMPP (рис. 2.2).

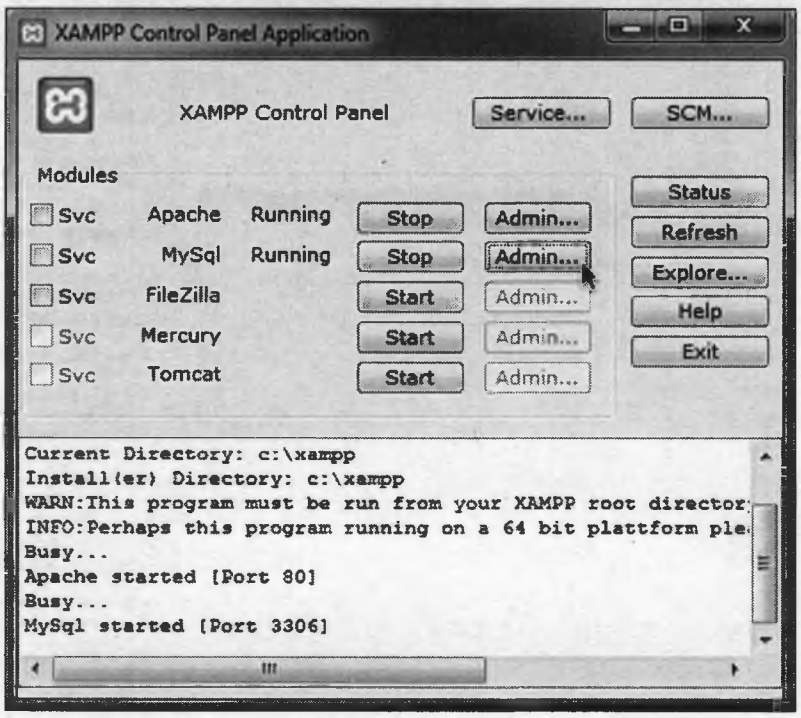


Рис. 2.2. Кнопка **Admin** поможет открыть phpMyAdmin

Чтобы получить доступ к phpMyAdmin с помощью MAMP в Mac OS X, нажмите кнопку **Open start page** в окне **MAMP** и перейдите на вкладку **phpMyAdmin** вверху экрана (рис. 2.3).

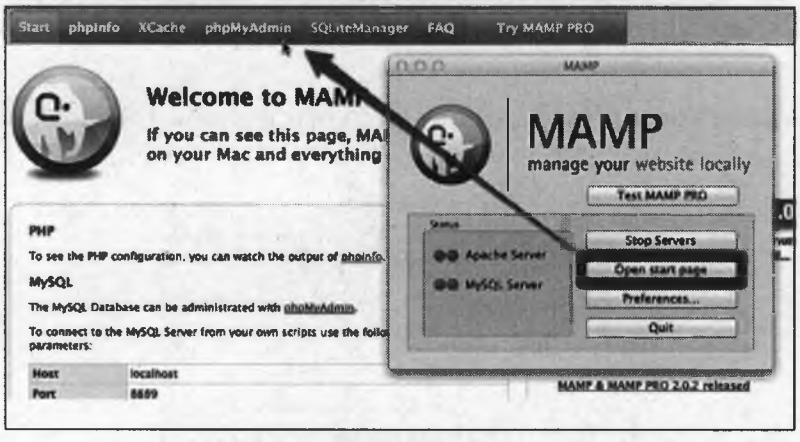


Рис. 2.3. Доступ к phpMyAdmin можно получить на странице приветствия MAMP

Итак, у вас в браузере должна открыться страница с phpMyAdmin (рис. 2.4). На момент написания книги в состав XAMPP входило приложение phpMyAdmin версии 3.5 с улучшенным интерфейсом (в дальнейшем будут представлены скриншоты именно этой версии). Более ранняя версия 3.3 отличается лишь внешним видом, принцип работы остался тем же.

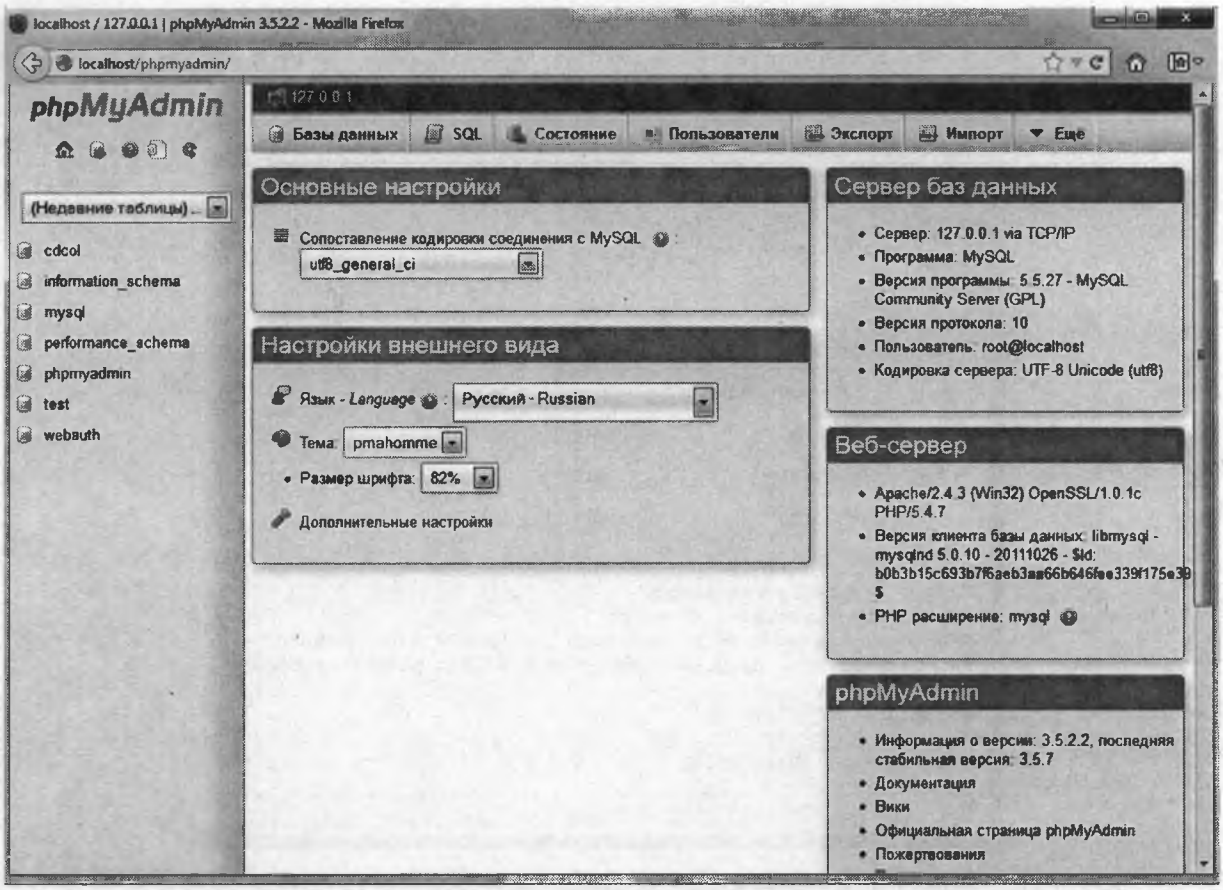


Рис. 2.4. Если вы видите такую страницу, значит, у вас есть phpMyAdmin

В phpMyAdmin вы обнаружите несколько ссылок, которые ведут к инструментам, необходимым для управления вашим MySQL-сервером и данными, которые он содержит. Пока пропустим эти возможности и сосредоточимся на окне для ввода SQL-запросов.

Внизу под логотипом phpMyAdmin находятся значки. Нажав на второй слева (рис. 2.5), вы откроете окно для ввода SQL-запросов (рис. 2.6).

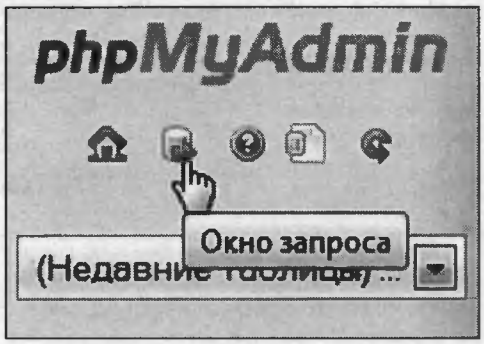


Рис. 2.5. Значок для открытия окна SQL-запросов

В большом пустом текстовом поле можно набирать команды, выполняя которые сервер баз данных будет решать определенные задачи. Попробуем ввести несколько команд, чтобы ознакомиться с работой MySQL-сервера.

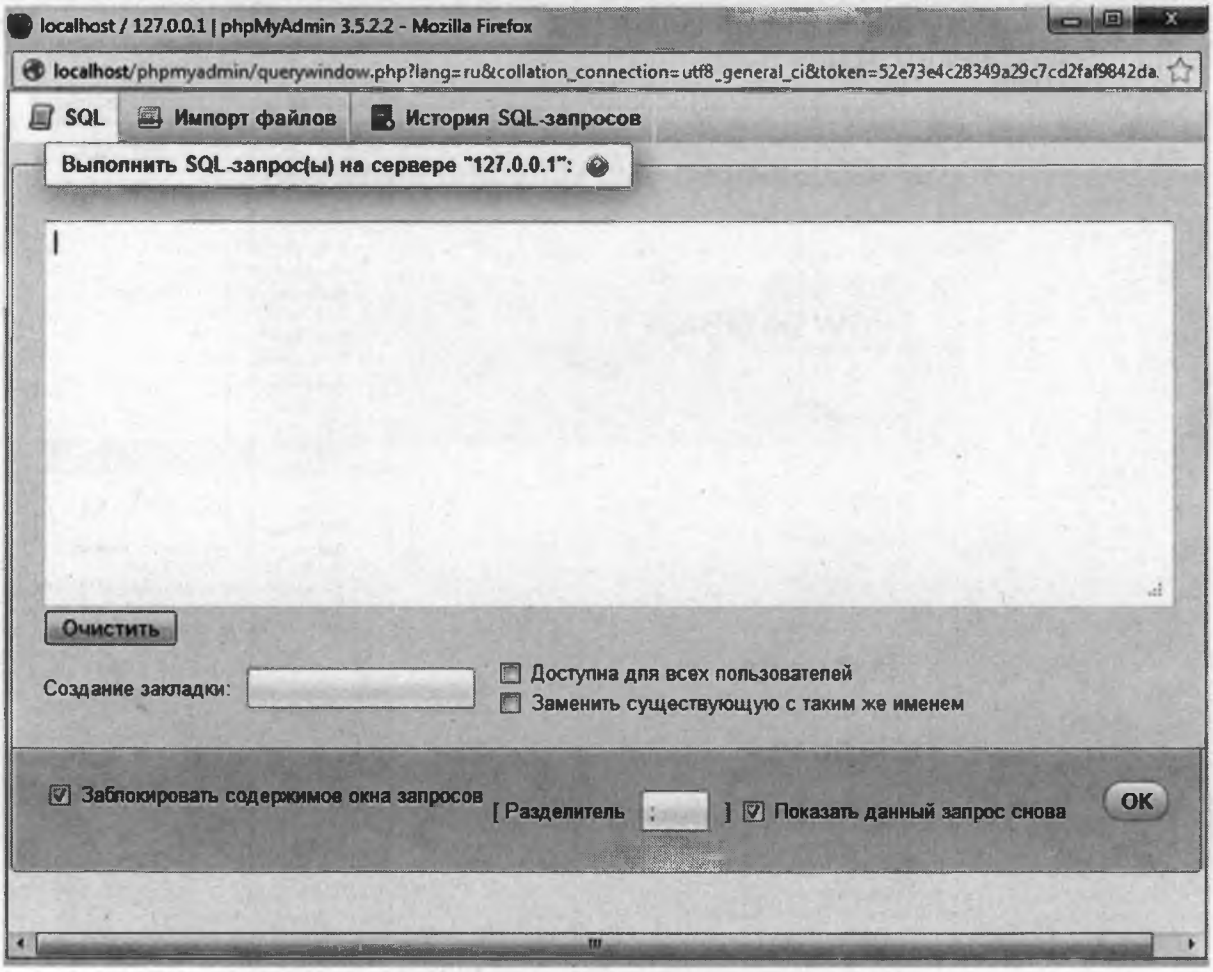


Рис. 2.6. Окно для ввода SQL-запросов

На самом деле сервер MySQL может управлять несколькими базами данных. Благодаря этому разные клиенты веб-хостинга могут использовать один и тот же экземпляр MySQL. В первую очередь после подключения к серверу вы должны выбрать базу данных, с которой будете работать. Для этого нужно получить список баз на текущем сервере.

Наберите в окне для SQL-запросов следующую команду и нажмите OK.

```
SHOW DATABASES
```

Сначала вам покажется, что ничего не происходит, однако вскоре в главном окне phpMyAdmin отобразятся результаты (рис. 2.7).

Если вы используете MAMP, полученный вами список будет не столь внушительным, как на рис. 2.7. XAMPP использует дополнительные базы данных для хранения собственных настроек, тогда как MAMP не засоряет ваш MySQL-сервер информацией и включает всего две необходимые. В любом случае вы обязательно обнаружите базы данных с названиями `information_schema` и `mysql`. Первую MySQL использует для учета всех остальных баз данных на сервере (если вы не планируете заниматься очень сложными проектами, лучше оставить ее как есть). Во второй MySQL хранит информацию о пользователях, их пароли и полномочия.

Более подробно мы рассмотрим эту тему в главе 10 при обсуждении вопросов администрирования MySQL.

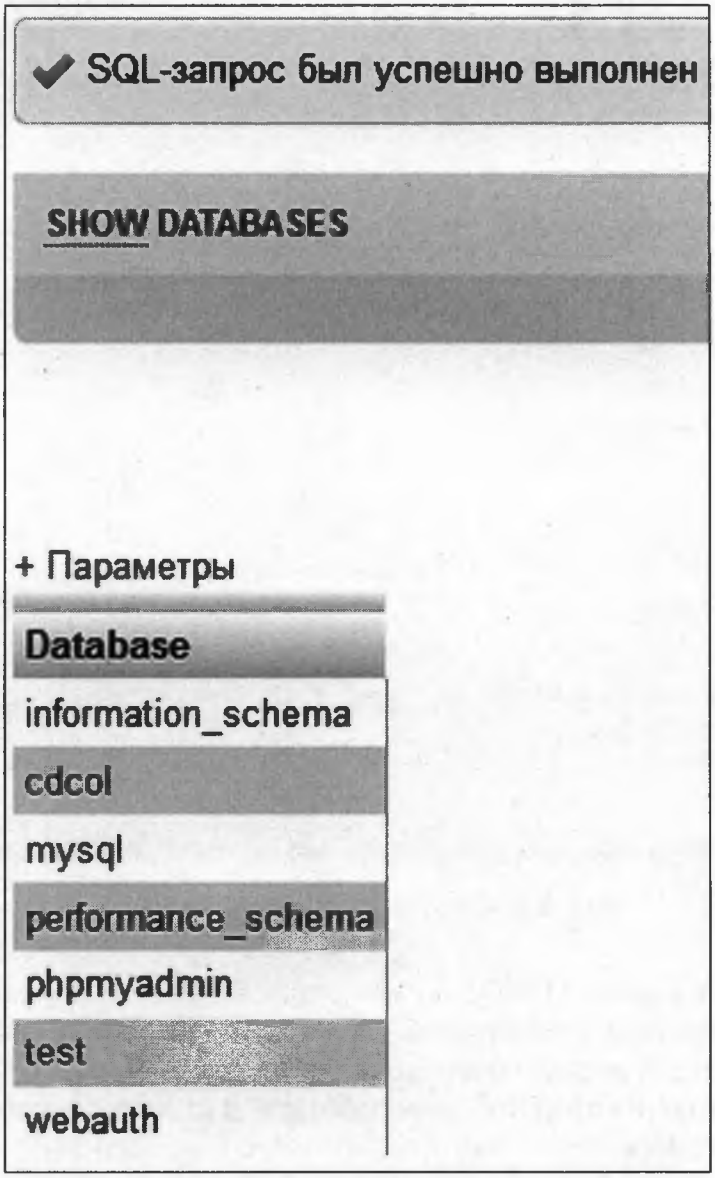


Рис. 2.7. Результаты запроса выводятся в главном окне phpMyAdmin

База данных `test` — это просто шаблон, который изначально поставляется вместе с MySQL (в MAMP она удалена). Ее можно смело удалить, поскольку вы займетесь созданием собственной базы данных.

Процесс удаления в MySQL выполняется командой DROP (Удалить):

```
DROP DATABASE test
```

После того как вы введете данную команду в окне для SQL-запросов и нажмете кнопку OK, скорее всего, phpMyAdmin выведет сообщение об ошибке Команда DROP DATABASE (Удалить базу данных) отключена. Таким образом встроенное в phpMyAdmin средство безопасности препятствует запуску критических запросов.

Чтобы удалить базу данных в phpMyAdmin, перейдите на вкладку **Базы данных** в главном окне phpMyAdmin (вверху крайняя слева) — вы увидите список баз вашего сервера. Установите флажок рядом с той, которую вы хотите удалить (в нашем случае это `test`), и нажмите кнопку **Удалить** внизу справа (рис. 2.8).

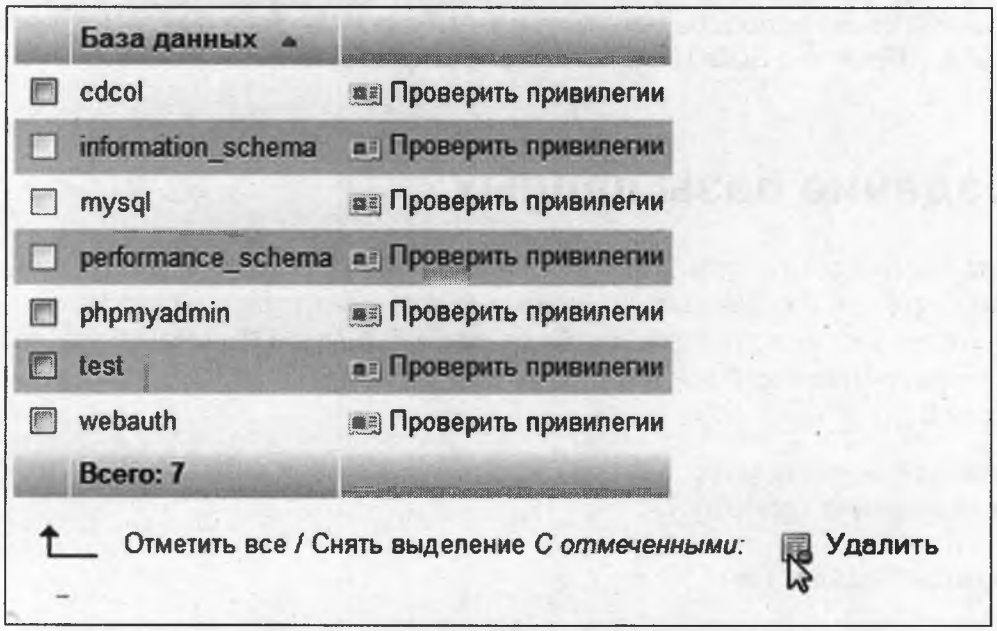


Рис. 2.8. Кнопка удаления баз данных в phpMyAdmin надежно спрятана

Приложение phpMyAdmin попросит вас подтвердить, действительно ли вы хотите удалить базу данных. В случае положительного ответа MySQL послушно выполнит вашу просьбу, а phpMyAdmin выведет контрольное сообщение о том, что операция прошла успешно.

Помимо `DROP DATABASE`, существуют и другие опасные команды, которые вы можете отправить серверу MySQL. Тем не менее далеко не всегда phpMyAdmin будет защищать вас от ваших же ошибок. Набирая SQL-запросы, соблюдайте осторожность, иначе с помощью одной команды можете уничтожить всю базу данных и ее содержимое.

Структурированный язык запросов

Команды, используемые для управления MySQL, являются частью стандарта — **структурированного языка запросов** (Structured Query Language, или **SQL**). Такие команды еще называют запросами. В дальнейшем в книге используются оба термина.

SQL — стандартный язык, который служит для взаимодействия со множеством баз данных. Даже если вы в будущем перейдете с MySQL на программу вроде Microsoft SQL Server, то обнаружите, что большая часть команд выглядит одинаково. Важно понять разницу между SQL и MySQL: MySQL — сервер баз данных, который вы используете, а SQL — язык, с помощью которого вы взаимодействуете с этими базами данных.

```
CREATE TABLE имя_таблицы (
  имяСтолбца1 типСтолбца1 подробностиОстолбце1,
  имяСтолбца2 типСтолбца2 подробностиОстолбце2,
  ⋮
) DEFAULT CHARACTER SET charset ENGINE=InnoDB
```

Давайте вернемся к примеру с таблицей `joke`, которая была изображена на рис. 2.1. Как вы помните, она состоит из трех столбцов: `id` (идентификатор шутки), `joketext` (текст шутки) и `jokedate` (дата добавления шутки в базу данных). Для создания такой таблицы необходимо выполнить следующую команду.

```
CREATE TABLE joke (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  joketext TEXT,
  jokedate DATE NOT NULL
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB
```

Выглядит жутко, правда? Давайте рассмотрим ее по частям.

```
CREATE TABLE joke (
```

Первая строка достаточно простая. Она говорит, что мы хотим создать новую таблицу под названием `joke`. Открывающая круглая скобка обозначает начало списка столбцов в таблице.

```
id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
```

Начало второй строки сообщает о том, что нам нужен столбец `id` для хранения значений типа integer (`INT`), так называемых целых чисел. В оставшейся части строки описываются дополнительные характеристики столбца:

- создавая строку в таблице, этот столбец нельзя оставить пустым (`NOT NULL`);
- если, добавляя запись, вы не укажите значение для этого столбца, то MySQL автоматически подберет для него число, которое будет превышать наибольшее значение в таблице на единицу (`AUTO_INCREMENT`);
- поскольку в таблице данный столбец играет роль уникального идентификатора для записей, все значения в нем должны быть уникальными (`PRIMARY KEY`).

```
joketext TEXT,
```

Третья строка говорит о том, что нам нужен столбец под названием `joketext`, который будет содержать текст (`TEXT`).

```
jokedate DATE NOT NULL
```

Четвертая строка описывает последний столбец `jokedate`, в котором будет храниться дата (`DATE`). Его также нельзя оставлять незаполненным (`NOT NULL`).

```
) DEFAULT CHARACTER SET utf8
```

Закрывающая круглая скобка обозначает конец списка столбцов в таблице.

`DEFAULT CHARACTER SET utf8` сообщает MySQL, что текст, содержащийся в этой таблице, должен быть закодирован в UTF-8. Это наиболее распространенная кодировка веб-страниц, поэтому она должна использоваться во всех таблицах базы данных.

```
ENGINE=InnoDB
```

Из этой строки MySQL узнает о том, какой движок для хранения данных нужно использовать при создании таблицы.

Говоря простым языком, **движок для хранения данных** — это формат файлов. Например, на готовых сайтах размещаются фотографии в формате JPEG, однако на этапе разработки дизайна страниц лучше использовать формат PNG. Оба формата поддерживаются браузерами, и у каждого из них есть свои сильные и слабые стороны. Аналогичным образом MySQL поддерживает несколько форматов таблиц для баз данных.

Формат InnoDB на сегодняшний день — наилучший для баз данных, которые используются при создании сайтов (именно с такой базой мы и будем работать). Однако по умолчанию в MySQL задается более ранний формат MyISAM, поэтому нужно указать, что мы хотим создать таблицу с использованием InnoDB.

Обратите внимание, что каждому столбцу мы назначили конкретный тип данных: столбец `id` будет содержать целые числа, `joketext` — текст, а `jokedate` — даты. MySQL требует, чтобы вы заранее указали типы данных для каждого столбца, — это помогает упорядочивать информацию и сравнивать значения в пределах одного столбца. Перечень типов данных, которые используются в MySQL, находится в приложении Г.

После того как вы введете вышеуказанную команду, нажмите **OK**. Если вы все набрали верно, в окне phpMyAdmin появится сообщение о том, что запрос выполнен успешно, и ваша первая таблица будет создана. В случае ошибки phpMyAdmin сообщит о том, что с запросом возникла проблема, и попытается указать место, где у MySQL возникли проблемы.

Убедимся в том, что полученная таблица создана должным образом. Введите в окне SQL-запросов следующую команду и нажмите **OK**.

```
SHOW TABLES
```

Приложение phpMyAdmin должно отобразить результат, показанный на рис. 2.11.

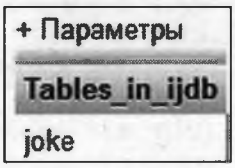


Рис. 2.11. phpMyAdmin перечислил таблицы в текущей базе данных

Это список всех таблиц в вашей базе данных под названием `ijdb`. В ней содержится всего одна таблица `joke`, которую вы только что создали. Пока все

выглядит хорошо. Давайте рассмотрим эту таблицу более детально, используя запрос DESCRIBE.

```
DESCRIBE joke
```

Как видно из рис. 2.12, таблица joke состоит из трех столбцов (или полей), которые в полученном нами ответе выведены в виде строк. Сейчас полученная информация выглядит немного странно, однако в дальнейшем все станет понятно. Пока у вас есть дела поважнее, например добавить шутки в таблицу.

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	*NULL*	auto_increment
joketext	text	YES		*NULL*	
jokedate	date	NO		*NULL*	

Рис. 2.12. phpMyAdmin перечисляет столбцы из таблицы joke в виде строк

Прежде чем приступить к заполнению таблицы, остановимся на том, как ее можно удалить. Выполнение этой задачи похоже на удаление базы данных с помощью команды DROP DATABASE, но в данном случае phpMyAdmin не станет вас уберегать от опасных запросов. Если вы на самом деле не хотите удалять таблицу joke, *не выполняйте* этот запрос! Однако если вы не прочь увидеть команду в действии, будьте готовы к тому, чтобы создать таблицу заново.

```
DROP TABLE имяТаблицы
```

Добавление данных в таблицу

Итак, вы создали базу данных и у вас есть готовая таблица, которую нужно заполнить шутками. Команда, которая добавляет данные в таблицу, называется INSERT. Она бывает двух видов.

```
INSERT INTO имяТаблицы SET
  имяСтолбца1 = значениеСтолбца1,
  имяСтолбца2 = значениеСтолбца2,
  ⋮

INSERT INTO имяТаблицы
  (имяСтолбца1, имяСтолбца2, …)
  VALUES (значениеСтолбца1, значениеСтолбца2, …)
```

Чтобы добавить шутку в таблицу, вы можете использовать любой вариант команды.

```
INSERT INTO joke SET
joketext = "Зачем цыпленок перешел дорогу? Чтобы попасть на другую
сторону!",
jokedate = "2012-04-01"
```

```
INSERT INTO joke
(joketext, jokedate) VALUES (
"Зачем цыпленок перешел дорогу? Чтобы попасть на другую сторону!",
"2012-04-01")
```

Заметим, что и в том, и в другом случае порядок следования столбцов в команде INSERT должен совпадать с порядком, в котором перечисляются значения. Сама очередность столбцов не важна. Попробуйте поменяйте местами пары столбец — значение и снова выполнить запрос.

При формировании этого запроса вы, должно быть, заметили, что для обозначения начала и конца текста шутки используются двойные кавычки ("). Заключенный подобным образом фрагмент текста называется **текстовой строкой**. Именно так представляется большая часть данных в SQL. К примеру, дата вводится в виде текста в формате "YYYY-MM-DD". Вместо двойных кавычек можно также использовать одинарные (').

```
INSERT INTO joke SET
joketext = 'Зачем цыпленок перешел дорогу? Чтобы попасть на другую
сторону!',
jokedate = '2012-04-01'
```

А как быть, если вам понадобится использовать кавычки внутри самой шутки? В таком случае при формировании запроса текст, в котором стоят одинарные кавычки, заключается в двойные, и наоборот.

Если же текст, который вы хотите добавить в запрос, содержит оба вида кавычек, вам придется **экранировать** конфликтующие символы в строке, поставив перед ними обратный слеш (\). Так MySQL узнает о том, что нужно игнорировать любое специальное значение данного символа. В случае с одинарными и двойными кавычками MySQL поймет, что их не нужно интерпретировать как конец текстовой строки.

Чтобы закрепить полученную информацию, рассмотрим пример, где с помощью команды INSERT добавляется шутка, которая содержит как одинарные, так и двойные кавычки.

```
INSERT INTO joke
(joketext, jokedate) VALUES (
'"Королеве - подвески, Констанции - подвязки, Портосу - подтяжки...", -
повторял, чтобы не перепутать, д\'Артаньян по дороге в Англию.',
"2012-04-01")
```

Как видите, начало и конец текста шутки отмечены одинарными кавычками. Поэтому одинарную кавычку для обозначения апострафа пришлось экранировать с помощью обратного слеша. Увидев его, MySQL поймет, что кавычку нужно воспринимать как символ внутри текста, а не как обозначение конца строки.

А как же в таком случае добавить в SQL-запрос обратный слеш? Нужно ввести два слеша подряд (\\). MySQL воспримет эту запись как одинарную косую черту, которая является частью текстовой строки.

Теперь, когда вы знаете, как добавить записи в таблицу, давайте разберем, как их оттуда извлечь.

Вывод сохраненных данных

Команда SELECT, которая используется для вывода информации, хранящейся в таблицах базы данных, одна из самых сложных в языке SQL. Причина этой сложности кроется в главном достоинстве базы данных — гибкости в извлечении информации. Поскольку ваш опыт работы с базами еще не богат, сосредоточимся на относительно примитивных результирующих списках и рассмотрим наиболее простые виды команды SELECT.

Следующая команда перечислит все, что хранится в таблице joke.

```
SELECT * FROM joke
```

Этот запрос можно перевести как «выбрать все из joke». Результат, полученный вами при выполнении данной команды, должен быть похож на тот, что приведен на рис. 2.13.

id	joketext	jokedate
1	Зачем цыпленок перешел дорогу? Чтобы попасть на др...	2012-04-01

Рис. 2.13. phpMyAdmin выведет все содержимое таблицы joke

Предположим, что вы работаете с этой базой данных над каким-нибудь серьезным проектом и не хотите отвлекаться на чтение шуток. Вы можете попросить MySQL пропустить столбец joketext с помощью следующего запроса.

```
SELECT id, jokedate FROM joke
```

На этот раз вместо команды «выбрать все» указываются конкретные столбцы, которые необходимо отобразить. Результат должен выглядеть, как на рис. 2.14.

id	jokedate
1	2012-04-01

Рис. 2.14. Вы можете выбрать только те столбцы, которые вам нужны

А что если вы захотите увидеть только часть шутки? Помимо возможности указать определенные столбцы, команда SELECT позволяет изменять способ вывода каждого из них с помощью определенных функций. Допустим, вам нужно извлечь только первые 20 символов из поля joketext. Воспользуйтесь для этого функцией LEFT, которая выводит в столбце определенное количество символов.

```
SELECT id, LEFT(joketext, 20), jokedate FROM joke
```

Результат запроса показан на рис. 2.15.

Еще одна полезная функция COUNT позволяет сосчитать количество полученных результатов. Например, чтобы узнать, сколько шуток в вашей таблице, наберите следующую команду.

```
SELECT COUNT(*) FROM joke
```

id	LEFT(joketext, 20)	jokedate
1	Зачем цыпленок переш	2012-04-01

Рис. 2.15. Функция LEFT сокращает количество символов в тексте до заданного значения

Как видно из рис. 2.16, в вашей таблице только одна шутка.

COUNT(*)
1

Рис. 2.16. Функция COUNT подсчитывает количество строк

Пока в рассмотренных нами примерах из таблицы извлекались все записи. Однако вы можете ограничить результат только теми записями, которые отвечают нужным требованиям. Подобные ограничения устанавливаются путем добавления к команде SELECT конструкции — оператора WHERE. Рассмотрим следующий пример.

```
SELECT COUNT(*) FROM joke WHERE jokedate >= "2012-01-01"
```

Данный запрос сосчитает количество добавленных шуток начиная с 1 января 2012 года. Когда речь идет о датах, понятие «больше или равно» означает «в тот же день или после». Используя тот же подход, можно искать записи, которые содержат определенный фрагмент текста. Взгляните на этот запрос:

```
SELECT joketext FROM joke WHERE joketext LIKE "%цыпленок%"
```

В результате его выполнения будет выведен текст тех шуток, где есть слово «цыпленок» в поле joketext. Ключевое слово LIKE сообщает MySQL о том, что указанный столбец должен удовлетворять заданному шаблону[1]. В нашем случае это "%цыпленок%". Символ % указывает на то, что до и (или) после слова «цыпленок» может находиться любой текст.

Операторе WHERE допускает сочетание нескольких условий, что позволяет получить более точный результат. Например, чтобы вывести только те шутки про д'Артаньяна, которые были добавлены в апреле 2012 года, воспользуйтесь следующим запросом.

```
SELECT joketext FROM joke WHERE
joketext LIKE "%д'Артаньян%" AND
jokedate >= "2012-04-01" AND
jokedate < "2012-05-01"
```

[1] Инструкция LIKE не чувствительна к регистру, поэтому данному шаблону также будут соответствовать шутки, содержащие подстроки «Цыпленок» или даже «ЗаБаВнЫйЦыПлЁнОк».

Глава 3
ЗНАКОМСТВО С PHP

PHP — язык, работающий на стороне сервера. Это может показаться немного сложным для понимания, особенно если до сих пор вы создавали сайты исключительно с помощью клиентских языков, таких как HTML, CSS и JavaScript.

Серверные языки во многом похожи на JavaScript, они точно так же позволяют встраивать небольшие программы (скрипты) в HTML-код веб-страницы. По сравнению с HTML такой подход дает больше возможностей для контролирования содержимого окна браузера. Главное отличие между JavaScript и PHP проявляется на этапе загрузки веб-страницы, когда эти встроенные программы запускаются.

Код, который написан на JavaScript и подобных ему языках, работающих на стороне клиента, считывается браузером и выполняется уже после загрузки веб-страницы с сервера. В то время как код, написанный с помощью серверных языков вроде PHP, выполняется веб-сервером до того, как страница будет отправлена в браузер. Клиентские языки позволяют управлять поведением страницы, которая уже отображается в браузере, серверные — генерируют модифицированные страницы налету, до того как они будут отправлены на сторону клиента.

Вместо встроенного PHP-кода веб-сервер вставляет в страницу результаты его выполнения. Получив эту страницу, браузеры видят стандартный HTML-код — отсюда термин серверный язык. Давайте вернемся к примеру today.php, о котором шла речь в главе 1 (`chapter3/today.php`).

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Сегодняшняя дата</title>
  </head>
  <body>
    <p>Сегодняшняя дата (согласно данному веб-серверу):
      <?php

      echo date('l, F jS Y.');

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

Значительная часть данного примера написана на языке HTML, и только участок между `<?php` и `?>` содержит PHP-код. Выражение `<?php` обозначает начало

встроенного PHP-скрипта, а ?> — его окончание. Веб-сервер интерпретирует все, что находится между этими разделителями, преобразует результат в обычный HTML-код и отправит полученную веб-страницу в браузер, который посылал запрос. Браузер получит следующий код.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Сегодняшняя дата</title>
  </head>
  <body>
    <p>Сегодняшняя дата (согласно данному веб-серверу):
      Sunday, March 24th 2013.    </p>
  </body>
</html>
```

Обратите внимание, что PHP-код исчез. Вместо него теперь находится результат выполнения скрипта, который выглядит как обычный HTML-код. Этот пример демонстрирует несколько преимуществ серверных скриптов.

- **Отсутствие проблем, связанных с совместимостью с браузерами.** Интерпретацией PHP-скриптов занимается исключительно веб-сервер, поэтому не стоит волноваться о том, поддерживает ли браузер пользователя возможности того языка, который вы использовали.
- **Доступ к серверным ресурсам.** В вышеописанном примере на веб-страницу добавлялась дата, полученная от сервера. JavaScript позволил бы получить только ту дату, которая установлена на компьютере пользователя. Примеры использования серверных ресурсов могут быть и куда более впечатляющими, например добавление на страницу содержимого, извлеченного из базы данных MySQL.
- **Снижение нагрузки на клиентскую часть.** JavaScript может существенно замедлить отображение веб-страницы (особенно на мобильных устройствах), поскольку, перед тем как вывести ее содержимое, браузеру необходимо выполнить скрипт. Серверный код переносит всю нагрузку на веб-сервер, работу которого можно сделать скоростной на столько, на сколько того требует ваше приложение (и позволяют денежные средства).

Базовые выражения и синтаксис

Синтаксис PHP покажется знакомым любому, кто понимает JavaScript, C, C++, C#, Objective-C, Java, Perl или любой другой язык, основанный на си. Но не стоит волноваться, если вы не имели с ними дела или новичок в программировании.

PHP-скрипт состоит из **наборов команд**, или **выражений**. Каждое из них представляет собой инструкцию, которая должна быть выполнена, прежде чем веб-сервер перейдет к следующей команде. Как и в вышеупомянутых языках, любое выражение в PHP всегда заканчивается точкой с запятой (;).

вместе они называются **арифметическими операторами**, поскольку выполняют арифметические действия с числами.

После каждого арифметического выражения следует **комментарий**. С его помощью вы можете описать то, что выполняет код. Хотя поясняющий текст и находится внутри программы, интерпретатор PHP его игнорирует. Комментарии начинаются с символов `//` и заканчиваются вместе с текущей строкой. Если комментарий занимает несколько строк, его следует выделить символами `/*` и `*/`. Интерпретатор PHP проигнорирует все, что находится между этими двумя разделителями. В оставшейся части книги комментарии помогут вам понять некоторые части представленного кода.

Но вернемся к операторам. Конструкция, которая соединяет части текста, называется **оператором конкатенации строк**.

```
$testVariable = 'Привет ' . 'всем!'; // присваивает значение 'Привет всем!'
```

Переменные можно использовать везде, где применяются конкретные символьные значения. Взгляните на следующий набор выражений.

```
$var1 = 'PHP'; // присваивает значение 'PHP' переменной $var1
$var2 = 5; // присваивает значение 5 переменной $var2
$var3 = $var2 + 1; // присваивает значение 6 переменной $var3
$var2 = $var1; // присваивает значение 'PHP' переменной $var2
echo $var1; // выводит 'PHP'
echo $var2; // выводит 'PHP'
echo $var3; // выводит '6'
echo $var1 . ' - наше все!'; // выводит 'PHP - наше все!'
echo "$var1 - наше все!"; // выводит 'PHP - наше все!'
echo '$var1 - наше все!'; // выводит '$var1 - наше все!'
```

Обратите внимание на две последние строки. Вы можете добавить имя переменной внутрь текста и получить ее значение в том же месте, если заключите строку в двойные кавычки вместо одинарных. Процесс преобразования имен переменных в их значения называется **интерполяцией**. Как видно на примере последней строки, текст, заключенный в одинарные кавычки, не интерполирует имена переменных в их содержимое.

Массивы

Массив представляет собой особый вид переменных, который содержит несколько значений. Если переменную можно представить в виде ящика, внутри которого находится значение, то массив — это ящик с отделениями, хранящий множество значений в отдельных секциях.

Самый простой способ создать массив в PHP — использовать команду `array`.

```
$myArray = array('один', 2, '3');
```

Приведенный код создает массив `$myArray`, который содержит три значения: `'один'`, `2` и `'3'`. Как и в случае с переменными, значение каждого элемента

массива может быть любого типа. В нашем примере первый и третий элементы хранят строки, второй — число.

Чтобы получить доступ к значению, хранящемуся в массиве, нужно знать его **индекс**. Как правило, для индексации значений внутри массива используются числа начиная с нуля: первое значение (или элемент) массива имеет индекс 0, второе — 1, третье — 2 и т. д, индекс *n*-го элемента массива равен *n* – 1. Таким образом, чтобы получить значение нужной вам переменной массива, укажите после ее имени соответствующий индекс в квадратных скобках.

```
echo $myArray[0]; // выводит 'один'
echo $myArray[1]; // выводит '2'
echo $myArray[2]; // выводит '3'
```

Каждое значение, хранящееся в массиве, называется **элементом** массива. Индекс в квадратных скобках используется как для добавления новых элементов, так и для перезаписи уже существующих значений.

```
$myArray[1] = 'два'; // присваиваем новое значение
$myArray[3] = 'четыре'; // создаем новый элемент
```

Элементы в конец массива можно добавлять с помощью обычного оператора присваивания. Для этого содержимое скобок, которые следуют за именем переменной, оставляют пустым.

```
$myArray[] = 'пятый элемент';
echo $myArray[4]; // выводит 'пятый элемент'
```

Как правило, в качестве индексов массива чаще используются числа, но их можно заменить и строками. В этом случае массив будет называться **ассоциативным**, поскольку он ассоциирует (сопоставляет) значения с индексами, имеющими определенный смысл. В следующем примере каждое из трех имен сопоставляется с датой, приведенной в виде строки.

```
$birthdays['Кевин'] = '1978-04-12';
$birthdays['Стефани'] = '1980-05-16';
$birthdays['Дэвид'] = '1983-09-09';
```

Ассоциативные массивы также создаются с помощью команды `array`. Вот как это делается на примере массива `$birthdays`.

```
$birthdays = array('Кевин' => '1978-04-12', 'Стефани' => '1980-05-16',
'Дэвид' => '1983-09-09');
```

Теперь, чтобы узнать день рождения Кевина, можно использовать его имя в качестве индекса:

```
echo 'Мой день рождения: ' . $birthdays['Кевин'];
```

Важность этого вида массивов вы сполна оцените в следующем разделе, где речь пойдет о взаимодействии с пользователями. С другими возможностями применения массивов вы познакомитесь в ходе чтения книги.

Формы для обеспечения взаимодействия с пользователями

Сегодня, создавая сайт, основанный на базе данных, нужно позаботится не только о том, чтобы просто динамически генерировать его страницы, но и о том, чтобы придать ему интерактивность, даже если это всего лишь строка поиска.

Опытные JavaScript-программисты под интерактивностью подразумевают механизм обработки событий, который позволяет реагировать на действия пользователя, например на движение курсора над ссылкой. Языки, работающие на стороне сервера, такие как PHP, в вопросах взаимодействия с пользователем более ограничены. PHP-код активируется только тогда, как на сервер поступает запрос, поэтому такого рода взаимодействие происходит исключительно по принципу «туда и обратно»: пользователь отправляет запрос на сервер, а сервер отсылает в ответ динамически сгенерированную страницу[1].

Чтобы обеспечить интерактивность в PHP, можно отправить информацию о действиях пользователя вместе с запросом новой веб-страницы. Сделать это довольно легко.

Передача переменных через ссылки

Самый простой способ переслать информацию, открывая веб-страницу, — использовать строку запроса в адресе URL. Если вы встречали адрес, где имя файла содержало вопросительный знак, значит, вы уже видели этот подход в действии. Например, если вы захотите найти в Google строку SitePoint, то при выводе результатов поиска он перенаправит вас на адрес URL следующего вида.

```
http://www.google.com/search?hl=en&q=SitePoint
```

Видите вопросительный знак внутри URL? Заметили, что текст, который идет после него, содержит ваш поисковый запрос (SitePoint)? Эта информация была отправлена при открытии страницы **http://www.google.com/search**.

Напишем собственный небольшой пример. Создайте обычный HTML-файл под названием **name.html** (расширение **.php** указывать не нужно, поскольку в этом файле PHP-кода не будет) и добавьте в него следующую ссылку (`chapter3/links1/name.html`, фрагмент):

```
<a href="name.php?name=Кевин">Привет, я Кевин!</a>
```

Это ссылка на файл с именем **name.php**, при нажатии которой выполняется не только запрос страницы, но и отправка переменной. Переменная передается как часть строки запроса, которая входит в состав URL и следует за вопросительным

[1] После того как в среде JavaScript выросла популярность Ajax-технологии, ситуация несколько изменилась. Сейчас код JavaScript в ответ на действия пользователя способен отправлять запросы веб-серверу и запускать PHP-скрипты. Однако, чтобы не отклоняться от основной темы, мы сосредоточимся на классических приложениях, которые не используют технологию Ajax. Для досконального изучения данной технологии прочтите книгу Эрла Каслдайна (Earle Castledine) и Крэйга Шарки (Craig Sharkie) «Изучаем jQuery»(Query: Novice to Ninja).

знаком. Она называется name и имеет значение Кевин. Другими словами, вы создали ссылку, которая загружает файл name.php и сообщает PHP-коду из этого файла, что name равно Кевин.

Чтобы понять, к чему приводит нажатие данной ссылки, нам понадобится скрипт name.php. Создайте новый HTML-файл, но на этот раз сохраните его с расширением .php. Так веб-сервер поймет, что файл содержит код на языке PHP. Внутри тега body наберите следующий текст (chapter3/links1/name.php, фрагмент):

```
<?php
$name = $_GET['name'];
echo 'Добро пожаловать на наш веб-сайт, ' . $name . '!';
?>
```

Поместите оба файла (name.html и name.php) в каталог вашего веб-сервера и откройте первый из них в браузере. Адрес URL должен выглядеть как http://localhost/name.html (или http://localhost:8888/name.html, если ваш веб-сервер не использует 80-й порт). Щелкните на ссылке, которая отобразилась на странице, чтобы запросить PHP-скрипт. В результате вы увидите надпись: «Добро пожаловать на наш веб-сайт, Кевин!» (рис. 3.1).

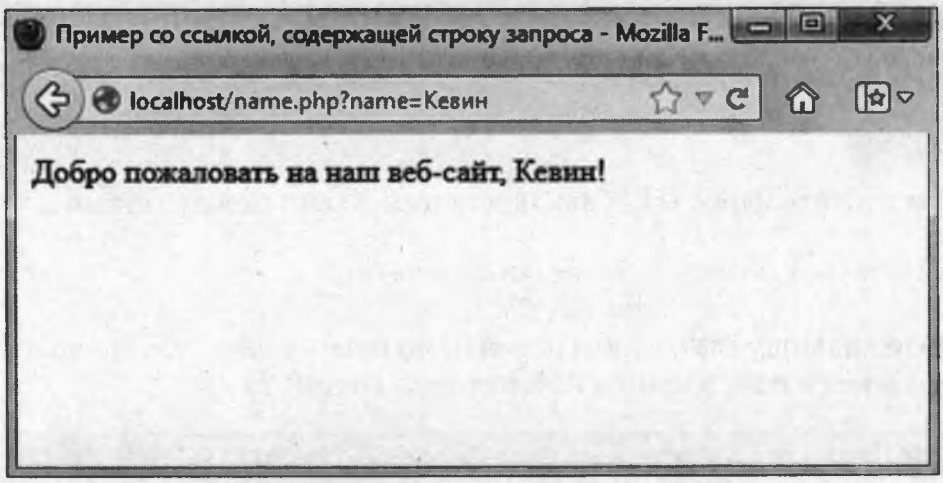


Рис. 3.1. Приветствие, адресованное пользователю

Теперь рассмотрим код, благодаря которому получен такой результат. Вот самая важная строка (chapter3/links1/name.php, фрагмент):

```
$name = $_GET['name'];
```

Если вы внимательно прочитали раздел «Массивы», то поймете, что делает эта строка. Она присваивает новой переменной $name значение, хранящееся в элементе 'name' массива $_GET. Но откуда взялся $_GET?

$_GET — это одна из целого ряда переменных, которые PHP создает автоматически при поступлении запроса от браузера. Она представляет собой массив, содержащий все значения, передаваемые через строку запроса в адресе URL. Данный массив ассоциативный, поэтому доступ к переменной name, переданной в строке запроса, можно получить с помощью выражения $_GET['name']. Скрипт

name.php присваивает это значение обычной переменной (`$name`), а затем выводит его как часть текстовой строки, используя команду echo (`chapter3/links1/name.php`, отрывок).

```
echo 'Добро пожаловать на наш веб-сайт, ' . $name . '!';
```

Значение переменной `$name` добавлено в строку вывода с помощью оператора конкатенации (.), который рассмотрен в разделе «Переменные, операторы и комментарии».

Язык программирования PHP прост не только для изучения, но и для создания небезопасных сайтов. Поэтому, прежде чем углубится в дальнейшее изучение языка, попробуем выявить и исправить конкретную проблему с безопасностью, которая возникла в этом коде и которая довольно часто встречается в Интернете в наши дни. Она связана с тем, что содержимое страницы, которую генерирует скрипт **name.php**, находится под контролем пользователя (речь идет о переменной `$name`). И хотя эта переменная получает значение из строки запроса по ссылке на странице **name.html**, злоумышленник может отредактировать адрес URL и послать скрипту другое значение.

Чтобы увидеть, как это работает, щелкните на ссылке в файле **name.html** еще раз. Перейдя на страницу с приветственным сообщением, взгляните на URL в адресной строке своего браузера. Он должен выглядеть примерно так:

```
http://localhost/name.php?name=Кевин
```

Отредактируйте адрес URL, заключив имя Кевин между тегами `b` и `/b`:

```
http://localhost/name.php?name=<b>Кевин</b>
```

Нажмите клавишу **Enter**, чтобы перейти по новому адресу. Теперь имя на странице отображается полужирным начертанием (рис. 3.2)[1].

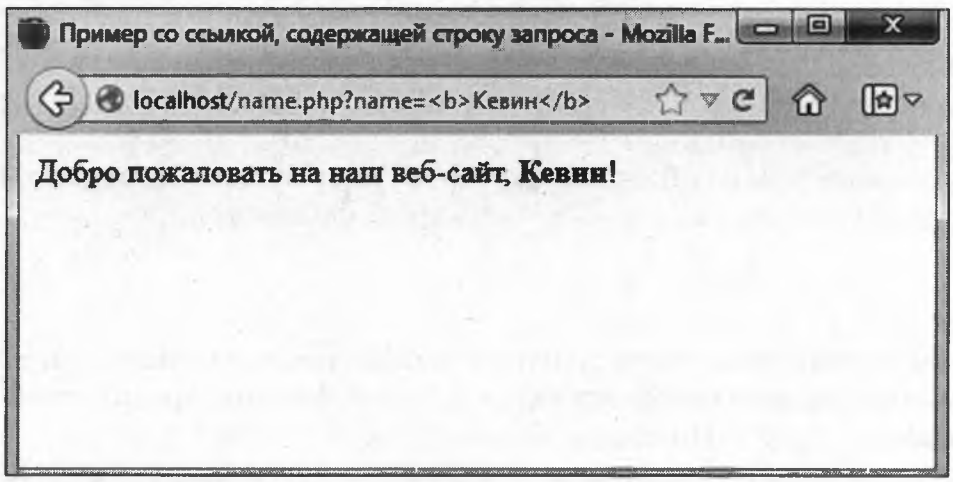


Рис. 3.2. Уязвимость в безопасности всегда на руку злоумышленникам

[1] Возможно, вы уже заметили, что некоторые браузеры автоматически преобразуют символы < и > в набор символов, соответствующих кодировке в формате URL (`%3C` и `%3E` соответственно). В любом случае PHP получит одно и то же значение.

Таким образом, пользователь может ввести в адрес URL любой HTML-код, и ваш PHP-скрипт без лишних вопросов добавит его в код генерируемой страницы. В нашем случае это безобидный тег b, но злоумышленник может использовать сложный код на языке JavaScript и, например, украсть пароль пользователя. Ему понадобится лишь опубликовать видоизмененную ссылку на подконтрольном сайте и привлечь внимание пользователя, чтобы заставить его на ней щелкнуть. Взломщик может добавить такую ссылку в электронное письмо и разослать его вашим посетителям. Если ему это удастся, код будет встроен в вашу страницу — и ловушка сработает!

Не хотелось бы пугать вас рассказами о хакерах, которые заставляют ваш PHP-код работать против вас самих, особенно когда вы только приступили к изучению языка. Но факт остается фактом: самая слабая сторона PHP заключается в его простоте, которая способна порождать подобную уязвимость. При изучении языка PHP на профессиональном уровне бóльшая часть времени уходит на то, чтобы научиться избегать проблем с безопасностью. Однако чем раньше вы столкнетесь с этими проблемами и научитесь обходить их стороной, тем легче вам придется в будущем.

Так как же сгенерировать страницу, содержащую имя пользователя, не боясь подвергнуться опасности со стороны взломщиков? Решение заключается в том, чтобы воспринимать значение, предоставленное для переменной `$name`, как обычный текст, который выводится на сайте, а не как фрагмент HTML, встраиваемый в код страницы.

Еще раз откройте файл **name.php** и отредактируйте код, чтобы он выглядел следующим образом (`chapter3/links2/name.php`, фрагмент):

```
<?php
$name = $_GET['name'];
echo 'Добро пожаловать на наш веб-сайт, ' .
    htmlspecialchars($name, ENT_QUOTES, 'UTF-8') . '!';
?>
```

Этот код содержит много интересного. Первая строка осталась прежней: переменной `$name` присваивается значение элемента `'name'` из массива `$_GET`. А вот идущая далее команда `echo` изменилась до неузнаваемости. В предыдущей версии кода переменная `$name` выводилась как есть, теперь используется встроенная в PHP функция `htmlspecialchars`, которая выполняет важное преобразование.

Как вы помните, дыра в безопасности скрипта **name.php** возникла из-за того, что страница генерировалась с помощью кода из переменной `$name`, который мог использовать все возможности языка разметки HTML. Функция `htmlspecialchars` преобразовывает специальные HTML-символы < и > в строки вида `<` и `>`. Благодаря этому браузер не может интерпретировать их как HTML-код.

Рассмотрим подробнее строку, где вызывается функция `htmlspecialchars`. Она первая в этой книге принимает более одного аргумента.

```
htmlspecialchars($name, ENT_QUOTES, 'UTF-8')
```

В качестве первого аргумента выступает переменная `$name` (текст, который нужно преобразовать). Второй аргумент — константа[1] `ENT_QUOTES`, которая сообщает функции `htmlspecialchars`, что, кроме прочих специальных символов, нужно преобразовать еще одинарные и двойные кавычки. Третий параметр — строка `'UTF-8'`, благодаря которой PHP определят, какую кодировку нужно использовать для интерпретации заданного текста.

ПРЕИМУЩЕСТВА И ПОДВОДНЫЕ КАМНИ UTF-8 В PHP

Как вы уже заметили, в верхней части HTML-кода из примеров в данной книге находится тег `meta` в виде:

```
<meta charset="utf-8">
```

Данный тег говорит браузеру о том, что страница, которую он загружает, закодирована в UTF-8[2]. Благодаря кодировке пользователи могут отправлять текст, содержащий тысячи символов, используемых в различных языках, с которыми в противном случае ваш сайт не справился бы. Многие встроенные в PHP функции, например `htmlspecialchars`, исходят из того, что по умолчанию используется более простая кодировка ISO-8859-1 (или Latin-1). Поэтому вызывая их, нужно сообщать о том, что вы работаете с UTF-8.

Если вы хотите использовать в коде дополнительные символы (например, фигурные кавычки, тире) или буквы, не входящие в английский алфавит (например, é), нужно также настроить текстовый редактор, чтобы он сохранял ваши HTML- и PHP-файлы в кодировке UTF-8. Код, представленный в этой книге, прежде всего призван обеспечить безопасность и использует последовательности символов из языка HTML (например, `&rsquo;` для правой фигурной кавычки), которые будут работать, невзирая ни на что.

Откройте в браузере файл **name.html** и щелкните на ссылке, которая теперь указывает на обновленную версию **name.php**. Вы снова увидите сообщение с приветствием «Добро пожаловать на наш веб-сайт, Кевин!». Еще раз отредактируйте адрес URL, заключив имя в теги `b` и `/b`.

```
http://localhost/name.php?name=<b>Кевин</b>
```

На этот раз после нажатия клавиши **Enter** вы увидите набранный вами текст (рис. 3.3), а не имя, выведенное полужирным шрифтом.

Если вы посмотрите на исходный код страницы, то убедитесь в том, что функция `htmlspecialchars` выполнила свою работу и преобразовала символы < и >, указанные в имени, в соответствующие HTML-последовательности `<` и `>`. Теперь, когда злоумышленники попробуют встроить в ваш сайт нежелательный код, он отобразится на странице в виде простого текста.

---

[1] В языке PHP константа — это переменная, значение которой остается неизменным. Ее имя не должно начинаться со знака $. PHP содержит несколько встроенных констант, с помощью которых можно управлять стандартными функциями.

[2] UTF-8 — кодировка символов, один из многочисленных стандартов для представления текста в компьютерной памяти в виде последовательности единиц и нулей. Ознакомиться с символьными кодировками можно в статье The definitive guide to web character encoding («Полное руководство по кодировкам символов в Интернете») по адресу **http://www.sitepoint.com/article/guide-web-character-encoding/**.

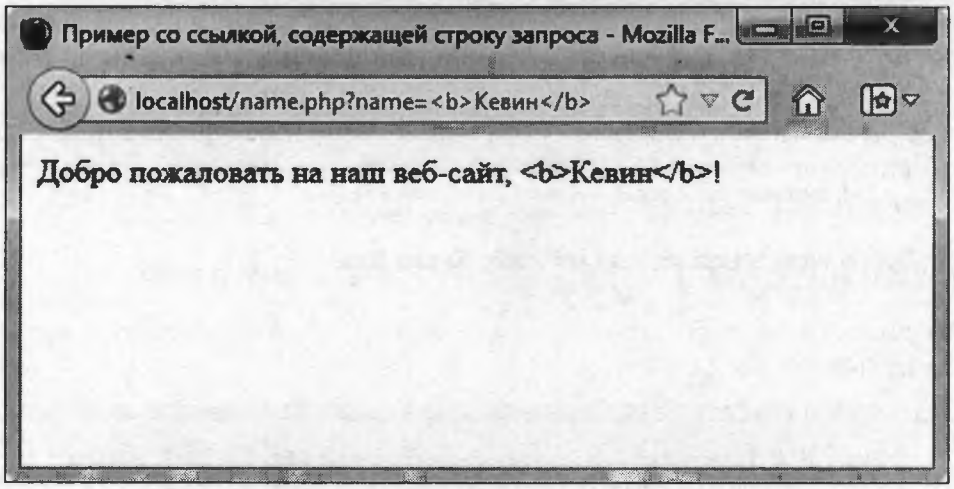


Рис. 3.3. Хотя эта надпись не такая красивая, зато безопасная

Далее мы будем часто прибегать к использованию функции `htmlspecialchars`, чтобы избавиться от такого рода уязвимостей. Не переживайте, если прямо сейчас вы не уловили все тонкости применения данного подхода. Вы и сами не заметите, как использование этой функции войдет у вас в привычку. А пока рассмотрим некоторые более сложные методики передачи значений PHP-скриптам при их вызове.

Согласитесь, передача одной переменной в строке запроса — уже неплохо. Однако при желании вы можете передавать более одного значения. Рассмотрим усложненный вариант предыдущего примера. Еще раз откройте файл **name.html** и измените ссылку, указывающую на **name.php**, используя следующую строку запроса (`chapter3/links3/name.html`, фрагмент):

```
<a href="name.php?firstname=Кевин&amp;lastname=Янк">Привет, я Кевин
Янк!</a>
```

На этот раз наша ссылка передает две переменные: `firstname` и `lastname`, разделенные в строке запроса амперсандом (знак `&`, который в HTML и в адресе URL записывается как `&`). Разделяя каждую пару `имя=значение` амперсандом, вы можете передавать еще больше переменных.

В файле **name.php** для значений используются две переменные (`chapter3/links3/name.php`, фрагмент).

```
<?php
$firstName = $_GET['firstname'];
$lastName = $_GET['lastname'];
echo 'Добро пожаловать на наш веб-сайт, ' .
    htmlspecialchars($firstName, ENT_QUOTES, 'UTF-8') . ' ' .
    htmlspecialchars($lastName, ENT_QUOTES, 'UTF-8') . '!';
?>
```

Команда `echo` стала довольно объемной, но вам все равно следует разобраться, как она работает. Используя операторы конкатенации (`.`), данная команда выводит строку `Добро пожаловать на наш веб-сайт`, после которой следуют значение `$firstName`, пробел, значение `$lastName` и восклицательный знак. При этом оба

значения `$firstName` и `$lastName` обработаны функцией `htmlspecialchars` для безопасного вывода. Результат выполнения команды показан на рис. 3.4.

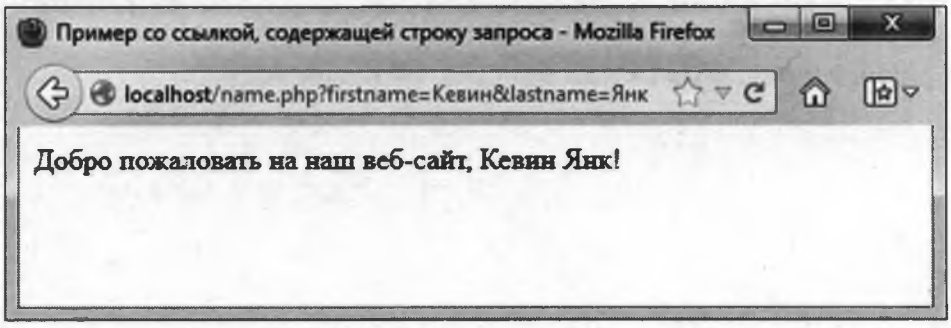


Рис. 3.4. Приветственное сообщение стало более детальным

Однако наша цель — достичь полноценного взаимодействия с пользователем, при котором PHP будет обрабатывать произвольно введенную информацию. Продолжая работу с кодом, который выводит приветственное сообщение, предложим пользователю ввести свое имя, чтобы оно отобразилось на итоговой веб-странице. Для этого нам понадобится HTML-форма.

Передача переменных с помощью форм

Удалите из файла **name.html** ссылку и вставьте на ее место HTML-код для создания формы (`chapter3/forms1/name.html`, фрагмент).

```
<form action="name.php" method="get">
  <div><label for="firstname">Имя:      &n
bsp; 
    <input type="text" name="firstname" id="firstname"></label>
  </div>
  <div><label for="lastname">Фамилия:
    <input type="text" name="lastname" id="lastname"></label>
  </div>
  <div><input type="submit" value="Поехали"></div>
</form>
```

Форма, полученная в результате выполнения этого кода, показана на рис. 3.5.

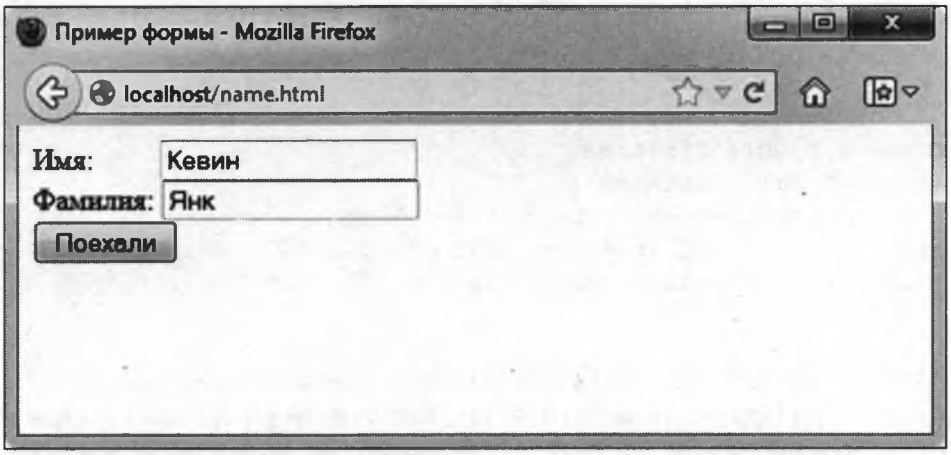


Рис. 3.5. Создаем персональное приветственное сообщение

СОДЕРЖАНИЕ ПРЕВЫШЕ ФОРМЫ

Вне всяких сомнений, созданная форма выглядит довольно просто. Однако, применив CSS в этом и последующих примерах, вы можете сделать их более привлекательными. Поскольку данная книга посвящена PHP и MySQL, мы будем придерживаться стандартного внешнего вида. О стилизации форм с помощью CSS, можно прочесть в книге The CSS3 Anthology («Антология CSS3») издательства SitePoint (http://www.sitepoint.com/books/cssant4/).

Принцип работы полученной формы точно такой же, как и у второй ссылки, рассмотренной в разделе «Передача переменных через ссылки» (с текстом `firstname=Кевин&lastname=Янк` в строке запроса), только теперь вы можете ввести абсолютно любое имя. После того как вы нажмете кнопку отправки **Поехали**, браузер загрузит скрипт **name.php**, автоматически добавив переменные и их значения в строку запроса. Имена переменных будут получены из атрибутов `name`, принадлежащих тегам `<input type="text">`, а значения — из текста, набранного пользователем в полях ввода.

АПОСТРОФЫ В ПОЛЯХ ФОРМЫ

Если фамилия, которую вы ввели содержит апостроф (например, Молли О'Рейлли), то в приветственном сообщении перед апострофом вы увидите косую черту: `Добро пожаловать на наш веб-сайт, Молли О\'Рейлли!` Ее появление связано с одной из особенностей встроенной в PHP системы безопасности, которая называется **волшебные кавычки**. Вы изучите ее в главе 4.

С помощью атрибута `method`, принадлежащего тегу `form`, браузер узнает о том, каким образом отправлять переменные и их значения вместе с запросом. Если атрибут имеет значение `get`, использованное нами в файле **name.html**, то информация передается в виде строки запроса (и появляется в массиве `$_GET`). Однако иногда появление данных в строке запроса может быть нежелательным или даже технически неосуществимым. Что если мы добавим в нашу форму тег `textarea`, чтобы пользователи могли вводить большие объемы текста? Строка запроса в адресе, включающая несколько абзацев, будет неимоверно длинной и может превысить максимальную длину URL, допустимую в современных браузерах. Альтернативный вариант заключается в том, чтобы передавать информацию скрытно.

Еще раз отредактируйте файл **name.html**. Измените атрибут `method`, установив для него значение `post` (`chapter3/forms2/name.html`, фрагмент).

```
<form action="name.php" method="post">
  <div><label for="firstname">Имя:      &n
bsp; 
    <input type="text" name="firstname" id="firstname"></label>
  </div>
  <div><label for="lastname">Фамилия:
    <input type="text" name="lastname" id="lastname"></label>
  </div>
  <div><input type="submit" value="Поехали"></div>
</form>
```

Новое значение атрибута `method` заставит браузер отправлять переменные формы незаметно, как часть запроса к странице, не добавляя их в строку запроса в адресе URL.

Поскольку переменные больше не отправляются внутри строки запроса, в массиве $_GET их тоже больше не будет. Вместо этого они помещаются в другой массив $_POST, созданный специально для значений, отправленных методом post. Теперь, чтобы получить переменные из нового массива, нужно отредактировать скрипт **name.php** (chapter3/forms2/name.php, фрагмент).

```
<?php
$firstname = $_POST['firstname'];
$lastname = $_POST['lastname'];
echo 'Добро пожаловать на наш веб-сайт, ' .
    htmlspecialchars($firstname, ENT_QUOTES, 'UTF-8') . ' ' .
    htmlspecialchars($lastname, ENT_QUOTES, 'UTF-8') . '!';
?>
```

В результате отправки новой формы появится страница, показанная на рис. 3.6.

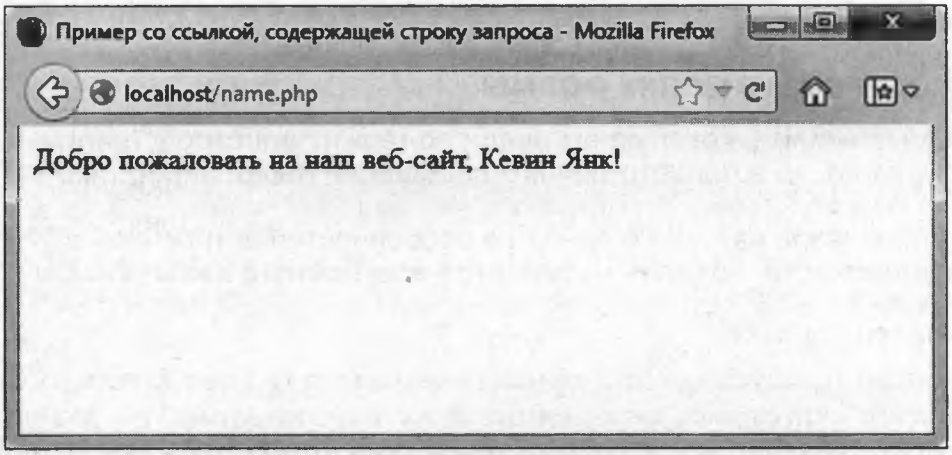


Рис. 3.6. Приветствие, полученное без строки запроса

Данная форма по своим функциям ничем не отличается от предыдущей. Единственное различие между ними заключается в том, что теперь адрес страницы, которая загружается при нажатии кнопки **Поехали**, не будет содержать строку запроса. С одной стороны, это позволяет передавать с помощью формы достаточно большой объем информации (или конфиденциальные данные, например пароли), не показывая ее в адресной строке. С другой стороны, если пользователь добавит в закладки страницу, которая загрузилась в результате отправки формы, то такая закладка окажется совершенно бесполезной, поскольку ей не хватит переданных данных. Именно по этой причине поисковые системы используют строки запроса для передачи критериев поиска. Если вы добавите в закладки страницу с результатами поиска на сайте Google, то позднее с ее помощью вы сможете выполнить точно такой же запрос, потому что условия поиска содержатся в адресе URL.

Иногда доступ к переменной необходимо получить вне зависимости от того, как ее отправили — с помощью формы или как часть строки запроса. В такой ситуации вам пригодится массив $_REQUEST. Он содержит все значения, которые передаются в массивы $_GET и $_POST. Попробуйте еще раз изменить скрипт обработки формы таким образом, чтобы он получал имя и фамилию пользователя из любого источника (chapter3/forms3/name.php, фрагмент):

```
<?php
$firstname = $_REQUEST['firstname'];
$lastname = $_REQUEST['lastname'];
echo 'Добро пожаловать на наш веб-сайт, ' .
    htmlspecialchars($firstname, ENT_QUOTES, 'UTF-8') . ' ' .
    htmlspecialchars($lastname, ENT_QUOTES, 'UTF-8') . '!';
?>
```

На этом знакомство с основами использования форм для обеспечения взаимодействия с пользователем посредством PHP закончено. В последующих примерах рассмотрены более серьезные проблемы и методы их решения.

Управляющие конструкции

До сих пор все примеры PHP-кода в этой книге представляли собой либо несложный скрипт, который выводит текст на веб-страницу, либо последовательность выражений, которые выполняются по порядку. Если вы когда-либо писали программы на других языках (JavaScript, Objective-C, Ruby или Python), то вы уже знаете, что они очень редко бывают настолько простыми.

Как и любой другой язык программирования, PHP предоставляет средства, помогающие управлять ходом выполнения программы. Он содержит специальные выражения, которые позволяют изменить прямую последовательность команд, характерную для предыдущих примеров. Такие выражения называются **управляющими конструкциями**.

Самая простая и часто используемая управляющая конструкция — выражение `if`. Ход выполнения программы с ее использованием представлен на рис. 3.7.

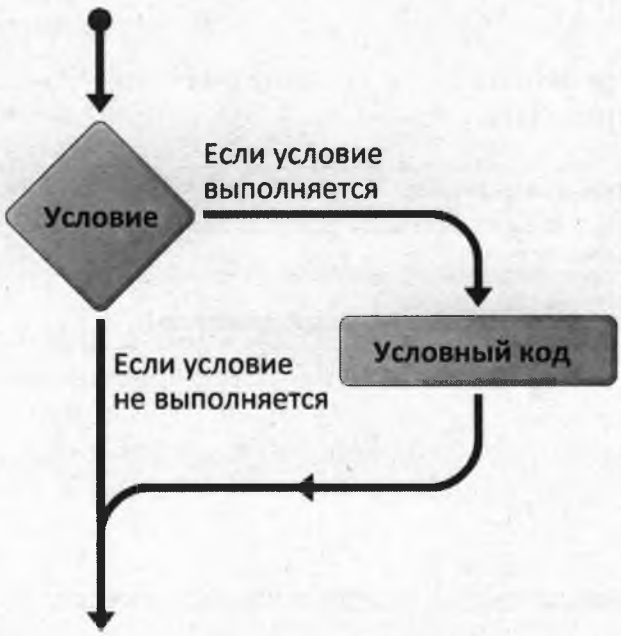


Рис. 3.7. Принцип работы выражения `if`[1]

[1] Эта и несколько аналогичных диаграмм, представленных в книге, изначально созданы Кэмероном Адамсом (Cameron Adams) для Simply JavaScript («Просто JavaScript»; Мельбурн: SitePoint, 2006), написанной совместно с автором этого издания, и используются с его согласия.

Вот как выражение `if` выглядит в коде на языке PHP.

```
if (условие)
{
  : код, который будет запущен, если условие выполняется
}
```

С помощью данной управляющей конструкции PHP выполнит набор команд, если сработает какое-то условие.

Выполним небольшой трюк на примере созданной ранее страницы с приветствием. Для начала откройте файл **name.html**, чтобы внести в него некоторые изменения. Упростите форму, которая в нем находится: теперь она будет отправлять скрипту `name.php` только одну переменную `name` (`chapter3/if/name.html`, фрагмент).

```
<form action="name.php" method="post">
  <div><label for="name">Имя:
    <input type="text" name="name" id="name"></label>
  </div>
  <div><input type="submit" value="Поехали"/></div>
</form>
```

Теперь отредактируйте файл **name.php**. Замените имеющийся PHP-код следующим фрагментом (`chapter3/if/name.php`, фрагмент).

```
$name = $_REQUEST['name'];
if ($name == 'Кевин')
{
  echo 'Добро пожаловать, о блистательный правитель!';
}
```

Теперь, если переменная `name` получит значение `'Кевин'`, появится специальное сообщение (рис. 3.8).

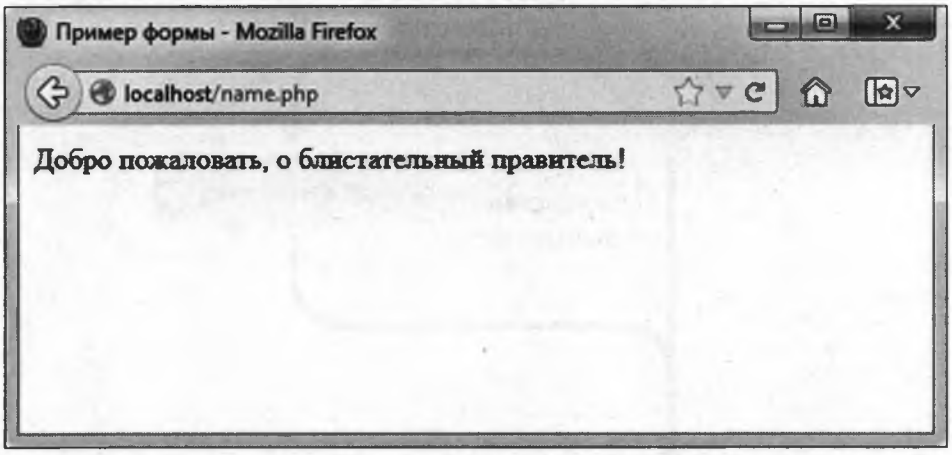


Рис. 3.8. Хорошо быть королем

Если вместо имени Кевин будет введено какое-то другое, приветствие не будет столь радушным: условный код внутри выражения `if` не выполнится и страница окажется пустой.

Чтобы не обделить теплым приветствием всех остальных, можно воспользоваться выражением `if – else`. Ход выполнения этой конструкции показан на рис. 3.9.

Рис. 3.9. Принцип работы выражения `if – else`

Блок `else` в выражении `if – else` следует за блоком `if` (`chapter3/ifelse1/name.php`, фрагмент):

```
$name = $_REQUEST['name'];
if ($name == 'Кевин')
{
  echo 'Добро пожаловать, о блистательный правитель!';
}
else
{
  echo 'Добро пожаловать на наш веб-сайт, ' .
      htmlspecialchars($name, ENT_QUOTES, 'UTF-8') . '!';
}
```

Теперь, отправив вместо имени Кевин любое другое имя, вы увидите обычное приветственное сообщение (рис. 3.10).

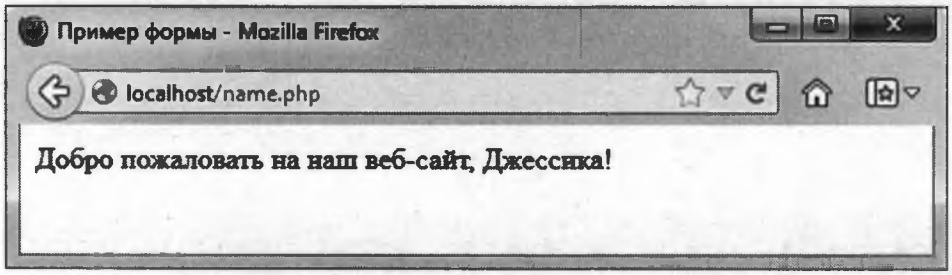


Рис. 3.10. Вы должны помнить о других людях

Использованная в вышеприведенном условии инструкция == называется **оператором равенства**. Этот оператор помогает сравнить два значения и определить, равны ли они друг другу.

ДВОЙНАЯ ПРОБЛЕМА

При использовании оператора равенства не забывайте набирать два знака «равно» (==). Начинающие программисты на языке PHP часто пишут подобное условие с использованием одного знака.

```
if ($name = 'Кевин') // Пропущен знак равенства!
```

В этом условии вместо оператора равенства указан оператор присваивания (=), с которым вы познакомились в разделе «Переменные, операторы и комментарии». Таким образом, вместо сравнения содержимого `$name` и строки `'Кевин'` произойдет присваивание значения `'Кевин'` переменной `$name`.

Еще хуже, если выражение `if` станет использовать оператор присваивания в качестве условия, которое подразумевается истинным; условный код внутри выражения `if` будет запускаться всегда, независимо от того, каким было исходное значение `$name`.

Условия могут быть куда более сложными, чем единственная проверка на равенство. Помните, ранее в примере с формой мы вводили имя и фамилию? Если теперь вы захотите вывести специальное сообщение для конкретного человека, придется проверить оба значения.

Отредактируйте файл **name.html** так, чтобы создаваемая форма снова содержала два поля (`chapter3/ifelse2/name.html`, фрагмент).

```
<form action="name.php" method="post">
  <div><label for="firstname">Имя:      &n
bsp; 
    <input type="text" name="firstname" id="firstname"></label>
  </div>
  <div><label for="lastname">Фамилия:
    <input type="text" name="lastname" id="lastname"></label>
  </div>
  <div><input type="submit" value="Поехали"></div>
</form>
```

Затем откройте файл **name.php** и приведите PHP-код к следующему виду (изменения выделены полужирным начертанием (`chapter3/ifelse2/name.php`, фрагмент)).

```
$firstName = $_REQUEST['firstname'];
$lastName = $_REQUEST['lastname'];
if ($firstName == 'Кевин' and $lastName == 'Янк')
{
  echo 'Добро пожаловать, о блистательный правитель!';
}
else
{
  echo 'Добро пожаловать на наш веб-сайт, ' .
      htmlspecialchars($firstName, ENT_QUOTES, 'UTF-8') . ' ' .
      htmlspecialchars($lastName, ENT_QUOTES, 'UTF-8') . '!';
}
```

Обновленное условие станет выполняться лишь в том случае, если переменные `$firstName` и `$lastName` примут значения `'Кевин'` и `'Янк'` соответственно. Оператор `AND` делает все условие истинным только тогда, когда истинны оба сравнения. При использовании похожего оператора `OR` условие будет выполняться, если истинно хотя бы одно из двух простых условий. Если для вас более привычны используемые в языках JavaScript или си операторы `&&` и `||` (для `AND` и `OR` соответственно), то в PHP они тоже работают.

На рис. 3.11 видно, что совпадения только одной части имени в этом примере не достаточно, чтобы соответствовать высокому званию правителя.

По мере необходимости мы рассмотрим более сложные условия, а пока вам нужно иметь лишь общее представление о выражении `if – else`.

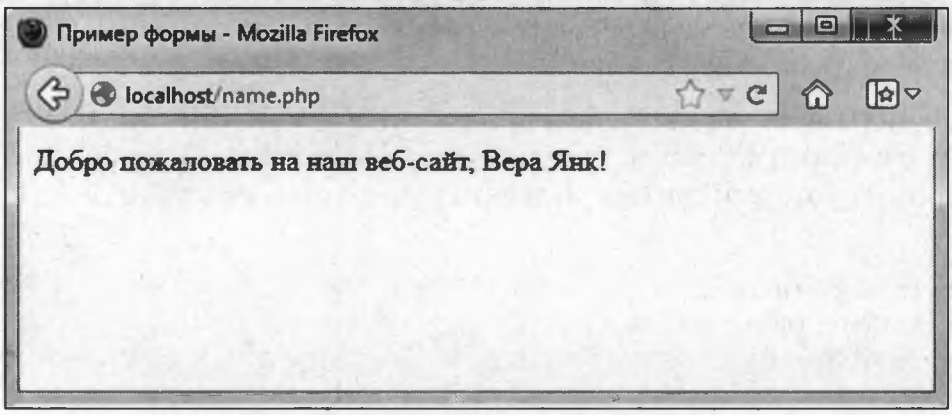


Рис. 3.11. Вы не похожи на правителя

Цикл `while` — еще одна управляющая структура, которая часто используется в PHP. Если в конструкции `if – else` проверка условия позволяет решать, нужно ли выполнять набор команд, то в цикле `while` условие используется для определения того, сколько раз один и тот же набор команд необходимо выполнить. На рис. 3.12 показано, как работает цикл while.

Рис. 3.12. Принцип работы цикла `while`

Вот как это выглядит в программном коде.

```
while (условие)
{
  : выражения (одно или несколько), которые выполняются, пока условие
истинно
}
```

Работа цикла `while` похожа на работу выражения `if`. Разница проявляется во время выполнения действий после проверки условия. В цикле `while` вместо перехода к выражению, которое идет за закрывающей фигурной скобкой (}), происходит повторная проверка условия. Если условие по-прежнему верно, то выражения выполняются во второй и в третий раз — до тех пор, пока условие возвращает значение `true`. Как только условие перестает быть истинным (будь то первая или сотая проверка), программа переходит к выражениям, которые следуют за циклом `while` после закрывающей скобки.

Такие циклы могут пригодиться при работе с длинными списками элементов (например, с нашими шутками, хранящимися в базе данных), но пока рассмотрим простой пример, который считает до десяти (`chapter3/count10.php`, фрагмент).

```
$count = 1;
while ($count <= 10)
{
  echo "$count ";
  ++$count;
}
```

Возможно, код показался вам немного запутанным, давайте разберем каждую строку.

```
$count = 1;
```

В первой строке создается переменная с именем `$count`, которой присваивается значение 1.

```
while ($count <= 10)
```

Вторая строка — начало цикла. По условию значение `$count` должно быть меньше или равно 10.

```
{
```

Открывающая фигурная скобка обозначает в цикле `while` начало блока с условным кодом, который иначе называют **телом цикла**. Он выполняется снова и снова, пока условие остается истинным.

```
echo "$count ";
```

Эта строка выводит значение переменной `$count`, за которым следует пробел. Чтобы облегчить восприятие кода, мы воспользовались преимуществом интерполяции переменных и заменили оператор конкатенации двойными кавычками (см. раздел «Переменные, операторы и комментарии»).

```
++$count;
```

В пятой строке к значению переменной `$count` добавляется единица. `++$count` — это сокращенный вариант записи выражения `$count=$count+1`. Обе конструкции работают одинаково.

```
}
```

Закрывающая фигурная скобка обозначает конец тела цикла `while`.

В результате запуска этого кода произойдет следующее. В ходе первой проверки значение `$count` окажется равным 1 — условие определенно выполняется, поэтому будет выведено содержимое переменной `$count` (то есть 1), а ей присвоено новое значение, равное 2. При второй проверке условие все еще сохранит истинность, аналогичным образом будет выведено старое значение 2 и присвоено новое, равное 3. Точно так же будут выведены значения 3, 4, 5, 6, 7, 8, 9 и 10. В конце концов, когда значение переменной `$count` станет равным 11, условие перестанет выполняться и цикл завершится.

Конечный результат выполнения кода показан на рис. 3.13.

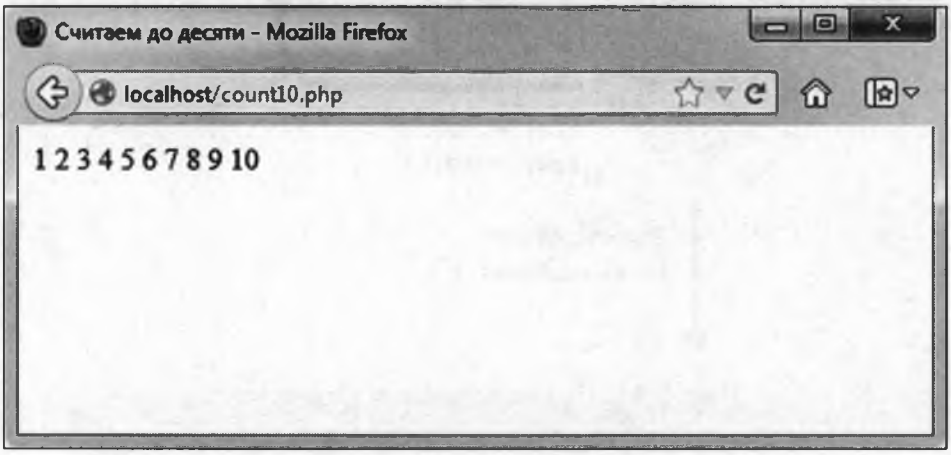


Рис. 3.13. PHP демонстрирует навыки счета

В этом примере в условии использовался новый оператор <= (меньше или равно). Существуют и другие числовые операторы подобного типа: >= (больше или равно), < (меньше), > (больше) и != (не равно). Последний также применяется для сравнения текстовых строк.

Еще одна конструкция, созданная специально для тех случаев, когда перебирается набор значений, пока выполняется какое-то условие (наподобие описанного в предыдущем примере), называется циклом `for`. Принцип работы данного цикла приведен на рис. 3.14.

В программном коде цикл `for` выглядит следующим образом.

```
for (объявляем счетчик; условие; увеличиваем счетчик)
{
  : выражения (одно или несколько), которые выполняются раз за разом,
пока условие истинно
}
```

Рис. 3.14. Принцип работы цикла `for`

Счетчик объявляется только один раз в начале цикла, в то время как проверка условия (перед запуском тела цикла) и увеличение счетчика (после завершения работы тела цикла) происходят при каждом повторении.

Вот как выглядит пример со счетом до десяти, реализованный посредством цикла `for` (`chapter3/count10-for.php`, фрагмент).

```
for ($count = 1; $count <= 10; ++$count)
{
  echo "$count";
}
```

Как видите, выражения, которые инициализируют переменную `$count` и увеличивают ее значение, находятся в первой строке цикла `for` вместе с условием. И хотя на первый взгляд код стал немного сложнее, когда вы привыкните к синтаксису, вам станет легче его воспринимать: теперь все конструкции, связанные с управлением циклом, собраны в одном месте. Выражение `for` используется во многих примерах этой книги, поэтому у вас будет достаточно возможностей попрактиковаться в его чтении.

Полируем до блеска

Итак, теперь вам знакомы основы синтаксиса языка программирования PHP. Вы можете взять любую статическую веб-страницу, изменить ее расширение на .php и добавить PHP-код для генерирования содержимого налету. Вполне неплохо для одного дня занятий!

Но прежде чем двигаться дальше, немного остановимся и критически взглянем на созданные ранее примеры. Поскольку наша цель — создание сайтов на профессиональном уровне, нам предстоит исправить несколько недочетов.

Знание методик, описанных в оставшейся части третьей главы, поможет придать вашей работе профессиональный вид, а вам выделиться из толпы программистов-любителей. В дальнейшем эти методики будут использоваться даже в самых простых примерах, чтобы вы оставались уверенными в качестве разрабатываемого продукта.

Не демонстрируйте свои технологические решения

Среди рассмотренных примеров встречались как обычные HTML-файлы, чьи имена заканчивались на .html, так и файлы, которые содержали коды HTML и PHP и имели расширение .php. Несмотря на то что вам как разработчику может пригодиться такое разграничение файлов, пользователям вовсе необязательно знать, какие страницы сайта были сгенерированы с помощью PHP.

Кроме того, хотя язык PHP — это мощное технологическое решение, которое подходит для создания практически любого сайта, основанного на базе данных, не исключено, что когда-нибудь вам захочется перейти на другую технологию. В таком случае все адреса URL для динамических страниц на вашем сайте перестанут работать, поскольку изменятся имена файлов в соответствии с выбранным вами языком. Неужели вам это нужно?

В наши дни разработчики уделяют много внимания адресам URL, которые видны пользователю. Они должны быть неизменными, насколько это возможно, и нет никакого смысла отягощать их своеобразной рекламой языка программирования, с помощью которого была создана та или иная страница.

Чтобы избавиться от расширений файлов в адресах, воспользуйтесь преимуществами, которые дают индексы директорий. Когда вместо конкретного скрипта в адресе URL указана директория, веб-сервер ищет внутри нее файлы с названиями index.html или index.php и выводит их в ответ на запрос.

Возьмите, например, страницу today.php из главы 1. Переименуйте ее в index.php. Затем вместо того, чтобы поместить файл в корневую директорию своего веб-сервера, создайте для него подкаталог с названием today. Теперь наберите в браузере адрес http://localhost/today/ (или что-то похожее на http://localhost:8888/today/, если вам пришлось указать номер порта для своего сервера). На рис. 3.15 приведен пример с новым URL. Полученный адрес не содержит ненужного расширения .php, он стал более коротким и запоминающимся, что в наши дни весьма желательно для URL.

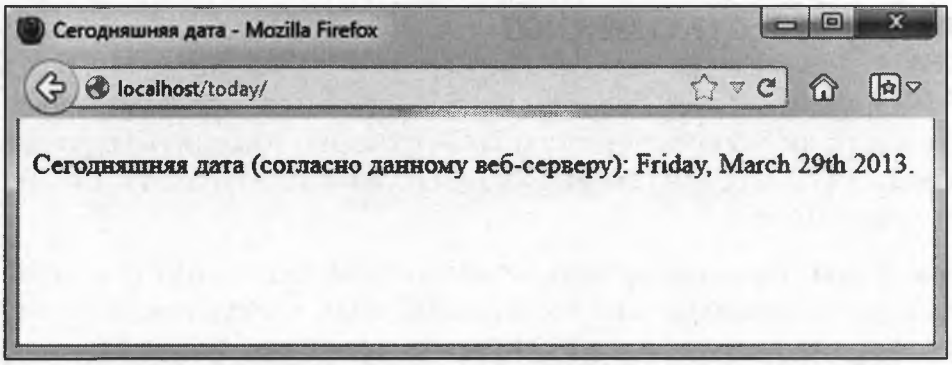


Рис. 3.15. Более стильный адрес URL

Используйте шаблоны

В тех простых примерах, с которыми мы до сих пор имели дело, добавление PHP-кода прямо на HTML-страницу было оправданно. Однако с увеличением объема кода, необходимого для генерирования страницы, поддерживать в надлежащем виде такую «смесь» HTML и PHP станет непросто. Представьте, что вы работаете совместно с веб-дизайнером, который не особо силен в технических вопросах. В этом случае большие блоки непонятного PHP-кода, смешанные с HTML, могут привести к катастрофе: дизайнер по неосторожности изменит программный код, что вызовет ошибки, которые он не в состоянии исправить самостоятельно.

Лучше вынести основную часть PHP-кода в отдельный файл, а для веб-страницы оставить преимущественно HTML. С этим поможет справиться выражение `include`, которое позволяет вставлять в код содержимое других файлов. Чтобы разобраться, как оно работает, перепишем рассмотренный ранее пример использования цикла `for` для счета до десяти.

Для начала создайте директорию **count10** и поместите туда новый файл **index.php**. Откройте его для редактирования и наберите следующий код (`chapter3/count10/index.php`).

```
<?php
$output = ''; ❶
for ($count = 1; $count <= 10; ++$count)
{
  $output .= "$count "; ❷
}

include 'count.html.php'; ❸
```

Не удивляйтесь, это на самом деле *все* содержимое данного файла. В нем нет ни строки HTML-кода. С циклом `for` вы уже знакомы, тем не менее обратите внимание на некоторые интересные участки этого скрипта.

1. Вместо того чтобы вывести числа от 1 до 10 с помощью команды `echo`, скрипт добавляет их в переменную с именем `$output`, которой в самом начале кода присваивается пустая строка.
2. В данной строке каждое число, за которым следует пробел, добавляется в конец переменной `$output`. Оператор `.=` сочетает в себе операторы при-

сваивания и конкатенации и служит для добавления значений в конец уже существующей строковой переменной. С его помощью записывается упрощенный вариант строки `$output = $output . "$count ";`.

3. Это команда `include`, которая сообщает PHP, что в данном месте нужно выполнить содержимое файла **count.html.php**[1].

   Должно быть, вы также заметили, что в конце файла отсутствует выражение `?>`, которое соответствует началу `<?php`. При желании вы можете его добавить, но в этом нет особой необходимости. Если скрипт заканчивается PHP-кодом, то об окончании кода специально сигнализировать не нужно — за вас это сделает конец файла. Именно так предпочитают поступать настоящие профессионалы.

Теперь создайте файл **count.html.php**, который подключается в последней строке скрипта (`chapter3/count10/count.html.php`).

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Counting to Ten</title>
  </head>
  <body>
    <p>
      <?php echo $output; ?>
    </p>
  </body>
</html>
```

Данный файл практически полностью состоит только из HTML-кода, за исключением всего лишь одной строки, которая предназначена для того, чтобы выводить значение переменной `$output`. Это и есть та самая переменная, которая создана в файле **index.php**.

Написанный выше код называется **шаблоном**. Статическая веб-страница содержит небольшой фрагмент PHP-кода, с помощью которого в нее вставляются динамически сгенерированные значения. Вместо того чтобы генерировать такие значения внутри сложного кода в самой странице, используется отдельный PHP-скрипт (**index.php**).

Рассмотренный подход позволяет переложить работу с шаблонами на плечи дизайнеров, которые умеют обращаться с HTML, не беспокоясь о том, что они что-то сделают с вашим PHP-кодом. А вы сможете сосредоточиться на написании скриптов, не отвлекаясь на код HTML-разметки.

Чтобы как-то выделить файлы с шаблонами, к их названиям можно добавлять расширение **.html.php**. Веб-сервер воспримет такие файлы как обычные PHP-скрипты, а расширение **.html.php** послужит вам полезным напоминаем о том, что они содержат как HTML-, так и PHP-код.

[1] Иногда при записи команды `include` имена файлов могут указываться в круглых скобках, наподобие записи функций `date` или `htmlspecialchars`. На наш взгляд, это только усложняет представление имени файла, поэтому в данной книге вы такой вариант не встретите. То же самое относится к еще одной популярной однострочной команде `echo`.

Шаблонов много, контроллер один

Преимущество использования команды `include` для загрузки файлов заключается в том, что в одном скрипте она может применятся *несколько раз*, выводя разные шаблоны.

PHP-скрипт, который отвечает на запросы браузера, выбирает для заполнения нужный шаблон и отправляет результат обратно, называют **контроллером**. Контроллер содержит логику, на основе которой решается, какой шаблон получит браузер.

Вернемся к еще одному примеру из этой главы — к форме с приветствием, которая предлагала посетителям ввести имя и фамилию. Начнем с шаблона для формы. Можно воспользоваться написанным ранее файлом **name.html**. Создайте директорию **welcome** и сохраните в нее копию **name.html**, переименованную в **form.html.php**. В полученной копии измените атрибут `action` в теге `form` (`chapter3/welcome/form.html.php`):

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Пример формы</title>
  </head>
  <body>
    <form action="" method="post">
      <div><label for="firstname">Имя:      
;  
        <input type="text" name="firstname" id="firstname"></label>
      </div>
      <div><label for="lastname">Фамилия:
        <input type="text" name="lastname" id="lastname"></label>
      </div>
      <div><input type="submit" value="Поехали"></div>
    </form>
  </body>
</html>
```

Пустой атрибут `action` означает, что браузер должен отправить форму обратно по тому же адресу, с которого он ее получил. В нашем случае это URL контроллера, подключающего данный шаблон.

Давайте рассмотрим котроллер для текущего примера. Создайте в директории **welcome**, где находится шаблон формы, скрипт **index.php**. Наберите в нем следующий код (`chapter3/welcome/index.php`).

```
<?php
if (!isset($_REQUEST['firstname'])) ❶
{
  include 'form.html.php'; ❷
}
else ❸
{
  $firstName = $_REQUEST['firstname'];
```

```
    $lastName = $_REQUEST['lastname'];
    if ($firstName == 'Kevin' and $lastName == 'Yank')
    {
      $output = 'Добро пожаловать, о блистательный правитель!'; ❹
    }
    else
    {
      $output = 'Добро пожаловать на наш веб-сайт, ' .
      htmlspecialchars($firstName, ENT_QUOTES, 'UTF-8') . ' ' .
      htmlspecialchars($lastName, ENT_QUOTES, 'UTF-8') . '!';
    }

    include 'welcome.html.php'; ❺
}
```

Приведенный выше код похож на написанный ранее скрипт **name.php**, но есть и некоторые различия.

1. Первым делом контроллер должен определить, является ли текущий запрос отправкой формы из файла **form.html.php**. Для этого нужно проверить наличие в запросе переменной `firstname`. Если она присутствует, то PHP должен был поместить ее значение в `$_REQUEST['firstname']`.

   Встроенная в PHP функция `isset` определяет, имеет ли значение конкретная переменная (или элемент массива). Если `$_REQUEST['firstname']` было присвоено значение, то выражение `isset($_REQUEST['firstname'])` вернет `true`, если нет — `false`.

   Чтобы облегчить восприятие скрипта, который содержит контроллер, вначале лучше указывать код, отвечающий за отправку формы. Выражение `if` должно проверять, задано ли значение для `$_REQUEST['firstname']`. Для этого воспользуемся оператором отрицания (`!`). Находясь перед именем функции, он изменит ее значение на противоположное: с `true` на `false` или с `false` на `true`. Таким образом, если запрос не содержит переменной `firstname`, то `!isset($_REQUEST['firstname'])` вернет `true` и будет выполнено тело выражения `if`.

2. Если форма не была отправлена вместе с запросом, то контроллер подключит ее с помощью файла **form.html.php**.

3. Если форма была отправлена, то вместо этого выполнится тело выражения `else`.

   Этот код извлекает переменные `firstname` и `lastname` из массива `$_REQUEST`, а затем генерирует соответствующее приветственное сообщение на основе введенного имени.

4. Вместо того чтобы вывести приветствие посредством команды `echo`, контроллер поместит его в переменную с именем `$output`.

5. Сгенерировав соответствующее сообщение, контроллер подключит шаблон **welcome.html.php**, который и выведет приветствие на страницу.

Осталось написать шаблон **welcome.html.php** (`chapter3/welcome/welcome.html.php`).

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Пример формы</title>
  </head>
  <body>
    <p>
      <?php echo $output; ?>
    </p>
  </body>
</html>
```

Теперь запустите любой браузер и укажите в нем адрес http://localhost/welcome/ (или http://localhost:8888/welcome/, если ваш веб-сервер не использует 80-й порт). Вам будет предложено ввести свое имя. После того как вы отправите форму, на экране появится соответствующее приветственное сообщение. Адрес URL при этом меняться не должен.

Одно из преимуществ сохранения постоянного адреса при выводе формы и отображении приветствия заключается в том, что пользователь может добавить страницу в закладки на любом этапе работы с ней. Какая бы страница ни была добавлена (с полями ввода или приветствием), при возвращении на нее пользователю будет еще раз предложено заполнить форму. В предыдущей версии этого примера, когда страница с приветствием имела собственный URL, возвращение по этому адресу без отправки формы приводило к генерированию некорректного сообщения `Добро пожаловать на наш веб-сайт` или к ошибке PHP (в зависимости от конфигурации вашего сервера).

ОТКУДА ТАКАЯ ЗАБЫВЧИВОСТЬ?

В главе 9 вы узнаете, как запоминать имена пользователей между визитами.

Переходим к базе данных

В этой главе вы увидели в действии серверный скриптовый язык PHP и изучили его основные выражения, переменные, операторы, комментарии и управляющие конструкции. Несмотря на то что приведенные примеры были довольно просты, нам удалось рассмотреть использование привлекательных адресов URL и отделение HTML-шаблонов от PHP-кода, управляющего генерированием страницы.

Вы уже, пожалуй, догадались, что настоящая мощь языка PHP кроется в сотнях (и даже тысячах) встроенных функций, которые позволяют получать доступ к данным в MySQL, отправлять электронные письма, динамически генерировать изображения и даже создавать налету файлы в формате Adobe Acrobat PDF.

В главе 4 вы изучите функции для работы с MySQL, встроенные в PHP, и узнаете, как опубликовать в Интернете базу данных с шутками, созданную в главе 2. Мы подготовим почву для достижения главной цели, которую преследует эта книга, — создание с помощью PHP и MySQL полноценной системы управления содержимым сайта.

Глава 4
ПУБЛИКАЦИЯ ДАННЫХ ИЗ MySQL В ИНТЕРНЕТЕ

Эта глава научит вас извлекать информацию, хранящуюся в базах данных MySQL, и выводить ее в виде веб-страницы. На данном этапе вы уже владеете основными навыками работы с системами реляционных баз данных (MySQL) и серверными скриптовыми языками (PHP). Теперь вы готовы к тому, чтобы научиться создавать с их помощью настоящие сайты с поддержкой баз данных.

Общие сведения

Прежде чем продолжить, стоит вернуться немного назад и кое-что прояснить. В вашем распоряжении есть два мощных инструмента — скриптовый язык PHP и движок реляционных баз данных MySQL. Важно понять, как они сочетаются друг с другом.

Содержимое сайта, основанного на базе данных, хранится в этой самой базе, откуда затем динамически извлекается для создания веб-страниц, отображаемых в обычном браузере. Итак, с одной стороны есть пользователь, который зашел на ваш сайт и запросил страницу с помощью браузера. Браузер при этом ожидает получить в ответ стандартный документ в формате HTML. С другой стороны есть содержимое вашего сайта, которое хранится в одной или нескольких таблицах внутри базы данных MySQL, умеющей отвечать только на SQL-запросы (команды). PHP — это посредник, который умеет разговаривать на обоих языках. Он обрабатывает запрос страницы и извлекает информацию из базы данных MySQL, используя SQL-запросы, похожие на те, с помощью которых вы создавали таблицу с шутками в главе 2. Полученная PHP информация выдается в виде хорошо отформатированной HTML-страницы, которая и была необходима браузеру (рис. 4.1).

Таким образом, когда пользователь заходит на страницу сайта, основанного на базе данных, происходит следующее.

1. Браузер пользователя запрашивает веб-страницу у вашего сервера.
2. Программное обеспечение веб-сервера (обычно это Apache) распознает запрашиваемый PHP-скрипт и запускает интерпретатор для выполнения кода, содержащегося в соответствующем файле.
3. С помощью определенных команд языка PHP (о них вы узнаете в этой главе) происходит подключение к базе данных MySQL и выполняется запрос содержимого, которое должно находиться на странице.

4. База данных MySQL посылает в ответ PHP-скрипту запрашиваемое содержимое.
5. PHP-скрипт сохраняет содержимое в одной или нескольких переменных, а затем выводит его как часть веб-страницы с помощью выражений `echo`.
6. Интерпретатор PHP завершает свою работу, отправляя созданную им копию HTML-документа веб-серверу.
7. Веб-сервер отправляет браузеру документ, словно обычный HTML-файл. Причем браузер даже не догадывается, что документ выдан интерпретатором PHP, а не взят из файла. Ему кажется, что он запрашивает и получает обычную веб-страницу.

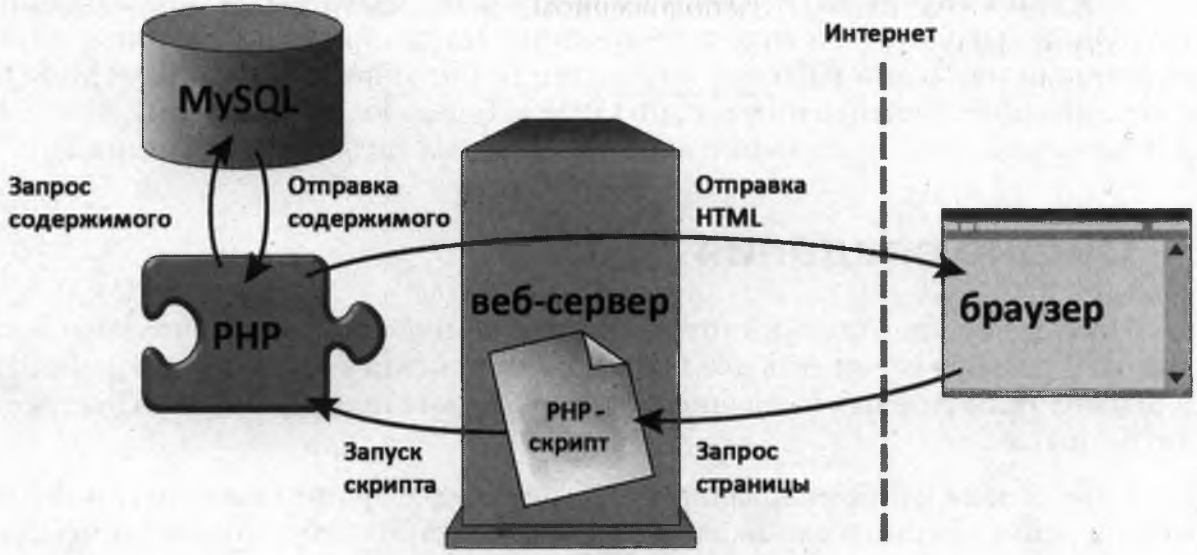


Рис. 4.1. PHP извлекает данные из MySQL для создания веб-страниц

Создание учетной записи пользователя в MySQL

Чтобы PHP подключился к серверу баз данных MySQL, ему необходимо указать имя пользователя и пароль. Сейчас в вашей базе содержится всего несколько забавных шуток, но там вполне может оказаться и конфиденциальная информация, например адреса электронной почты или другие подробности о посетителях вашего сайта. Сервер MySQL поддерживает высокий уровень безопасности и позволяет задать допустимые подключения и возможные действия.

В главе 1 вы установили пароль для учетной записи администратора MySQL-сервера (`root`). Вы *могли бы* использовать данные логин и пароль и в PHP-скриптах для подключения к MySQL, однако не стоит этого делать. `root` — это всесильная административная учетная запись. Если пароль от нее попадет не в те руки, ваша база подвергнется серьезной угрозе. В большинстве случаев на помощь придут системы безопасности, работающие на других уровнях (например, брандмауэр, который не даст подключиться к вашей базе данных из внешней сети), но лучше перестраховаться, чем потом жалеть.

Самое лучшее решение в таком случае — создать новую пользовательскую учетную запись, обладающую только теми привилегиями, которые необходимы для работы с базой данных `ijdb`, лежащей в основе вашего сайта. Создадим ее прямо сейчас.

1. Откройте phpMyAdmin одним из способов:

   1) в Windows запустите панель управления XAMPP и нажмите кнопку **Admin** рядом с пунктом **MySql**;

   2) в Mac OS X запустите MAMP, нажмите кнопку **Open start page** (если домашняя страница не открылась автоматически), а затем в верхней части домашней страницы перейдите на вкладку **phpMyAdmin**.

2. В списке баз данных, расположенном в левой части экрана, выберите `ijdb` (рис. 4.2).

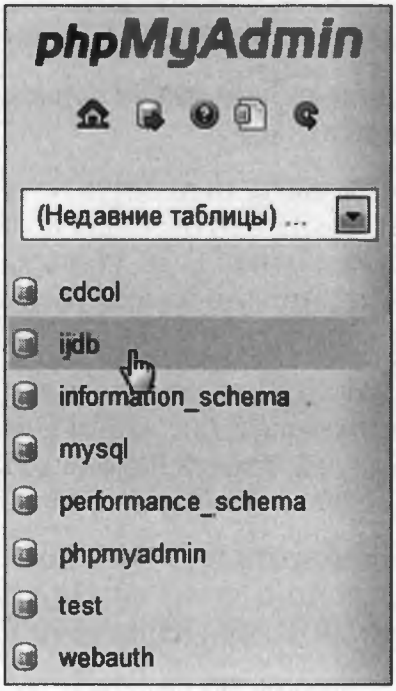


Рис. 4.2. Выбор базы данных `ijdb`

3. Отобразится список таблиц, которые находятся в этой базе данных (сейчас там только одна таблица `joke`). В меню верхней части окна перейдите на вкладку **Привилегии** (рис. 4.3).

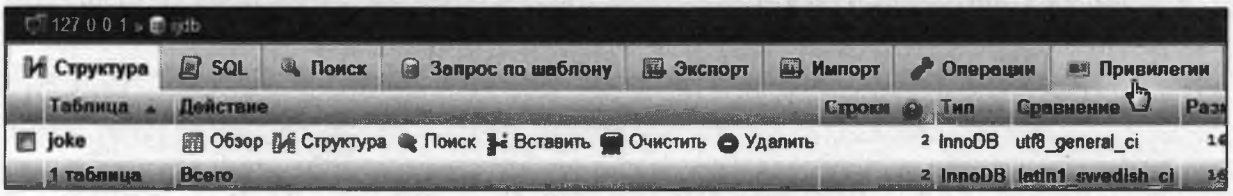


Рис. 4.3. Переход на вкладку **Привилегии**

4. Появится список пользователей с правами доступа к базе данных `ijdb` (рис. 4.4).

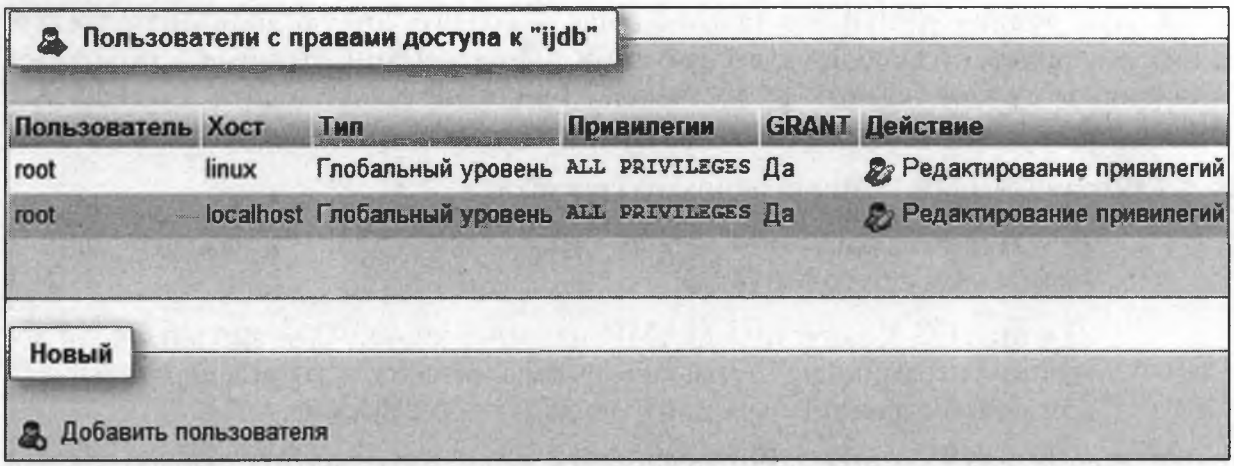


Рис. 4.4. Список тех, кто имеет доступ к базе данных `ijdb`

На этом этапе правами доступа обладает только пользователь root[1].

5. Перейдите по ссылке **Добавить пользователя** внизу списка и введите следующую информацию об учетной записи.

   Имя пользователя. Для ввода используйте текстовое поле. Если хотите, назовите пользователя `ijdb`. Это обычная практика, когда учетная запись, имеющая доступ только к одной базе данных, носит имя самой базы. Мы выбрали название `ijdbuser`, чтобы имя базы данных (`ijdb`) отличалось от имени пользователя, имеющего к ней доступ.

   Хост. Поскольку ваш MySQL-сервер работает на том же компьютере, что и веб-сервер, вы можете сделать так, чтобы учетная запись принимала подключения только локального хоста (localhost). Для приема подключений от других компьютеров, оставьте выбор по умолчанию — **Любой хост**[2].

   Пароль. Для ввода используйте текстовое поле. В данной книге выбран пароль mypassword. Вам, вероятно, стоит подобрать свой уникальный вариант. Запомните его, он пригодится при написании PHP-скриптов.

6. В разделе **База данных для пользователя** выберите пункт **Выставить полные привилегии на базу данных «ijdb»**. Это предоставит учетной записи *полную свободу действий* во всем, что касается базы данных `ijdb` (но только в отношении ее).

7. В разделе **Глобальные привилегии** оставьте неотмеченными все пункты. Настройки раздела разрешают пользователю выполнять определенный вид запросов с *любой* базой данных. В нашем случае необходимо ограничить учетную запись единственной базой.

8. Нажмите внизу формы кнопку **Добавить пользователя**.

[1] Пользователь root упоминается дважды, потому что сервер MySQL поставляется с двумя учетными записями администратора. Одна принимает подключения по IP-адресу 127.0.0.1, а другая от имени localhost. Как правило, оба идентификатора указывают на локальный компьютер, но в зависимости от способа подключения сервер воспринимает соединение по-разному. Более подробно об этой проблеме – в главе 10.

[2] На данном этапе лучше использовать локальную учетную запись. Пункт **Любой хост** может привести к проблемам, о которых вы узнаете в главе 10.

9. Программа phpMyAdmin подтвердит добавление нового пользователя и выведет SQL-запрос, отправленный серверу баз данных (рис. 4.5). Значение этих команд (если они вам понадобятся) можно найти в руководстве по MySQL.

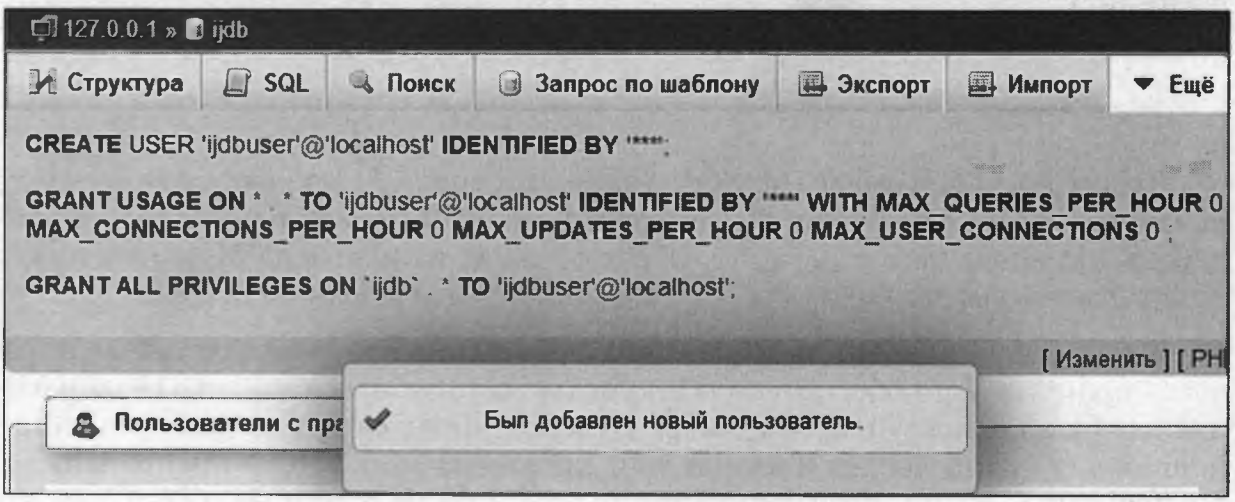


Рис. 4.5. Окно phpMyAdmin подтвердит, что вы добавили нового пользователя

Подключение к MySQL с помощью PHP

Прежде чем вы научитесь извлекать содержимое из базы данных и добавлять его на веб-страницы, нужно рассмотреть, как осуществляется соединение с MySQL с помощью PHP-скрипта. До этого момента вы использовали веб-приложение phpMyAdmin, тем не менее написанные вами PHP-скрипты могут подключаться к MySQL-серверу напрямую. Поддержка таких подключений обеспечивается встроенным расширением PHP Data Objects (PDO).

Вот как записывается команда соединения с сервером MySQL с помощью PDO.

```
new PDO('mysql:host=имя_сервера;dbname=база_данных', 'имя_пользователя',
'пароль')
```

На данном этапе воспринимайте `new PDO` как встроенную функцию, аналогичную, например, функции `date`, которая использована в главе 3. Если вы усомнились в том, что в имени функции могут стоять *пробелы*, значит, вы довольно сообразительны. Вскоре мы разберемся, в чем тут дело, а пока остановимся на аргументах, которые принимает данная функция. В их число входят:

1) строка с описанием типа базы данных (`mysql:`), именем сервера (`host=имя_сервера`) и названием базы данных (`dbname=база_данных`);

2) имя учетной записи MySQL, которую вы хотите использовать в PHP;

3) пароль для учетной записи.

Как вы, наверное, помните, функции, вызванные в PHP, обычно возвращают значение. Так называемая функция `new PDO` возвращает **объект PDO**, который

определяет установленное соединение. Поскольку мы планируем использовать соединение в дальнейшем, этот объект следует поместить в переменную. Если теперь подставить в вызов `new PDO` значения, необходимые для подключения к нашей базе данных, то он примет следующий вид (`chapter4/connect/index.php`, фрагмент):

```
$pdo = new PDO('mysql:host=localhost;dbname=ijdb', 'ijdbuser',
'mypassword');
```

Скорее всего, для вашего MySQL-сервера эта запись будет выглядеть несколько иначе хотя бы потому, что вы, наверняка, применяли для пользователя ijdbuser пароль, отличный от `mypassword`. Обратите также внимание на то, что значение, возвращаемое выражением `new PDO`, сохраняется в переменной `$pdo`.

Важно понимать, что MySQL — это совершенно отдельная от веб-сервера часть программного обеспечения. Вполне может оказаться так, что сервер баз данных будет недоступен, например, из-за поломки в сети, неверно предоставленной вами пары логина и пароля или даже из-за того, что вы просто забыли его запустить. В этом случае выражение `new PDO` не сработает и сгенерирует исключение.

Исключение возникает в том случае, когда вы заставляете PHP выполнить задачу, которую выполнить нельзя. Конечно, он попытается сделать то, что ему велено, но у него ничего не получится. Чтобы сообщить о своей неудаче, PHP сгенерирует (выбросит) для вас исключение. Как ответственный разработчик, вы обязаны это исключение перехватить и что-нибудь с ним сделать.

ЛОВИТЕ!

Если вы не перехватите исключение, то PHP прекратит выполнение скрипта и выведет внушительное сообщение об ошибке, в котором, кроме всего прочего, разместит и код этого самого скрипта. Как вы помните, код содержит логин и пароль для MySQL, поэтому очень важно, чтобы сообщение об ошибке не попалось на глаза пользователям!

Чтобы перехватить исключение, поместите код, который может привести к его выбросу, внутрь **выражения** `try — catch`:

```
try
{
  ⋮ делаем что-то опасное
}
catch (ExceptionType $e)
{
  ⋮ обрабатываем исключение
}
```

Выражение `try — catch` похоже на выражение `if — else`. Разница в том, что второй блок кода в нем запускается лишь тогда, когда не смог выполниться первый.

Запутались? Тогда подытожим вышесказанное и попробуем представить общую картину.

```
try
{
  $pdo = new PDO('mysql:host=localhost;dbname=ijdb', 'idjbuser',
    'mypassword');
}
catch (PDOException $e)
{
  $output = 'Невозможно подключиться к серверу баз данных.';
  include 'output.html.php';
  exit();
}3
```

Итак, код представляет собой выражение `try - catch`. В верхнем блоке `try` происходит подключение к базе данных с помощью команды `new PDO`. В случае успеха полученный объект `PDO` будет помещен в переменную `$pdo` — это позволит работать с новым подключением. Если попытка закончилась неудачей, PHP сгенерирует объект `PDOException` — разновидность исключений, которую выбрасывает `new PDO`. Блок `catch` перехватит объект `PDOException` и поместит его в переменную `$e`. Внутри блока переменной `$output` будет присвоено сообщение, информирующее пользователя о том, что произошло. В следующей строке кода подключается шаблон **output.html.php** — стандартная страница, на которой отображается значение `$output` (`chapter4/connect/output.html.php`).

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Вывод скрипта</title>
  </head>
  <body>
    <p>
      <?php echo $output; ?>
    </p>
  </body>
</html>
```

После того как сообщение появится на экране, последнее выражение в блоке `catch` вызовет встроенную функцию `exit`. Это первый пример функции, которую можно запустить без параметров. Такой вызов `exit` приведет к тому, что PHP-скрипт тут же закончит свою работу, и в случае неудачного подключения оставшаяся часть кода в контроллере (которая, скорее всего, зависит от успешного соединения с базой данных) выполнена не будет.

Надеемся, что теперь приведенный выше код стал более понятен. Если вы в чем-то еще не разобрались, попробуйте вернуться к началу раздела и перечитать его заново. Осталось выяснить, что же такое `new PDO` и какой объект оно возвращает.

Ускоренный курс объектно-ориентированного программирования

Аббревиатура PDO расшифровывается как объекты данных PHP (PHP Data Objects). Выражение `new PDO` как раз и возвращает такой объект. Рассмотрим, что он собой представляет.

Возможно, в процессе самостоятельного изучения PHP или других языков программирования вы уже встречали термин **объектно-ориентированное**

программирование (ООП). Этот стиль разработки, который великолепно подходит для создания по-настоящему сложных программ с множеством динамических компонентов. Большинство популярных языков программирования поддерживают ООП, а некоторые даже обязывают его использовать. PHP позволяет разработчику самому решать — придерживаться данной концепции при написании скриптов или нет.

До сих пор вы создавали PHP-код, работая в стиле **процедурного программирования**, и будете продолжать это делать. Данный стиль хорошо подходит для относительно простых проектов, которые рассматриваются в этой книге. Не менее успешно он используется и для написания более сложных приложений, например WordPress (http://wordpress.org/).

Расширение PDO, которое позволяет подключаться к MySQL и работать с базами данных, спроектировано в объектно-ориентированном стиле. Это значит, что вместо двух вызовов функции — для связи с сервером и использования установленного соединения — необходимо создать объект PDO, который будет предоставлять подключение к базе данных, а затем использовать возможности этого объекта для взаимодействия с базой.

Создание объекта во многом схоже с вызовом функции (`chapter4/connect/index.php`, фрагмент).

```
$pdo = new PDO('mysql:host=localhost;dbname=ijdb', 'ijdbuser',
'mypassword');
```

Ключевое слово `new` говорит интерпретатору о том, что вы хотите создать новый объект. Поставив пробел и указав **имя класса**, вы сообщаете PHP о том, какого типа объект вам нужен. В стандартной комплектации PHP имеется не только множество встроенных функций, но и специальная библиотека классов, которая позволяет создавать объекты. Таким образом, выражение `new PDO` указывает интерпретатору на то, что следует создать новый объект встроенного класса PDO.

Подобно строкам, числам и массивам объекты в PHP представляют собой некоторые значения. Следовательно, вы точно так же можете сохранять их в переменной или передавать в функцию в качестве аргумента. Кроме того, объекты обладают несколькими дополнительными полезными возможностями.

Прежде всего, объект может выступать в роли контейнера для других значений. Этим он во многом похож на массив. Из главы 3 вы уже знаете, что доступ к значению внутри массива получают через его индекс (например, `birthdays['Кевин']`). Тот же принцип работает и с объектами, только называется и выглядит это немного по-другому. В случае объектов доступ к значению по индексу называется доступом к **свойству**, а желаемое имя свойства задается не в квадратных скобках, а с помощью **стрелки**. Например, `$myObject->someProperty`.

```
$myObject = new SomeClass(); // создаем объект
$myObject->someProperty = 123; // задаем свойству значение
echo $myObject->someProperty; // получаем значение свойства
```

Как правило, массив содержит список *похожих* значений (например, даты рождения), объекты, в свою очередь, предназначены для хранения *связанных между собой* сущностей (например, свойств подключения к базе данных). Однако если бы

объекты отличались только этим, в них не было бы особого смысла: для этих же целей можно использовать массивы. Безусловно, объекты способны на большее.

Помимо свойств и их значений, объекты способны хранить внутри функции — это предоставляет гораздо больше возможностей. Функция, размещенная внутри объекта, называется **методом**. Чтобы вызвать метод, применяют все ту же стрелку. Например, `$myObject->someMethod()`.

```
$myObject = new SomeClass(); // создаем объект
$myObject->someMethod(); // вызываем метод
```

Аналогично отдельным функциям методы принимают аргументы и возвращают значения.

Скорее всего, на данном этапе вам все кажется немного запутанным и бессмысленным. Тем не менее в определенных ситуациях связывание воедино наборов переменных (свойств) и функций (методов) в небольшие пакеты, именуемые объектами, значительно повышает аккуратность кода и делает его более простым для восприятия. Работа с базами данных — один из таких примеров.

Возможно, когда-нибудь вы захотите написать новые классы, чтобы затем создать из них объекты собственной разработки, но пока продолжим работу с классами в составе PHP. Вернемся к объекту PDO и посмотрим, что можно сделать с помощью одного из его методов.

Настройка подключения

Вы уже видели, как создается объект PDO для подключения к серверу MySQL и как выводится сообщение об ошибке, если возникают проблемы с установкой соединения.

```
try
{
  $pdo = new PDO('mysql:host=localhost;dbname=ijdb', 'idjbuser',
    'mypassword');
}
catch (PDOException $e)
{
  $output = 'Невозможно подключиться к серверу баз данных.';
  include 'output.html.php';
  exit();
}
```

Если предположить, что подключение прошло успешно, то перед дальнейшим использованием его необходимо настроить. Для этого следует вызвать некоторые методы нового объекта `PDO`.

Вначале нужно сконфигурировать поведение объекта PDO при обработке ошибок. Рассмотренная выше конструкция `try — catch` позволяет справиться с любыми проблемами, которые возникают во время подключения к базе данных. Однако, после того как соединение успешно установлено, `PDO` по умолчанию переключается в «тихий» режим, поэтому выявить проблему и оперативно на нее среагировать становится сложнее. Чтобы экземпляр PDO генерировал исключение `PDOException`

каждый раз, когда он не в состоянии выполнить запрос, необходимо вызвать из объекта PDO метод `setAttribute` (`chapter4/connect/index.php`, фрагмент).

```
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Два значения, передаваемые в качестве аргументов, — это константы, такие же как `ENT_QUOTES`, которую вы научились передавать в функцию `htmlspecialchars` в главе 3. Пусть вас не смущает запись `PDO::` в начале имени каждой из них: она указывает на то, что константы не встроены в язык PHP и составляют часть используемого класса PDO. По сути, эта строка говорит о том, что вы хотите задать атрибуту, управляющему выводом ошибок (`PDO::ATTR_ERRMODE`), режим выброса исключений (`PDO::ERRMODE_EXCEPTION`)[1].

Далее необходимо настроить кодировку символов для подключения к базе данных. В главе 3 упоминалось о том, что для сайтов лучше использовать разновидность UTF-8 — это максимально расширит диапазон символов, вводимых пользователями при заполнении форм. По умолчанию при подключении к MySQL PHP-скрипт применяет более простую кодировку ISO-8859-1 (или Latin-1). Следовательно, теперь нужно сделать так, чтобы новый объект PDO использовал UTF-8.

Существует несколько способов, позволяющих задать необходимую кодировку для подключения к MySQL. Наиболее надежный вариант — отправить SQL-запрос `SET NAMES "utf8"` с помощью метода `exec`. Этот метод принадлежит объекту PDO, хранящемуся в переменной `$pdo`, и позволяет осуществлять запросы к базе данных (`chapter4/connect/index.php`, фрагмент).

```
$pdo->exec('SET NAMES "utf8"');
```

Если по какой-то причине метод `exec` *не сможет* выполнить запрос (например, MySQL-сервер упал сразу после того, как вы к нему подключились), вы должны быть готовы к тому, чтобы перехватить сгенерированное исключение `PDOException`. Проще всего это сделать, разместив соответствующее выражение внутри блока `try`, где создается объект PDO. Вот как выглядит итоговый код, выполняющий подключение к MySQL и настройку установленного соединения (`chapter4/connect/index.php`, фрагмент).

```
<?php
try
{
  $pdo = new PDO('mysql:host=localhost;dbname=ijdb', 'ijdbuser',
    'mypassword');
  $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $pdo->exec('SET NAMES "utf8"');
}
catch (PDOException $e)
{
  $output = 'Невозможно подключиться к серверу баз данных.';
  include 'output.html.php';
  exit();
}
```

[1] Более подробную информацию о режимах обработки ошибок в объекте PDO вы найдете в руководстве по PHP(http://php.net/manual/ru/pdo.error-handling.php).

Чтобы усовершенствовать пример, отобразим сообщение, информирующее о том, что все прошло успешно. Таким образом, окончательная версия контроллера будет иметь следующий вид (`chapter4/connect/index.php`).

```
<?php
try
{
  $pdo = new PDO('mysql:host=localhost;dbname=ijdb', 'ijdbuser',
    'mypassword');
  $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $pdo->exec('SET NAMES "utf8"');
}
catch (PDOException $e)
{
  $output = 'Невозможно подключиться к серверу баз данных.';
  include 'output.html.php';
  exit();
}
$output = 'Соединение с базой данных установлено.';
include 'output.html.php';
```

Откройте данный пример в браузере. Если вы поместите файлы **index.php** и **output.html.php** в директорию **connect** своего веб-сервера, то адрес URL будет выглядеть как **http://localhost/connect/**. При условии, что сервер запущен и все остальное работает как положено, вы увидите сообщение об успешном подключении (рис. 4.6).

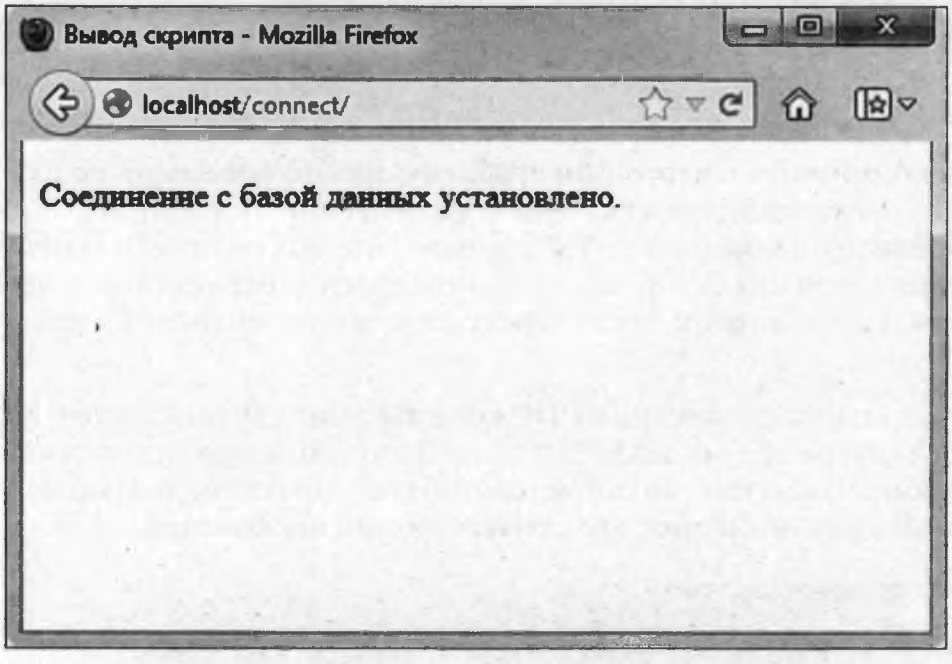


Рис. 4.6. Сообщение об успешном подключении к базе данных

Если PHP не в состоянии подключиться к MySQL-серверу или вы предоставили неверные логин и пароль, на экране появится другая надпись, похожая на ту, что приведена на рис. 4.7. Чтобы убедиться в том, что обработка ошибок функционирует должным образом, укажите для проверки некорректный пароль.

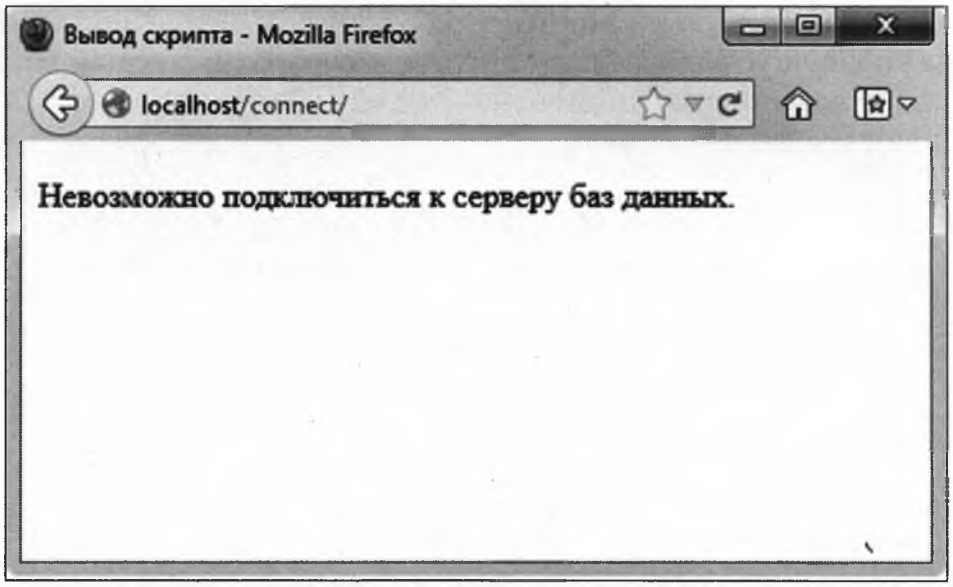


Рис. 4.7. Сообщение о невозможности подключения к серверу баз данных

Для посетителей сайта такого сообщения об ошибке вполне достаточно. Но что если вы — разработчик? Где искать решение проблемы? Прежде всего необходимо выяснить, что именно пошло не так. Для этого стоит немного изменить блок `catch`, который выводит сообщение (`chapter4/connect/index.php`, фрагмент).

```
catch (PDOException $e)
{
  $output = 'Невозможно подключиться к серверу баз данных.';
  include 'output.html.php';
  exit();
}
```

Чтобы обнаружить и исправить проблему, необходимо знать подробности об ошибке. Их предоставит перехваченное исключение. Посмотрите внимательно на первую строку блока `catch`, вы заметите, что мы не просто сообщаем PHP о перехвате исключения `PDOException`, но и просим сохранить его в переменную с именем `$e`. Таким образом, после перехвата в этой переменной будет находиться еще один объект.

Фактически *все* исключения в PHP представлены в виде объектов. Как и PDO, хранящийся внутри `$pdo` объект `PDOException` обладает собственными свойствами и методами. Чтобы выяснить причину, по которой возникла проблема, следует извлечь сообщение об ошибке, хранящееся внутри исключения.

```
catch (PDOException $e)
{
  $output = 'Невозможно подключиться к серверу баз данных: ' .
$e->getMessage();
  include 'output.html.php';
  exit();
}
```

Метод `getMessage` вызывается из объекта, хранящегося в переменной `$e`, и вставляет возвращенное им значение в конец сообщения об ошибке, используя

оператор конкатенации (.). Теперь, если вы укажете в PHP-коде неверный пароль, отобразится сообщение, представленное на рис. 4.8.

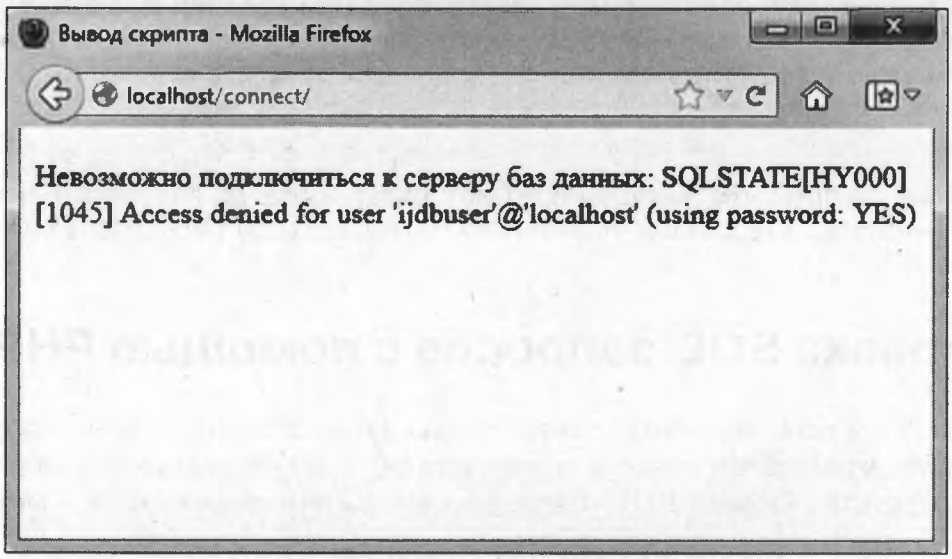


Рис. 4.8. Подробное сообщение об ошибке, которая возникает при вводе неверного пароля

Если ошибка возникнет во время подключения при задании кодировки, то на экране появится другое сообщение (рис. 4.9).

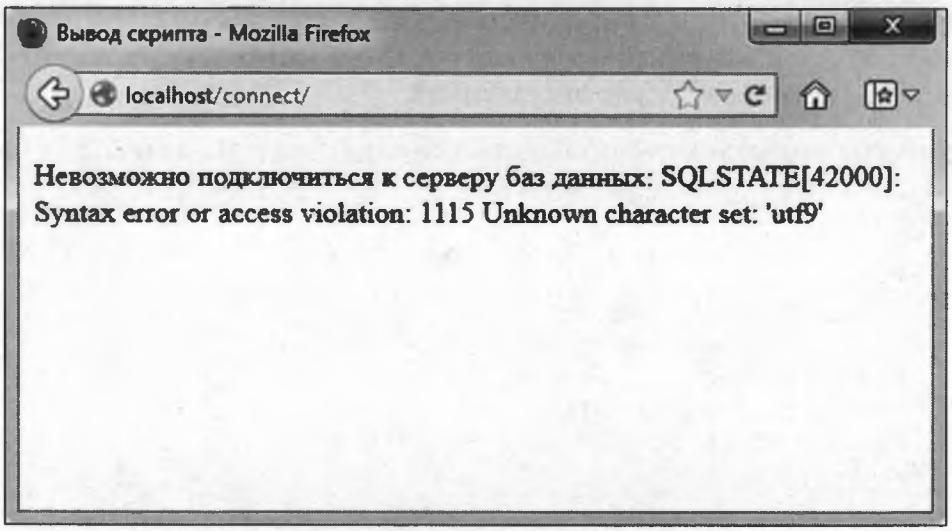


Рис. 4.9. Сообщение об ошибке, информирующее о неверно заданной кодировке

Попробуйте внести в код для подключения к базе данных другие неточности (например, укажите неправильное имя базы данных), чтобы увидеть остальные подробные сообщения об ошибках. Проработав несколько вариантов, добейтесь успешного соединения с базой данных и верните код с простым сообщением — это избавит ваших посетителей от лишней информации.

Установив соединение и выбрав базу данных, самое время приступить к использованию информации, которая в ней хранится.

PHP АВТОМАТИЧЕСКИ РАЗРЫВАЕТ СОЕДИНЕНИЕ

Вам, конечно, интересно, что происходит с подключением к серверу MySQL после того, как скрипт завершает свою работу. При желании вы можете заставить PHP разорвать соединение с сервером, обнулив соответствующий объект PDO. Для этого присвойте переменной, хранящей объект, значение `null`.

```
$pdo = null; // отключаемся от сервера баз данных
```

Стоит сказать, что, выполнив скрипт, интерпретатор закрывает любые подключения к базе данных, поэтому лучше позволить PHP сделать все за вас.

Отправка SQL-запросов с помощью PHP

В главе 2 вы подключались к серверу баз данных MySQL, используя приложение phpMyAdmin. Оно позволяло вводить SQL-запросы (команды) и сразу же получать результат. Объект PDO предлагает аналогичный механизм — метод `exec`.

```
$pdo->exec(запрос)
```

В качестве *запроса* здесь выступает строка, содержащая любые необходимые для выполнения команды. Вы применяли метод `exec` в подразделе «Настройка подключения» этой главы, где с его помощью отправляли запрос `SET NAMES "utf8"`, чтобы установить кодировку для подключения к базе данных.

Как вам уже известно, если при выполнении запроса возникает проблема (например, в нем содержится синтаксическая ошибка), метод генерирует исключение `PDOException`, которое следует перехватить.

Для примера попытаемся воссоздать таблицу `joke` из главы 2 (`chapter4/createtable/index.php`, фрагмент).

```
try
{
  $sql = 'CREATE TABLE joke (
        id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
        joketext TEXT,
        jokedate DATE NOT NULL
      ) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB';
  $pdo->exec($sql);
}
catch (PDOException $e)
{
  $output = 'Ошибка при создании таблицы joke: ' . $e->getMessage();
  include 'output.html.php';
  exit();
}

$output = 'Таблица joke была успешно создана.';
include 'output.html.php';
```

Обратите внимание, что для обработки потенциальных ошибок запроса используется знакомое выражение `try – catch`. Приведенный код содержит также метод `getMessage`, который позволяет получить от сервера MySQL подробную

информацию о возникшей ошибке. На рис. 4.10 приведено сообщение, которое выводится на экран в случае, если создаваемая нами таблица `joke` уже существует.

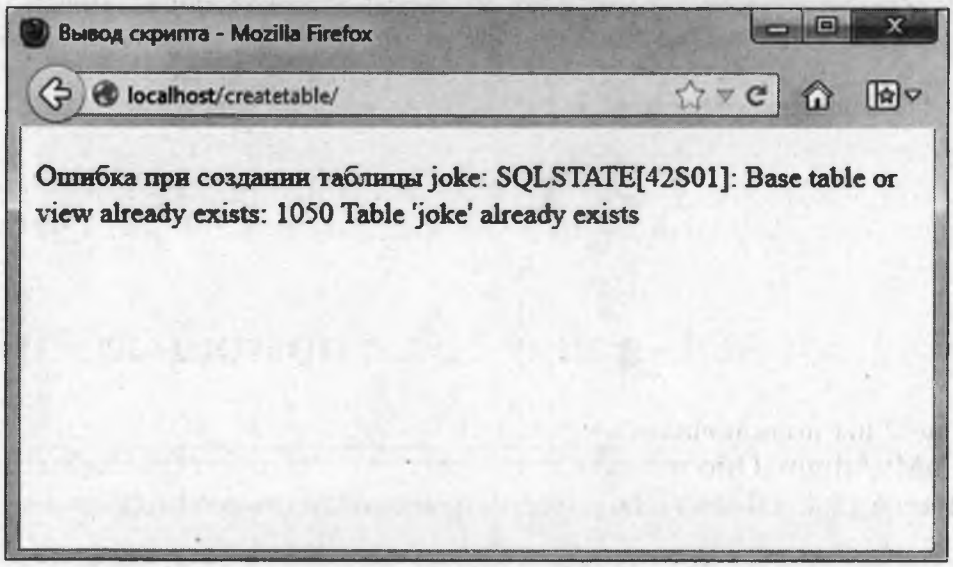


Рис. 4.10. Запрос `CREATE TABLE` не выполним, поскольку таблица уже существует

В результате выполнения запросов `DELETE`, `INSERT` и `UPDATE`, которые служат для изменения данных, хранящихся в базе, метод `exec` вернет количество строк (записей), в которых произошли преобразования. Следующая команда (одну из ее версий мы использовали в главе 2) позволит установить дату для всех шуток со словом «цыпленок» (`chapter4/updatechicken/index.php`, фрагмент).

```
try
{
  $sql = 'UPDATE joke SET jokedate="2012-04-01"
      WHERE joketext LIKE "%цыпленок%"';
  $affectedRows = $pdo->exec($sql);
}
catch (PDOException $e)
{
  $output = 'Ошибка при выполнении обновления: ' . $e->getMessage();
  include 'output.html.php';
  exit();
}
```

Сохранив в переменную `$affectedRows` значение, которое вернул метод `exec`, можно вывести количество строк, затронутых этим обновлением (`chapter4/updatechicken/index.php`, фрагмент).

```
$output = "Обновлено столбцов: $affectedRows";
include 'output.html.php';
```

На рис. 4.11 показан результат выполнения этого кода (предполагается, что в базе данных находится только одна шутка про цыпленка).

Если выполнить тот же запрос и повторно обновить страницу, сообщение изменится (рис. 4.12). В результате работы кода ни одна строка не изменится, поскольку дата, присваиваемая шуткам в этот раз, совпадет с уже существующей.

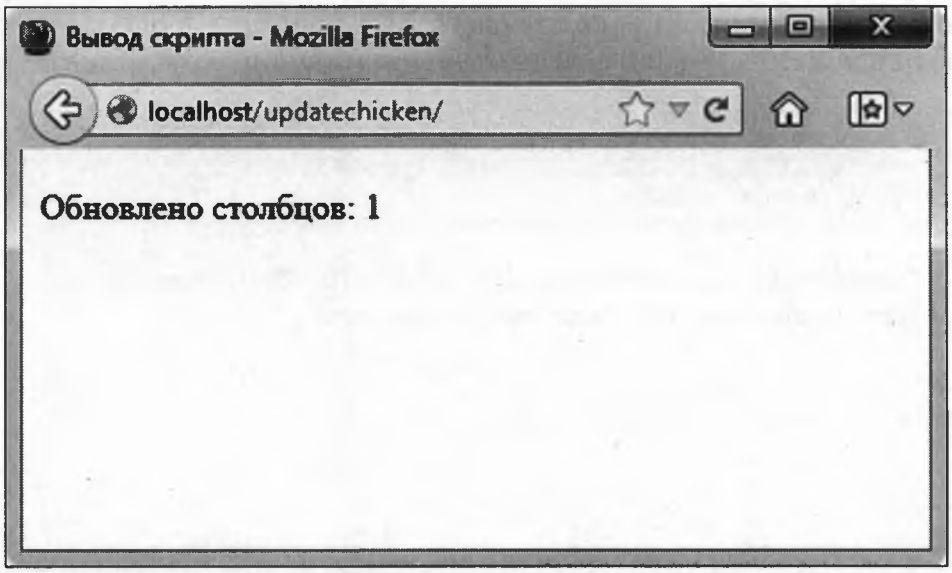


Рис. 4.11. Сообщение о количестве измененных записей в базе данных

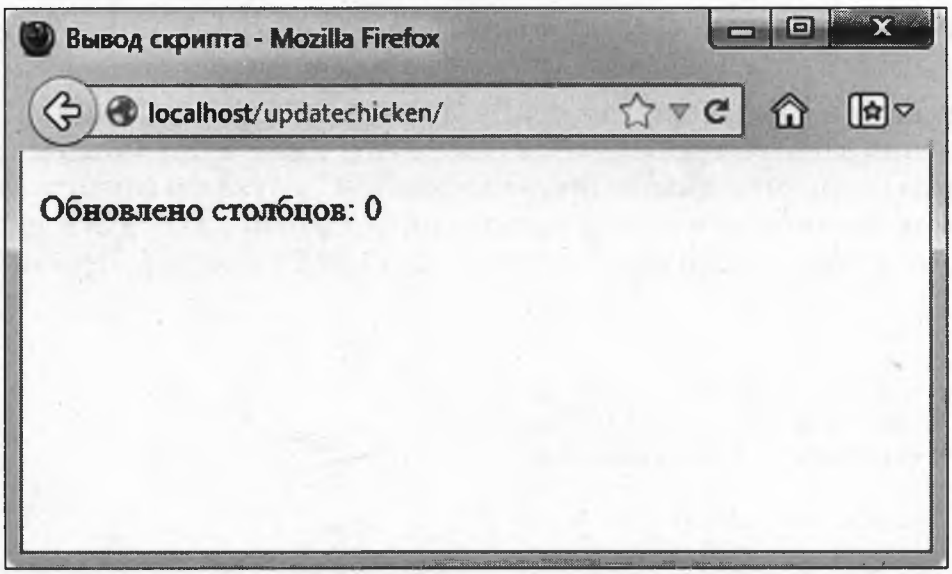


Рис. 4.12. MySQL даст знать, что вы попусту тратите время

Запросы с командой `SELECT` требуют немного другого подхода. Тем не менее PHP предоставляет все необходимые средства для работы с информацией, которая извлекается с их помощью.

Обработка результатов выполнения команды SELECT

Метод `exec` срабатывает для большинства SQL-запросов: с базой данных выполняются определенные действия, а в ответ возвращается количество строк, измененных в ходе запроса (если таковые имеются). Команда `SELECT` требует более изящного решения. Запросы, содержащие данную команду, используются для вывода хранящейся в базе данных информации. Таким образом, `SELECT`

не просто воздействует на содержимое базы данных, а несет в себе результаты, поэтому необходим метод, который поможет эти результаты вернуть.

Метод `query` похож на метод `exec` тем, что в качестве аргумента принимает SQL-запрос, направленный базе данных. Отличие заключается в том, что он возвращает объект `PDOStatement`, который представляет собой **результирующий набор**, иначе говоря, список всех строк (записей), полученных в результате запроса (`chapter4/listjokes/index.php`, фрагмент).

```
try
{
  $sql = 'SELECT joketext FROM joke';
  $result = $pdo->query($sql);
}
catch (PDOException $e)
{
  $error = 'Ошибка при извлечении шуток: ' . $e->getMessage();
  include 'error.html.php';
  exit();
}
```

Как и прежде, ошибки выводятся с помощью простого шаблона (`chapter4/listjokes/error.html.php`).

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Ошибка скрипта</title>
  </head>
  <body>
    <p>
      <?php echo $error; ?>
    </p>
  </body>
</html>
```

Если в результате выполнения запроса ошибок не возникло, код сохранит результирующий набор (в виде объекта `PDOStatement`) в переменную `$result`. Этот набор содержит тексты всех шуток, хранящихся в таблице `joke`. Поскольку на количество шуток в базе данных не накладывается практически никаких ограничений, результат может оказаться довольно объемным.

В главе 3 упоминалось о том, что для работы с большим объемом данных подходит цикл `while`. Используем его для поочередной обработки строк результирующего набора.

```
while ($row = $result->fetch())
{
  ⋮ обрабатываем строку
}
```

Условие данного цикла отличается от тех, которые вы привыкли видеть, поэтому поясним, как оно работает. Рассмотрим условие в виде отдельного выражения.

```
$row = $result->fetch();
```

Метод fetch объекта PDOStatement возвращает строки результирующего набора в виде массива (подробнее массивы рассмотрены в главе 3). Когда строки в результирующем наборе заканчиваются, метод возвращает false[1].

В вышеприведенной строке кода переменной $row присваивается значение, к которому сводится все выражение целиком. Благодаря этому вы можете использовать выражение в качестве условия для цикла while. Цикл продолжается до тех пор, пока условие не равно false, поэтому количество повторений совпадет с количеством строк в результирующем наборе. При этом переменная $row каждый раз будет принимать значение следующей строки. Теперь вам остается только извлечь значение из этой переменной при каждой итерации цикла.

Строки результирующего набора, возвращаемые методом fetch, представлены в виде ассоциативных массивов, где в качестве индексов используются названия столбцов таблицы. Если $row — строка в результирующем наборе, то $row['joketext'] — значение столбца joketext для этой строки.

Чтобы вывести все шутки внутри PHP-шаблона, необходимо получить их тексты. Для этого лучше всего сохранить каждую шутку в виде элемента массива $jokes (chapter4/listjokes/index.php, фрагмент).

```
while ($row = $result->fetch())
{
  $jokes[] = $row['joketext'];
}
```

Извлеченные из базы данных шутки передаются в шаблон jokes.html.php для последующего вывода.

Следовательно, итоговый код контроллера для данного примера выглядит следующим образом (chapter4/listjokes/index.php).

```
<?php
try
{
  $pdo = new PDO('mysql:host=localhost;dbname=ijdb', 'ijdbuser',
    'mypassword');
  $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $pdo->exec('SET NAMES "utf8"');
}
catch (PDOException $e)
{
  $error = 'Невозможно подключиться к серверу баз данных.';
  include 'error.html.php';
  exit();
}
try
{
  $sql = 'SELECT joketext FROM joke';
  $result = $pdo->query($sql);
}
catch (PDOException $e)
```

[1] Это один из тех случаев, когда объект PDO не может выполнить запрос: метод fetch не способен вернуть следующую строку, если в результирующем наборе их больше не осталось. При этом метод не генерирует исключение PDOException, иначе он бы не использовался в условии цикла while.

```
{
  $error = 'Ошибка при извлечении шуток: ' . $e->getMessage();
  include 'error.html.php';
  exit();
}
while ($row = $result->fetch())
{
  $jokes[] = $row['joketext'];
}

include 'jokes.html.php';
```

В завершение осталось написать шаблон **jokes.html.php**. В нем впервые придется выводить не одну простую переменную, а содержимое целого массива. Чаще всего для работы с массивами в PHP используют циклы, например уже знакомые вам `while` и `for`. В данном случае для перебора содержимого массива наиболее приемлем цикл `foreach`.

```
foreach (массив as $item)
{
  ⋮ обрабатываем каждый элемент $item
}
```

В верхней части цикла `foreach` в круглых скобках вместо условия указывают массив, за ним следует ключевое слово `as` и имя новой переменной, которая используется для поочередного хранения каждого элемента. Тело цикла выполняется для каждого значения внутри массива, а переменная, в которой хранится это значение, позволяет обращаться к нему напрямую.

Обычно цикл `foreach` используется в шаблонах для поочередного вывода каждого элемента массива. В случае с массивом `$jokes` это может выглядеть так.

```
<?php
foreach ($jokes as $joke)
{
?>
  ⋮ HTML-код для вывода каждого элемента $joke
<?php
}
?>
```

Сочетание PHP-кода, описывающего цикл, и HTML-кода, отображающего массив, выглядит не совсем аккуратно, поэтому в шаблонах часто используют альтернативную запись цикла `foreach`.

```
foreach (array as $item):
  ⋮ обрабатываем каждый элемент $item
endforeach;
```

Вот как выглядит эта версия кода внутри шаблона.

```
<?php foreach ($jokes as $joke): ?>
  ⋮ HTML-код для вывода каждого элемента $joke
<?php endforeach; ?>
```

Вооружившись данным инструментом, создадим шаблон для вывода списка шуток (`chapter4/listjokes/jokes.html.php`). Каждая запись выводится в виде отдельного абзаца внутри блока с цитатой `blockquote`.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Список шуток</title>
  </head>
  <body>
    <p>Вот все шутки, которые есть в базе данных:</p>
      <?php foreach ($jokes as $joke): ?>
      <blockquote>
        <p><?php echo htmlspecialchars($joke, ENT_QUOTES, 'UTF-8');
            ?>
        </p>
    </blockquote>
    <?php endforeach; ?>
  </body>
</html>
```

Шутки могут содержать символы, интерпретируемые как элементы разметки. Чтобы они корректно отображались, их следует преобразовать в специальные последовательности языка HTML. Например, заменить <, > и & на <, > и & соответственно. Для этого используется функция `htmlspecialchars`.

На рис. 4.13 показано, как будет выглядеть страница, после того как вы добавите в базу данных несколько шуток.

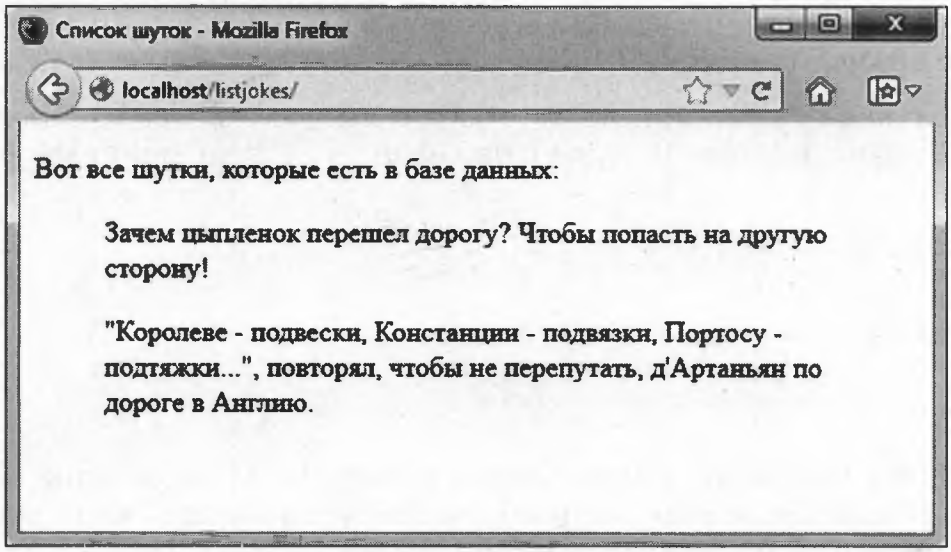


Рис. 4.13. Все шутки собраны в одном месте

ИСПОЛЬЗОВАНИЕ ЦИКЛА FOREACH ДЛЯ ПЕРЕХОДА ПО ЭЛЕМЕНТАМ РЕЗУЛЬТИРУЮЩЕГО НАБОРА

В приведенном выше примере для поочередного извлечения строк из результирующего набора `PDOStatement` применялся цикл `while` (`chapter4/listjokes/index.php`, фрагмент).

```
while ($row = $result->fetch())
{
  $jokes[] = $row['joketext'];
}
```

Оказывается, объект `PDOStatement`, помещенный внутрь цикла `foreach`, ведет себя подобно массиву. Таким образом, заменив `while` циклом `foreach`, можно немного упростить код для работы с базой данных.

```
foreach ($result as $row)
{
  $jokes[] = $row['joketext'];
}
```

В дальнейшем в оставшейся части книги используется именно вышеуказанный вариант перебора.

Добавление информации в базу данных

В этом разделе вы узнаете, как разрешить посетителям вашего сайта добавлять в базу данных собственные шутки.

Первое, что вам понадобится, — форма ввода. Вот шаблон, который отвечает нужным требованиям (`chapter4/addjoke/form.html.php`).

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Добавление шутки</title>
    <style type="text/css">
    textarea {
      display: block;
      width: 100%;
    }
    </style>
  </head>
  <body>
    <form action="?" method="post">
      <div>
        <label for="joketext">Введите сюда свою шутку:</label>
        <textarea id="joketext" name="joketext" rows="3" cols="40">
        </textarea>
      </div>
      <div><input type="submit" value="Добавить"></div>
    </form>
  </body>
</html>
```

Данная форма отправляет запрос тому самому скрипту, который ее сгенерировал, то есть контроллеру **index.php** (подобный подход мы уже рассматривали). Однако теперь вместо того, чтобы оставить атрибут `action` пустым, мы присвоили ему значение `?`. Позднее вы увидите, что адрес URL, который используется в этом примере для вывода формы, содержит строку запроса. Заданное значение атрибута `action` служит как раз для того, чтобы обнулить эту строку при отправке формы.

Вот как выглядит созданная форма в браузере (рис. 4.14).

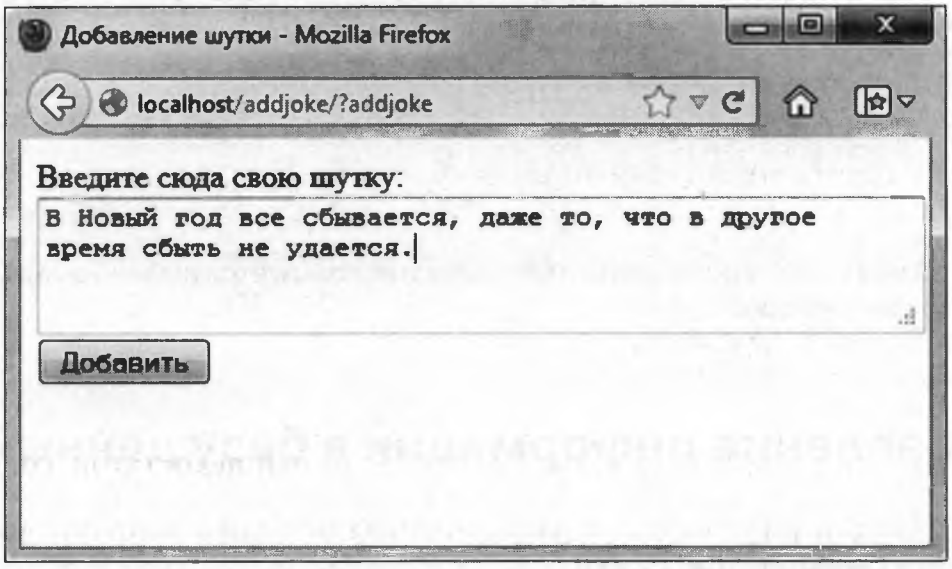


Рис. 4.14. В базу данных добавлена еще одна шутка

После отправки формы в запрос попадет переменная `joketext`, которая содержит текст шутки, набранной в поле ввода. Затем переменная появится в массивах `$_POST` и `$_REQUEST`, созданных интерпретатором PHP.

Свяжем разработанную форму с предыдущим примером, где выводится список шуток из базы данных. Для этого разместите над списком ссылку, предлагающую добавить шутку (`chapter4/addjoke/jokes.html.php`, фрагмент).

```
<body>
  <p><a href="?addjoke">Добавьте собственную шутку</a></p>
  <p> Вот все шутки, которые есть в базе данных:</p>
```

Как и форма, ссылка указывает на тот самый PHP-скрипт, с помощью которого сгенерирована данная страница, но при этом добавляется строка запроса `?addjoke`, сигнализирующая о том, что пользователь намерен пополнить базу новой шуткой. Контроллер определяет наличие этой строки и использует ее как сигнал для того, чтобы вместо списка шуток вывести форму.

Внесем в контроллер необходимые изменения (`chapter4/addjoke/index.php`, фрагмент).

```
if (isset($_GET['addjoke']))
{
  include 'form.html.php';
  exit();
}
```

Выражение `if` проверяет, включает ли строка запроса переменную с именем `addjoke`, — так определяется нажатие ссылки пользователем. Но даже если в строке запроса (`?addjoke`) не содержится значения для `addjoke`, переменная все равно будет создана. Выражение `isset($_GET['addjoke'])` позволяет в этом убедиться. При обнаружении переменной выводится форма и подключается шаблон form.html.php. Завершается скрипт командой `exit`.

Как только пользователь заполнит и отправит форму, к контроллеру поступит еще один запрос. Чтобы его распознать, необходимо проверить, установлено ли значение для `$_POST['joketext']` (`chapter4/addjoke/index.php`, фрагмент).

```
if (isset($_POST['joketext']))
{
```

Чтобы добавить отправленную шутку в базу данных, следует выполнить команду `INSERT`, используя для заполнения столбца `joketext` значение, хранящееся в `$_POST['joketext']`.

```
$sql = 'INSERT INTO joke SET
    joketext="' . $_POST['joketext'] . '",
    jokedate="today's date"';
$pdo->exec($sql);
```

Однако у этого кода есть один весьма серьезный недостаток: содержимое `$_POST['joketext']` полностью подвластно пользователю, который отправляет форму. Если в нее ввести опасный SQL-код, скрипт безропотно передаст его серверу MySQL. Такой вид атак называют **внедрением SQL-кода**. На ранних стадиях разработки PHP хакеры широко использовали эту распространенную уязвимость сайтов[1].

Команда разработчиков PHP добавила в язык средства для борьбы с внедрением SQL-кода. Во многих установленных копиях они до сих пор остаются включенными по умолчанию. Этот защитный механизм под названием **волшебные кавычки** автоматически анализирует переданные браузером значения и добавляет обратный слеш (\) перед всеми потенциально опасными символами, например кавычками, которые могут вызвать проблемы, если случайно попадут в SQL-запрос.

Однако сами волшебные кавычки вызывают не меньше проблем. Во-первых, метод, с помощью которого определяются и обрабатываются «опасные» символы, эффективен лишь в отдельных ситуациях. Некоторые кодировки, используемые на сайте и сервере баз данных, делают эти меры предосторожности совершенно бесполезными. Во-вторых, когда переданное значение используется не для создания SQL-запросов, а в каких-то *других* целях, обратный слеш появляется в нем очень некстати. Об этом упоминалось в главе 3, когда рассматривался пример с приветственным сообщением: механизм волшебных кавычек добавлял ненужную косую черту в фамилию пользователя, содержащую апостроф.

Проще говоря, волшебные кавычки оказались неудачной идеей, причем настолько, что уже в шестой версии разработчики PHP планируют от них избавиться. Однако пока этого не произошло, придется иметь дело с теми неудобствами, которые они вызывают. Самое простое решение проблемы — определить, включены ли волшебные кавычки на вашем веб-сервере, и при необходимости отменять те изменения, которые они вносят в передаваемые переменные[2]. В руководстве

[1] Внедрения SQL-кода по сей день остается весьма распространенным видом хакерской атаки, поскольку некоторые разработчики не подозревают о такой слабости PHP. Взгляните, например, на попытку заставить камеры дорожного видеонаблюдения удалить собственные базы данных (http://www.gizmodo.com.au/2010/03/sql-injection-license-plate-hopes-to-foil-euro-traffic-cameras/).

[2] Чтобы отключить волшебные кавычки и избавить свой веб-сервер от множества лишней работы, присвойте параметру `magic_quotes_gpc`, который находится внутри файла php.ini, значение `Off`. Затем убедитесь, что после изменения параметра ваш код работает корректно. Если же волшебные кавычки включены, вам придется их учитывать.

PHP[1] приводится фрагмент кода, который специально для этого предназначен (chapter4/addjoke/index.php, фрагмент).

```
if (get_magic_quotes_gpc())
{
  $process = array(&$_GET, &$_POST, &$_COOKIE, &$_REQUEST);
  while (list($key, $val) = each($process))
  {
    foreach ($val as $k => $v)
    {
      unset($process[$key][$k]);
      if (is_array($v))
      {
        $process[$key][stripslashes($k)] = $v;
        $process[] = &$process[$key][stripslashes($k)];
      }
      else
      {
        $process[$key][stripslashes($k)] = stripslashes($v);
      }
    }
  }
  unset($process);
}
```

Не пытайтесь разобраться в данном коде. В нем использованы расширенные возможности языка PHP, о которых вы пока не знаете, а также те, которые в этой книге вовсе не рассматриваются. Просто добавьте этот фрагмент в верхнюю часть контроллера или любой другой PHP-скрипт, принимающий данные от пользователя в виде строки запроса, через форму или в виде куки (об этом пойдет речь в главе 9). Если данный код потребуется в одном из дальнейших примеров, вы об этом узнаете[2].

Избавившись от проблем, к которым приводят волшебные кавычки, вы можете более эффективно взаимодействовать с базой данных и передавать данные с помощью подходящих механизмов, например параметризированных запросов.

Параметризированным называют SQL-запрос, который отправляется серверу баз данных заранее, чтобы подготовить его к выполнению дальнейших команд (непосредственно выполнения запроса в этот момент не происходит). Как правило, направляемый код содержит **псевдопеременные**, которые при формировании настоящего запроса заменяются необходимыми значениями. В процессе подстановки объект PDO автоматически защитит вас от «опасных» символов.

Рассмотрим подготовку запроса с командой INSERT и его безопасное выполнение с подстановкой $_POST['joketext'] вместо текста шутки.

```
$sql = 'INSERT INTO joke SET
  joketext = :joketext,
  jokedate = "сегодняшняя дата"';
```

[1] http://www.php.net/manual/en/security.magicquotes.disabling.php.

[2] В главе 6 вы увидите, как избежать необходимости добавлять этот код в контроллер.

```
$s = $pdo->prepare($sql);
$s->bindValue(':joketext', $_POST['joketext']);
$s->execute();
```

Разобьем код на отдельные выражения. Для начала SQL-запрос записывается в виде строки и сохраняется, как обычно, в переменную `$sql`. Однако в записи команды `INSERT` есть кое-что необычное: не указано значение для столбца `joketext`, вместо него находится псевдопеременная `:joketext`. К полю `jokedate` мы вернемся несколько позже.

На следующем этапе из объекта PDO (`$pdo`) вызывается метод `prepare`, которому в качестве аргумента передается SQL-запрос. Этот запрос направляется серверу MySQL, чтобы подготовить его к последующему выполнению команд. Обработать запрос MySQL пока не в состоянии — ему не хватает значения для столбца `joketext`. Метод `prepare` возвращает объект `PDOStatement` (того самого типа, с помощью которого мы получаем результаты выполнения команды `SELECT`) и сохраняет его в переменную `$s`.

Теперь, когда MySQL-сервер готов к выполнению запроса, ему можно отправить пропущенные значения (одно или несколько), вызвав метод `bindValue` из объекта `PDOStatement ($s)`. Данный метод вызывается при передаче каждого значения (в нашем случае всего один раз для текста шутки). В качестве аргумента кроме самого значения `$_POST['joketext']`,ему передается псевдопеременная `':joketext'`, которую необходимо заменить. Сервер MySQL знает, что он получает содержимое отдельных переменных, а не SQL-код. Следовательно, исчезает риск того, что отправленные символы интерпретируются как команды на языке SQL. Таким образом, использование параметризированных запросов исключает уязвимости, связанные с внедрением SQL-кода.

В завершение из объекта `PDOStatement` вызывается метод `execute`, чтобы сервер MySQL смог выполнить запрос с теми значениями, которые ему предоставили[1].

Осталось разобраться, как присвоить полю `jokedate` сегодняшнюю дату. Конечно, можно было бы написать какой-нибудь причудливый код, который бы генерировал ее в подходящем для MySQL формате (YYYY-MM-DD). Однако не стоит этого делать: у сервера баз данных есть специальная функция `CURDATE`.

```
$sql = 'INSERT INTO joke SET
    joketext = :joketext,
    jokedate = CURDATE()';
$s = $pdo->prepare($sql);
$s->bindValue(':joketext', $_POST['joketext']);
$s->execute();
```

Функция `CURDATE`, входящая в состав MySQL, используется для присваивания текущей даты в качестве значения для столбца `jokedate`. На самом деле в MySQL есть множество подобных функций, но мы будем останавливаться на них только по мере необходимости. Те из них, которые используются наиболее часто, вы найдете в приложении Б.

[1] В отличие от аналогичного по действию метода `exec` метод из объекта `PDOStatement` носит другое название — `execute`. У PHP есть много сильных сторон, но единообразие не входит в их число.

Теперь, когда запрос составлен, завершим выражение `if`, созданное ранее для обработки данных, поступающих из формы (`chapter4/addjoke/index.php`, фрагмент).

```
if (isset($_POST['joketext']))
{
  try
  {
    $sql = 'INSERT INTO joke SET
        joketext = :joketext,
        jokedate = CURDATE()';
    $s = $pdo->prepare($sql);
    $s->bindValue(':joketext', $_POST['joketext']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при добавлении шутки: ' . $e->getMessage();
    include 'error.html.php';
    exit();
  }

  header('Location: .');
  exit();
}
```

Конструкция `if` имеет еще одну интересную особенность. После того как новая шутка добавлена в базу данных, вместо того чтобы вывести привычный шаблон, необходимо перенаправить браузер пользователя к списку шуток, среди которых уже есть новый текст. Именно этим занимаются две строки кода, выделенные полужирным начертанием внутри конструкции.

Первое, что приходит на ум для достижения подобного результата, — позволить контроллеру извлечь список шуток из базы данных после добавления новой записи и вывести полученный результат с помощью шаблона **jokes.html.php**. Однако проблема в том, что браузер отображает список шуток в результате отправки формы. Если по какой-то причине пользователь обновит страницу, браузер повторно отправит ту же форму, и в в базе данных появится копия только что добавленной шутки, что нежелательно.

Необходимо сделать так, чтобы браузер обходился с дополненным списком шуток как с обычной веб-страницей и ее обновление не приводило к повторной отправке формы. Для этого в ответ на получение формы нужно отправить браузеру специальный **HTTP-заголовок для перенаправления**[1], который сообщит ему примерно следующее: «Страница, которую вы ищите, находится *здесь*».

Функция `header` предоставляет серверу средства для отправки специальных ответов, как тот, что приведен выше, и позволяет возвращать браузеру особые **заголовки**. Чтобы объявить о перенаправлении, вы должны отослать заголовок `Location` вместе с адресом той страницы, на которую хотите направить браузер.

```
header('Location: URL');
```

[1] HTTP расшифровывается как HypertextTransferProtocol, что в переводе означает «протокол передачи гипертекста». Это язык, который описывает взаимодействие между браузером посетителя и веб-сервером на уровне запрос — ответ.

В нашем примере браузеру требуется вернуться на ту же страницу (контроллер). Мы не отсылаем его в другое место, а просим выполнить еще один запрос, но на этот раз без отправки формы. Поскольку браузер направляется к контроллеру index.php, ему следует перезагрузить текущую директорию (она обозначается точкой), которая является родительской по отношению к скрипту.

Ниже приведены две строки кода, которые после добавления новой шутки в базу данных перенаправляют браузер обратно к контроллеру (`chapter4/addjoke/index.php`, фрагмент).

```
  header('Location: .');
  exit();
}
```

$_SERVER['PHP_SELF'] — адрес URL текущей страницы

Еще один способ получить адрес текущей страницы в PHP — использовать переменную `$_SERVER['PHP_SELF']`. Как и `$_GET`, `$_POST` и `$_REQUEST`, она представляет собой массив, который автоматически создается интерпретатором. Этот массив содержит полный набор информации, предоставляемой вашим сервером. В частности, здесь хранится адрес PHP-скрипта, с помощью которого веб-сервер генерирует текущую страницу.

К сожалению, сервер автоматически преобразует адрес http://localhost/addjoke/ в http://localhost/addjoke/index.php, поэтому переменная `$_SERVER['PHP_SELF']` содержит длинную версию URL. Перенаправление браузера с помощью точки позволяет сохранить укороченный и более удобный для восприятия адрес. В связи с этим `$_SERVER['PHP_SELF']` в книге не используется. Однако, учитывая то, что она фигурирует во множестве стандартных примеров, рассматриваемых в Интернете, вам будет полезно знать ее предназначение.

Оставшаяся часть контроллера, как и прежде, ответственна за вывод списка с шутками. Вот полный код скрипта (`chapter4/addjoke/index.php`).

```
<?php
if (get_magic_quotes_gpc())
{
  $process = array(&$_GET, &$_POST, &$_COOKIE, &$_REQUEST);
  while (list($key, $val) = each($process))
  {
    foreach ($val as $k => $v)
    {
      unset($process[$key][$k]);
      if (is_array($v))
      {
        $process[$key][stripslashes($k)] = $v;
        $process[] = &$process[$key][stripslashes($k)];
      }
      else
      {
        $process[$key][stripslashes($k)] = stripslashes($v);
      }
    }
  }
  unset($process);
```

```
}

if (isset($_GET['addjoke']))
{
  include 'form.html.php';
  exit();
}

try
{
  $pdo = new PDO('mysql:host=localhost;dbname=ijdb', 'ijdbuser',
      'mypassword');
  $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $pdo->exec('SET NAMES "utf8"');
}
catch (PDOException $e)
{
  $error = 'Невозможно подключиться к серверу баз данных.';
  include 'error.html.php';
  exit();
}

if (isset($_POST['joketext']))
{
  try
  {
    $sql = 'INSERT INTO joke SET
        joketext = :joketext,
        jokedate = CURDATE()';
    $s = $pdo->prepare($sql);
    $s->bindValue(':joketext', $_POST['joketext']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при добавлении шутки: ' . $e->getMessage();
    include 'error.html.php';
    exit();
  }

  header('Location: .');
  exit();
}

try
{
  $sql = 'SELECT joketext FROM joke';
  $result = $pdo->query($sql);
}
catch (PDOException $e)
{
  $error = 'Ошибка при извлечении шуток: ' . $e->getMessage();
  include 'error.html.php';
  exit();
}

while ($row = $result->fetch())
```

```
{
  $jokes[] = $row['joketext'];
}

include 'jokes.html.php';
```

Перечитайте код, чтобы убедиться в том, что вам все понятно, и обратите внимание на ту часть, где выполняется подключение к базе данных. Создание объекта PDO должно предшествовать выполнению любых запросов. Тем не менее для вывода формы, позволяющей добавлять шутки, соединение с базой данных вовсе необязательно, поэтому данный фрагмент кода может размещаться в самом верху контроллера.

Загрузите скрипт в браузер и добавьте в базу данных несколько шуток (рис. 4.15).

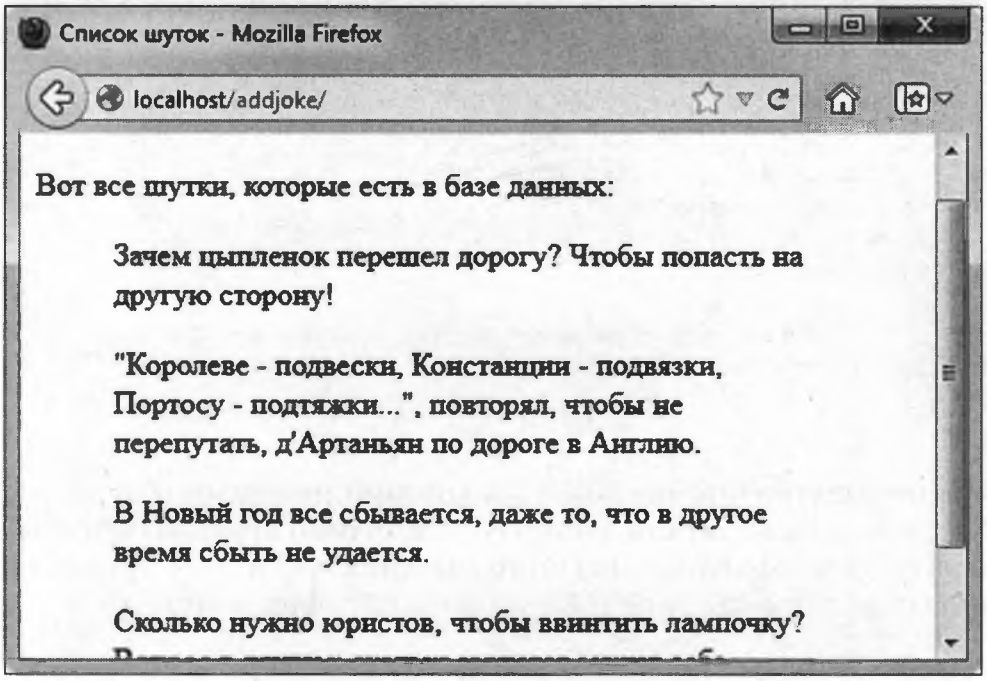


Рис. 4.15. И никакого SQL!

Вот и все. С помощью единственного контроллера index.php, управляющего всем процессом, можно пополнять базу данных MySQL новыми шутками и извлекать те, которые в ней хранятся.

Удаление информации из базы данных

Добавим к нашему примеру последний штрих: разместим на странице рядом с каждой шуткой кнопку Удалить. Нажав ее, пользователь удалит запись из базы данных, а на экране появится обновленный список шуток.

Прежде чем ознакомиться с предложенным ниже вариантом решения, попробуйте справиться с задачей самостоятельно. Несмотря на совершенно новую функциональность, вам потребуются все те же инструменты из предыдущих примеров главы 4. Вот несколько подсказок.

- Для выполнения задачи достаточно одного-единственного контроллера index.php.
- Вам понадобится команда DELETE из языка SQL, с которой вы познакомились в главе 2.
- Нужно однозначно идентифицировать ту шутку, которую вы хотите удалить в своем контроллере. Передайте идентификатор соответствующей записи вместе с запросом на удаление, используя столбец id в таблице joke. Проще всего это сделать с помощью скрытого поля в форме.

Подумайте несколько минут над тем, как это все реализовать, и, когда почувствуете, что готовы увидеть решение, вернитесь к чтению книги.

Первым делом следует изменить запрос с командой SELECT, который извлекает список шуток из базы данных. Чтобы однозначно идентифицировать каждую запись, помимо joketext, нужно получить столбец id (chapter4/deletejoke/index.php, фрагмент).

```
try
{
  $sql = 'SELECT id, joketext FROM joke';
  $result = $pdo->query($sql);
}
catch (PDOException $e)
{
  $error = ' Ошибка при извлечении шуток: ' . $e->getMessage();
  include 'error.html.php';
  exit();
}
```

Теперь отредактируйте цикл while, который помещает результаты запроса в массив $jokes. Каждому его элементу необходимо присвоить не только текст шутки, но и ее идентификатор. Для этого значения массива $jokes должны стать массивами (chapter4/deletejoke/index.php, фрагмент).

```
while ($row = $result->fetch())
{
  $jokes[] = array('id' => $row['id'], 'text' => $row['joketext']);
}
```

ИСПОЛЬЗОВАНИЕ FOREACH

Такой подход пригоден, даже если при обработке записей из базы данных вы предпочитаете использовать цикл foreach.

```
foreach ($result as $row)
{
  $jokes[] = array('id' => $row['id'], 'text' =>
    $row['joketext']);
}
```

В результате работы цикла while сформируется массив $jokes, где каждый элемент является ассоциативным массивом с двумя значениями — идентификатором шутки и ее текстом. Таким образом, из каждой шутки $jokes[n] мы можно извлечь поля id ($jokes[n]['id']) и joketext ($jokes[n]['text']).

На следующем этапе потребуется изменение шаблона **jokes.html.php**. Необходимо, чтобы он извлекал из нового структурированного массива тексты шуток и добавлял возле каждой записи кнопку **Удалить** (`chapter4/deletejoke/jokes.html.php`, фрагмент).

```
<?php foreach ($jokes as $joke): ?>
  <form action="?deletejoke" method="post"> ❶
    <blockquote>
      <p>
        <?php echo htmlspecialchars($joke['text'], ENT_QUOTES, 'UTF-8');
        ?> ❷
        <input type="hidden" name="id" value="<?php
          echo $joke['id']; ?>"> ❸
        <input type="submit" value="Удалить"> ❹
      </p>
    </blockquote>
  </form> ❺
<?php endforeach; ?>
```

Рассмотрим наиболее важные участки обновленного кода.

1. Каждая шутка выводится в виде формы, отправляя последнюю, вы удаляете запись. Контроллер узнает об этом с помощью строки запроса `?deletejoke` в атрибуте `action`.
2. Любая шутка из массива `$jokes` представляет собой не простой текст, а двухэлементный массив. Чтобы получить ее содержимое, необходимо обновить данную строку. Для этого вместо `$joke` используется `$joke['text']`.
3. Идентификатор удаляемой шутки передается при отправке формы для удаления. Для этого id сохраняется в одно из ее полей. Поскольку пользователь не должен видеть это поле, оно делается скрытым (`<input type="hidden">`). Поле обладает именем `id` и содержит идентификатор шутки, которую нужно удалить (`$joke['id']`).
4. В отличие от текста шутки идентификатор пользователем не передается, поэтому нет необходимости применять функцию `htmlspecialchars`, чтобы сделать его безопасным с точки зрения HTML. Можно быть уверенными в том, что это будет число, так как оно автоматически генерируется для столбца `id` самим сервером MySQL, когда происходит добавление шутки.
5. Нажатие кнопки `<input type="submit">` обеспечивает отправку формы. Текст кнопки определяется значением атрибута `value`.
6. В конце форма для шутки закрывается.

УЛУЧШЕНИЕ РАЗМЕТКИ

Если вы разбираетесь в HTML, то, скорее всего, подумаете, что теги `input` должны находиться за пределами элемента `blockquote`, поскольку они не являются частью цитируемого текста (шутки). В действительности так оно и есть: форма и ее поля должны находиться либо перед элементом `blockquote`, либо после него. К сожалению, чтобы такая структура тегов отображалась нормально, пришлось бы воспользоваться каскадными таблицами стилей (Cascading Style Sheets, или CSS), что идет вразрез с тематикой книги.

Использование не совсем корректной разметки позволило оградить вас от тонкостей CSS в книге про PHP и MySQL. Если вы планируете использовать приведенный выше код в реальном проекте, уделите некоторое время изучению CSS или воспользуйтесь услугами того, кто разбирается в этой области. А пока вы получите полный контроль над HTML-кодом, не беспокоясь о стилях CSS, обеспечивающих красивый внешний вид страницы.

На рис. 4.16 представлен список шуток с кнопками удаления.

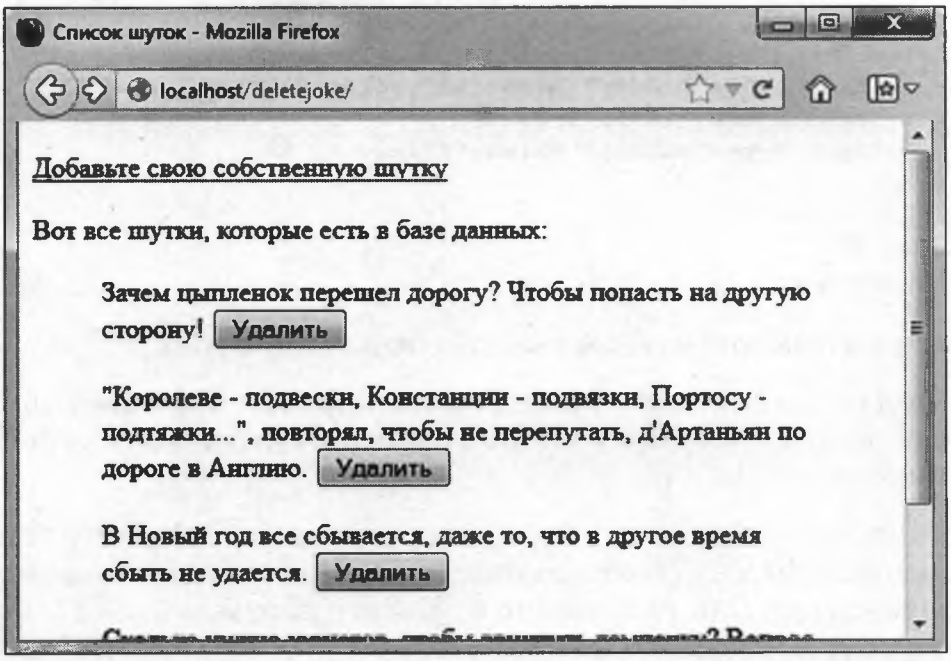


Рис. 4.16. Каждая кнопка удаляет отдельную шутку

Теперь, чтобы функция заработала, осталось обновить контроллер. После этого скрипт сможет обрабатывать форму, отправляемую при нажатии одной из кнопок (`chapter4/deletejoke/index.php`, фрагмент).

```
if (isset($_GET['deletejoke']))
{
  try
  {
    $sql = 'DELETE FROM joke WHERE id = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при удалении шутки: ' . $e->getMessage();
    include 'error.html.php';
    exit();
  }

  header('Location: .');
  exit();
}
```

Приведенный фрагмент кода работает аналогично тому, с помощью которого ранее в этой главе шутки добавлялись. Для начала подготавливается команда DELETE с псевдопеременной для идентификатора удаляемой записи[1]. Затем вместо псевдопеременной подставляется полученное значение `$_POST['id']` и выполняется запрос. В завершение кода функция `header` обеспечивает отправку нового запроса браузером для вывода обновленного списка шуток.

ПОЧЕМУ НЕ ССЫЛКА?

Если бы вы решали данный пример самостоятельно, то для удаления шуток вы, наверное, интуитивно выбрали бы гиперссылки. Это позволило бы избежать проблем с написанием полноценных HTML-форм для каждой записи на странице. Действительно, код для таких ссылок выглядит намного проще.

```
<?php foreach ($jokes as $joke): ?>
  <blockquote>
    <p>
    <?php echo htmlspecialchars($joke['text'], ENT_QUOTES,
        'UTF-8'); ?>
      <a href="?deletejoke&amp;id=<?php echo $joke['id'];
      ?>">Удалить</a>
    </p>
  </blockquote>
<?php endforeach; ?>
```

Однако гиперссылки не должны использоваться для выполнения каких-либо действий (например, удаления шутки). Единственное их назначение — обеспечивать переход к соответствующей информации. Это же относится и к формам с атрибутом `method="get"`: они применяются только для тех запросов, которые касаются уже сохраненных данных. Выполнение действий необходимо осуществлять исключительно с помощью форм, имеющих атрибут `method="post"`.

Дело в том, что браузеры и сопутствующее программное обеспечение воспринимают формы с атрибутом `method="post"` по-особому. Если вы, например, отправите такую форму, а затем обновите страницу, браузер спросит, действительно ли вы хотите отправить данные повторно. Для ссылок и форм с атрибутом `method="get"` подобная защита не предусмотрена.

Программное обеспечение для ускорения доступа в Интернет (и некоторые современные браузеры) осуществляют автоматический переход по ссылкам в фоновом режиме, чтобы при необходимости моментально отобразить страницы, на которые они ведут. Если шутки на вашем сайте удаляются с помощью ссылок, то браузер посетителя может автоматически удалить ваши записи.

Ниже приведена окончательная версия контроллера (`chapter4/deletejoke/index.php`). Все возникшие вопросы вы можете задать на форуме SitePoint (http://www.sitepoint.com/forums/).

[1] Вы, наверное, думаете, что здесь нет необходимости использовать параметризированные запросы и защищать базу данных от внедрения SQL-кода, поскольку идентификатор записи поступает из скрытого поля, незаметного для пользователя. На самом деле все поля формы (даже скрытые) подвластны контролю. Например, существуют широко распространенные дополнения для браузеров, которые способны делать скрытые поля видимыми и доступными для редактирования. Помните: когда дело касается безопасности сайта, любое отправленное браузером значение так или иначе подозрительно.

```
<?php
if (get_magic_quotes_gpc())
{
  $process = array(&$_GET, &$_POST, &$_COOKIE, &$_REQUEST);
  while (list($key, $val) = each($process))
  {
    foreach ($val as $k => $v)
    {
      unset($process[$key][$k]);
      if (is_array($v))
      {
        $process[$key][stripslashes($k)] = $v;
        $process[] = &$process[$key][stripslashes($k)];
      }
      else
      {
        $process[$key][stripslashes($k)] = stripslashes($v);
      }
    }
  }
  unset($process);
}

if (isset($_GET['addjoke']))
{
  include 'form.html.php';
  exit();
}

try
{
  $pdo = new PDO('mysql:host=localhost;dbname=ijdb', 'ijdbuser',
      'mypassword');
  $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $pdo->exec('SET NAMES "utf8"');
}
catch (PDOException $e)
{
  $error = 'Невозможно подключиться к серверу баз данных.';
  include 'error.html.php';
  exit();
}

if (isset($_POST['joketext']))
{
  try
  {
    $sql = 'INSERT INTO joke SET
        joketext = :joketext,
        jokedate = CURDATE()';
    $s = $pdo->prepare($sql);
    $s->bindValue(':joketext', $_POST['joketext']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при добавлении шутки: ' . $e->getMessage();
```

```
    include 'error.html.php';
    exit();
  }

  header('Location: .');
  exit();
}

if (isset($_GET['deletejoke']))
{
  try
  {
    $sql = 'DELETE FROM joke WHERE id = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при удалении шутки: ' . $e->getMessage();
    include 'error.html.php';
    exit();
  }

  header('Location: .');
  exit();
}

try
{
  $sql = 'SELECT id, joketext FROM joke';
  $result = $pdo->query($sql);
}
catch (PDOException $e)
{
  $error = 'Ошибка при извлечении шуток: ' . $e->getMessage();
  include 'error.html.php';
  exit();
}

while ($row = $result->fetch())
{
  $jokes[] = array('id' => $row['id'], 'text' => $row['joketext']);
}

include 'jokes.html.php';
```

Миссия выполнена

В этой главе вы рассмотрели PDO, встроенные в PHP классы (`PDO`, `PDOException` и `PDOStatement`). Создавая новые объекты и вызывая предоставляемые ими методы, вы научились взаимодействовать с сервером баз данных MySQL и попутно освоили основы объектно-ориентированного программирования, что для новичка в PHP — весьма серьезное достижение. С помощью

объектов PDO вы создали свой первый сайт с поддержкой MySQL, предоставили его посетителям доступ к базе данных `ijdb`, позволили им добавлять и удалять шутки.

Можно сказать, что в этой главе вы в некотором смысле достигли главной цели книги: научились создавать сайты на основе базы данных. Конечно, пока рассмотренные примеры содержали только самую необходимую информацию. Дальнейший материал поможет вам расширить знания.

В главе 5 вы снова вернетесь к окну для ввода команд в программе phpMyAdmin и узнаете, как использовать принципы реляционных баз данных и нетривиальные SQL-запросы для представления более сложных видов информации.

Глава 5
ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

Начиная с главы 2 вы работали с весьма простой базой данных для хранения шуток, состоящей из единственной таблицы с названием `joke`. Эта база данных послужила хорошим введением в MySQL, но на ее примере невозможно охватить обширную тему проектирования реляционных баз данных. В настоящей главе вы углубите уже приобретенные знания и изучите несколько новых возможностей MySQL, что позволит вам понять и оценить реальную мощь реляционной модели.

Обратите внимание, что некоторые темы рассмотрены поверхностно. Любой специалист в области информатики скажет вам, что проектирование баз данных — серьезная область исследований, в которой действуют проверенные на практике и математически доказуемые законы. Однако, несмотря на всю ценность подобного материала, он выходит за рамки данной книги.

Более полную информацию о принципах проектирования реляционных баз данных и о SQL в целом можно найти в книге Simply SQL (http://www.sitepoint.com/books/sql1/). Если вы хотите глубже изучить законы, на которых основана реляционная модель, вам стоит прочесть Database in Depth (Sebastopol: O'Reilly, 2005; http://oreilly.com/catalog/9780596100124/).

Отдаем должное

Для начала освежим в памяти структуру таблицы `joke`. Если вы помните, она содержит три столбца, благодаря которым можно идентифицировать шутки (`id`), отслеживать их текст (`joketext`) и дату добавления (`jokedate`). Ниже приведен SQL-код для создания этой таблицы и добавления в нее нескольких записей (`chapter5/sql/jokes1.sql`)[1].

```
# Код для создания простой таблицы joke.

CREATE TABLE joke (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  joketext TEXT,
```

[1] Если вам понадобилось воссоздать свою базу данных с нуля, удалите все таблицы с помощью phpMyAdmin, затем перейдите на вкладку **Импорт** с пустой базой данных `ijdb` и загрузите туда приведенный SQL-файл. Приложение phpMyAdmin выполнит все содержащиеся в нем команды и таким образом восстановит базу данных. Вы можете использовать файлы `.sql`, которые находятся в архиве с кодом для этой книги, в качестве своеобразных копий состояния базы данных и загружать их по мере необходимости.

```
  jokedate DATE NOT NULL
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB;

# Добавление шуток в таблицу.

INSERT INTO joke SET
joketext = 'Зачем цыпленок перешел дорогу? Чтобы попасть на другую
сторону!',
jokedate = '2009-04-01';

INSERT INTO joke
(joketext, jokedate) VALUES (
'"Королеве - подвески, Констанции - подвязки, Портосу - подтяжки..." -
повторял, чтобы не перепутать, д''Артаньян по дороге в Англию.',
"2009-04-01"
);
```

Теперь представим, что вам необходимы еще кое-какие сведения о шутках, например имена людей, которые их добавили. Решение, которое напрашивается само собой, — ввести в таблицу `joke` еще один столбец. В языке SQL для этого служит команда `ALTER TABLE`.

Откройте приложение phpMyAdmin (см. главу 2), выберите базу данных (если вы использовали те же названия, что и в книге, то это будет `ijdb`) и введите следующую команду.

```
ALTER TABLE joke ADD COLUMN authorname VARCHAR(255)
```

Этот код добавит в таблицу столбец под названием `authorname`. В качестве типа выбран `VARCHAR(255)` — **строка переменной длины** не более 255 символов, он позволит ввести любое, даже очень замысловатое и длинное имя. Аналогичным образом создайте столбец для адресов электронной почты авторов шуток. Больше сведений о команде `ALTER TABLE` вы найдете в приложении Б.

```
ALTER TABLE joke ADD COLUMN authoremail VARCHAR(255)
```

Чтобы убедиться, что оба столбца добавлены должным образом, попросите MySQL описать структуру таблицы[1].

```
DESCRIBE joke
```

В результате вы получите таблицу, показанную на рис. 5.1.

Выглядит неплохо, не так ли? Естественно, для работы с новой расширенной структурой таблицы понадобится внести изменения в HTML- и PHP-код из главы 4 и добавить информацию об авторах шуток с помощью запросов, содержащих команду `UPDATE`. Однако прежде чем тратить время на подобные изменения, следует остановиться и задуматься: верно ли выбрана структура таблицы? Оказывается, не совсем.

[1] Вместо того чтобы набирать запрос `DESCRIBE` вручную, выберите в phpMyAdmin таблицу `joke` и перейдите на вкладку **Структура**. Однако пока вы не набрались опыта в администрировании баз данных, лучше использовать любую возможность для работы с командами.

Имя	Тип	Null	По умолчанию	Дополнительно
id	int(11)	Нет	Нет	AUTO_INCREMENT
joketext	text	Да	NULL	
jokedate	date	Нет	Нет	
authorname	varchar(255)	Да	NULL	
authoremail	varchar(255)	Да	NULL	

Рис. 5.1. Теперь таблица `joke` содержит пять столбцов

Разные сущности лучше хранить отдельно

По мере углубления ваших знаний о сайтах, основанных на базе данных, вы можете решить, что представленный вами список шуток слишком мал. Безусловно, вы можете сделать так, чтобы количество записей, добавленных пользователями, превысило количество ваших собственных шуток. Допустим, вы решите создать сайт, на котором люди со всего мира станут обмениваться анекдотами. Сопровождать каждую шутку именем и электронным адресом автора довольно разумно, но используемый выше подход может привести к проблемам.

Что если Джоан Смит, активный пользователь вашего сайта, изменит адрес электронной почты? Она начнет отправлять шутки, указывая новые сведения, а добавленные ранее записи останутся привязанными к ее старому адресу. Заглянув в базу данных, вы подумаете, что записи сделаны двумя разными людьми с одинаковыми именами. Конечно, Джоан может уведомить вас об изменении личных данных и вы попытаетесь обновить ее старые шутки, указав новый адрес. Однако если вы пропустите хотя бы одну запись, содержимое вашей базы станет некорректным. Специалисты в области проектирования называют такого рода проблемы **аномалиями обновления**.

Наиболее очевидное решение — воспользоваться функциями базы данных, чтобы получить список всех пользователей, которые когда-либо добавляли шутки на сайт. Вы легко получите адреса электронной почты с помощью следующего запроса.

```
SELECT DISTINCT authorname, authoremail
FROM joke
```

Слово `DISTINCT` предотвращает вывод дублирующихся строк. Например, если Джоан Смит добавила на сайт 20 записей, то благодаря этому параметру ее имя появится в общем списке всего один раз, а не 20.

Если по какой-то причине вы решите удалить все шутки одного пользователя, вы сотрете не только его записи, но и любые сведения о нем, и больше никогда, например, не сможете отправить ему письмо с информацией о своем сайте. Специалисты в области проектирования баз данных называют это **аномалиями удаления**. Иногда список адресов электронной почты служит главным источником дохода для веб-ресурса, поэтому было бы глупо удалять всю информацию об авторе только потому, что вам не нравятся его шутки.

Еще один нюанс: вы не всегда можете быть уверены в том, что Джоан Смит каждый раз станет вводить свое имя одним и тем же образом. Джоан Смит, Дж. Смит, Смит, Джоан... Думаем, вы уловили суть. Отследить конкретного автора становится куда сложнее, особенно если Джоан Смит предпочитает использовать сразу несколько электронных адресов.

С этими и другими проблемами позволяют справиться общепринятые принципы проектирования. Вместо того чтобы хранить информацию об авторах в таблице `joke`, создайте еще одну таблицу. Используйте в ней для идентификации пользователей столбец `id` с уникальными числами аналогично тому, как это сделано для шуток. В дальнейшем эти идентификаторы можно задействовать в таблице `joke`, чтобы связать шутки с именами тех, кто их добавил. Окончательная структура таблицы показана на рис. 5.2.

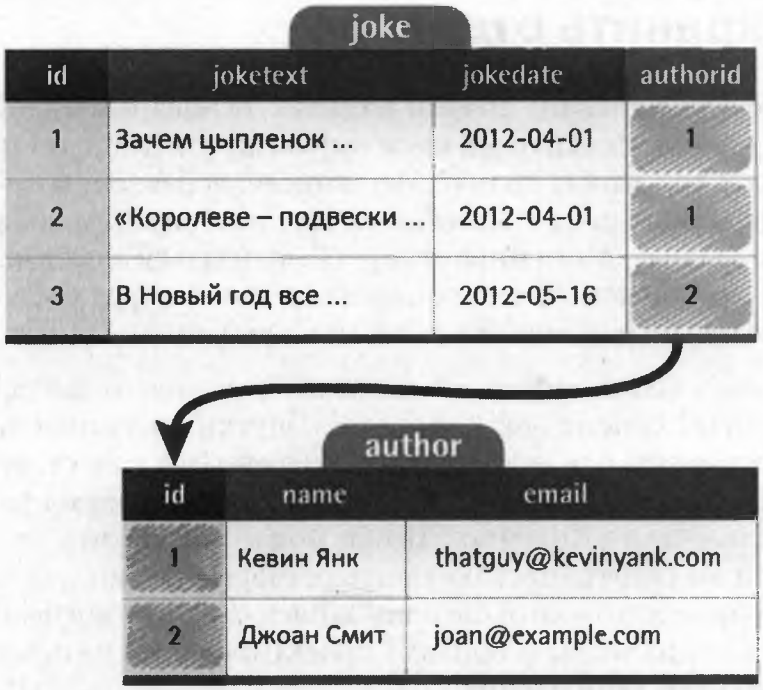


Рис. 5.2. Поле `authorid` в таблице `author` связано с соответствующей строкой таблицы `joke`

Итак, в представленных таблицах хранятся три шутки и два имени пользователя. Столбец `authorid` внутри таблицы `joke` устанавливает **связь** между двумя таблицами, указывая на то, что Кевин Янк отправил шутки с номерами 1 и 2, а Джоан Смит — автор шутки номер 3. Обратите внимание: поскольку каждый автор упоминается в базе данных всего один раз в отдельно взятой таблице, нам удалось избежать всех потенциальных проблем, о которых говорилось выше.

Используемая в примере структура базы данных подразумевает хранение двух видов сущностей (шуток и авторов), поэтому здесь наиболее подходит вариант с двумя таблицами. Всегда руководствуйтесь данным принципом при проектировании баз данных: *каждый тип объекта (или сущность), должен располагаться в отдельной таблице.*

Создать с нуля базу данных с вышеописанной структурой довольно легко: достаточно всего двух запросов с командами `CREATE TABLE`. Но поскольку нам

необходимо сохранить бесценные шутки, придется воспользоваться командой ALTER TABLE. Для начала удалим из таблицы joke столбцы с именами авторов и адресами их электронной почты.

```
ALTER TABLE joke DROP COLUMN authorname

ALTER TABLE joke DROP COLUMN authoremail
```

Затем создадим новую таблицу.

```
CREATE TABLE author (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255),
  email VARCHAR(255)
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB
```

В завершение добавим в таблицу joke столбец authorid.

```
ALTER TABLE joke ADD COLUMN authorid INT
```

Если вы решили создать обе таблицы с нуля, вот команды CREATE TABLE, которые вам понадобятся (chapter5/sql/2tables.sql, фрагмент).

```
# Код для создания простой таблицы joke,
# которая хранит идентификаторы авторов.

CREATE TABLE joke (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  joketext TEXT,
  jokedate DATE NOT NULL,
  authorid INT
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB;

# Код для создания простой таблицы author.

CREATE TABLE author (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255),
  email VARCHAR(255)
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB;
```

Теперь осталось добавить в новую таблицу несколько имен авторов и связать их с шутками в базе данных, заполнив столбец authorid[1]. Если вы хотите попрактиковаться в использовании запросов INSERT и UPDATE, сделайте это самостоятельно. Если же вы создали базу данных с нуля, то ниже приведен набор команд INSERT, которые выполнят все за вас (chapter5/sql/2tables.sql, фрагмент).

```
# Заполняем базу данных именами авторов.
# Чтобы их можно было распознать при добавлении шуток,
# указываем идентификаторы.

INSERT INTO author SET
```

[1] Пока вам придется сделать это вручную. Однако не стоит отчаиваться: в главе 7 вы узнаете, как вставить новые записи с корректными идентификаторами с учетом всех необходимых связей автоматически, используя PHP.

```
  id = 1,
  name = 'Кевин Янк',
  email = 'thatguy@kevinyank.com';

INSERT INTO author (id, name, email)
VALUES (2, 'Джоан Смит', 'joan@example.com');

# Добавляем шутки в базу данных.

INSERT INTO joke SET
  joketext = 'Зачем цыпленок перешел дорогу? Чтобы попасть на другую
сторону!',
  jokedate = '2012-04-01',
  authorid = 1;

INSERT INTO joke (joketext, jokedate, authorid)
VALUES (
  '"Королеве - подвески, Констанции - подвязки, Портосу - подтяжки..." -
повторял, чтобы не перепутать, д\'Артаньян по дороге в Англию.',
  '2012-04-01',
  1
);

INSERT INTO joke (joketext, jokedate, authorid)
VALUES (
  'В Новый год все сбывается, даже то, что в другое время сбыть не
удается."',
  '2012-04-01',
  2
);
```

Выборка из нескольких таблиц

Скорее всего, вам покажется, что после введения двух таблиц, процесс извлечения информации из базы данных усложнился. Как вы помните, перед нами стояла задача вывести список шуток с именами авторов и адресами их электронной почты. Если бы таблица была одна, то вам понадобилось бы использовать в PHP-коде запрос всего с одной командой SELECT.

```
try
{
  $sql = 'SELECT joketext, authorname, authoremail FROM joke';
  $result = $pdo->query($sql);
}
catch (PDOException $e)
{
  $error = 'Ошибка при извлечении шуток: ' . $e->getMessage();
  include 'error.html.php';
  exit();
}
foreach ($result as $row)
{
  $jokes[] = array(
    'id' => $row['id'],
    'text' => $row['joketext'],
    'name' => $row['authorname'],
    'email' => $row['authoremail']
  );
}
```

Для базы данных с новой структурой такой метод на первый взгляд не подходит. Информация об авторах больше не хранится в таблице joke, поэтому ее для соответствующей шутки придется извлекать отдельно. В этом случае для каждой выводимой записи потребуется вызывать метод query из объекта PDO. Получится неаккуратно и медленно, а по мере увеличения объема базы данных подобные запросы существенно снизят производительность сайта.

С учетом вышесказанного может сложиться впечатление, что старый подход, несмотря на определенные недостатки, гораздо лучше. К счастью, реляционные базы данных, в том числе и MySQL, облегчают работу с информацией, хранящейся в нескольких таблицах. Новый оператор JOIN (объединение) команды SELECT разрешает воспользоваться преимуществами обоих подходов. Он позволяет обращаться с данными из разных таблиц так, будто они хранятся в одной таблице. Вот как выглядит синтаксис этого оператора в простейшем случае.

```
SELECT столбцы
FROM таблица1 INNER JOIN таблица2
  ON условие(я) связанности данных
```

В нашем примере интерес представляют столбцы id и joketext из таблицы joke, а также name и email из таблицы author. Условие связанности записей в двух таблицах заключается в том, что значение поля authorid (из joke) обязано совпадать с содержимым столбца id (из author).

Первые два запроса выводят данные из двух таблиц — здесь в объединении нет необходимости. Результаты выполнения этих запросов представлены на рис. 5.3 и 5.4 соответственно.

```
SELECT id, LEFT(joketext, 20), authorid FROM joke
```

id	LEFT(joketext, 20)	authorid
1	Зачем цыпленок переш	1
2	"Королеве - подвески	1
3	В Новый год все сбыв	2

Рис. 5.3. Содержимое таблицы joke

```
SELECT * FROM author
```

id	name	email
1	Кевин Янк	thatguy@kevinyank.com
2	Джоан Смит	joan@example.com

Рис. 5.4. Содержимое таблицы author

В третьем запросе происходит самое интересное — используется оператор JOIN. Результат выполнения данного запроса показан на рис. 5.5.

```
SELECT joke.id, LEFT(joketext, 20), name, email
```

```
FROM joke INNER JOIN author
  ON authorid = author.id
```

id	LEFT(joketext, 20)	name	email
1	Зачем цыпленок переш	Кевин Янк	thatguy@kevinyank.com
2	"Королеве - подвески	Кевин Янк	thatguy@kevinyank.com
3	В Новый год все сбыв	Джоан Смит	joan@example.com

Рис. 5.5. Результат объединяющего запроса

Объединяющий запрос группирует значения из двух таблиц в единый результирующий набор, сопоставляя связанные между собой данные. Даже несмотря на то что необходимая вам информация хранится в двух разных местах, вы все равно способны получить ее с помощью всего одного запроса к базе данных. Обратите внимание: поскольку свой столбец `id` есть и у шуток, и у авторов, то при упоминании такого столбца в запросе следует указывать имя таблицы, из которой вы хотите его извлечь. Идентификаторы таблиц `joke` и `author` записываются соответственно как `joke.id` и `author.id`. Если этого не сделать, MySQL не поймет, какой именно столбец `id` вы имеете в виду, и выдаст ошибку (рис. 5.6).

```
SELECT id, LEFT(joketext, 20), name, email
FROM joke INNER JOIN author
  ON authorid = id
```

Рис. 5.6. MySQL не терпит двусмысленности

Теперь, когда вам известен эффективный способ извлечения данных из двух таблиц, перепишите код для составления списка шуток, используя преимущества оператора `JOIN` (`chapter5/jokes/index.php`, фрагмент).

```
try
{
  $sql = 'SELECT joke.id, joketext, name, email
      FROM joke INNER JOIN author
        ON authorid = author.id';
  $result = $pdo->query($sql);
}
catch (PDOException $e)
{
  $error = 'Ошибка при извлечении шуток: ' . $e->getMessage();
  include 'error.html.php';
  exit();
}

foreach ($result as $row)
{
  $jokes[] = array(
    'id' => $row['id'],
    'text' => $row['joketext'],
    'name' => $row['name'],
    'email' => $row['email']
```

```
    );
}

include 'jokes.html.php';
```

На следующем этапе обновите шаблон таким образом, чтобы для каждой шутки в нем выводились данные об авторе (`chapter5/jokes/jokes.html.php`, фрагмент).

```
<?php foreach ($jokes as $joke): ?>
  <form action="?deletejoke" method="post">
    <blockquote>
      <p>
        <?php echo htmlspecialchars($joke['text'], ENT_QUOTES, 'UTF-8');
        ?>
        <input type="hidden" name="id" value="<?php echo $joke['id'];
        ?>">
        <input type="submit" value="Удалить">
        (автор <a href="mailto:<?php
        echo htmlspecialchars($joke['email'], ENT_QUOTES,
        'UTF-8'); ?>"><?php
        echo htmlspecialchars($joke['name'], ENT_QUOTES,
        'UTF-8'); ?></a>)
      </p>
    </blockquote>
  </form>
<?php endforeach; ?>
```

Итоговая страница будет выглядеть следующим образом (рис. 5.7).

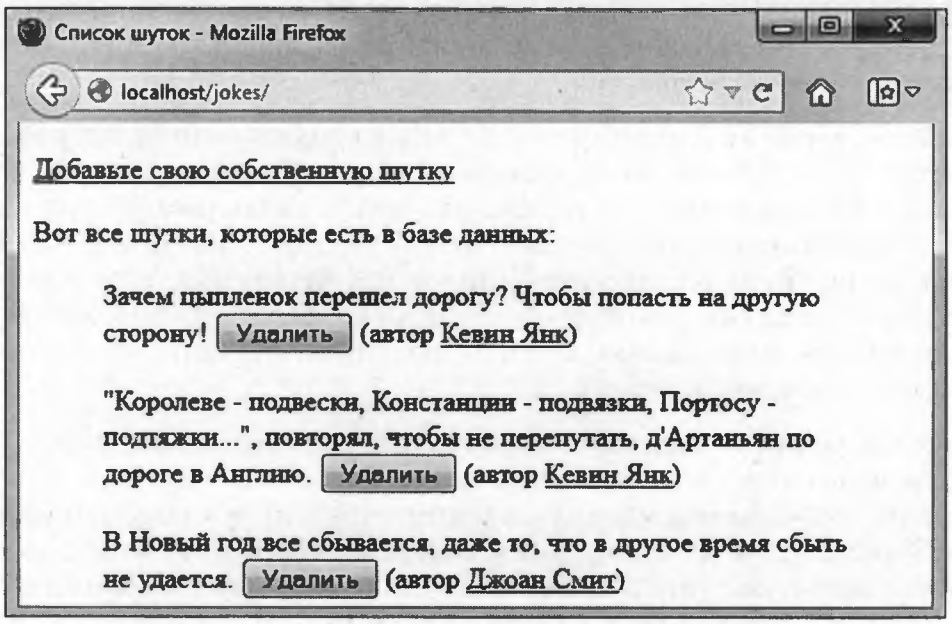


Рис. 5.7. Конечно же, лучшие шутки придуманы автором этой книги

Чем дольше вы работаете с базой данных, тем отчетливей вы должны осознавать возможности, которые открывает перед вами объединение информации из разных таблиц в единый результирующий набор. Рассмотрим еще один пример, где по запросу выводится список шуток, автор которых — Джоан Смит.

```
SELECT joketext
FROM joke INNER JOIN author
  ON authorid = author.id
WHERE name = "Джоан Смит"
```

Несмотря на то что результаты запроса (рис. 5.8) взяты только из таблицы `joke`, в коде использовался оператор `JOIN`. С его помощью выполнялся поиск шуток на основе значения, хранящегося в таблице `author`. Далее вам встретится еще немало таких остроумных примеров, но уже сейчас можно отметить, что область практического применения оператора `JOIN` широка и разнообразна. К тому же он почти всегда позволяет избежать лишней работы.

joketext
В Новый год все сбывается, даже то, что в другое в...

Рис. 5.8. Шутка, присланная Джоан

Простые связи

Структура базы данных в каждом конкретном случае обусловлена связями между теми наборами информации, которые в ней хранятся. В этом разделе вы познакомитесь с типичными видами связей и узнаете, как их лучше всего представить в рамках реляционной базы данных.

В случае простого отношения **«один к одному»** достаточно одной таблицы. Примером такой связи может служить адрес электронной почты для каждого автора в таблице `joke`. Поскольку все авторы имеют по одному адресу и у каждого адреса есть только один владелец, то выделять информацию об электронной почте в отдельную таблицу нет смысла[1].

Отношение **«многие к одному»** чуть сложнее, однако его пример вы тоже видели. Каждая шутка в базе данных связана только с одним автором, но этот же автор может добавить несколько шуток. Мы уже останавливались на проблемах, связанных с хранением таких данных в одной таблице. Напомним лишь, что это приводит к появлению дубликатов одних и тех же данных, которые занимают лишнее место и плохо синхронизируются. Если разделить данные на две таблицы и использовать для связи столбец `id` (что позволит выполнять объединяющие запросы), то все проблемы исчезнут.

Отношение **«один ко многим»** ничем не отличается от предыдущего. Это лишь иное восприятие одного и того же типа связи. Так, связь «шутки — автор» — это пример отношения «многие к одному», а «автор —шутки» — «один ко многим» (один автор способен записать много шуток). Однако, несмотря на кажущуюся простоту, все не так очевидно, как хотелось бы. В нашем примере формирование базы данных началось с библиотеки шуток («многих») и только потом для каждой из них был назначен автор («один»). Теперь представьте, что все начинается с «одного», которому следует добавить «многих».

[1] У этого правила есть исключения. Например, если таблица становится слишком объемной и содержит множество столбцов, которые редко используются в запросах с командой `SELECT`, то имеет смысл выделить такие столбцы в отдельную таблицу. Скорость выполнения запросов в уменьшенной таблице увеличится.

Допустим, вы хотите, чтобы каждый автор в нашей базе данных («один») имел несколько адресов электронной почты («много»). Когда за проектирование такой модели берется новичок, не имеющий опыта работы с базами данных, первая мысль, которая приходит ему в голову, — попытаться сохранить несколько значений в одном поле таблицы (рис. 5.9).

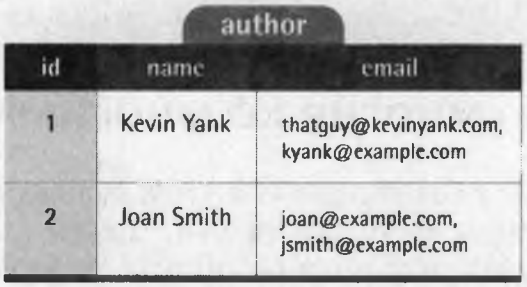


Рис. 5.9. Никогда не перегружайте поля таблицы несколькими значениями

Такой подход сработает. Однако, чтобы извлечь из базы данных отдельный адрес, вам придется разбивать полученную строку по запятым (или по другим специальным символам, используемым в качестве разделителя), что не так уж просто и отнимает много времени. А теперь представьте себе PHP-код для удаления какого-то одного адреса у конкретного автора! Кроме того, вам понадобится увеличить максимальную длину значений для столбца `email`, что приведет к бесполезной трате дискового пространства, так как у большинства авторов будет только один адрес.

Снова вернемся к тому, что отношение «один ко многим» — это всего лишь обратное восприятие отношения «многие к одному», с которым вы уже имели дело в случае с шутками и авторами. Следовательно, решение проблемы похоже: необходимо выделить новые сущности (в нашем случае это адреса электронной почты) в отдельную таблицу (рис. 5.10).

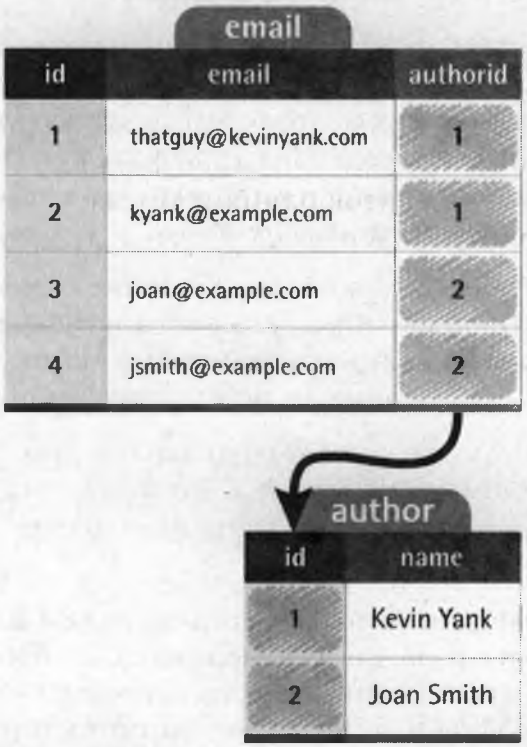


Рис. 5.10. Поле `authorid` таблицы `email` связано с каждой строкой таблицы `author`

Применив к такой структуре базы данных оператор JOIN, вы с легкостью выделите адреса, связанные с конкретным автором.

```
SELECT email
FROM author INNER JOIN email
  ON authorid = author.id
WHERE name = "Кевин Янк"
```

Отношение «многие ко многим»

Итак, у вас есть сайт с базой данных шуток, количество которых неуклонно растет. Причем темпы роста настолько высоки, что вам уже трудно с ними справляться. Посетителям сайта выводится гигантская страница с сотней совершенно несистематизированных шуток. Пора что-то менять.

Предположим, вы решили разбить шутки по следующим категориям: «О д'Артаньяне», «О переходе через дорогу», «Об адвокатах», «О лампочке» и «Политические шутки». Вспомнив упомянутый совет, вы пришли к выводу, что категории — это новые сущности, поэтому для них нужно создать новую таблицу.

```
CREATE TABLE category (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255)
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB
```

Теперь перед вами стоит непростая задача: распределить все записи по категориям. И тут выясняется, что шутки о политике могут быть связаны с переходом через дорогу, а шутки, в которых упоминается д'Артаньян, — с адвокатами. Одна и та же шутка попадает в несколько категорий, а каждая категория содержит множество шуток. Это и есть отношение **«многие ко многим»**.

Большинство неопытных разработчиков начинает искать пути, позволяющие хранить нескольких значений в одном столбце. Им кажется, что к таблице joke следует добавить поле category, а затем с его помощью выводить список идентификаторов тех категорий, в которые попадет каждая шутка. Здесь пригодится еще один совет: *если вам нужно сохранить несколько значений в одном поле, то, скорее всего, вы допустили ошибку при проектировании.*

Правильный способ представления связи «многие ко многим» заключается в использовании промежуточной таблицы. Такая таблица не содержит данных как таковых, в ней попарно перечисляются связанные записи. На рис. 5.11 показано, как могла бы выглядеть структура базы данных, если бы шутки были разбиты по категориям.

Таблица jokecategory связывает идентификаторы шуток jokeid и категорий categoryid. В данном примере шутка, которая начинается с фразы «Сколько нужно адвокатов...» принадлежит двум категориям одновременно — «Об адвокатах» и «О лампочке».

Промежуточная таблица создается аналогично любой другой. Отличие заключается в выборе первичного ключа. Во всех таблицах, с которыми вы до сих пор имели дело, был столбец id. При создании он обозначался как PRIMARY KEY (первичный ключ). В таких столбцах MySQL не позволяет хранить одинаковые значения, к тому же для них повышается скорость выполнения объединяющих запросов.

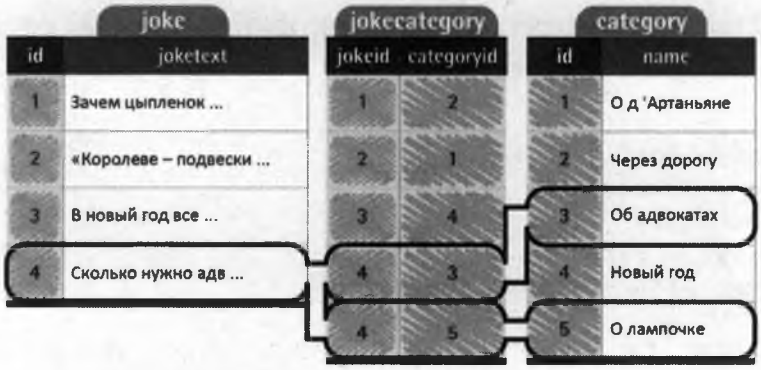


Рис. 5.11. Таблица `jokecategory` попарно связывает строки из таблиц `joke` и `category`

У промежуточной таблицы такого столбца с уникальными значениями нет. Каждый идентификатор шутки может быть указан более одного раза. То же самое касается идентификаторов категорий — в одной категории может быть много шуток. Избегать следует повторяющихся пар. Кроме того, задача данной таблицы — упростить объединяющие запросы, поэтому прирост производительности, который обеспечивается первичным ключом, здесь не помешает. В связи с этим при создании первичного ключа для промежуточных таблиц указывается несколько столбцов.

```
CREATE TABLE jokecategory (
  jokeid INT NOT NULL,
  categoryid INT NOT NULL,
  PRIMARY KEY (jokeid, categoryid)
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB
```

Данный запрос создает таблицу, в которой первичный ключ формируют два столбца — `jokeid` и `categoryid`. Это обеспечивает необходимую уникальность промежуточной таблицы, не дает присвоить шутке одну и ту же категорию более одного раза и повышает скорость объединяющих запросов, в которых данная таблица используется[1].

Теперь, когда промежуточная таблица готова и содержит назначенные категории, вы можете использовать оператор `JOIN`, чтобы сформулировать несколько интересных и практичных запросов, например вывести список всех шуток в категории «О д'Артаньяне».

```
SELECT joketext
FROM joke INNER JOIN jokecategory
  ON joke.id = jokeid
INNER JOIN category
  ON categoryid = category.id
WHERE name = "о д'Артаньяне"
```

Как видите, в этом запросе используются два оператора `JOIN`. Первый объединяет таблицы `joke` и `jokecategory`, а второй берет готовые данные и объединяет их с таблицей `category`. По мере того как структура вашей базы данных будет усложняться, подобные запросы с несколькими объединениями станут нормой.

[1] Если вы хотите получить вышеописанную структуру и создать таблицу `jokecategory` с нуля, используйте команды `CREATE TABLE` и `INSERT` (то же касается и других таблиц, включая записи с шутками).

Следующий запрос выводит список категорий, в которых содержатся шутки, начинающиеся словами «сколько адвокатов».

```
SELECT name
FROM joke INNER JOIN jokecategory
  ON joke.id = jokeid
INNER JOIN category
  ON categoryid = category.id
WHERE joketext LIKE «Сколько адвокатов%»
```

Еще один запрос, где используется таблица `author` для объединения четырех других таблиц, перечисляет имена всех авторов, которые добавили шутки о д'Артаньяне.

```
SELECT author.name
FROM joke INNER JOIN author
  ON authorid = author.id
INNER JOIN jokecategory
  ON joke.id = jokeid
INNER JOIN category
  ON categoryid = category.id
WHERE category.name = «о д'Артаньяне»
```

Один за многих, и многие за одного

В этой главе вы ознакомились с основными принципами правильного проектирования баз данных. Узнали, каким образом MySQL и любые другие системы по управлению реляционными базами данных связывают разного рода сущности. Начав с понимания простых отношений вроде «один к одному», вы перешли к более сложным отношениям «многие к одному», «один ко многим» и «многие ко многим».

В ходе чтения данной главы вы также изучили несколько новых возможностей популярных SQL-команд. В частности, вы узнали, как использовать запрос `SELECT` для объединения данных из разных таблиц в единый результирующий набор.

Теперь, получив в руки всю мощь, которой обладают базы данных, состоящие из множества таблиц, вы способны улучшить свой простенький сайт со списком шуток из главы 4 и добавить в него учет авторов и категорий. Именно этому будет посвящена глава 7. Но прежде следует уделить время совершенствованию навыков в PHP. Следующая глава познакомит вас с некоторыми неочевидными моментами программирования. Эти знания пригодятся вам в дальнейшем для создания расширенной версии сайта с шутками.

Глава 6
СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ

Прежде чем погрузиться с головой в процесс совершенствования базы данных с шутками, уделим немного времени оттачиванию PHP-мастерства. В частности, рассмотрим несколько способов улучшения структуры кода.

Методики структурного программирования применимы к любым PHP-проектам, кроме разве что самых простых. В главе 3 вы уже научились разделять код на несколько файлов: контроллер и набор связанных с ним шаблонов. Это позволило отделить серверную бизнес-логику сайта от HTML-кода, который генерирует динамические страницы. Так вы освоили использование команды `include`.

Язык PHP обладает множеством средств, облегчающих структурирование кода. Наиболее мощное из них, несомненно, поддержка объектно-ориентированного программирования, с которым вы поверхностно ознакомились в главе 4. Однако, чтобы создать сложное (и хорошо структурированное) приложение на PHP[1], совершенно необязательно знать все тонкости ООП. Существует возможность структурировать код за счет более простых механизмов.

В этой главе вы рассмотрите несколько простых способов, позволяющих сделать код хорошо управляемым и поддерживаемым. И для этого вам абсолютно не понадобится становиться асом программирования (хотя вы, без сомнения, к этому стремитесь).

Подключаемые файлы

В коде даже самых простых сайтов, написанных на PHP, часто требуется использовать один и тот же набор команд в разных местах. Вы уже научились работать с командой `include` при загрузке шаблонов внутрь контроллера. Оказывается, она также позволяет избежать многократного повторения одних и тех же фрагментов кода.

Подключаемые файлы (или **включения**) содержат фрагменты кода, которые доступны для загрузки другими PHP-скриптами, благодаря чему их не нужно набирать заново.

Подключение HTML-кода

Концепция подключения файлов появилась задолго до PHP. Если вы уже давно не молоды, как и автор этой книги (а в мире веб-технологий это значит,

[1] Пожалуй, самое популярное сегодня PHP-приложение WordPress написано без использования ООП.

что вам больше 25 лет), то вы, скорее всего, знакомы с технологией Server Side Includes (SSI) — **включения на стороне сервера**. Поддерживаемая практически любым веб-сервером, SSI позволяет группировать часто повторяемые фрагменты HTML, JavaScript и CSS в отдельные файлы, которые затем используются на разных страницах.

В PHP подключаемые файлы чаще всего содержат чистый PHP-код или, если речь идет о шаблонах, смесь PHP и HTML. Обратите внимание, что хранить в таких файлах код на языке PHP *не обязательно*. При желании вы можете поместить туда сугубо статический фрагмент HTML. Такой способ хорошо подходит для выделения общих элементов дизайна, которые часто используются на сайте. Например, это могут быть сведения об авторских правах, располагаемые внизу каждой страницы (`chapter6/static-footer/footer.inc.html.php`).

```
<div id="footer">
  Содержимое этой страницы принадлежит &copy; 1998&ndash;2012
  ООО «Рога и копыта». Все права защищены.
</div>
```

Такой код называется **фрагментом шаблона**. Он представляет собой подключаемый файл, который используется внутри шаблонов. Чтобы выделить файлы данного типа, для них рекомендуется указывать специальное расширение **.inc.html.php**.

В дальнейшем такие фрагменты включаются в шаблоны (chapter6/static-footer/samplepage.html.php).

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Пример страницы</title>
  </head>
  <body>
    <p id="main">
      Эта страница использует статические включения для вывода
      внизу стандартного предупреждения об авторских правах.
    </p>
    <?php include 'footer.inc.html.php'; ?>
  </body>
</html>
```

А вот как выглядит контроллер, загружающий такой шаблон (`chapter6/static-footer/index.php`).

```
<?php
include 'samplepage.html.php';
?>
```

При загрузке в браузер страница примет следующий вид (рис. 6.1).

Чтобы обновить сообщение об авторских правах на всем сайте, вам понадобится отредактировать только файл **footer.inc.html.php**. И никаких больше операций поиска или замены, отнимающих время и чреватых ошибками!

Если вы хотите сделать свою жизнь еще проще, переложите всю работу на PHP (`chapter6/dynamic-footer/footer.inc.html.php`).

```
<div id="footer">
  Содержимое этой веб-страницы принадлежит &copy;
  1998&ndash; <?php echo date('Y'); ?> ООО «Рога и копыта».
  Все права защищены.
</div>
```

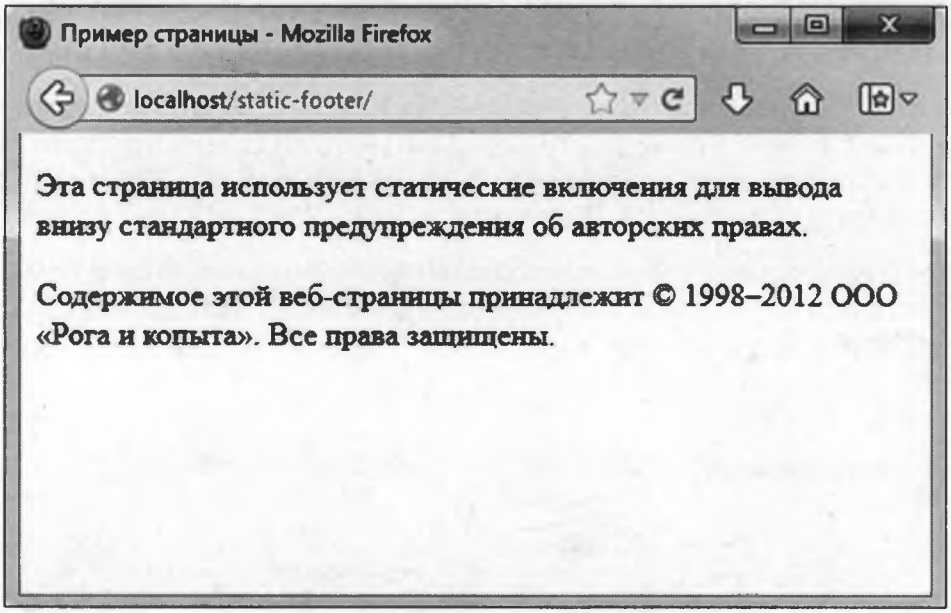


Рис. 6.1. Статическое включение выводит информацию об авторских правах

Подключение PHP-кода

Когда речь заходит о сайтах с поддержкой баз данных, то первое, что должен сделать практически любой контроллер, — установить соединение с этой базой данных. Как вы уже знаете, такое действие описывается довольно значительным фрагментом кода.

```
try
{
  $pdo = new PDO('mysql:host=localhost;dbname=ijdb', 'ijdbuser',
    'mypassword');
  $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $pdo->exec('SET NAMES "utf8"');
}
catch (PDOException $e)
{
  $error = 'Не удалось подключиться к серверу баз данных.';
  include 'error.html.php';
  exit();
}
```

Конечно, 12 строк не так и много, но если их набирать в начале каждого контроллера, то это быстро надоест. Многие начинающие PHP-разработчики перенабирают код и часто невнимательно, пропуская некоторые выражения (например, `try` или `catch`). В дальнейшем на поиск ошибок уходит много времени. Другие

интенсивно используют буфер обмена, копируя фрагменты кода из старых скриптов. Третьи сохраняют отдельные участки кода для последующего многократного использования с помощью текстового редактора.

А теперь представьте, что изменился пароль к базе данных или какие-то другие фрагменты кода. Вам придется пересматривать все файлы сайта, которые содержат соответствующий код, чтобы внести необходимые изменения. Работа становится еще более невеселой, если у фрагмента существует несколько версий.

На рис. 6.2 показано, как подключаемые файлы помогают в решении подобной проблемы. Вместо того чтобы повторять фрагмент кода во всех скриптах, где он необходим, вы записываете его один раз в отдельном файле, а затем при необходимости подключаете в любом PHP-скрипте.

Давайте воспользуемся этим подходом в нашем примере со списком шуток для подключения к базе данных и посмотрим, как он работает. Для начала создайте файл с именем db.inc.php[1] и разместите внутри него код для соединения с базой данных (`chapter6/jokes/db.inc.php`).

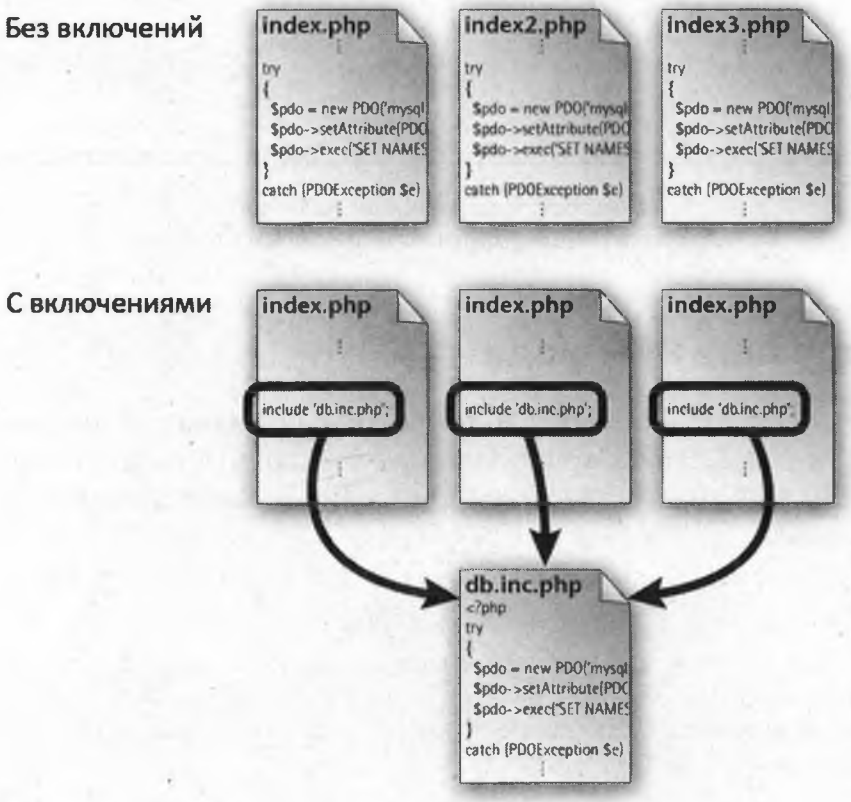


Рис. 6.2. Подключаемые файлы позволяют нескольким скриптам иметь общий код

[1] Текущее соглашение об именах подключаемых файлов (с использованием расширения .inc.php) позволяет легко выделить их среди обычных PHP-скриптов. Оно также гарантирует, что подключаемые файлы будут восприняты как файлы с PHP-кодом и обработаны веб-сервером и средствами разработки, которые вы используете. На практике вы вольны называть такие файлы как угодно. Когда-то широко использовалось расширение .inc. Однако, если веб-сервер не был настроен воспринимать подобные включения с PHP-кодом, это создавало угрозу безопасности. Пользователи, которые догадывались, что за таким именем скрывается подключаемый файл, могли загрузить его в виде обычного текста и получить доступ к конфиденциальной информации, которая в нем находилась, например узнать пароль к базе данных.

```
<?php
try
{
  $pdo = new PDO('mysql:host=localhost;dbname=ijdb', 'ijdbuser',
    'mypassword');
  $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $pdo->exec('SET NAMES "utf8"');
}
catch (PDOException $e)
{
  $error = 'Не удалось подключиться к серверу баз данных.';
  include 'error.html.php';
  exit();
}
```

Как видите, включения — это обычные PHP-файлы. Разница лишь в том, что тот фрагмент кода, который в них содержится, как правило, имеет смысл только в контексте другого, более объемного скрипта. Теперь включите файл db.inc.php в свой контроллер (`chapter6/jokes/index.php`).

```
<?php
if (get_magic_quotes_gpc())
{
  $process = array(&$_GET, &$_POST, &$_COOKIE, &$_REQUEST);
  while (list($key, $val) = each($process))
  {
    foreach ($val as $k => $v)
    {
      unset($process[$key][$k]);
      if (is_array($v))
      {
        $process[$key][stripslashes($k)] = $v;
        $process[] = &$process[$key][stripslashes($k)];
      }
      else
      {
        $process[$key][stripslashes($k)] = stripslashes($v);
      }
    }
  }
  unset($process);
}

if (isset($_GET['addjoke']))
{
  include 'form.html.php';
  exit();
}

if (isset($_POST['joketext']))
{
  include 'db.inc.php';

  try
```

```
    {
      $sql = 'INSERT INTO joke SET
          joketext = :joketext,
          jokedate = CURDATE()';
      $s = $pdo->prepare($sql);
      $s->bindValue(':joketext', $_POST['joketext']);
      $s->execute();
    }
    catch (PDOException $e)
    {
      $error = 'Ошибка при добавлении шутки: ' . $e->getMessage();
      include 'error.html.php';
      exit();
    }

    header('Location: .');
    exit();
}

if (isset($_GET['deletejoke']))
{
    include 'db.inc.php';

    try
    {
      $sql = 'DELETE FROM joke WHERE id = :id';
      $s = $pdo->prepare($sql);
      $s->bindValue(':id', $_POST['id']);
      $s->execute();
    }
    catch (PDOException $e)
    {
      $error = 'Ошибка при удалении шутки: ' . $e->getMessage();
      include 'error.html.php';
      exit();
    }

    header('Location: .');
    exit();
}

include 'db.inc.php';

try
{
    $sql = 'SELECT joke.id, joketext, name, email
        FROM joke INNER JOIN author
          ON authorid = author.id';
    $result = $pdo->query($sql);
}
catch (PDOException $e)
{
    $error = 'Ошибка при извлечении шуток: ' . $e->getMessage();
    include 'error.html.php';
    exit();
```

```
}

foreach ($result as $row)
{
  $jokes[] = array(
    'id' => $row['id'],
    'text' => $row['joketext'],
    'name' => $row['name'],
    'email' => $row['email']
  );
}

include 'jokes.html.php';
```

Каждый раз, когда вашему контроллеру необходимо соединиться с базой данных, он подключает файл db.inc.php с помощью команды `include`. Такое подключение занимает всего одну строку и делает код более легким для восприятия — перед каждым SQL-запросом в контроллере используется отдельное выражение `include`. Ранее соединение с базой данных устанавливалось в начале файла вне зависимости от того, использовалось ли оно после этого или нет.

Встретив команду `include`, интерпретатор приостанавливает текущий код и выполняет указанный PHP-скрипт. Затем возвращается к начальному скрипту и продолжает работу с того места, на котором остановился.

Подключаемые файлы — наиболее удобное средство структурирования кода в PHP. Именно поэтому они широко применяются. Даже в самом простом веб-приложении включения могут оказаться очень полезными.

Виды включений

Команда `include`, которую вы активно использовали, — это лишь одно из нескольких выражений, для подключения внешних PHP-файлов в текущем скрипте. К их числу относятся также `require`, `include_once` и `require_once`.

Команды `include` и `require` почти идентичны. Они лишь по-разному реагируют на ситуацию, когда указанный файл подключить невозможно: либо он не существует, либо у веб-сервера нет полномочий на его чтение. В таком случае `include` выводит предупреждение, при этом скрипт продолжает свою работу, а `require` отображает сообщение об ошибке, и скрипт останавливается[1].

Как правило, когда главный скрипт не способен продолжить работу без подключаемого файла, лучше использовать команду `require`. Однако по возможности старайтесь применять `include`. Если, например, файл db.inc.php не загрузится, скрипт сможет продолжить генерирование главной страницы. И пусть пользователь не увидит содержимого базы данных, но у него появится возможность перейти по ссылке Свяжитесь с нами внизу страницы и сообщить о проблеме.

[1] Как правило, предупреждения и сообщения об ошибке в файле php.ini отключены. Если что-то при выполнении кода пойдет не так, то в случае использования команды `include` вы ничего не заметите — на странице не будет хватать лишь той части содержимого, которая генерируется подключаемым файлом. При работе команды `require` создание страницы остановится на том месте, где произошла ошибка.

Команды `include_once` и `require_once` работают аналогично своим прототипам `include` и `require`. Разница заключается в том, что, если указанный в первых двух выражениях файл уже хотя бы один раз был подключен в ходе текущего запроса к странице (с использованием любой из четырех команд), выражения игнорируются. Такой подход пригодится при подключении файлов, выполняющих одноразовые задачи, например соединение с базой данных.

Принцип работы команды `include_once` показан на рис. 6.3. Скрипт **index.php** подключает два файла — **categories.inc.php** и **top10.inc.php**. Обоим файлам необходимо соединение с базой данных, поэтому команда `include_once` для подключения **db.inc.php** используется в каждом из них. PHP проигнорирует попытку подключения **db.inc.php** в скрипте **top10.inc.php**, поскольку данный файл уже был подключен внутри **categories.inc.php**. Таким образом, соединение с базой данных установится только один раз.

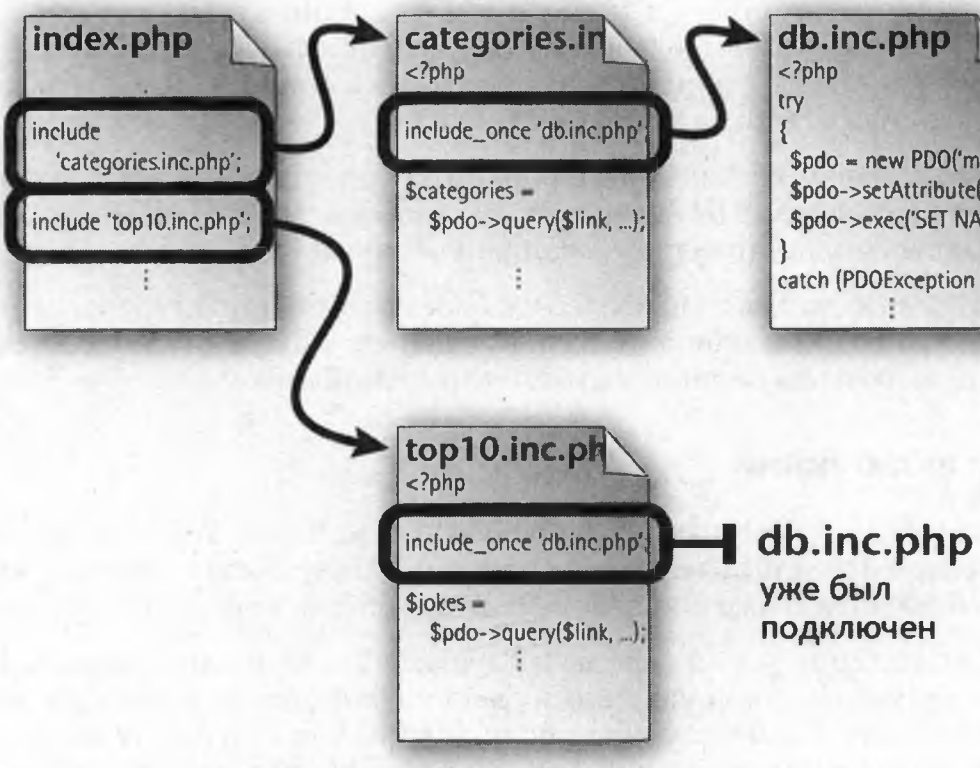


Рис. 6.3. Команда `include_once` позволяет избежать повторного соединения с базой данных

Команды `include_once` и `require_once` подходят также для загрузки библиотек с функциями (см. раздел «Нестандартные функции и библиотеки функций» этой главы).

Разделение подключаемых файлов

Во всех рассмотренных примерах предполагалось, что скрипты и подключаемые к ним файлы находятся в одной и той же директории веб-сервера. Однако так бывает не всегда. Часто возникает необходимость использовать подключаемые файлы в скриптах, которые разбросаны по дереву каталогов со сложной структу-

рой. На роль общего подключаемого файла отлично подойдет файл **db.inc.php** для соединения с базой данных.

Вопрос в том, как PHP-скрипт найдет подключаемый файл, если он находится в *другой* директории? Наиболее очевидное решение проблемы — указать местонахождение файла в виде **абсолютного пути**. Вот как это могло бы выглядеть на сервере под управлением Windows[1].

```
<?php include 'C:/Program Files/Apache Software Foundation/Apache2.2/
htdocs/includes/db.inc.php'; ?>
```

А это код для Linux-сервера.

```
<?php include '/usr/local/apache2/htdocs/includes/db.inc.php'; ?>
```

Несмотря на то что такой подход сработает, использовать его нежелательно, поскольку он привязывает код сайта к конфигурации вашего веб-сервера. В идеале должна сохраняться возможность запускать PHP-приложение на любом веб-сервере, поддерживающем этот язык программирования. Очень часто сайт, разработанный на одном сервере, может в дальнейшем функционировать на другом, и было бы непрактично ссылаться в коде на конкретные директории. И даже если вам посчастливилось использовать в своей работе всего один компьютер, вы сильно пожалеете, когда придется переместить свой сайт в другой раздел или каталог.

Гораздо разумнее в этом случае узнать с помощью PHP **корневую директорию** веб-сервера, а все остальные местоположения указывать относительно ее. Корневой директорией на сервере называется та, которая соотносится с корневой директорией вашего сайта. Например, чтобы файл **index.php** стал доступен по адресу **http://www.example.com/index.php**, нужно разместить его в корневой директории веб-сервера `www.example.com`. Путь к этой директории вы можете получить, используя переменную `$_SERVER['DOCUMENT_ROOT']` в любом PHP-скрипте. Как уже упоминалось в главе 4, переменная `$_SERVER` — это массив, который создается интерпретатором автоматически (подобно `$_GET`, `$_POST` и `$_REQUEST`). `$_SERVER` содержит весь набор информации, предоставляемой сервером, в том числе и `$_SERVER['DOCUMENT_ROOT']`.

Приведенный ниже пример сработает на любом из серверов Windows, Mac или Linux, где установлен Apache или IIS[2].

```
<?php include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php'; ?>
```

---

[1] При написании пути лучше использовать обычный слеш (/), даже если вы имеете дело с Windows-сервером. PHPспособен осуществить автоматическое преобразование, и вам не придется выполнять экранирование, используя два обратных слеша внутри строк (\\).

[2] На `$_SERVER['DOCUMENT_ROOT']` не стоит полагаться только в том случае, если PHP на сервере работает при поддержке Common Gateway Interface (CGI) — общего интерфейса шлюза. Спецификация CGI не требует, чтобы веб-сервер сообщал интерпретатору PHP о корневой директории сайта, поэтому данное значение там отсутствует. В реальных условиях CGI-версии PHP лучше избегать. К счастью, они встречаются все реже. Если вы следовали инструкциям по установке PHP, которые приводились в этой книге, то будьте уверены, что переменная `$_SERVER['DOCUMENT_ROOT']` сработает.

Еще один подходящий вариант для общего подключаемого файла — фрагмент кода, с помощью которого нивелируется действие волшебных кавычек (см. главу 4). Просто поместите нижеприведенный код в отдельный файл (`chapter6/includes/magicquotes.inc.php`).

```
<?php
if (get_magic_quotes_gpc())
{
  $process = array(&$_GET, &$_POST, &$_COOKIE, &$_REQUEST);
  while (list($key, $val) = each($process))
  {
    foreach ($val as $k => $v)
    {
      unset($process[$key][$k]);
      if (is_array($v))
      {
        $process[$key][stripslashes($k)] = $v;
        $process[] = &$process[$key][stripslashes($k)];
      }
      else
      {
        $process[$key][stripslashes($k)] = stripslashes($v);
      }
    }
  }
  unset($process);
}
```

Начиная с этого момента вы свели на нет действие волшебных кавычек с помощью одной строки в скрипте контроллера.

```
<?php
include $_SERVER['DOCUMENT_ROOT'] . '/includes/magicquotes.inc.php';
```

В оставшейся части книги оба включения (соединение с базой данных и избавление от волшебных кавычек) используются во многих примерах. Чтобы успешно применять тот и другой код, сохраните оба файла (db.inc.php и magicquotes.inc.php) в каталоге includes, расположенном в корневой директории вашего веб-сервера.

Нестандартные функции и библиотеки функций

До сих пор вы работали с готовыми функциями. Как правило, они нуждались в одном или нескольких **аргументах**, а в качестве результата **возвращали какое-то значение**. В PHP имеется обширная библиотека таких встроенных функций. Используя их в своем скрипте, вы можете выполнить практически любые действия — от получения текущей даты (`date`) до генерирования изображений налету (`imagecreatetruecolor`, http://www.php.net/imagecreatetruecolor).

Однако PHP предоставляет вам еще одну уникальную возможность, о которой вы пока не догадываетесь, — определять собственные **нестандартные функции**. Они работают точно так же, как и встроенные, и могут выполнять все, на что способен обычный PHP-скрипт.

Для начала возьмем простейший пример. Допустим, вам необходимо с помощью скрипта вычислить площадь прямоугольника, зная его ширину (3) и высоту (5). Из курса геометрии вы помните, что площадь прямоугольника вычисляется путем умножения обеих сторон.

```
$area = 3 * 5;
```

Вместо использования конкретной формулы лучше взять функцию `area`, позволяющую вычислять площадь прямоугольника по заданным размерам (`chapter6/calculate-area/index.php`, фрагмент).

```
$area = area(3, 5);
```

Так уж получилось, что в PHP встроенной функции `area` нет, однако ничто не мешает нам ее создать (`chapter6/calculate-area/area-function.inc.php`).

```
<?php
function area($width, $height)
{
  return $width * $height;
}
```

Вышеприведенный подключаемый файл описывает одну нестандартную функцию `area`. Скорее всего, единственная знакомая для вас строка в этом коде — `<?php`. То, с чем вы столкнулись, называется **объявлением функции**. Пройдемся по коду строка за строкой.

```
function area($width, $height)
```

Ключевое слово `function` сообщает PHP о том, что вы намереваетесь объявить новую функцию, которая будет использована в текущем скрипте. Далее следует имя функции (в данном случае `area`). Оно подчиняется тем же правилам, что и имя переменной: имя чувствительно к регистру, должно начинаться с буквы или символа подчеркивания (_) и может включать цифры, буквы и символы подчеркивания. Отличие заключается в том, что для функций нельзя использовать префикс в виде знака доллара. После имени функции всегда идут круглые скобки, содержимое которых иногда может оставаться пустым.

В скобках, следующих за именем функции, записываются аргументы, которые она принимает. Вам должно быть это знакомо по работе со встроенными функциями. Например, при использовании `date` для получения текущей даты вы указывали в скобках строку, описывающую формат вывода даты.

При объявлении собственной функции вместо значений аргументов указывают имена переменных. В нашем примере их две — `$width` и `$height`. В дальнейшем при вызове функция будет ожидать получения двух аргументов, присвоенных этим переменным, чтобы произвести с их помощью необходимые вычисления.

```
{
```

В оставшейся части объявления находится код, который выполняет арифметические операции или другие необходимые действия. Целиком код располагается внутри фигурных скобок (`{...}`). Эта строка содержит лишь первую из них — открывающую.

```
return $width * $height;
```

Код внутри фигурных скобок — это своеобразный миниатюрный PHP-скрипт. В нашем случае функция очень простая и содержит всего одно выражение — `return`.

Команда `return` позволяет мгновенно перейти из функции в главный скрипт. Встречая это выражение, интерпретатор прекращает выполнение кода текущей функции и возвращается в то место, откуда произведен вызов. Это своего рода катапультируемое кресло для функций.

Помимо выхода из функции, с помощью команды `return` можно указать значение, возвращаемое в код, который выполнил вызов. В данном случае возвращается `$width * $height` — результат умножения двух параметров.

```
}
```

Закрывающая скобка обозначает конец объявления функции.

Чтобы воспользоваться придуманной нами функцией, необходимо подключить файл, в котором она объявляется (`chapter6/calculate-area/index.php`).

```
<?php
include_once 'area-function.inc.php';
$area = area(3, 5);
include 'output.html.php';
```

Формально можно объявить функцию прямо внутри контроллера, но, вынося ее в отдельный подключаемый файл, вы значительно упрощаете ее повторное использование в других скриптах. К тому же код станет более аккуратным. Для подключения файла с функцией используется команда `include_once` (или `require_once`, если функция для скрипта очень важна).

Избегайте подключения файлов с функциями с помощью команд `include` или `require`. Как уже говорилось в подразделе «Виды включений», это чревато многократным объявлением одних и тех же функций из списка. В результате на экране пользователя возникнут предупреждения от интерпретатора.

Стандартная (но необязательная) практика — подключать все файлы с функциями в начале кода, чтобы сразу видеть, какие функции используются в конкретном скрипте.

Рассмотренный пример — это всего лишь основа **функциональной библиотеки**, подключаемый файл, который содержит набор связанных между собою функций. При желании вы можете переименовать его в `geometry.inc.php` и добавить туда структурированный код для выполнения различных расчетов по геометрии.

Принципиальное отличие между нестандартными функциями и подключаемыми файлами сводится к такому понятию, как **область видимости переменных**. Любая переменная в главном скрипте доступна для включения и может изменяться внутри него. Иногда это удобно, но чаще вызывает лишнюю головную боль.

Непреднамеренная перезапись одной из переменных главного скрипта внутри подключаемого файла часто приводит к возникновению ошибок, отслеживание и устранение которых способно отнять много времени. Чтобы избежать таких неприятностей, необходимо запоминать имена переменных в текущем скрипте и во всех используемых внутри него включениях.

Функции лишены подобных проблем. Переменные, созданные в рамках функции (в том числе любые аргументы), существуют только внутри нее и исчезают после того, как функция завершила свою работу. Принято говорить, что областью видимости таких переменных является функция, то есть они находятся в **области видимости функции**. Переменные, созданные внутри главного скрипта за пределами функций, этим функциям недоступны. Их область видимости — главный скрипт, поэтому говорят, что они находятся в **глобальной области видимости**.

Отбросив причудливые названия, разберемся, что все это значит. Если в главном скрипте есть переменная с именем `$width` и внутри функции также находится переменная `$width`, то PHP станет обращаться с ними как с отдельными сущностями. Что еще интереснее, две разные функции с одинаковыми именами переменных никак не повлияют друг на друга, потому что находятся в отдельных областях видимости.

В некоторых ситуациях переменная из глобальной области видимости — **глобальная переменная** — может понадобиться внутри функции. Например, файл db.inc.php создает соединение с базой данных, используемое скриптом и сохраняемое в глобальной переменной `$pdo`, и возникает необходимость использовать эту переменную, чтобы получить доступ к базе данных.

Если проигнорировать область видимости, то функцию можно записать следующим образом.

```
<?php
include_once $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

function totalJokes()
{
  try
  {
    $result = $pdo->query('SELECT COUNT(*) FROM joke');
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка базы данных при подсчете шуток!';
    include 'error.html.php';
    exit();
  }

  $row = $result->fetch();
  return $row[0];
}
```

ОБЩЕЕ ВКЛЮЧЕНИЕ ДЛЯ РАБОТЫ С БАЗОЙ ДАННЫХ

Обратите внимание, что в первой строке контроллера используется общая копия файла db.inc.php из директории includes, о которой говорилось в подразделе «Разделение подключаемых файлов». Убедитесь, что вы скопировали данный файл (и связанный с ним шаблон error.html.php для вывода ошибок) в каталог includes корневой директории веб-сервера. В противном случае PHP пожалуется на то, что не смог обнаружить файл db.inc.php.

Проблема в том, что глобальная переменная `$pdo` (выделенная в коде полужирным начертанием) недоступна в области видимости функции. Если вы попытаетесь вызвать функцию в том виде, в каком она записана сейчас, то получите сообщение об ошибках (рис. 6.4).

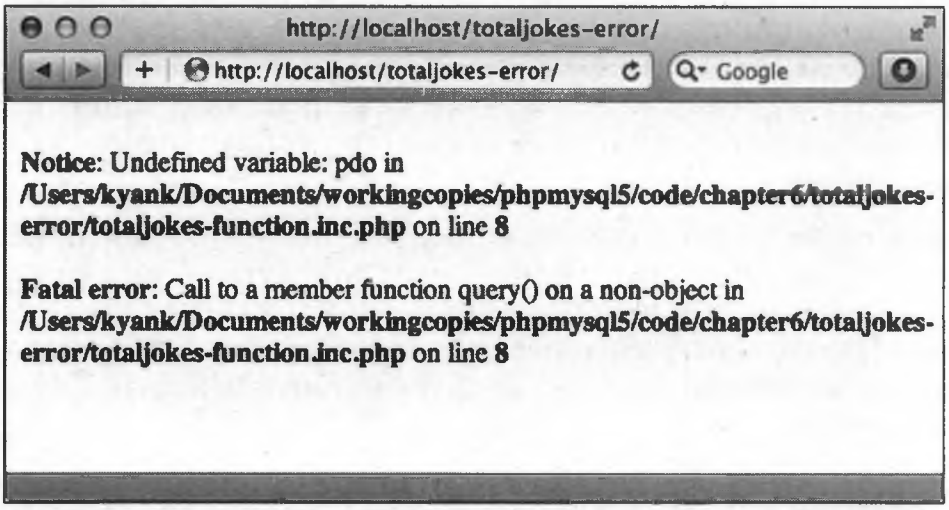


Рис. 6.4. У функции `totaljokes` нет доступа к переменной `$pdo`

Как вариант, можно добавить в функцию `totaljokes` новый аргумент и передать ей таким образом значение `$pdo`. Однако поступать так с каждой функцией, которой нужен доступ к базе данных, довольно утомительно.

Вместо этого воспользуемся глобальной переменной напрямую из функции, применив один из двух способов. Первый заключается в **импортировании** глобальной переменной в область видимости функции (`chapter6/totaljokes-global1/totaljokes-function.inc.php`).

```
<?php
include_once $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

function totalJokes()
{
  global $pdo;

  try
  {
    $result = $pdo->query('SELECT COUNT(*) FROM joke');
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка базы данных при подсчете шуток!';
    include 'error.html.php';
```

```
    exit();
  }

  $row = $result->fetch();
  return $row[0];
}
```

Выражение `global`, выделенное полужирным начертанием, позволяет задать список глобальных переменных, к которым вы хотели бы получить доступ внутри функции (если переменных несколько, они разделяются запятыми). Это называется импортированием переменных. Оно отличается от передачи значений в качестве аргумента, поскольку изменение импортированной переменной внутри функции приведет к ее изменению на глобальном уровне.

Еще один способ импортировать переменную — использовать массив `$GLOBALS` (`chapter6/totaljokes-global2/totaljokes-function.inc.php`).

```
<?php
include_once $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

function totalJokes()
{
  try
  {
    $result = $GLOBALS['pdo']->query('SELECT COUNT(*) FROM joke');
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка базы данных при подсчете шуток!';
    include 'error.html.php';
    exit();
  }

  $row = $result->fetch();
  return $row[0];
}
```

Как видите, мы всего лишь заменили `$pdo` выражением `$GLOBALS['pdo']`. Специальный массив `$GLOBALS` содержит элементы для каждой глобальной переменной и доступен во всех областях видимости, поэтому его называют **суперглобальным**. Используя его, вы получите доступ к любому глобальному значению с помощью записи `$GLOBALS['`*name*`']`, где *name* — имя глобальной переменной (без знака доллара). Преимущество данного подхода в том, что при необходимости вы все еще можете создать на уровне функции отдельную переменную с именем `$pdo`.

В PHP есть и другие специальные массивы, которые являются суперглобальными и доступны внутри функций: `$_SERVER`, `$_GET`, `$_POST`, `$_COOKIE`, `$_FILES`, `$_ENV`, `$_REQUEST` и `$_SESSION`. Подробнее о суперглобальных массивах рассказано в руководстве по PHP (http://php.net/manual/en/language.variables.superglobals.php).

Вспомогательные функции для шаблонов

Завершим изучение этой главы созданием действительно полезной функциональной библиотеки.

В языке PHP существует не так уж много функций, которые столь неудобны в использовании, как `htmlspecialchars`. В главе 3 мы уже говорили о том, что функцию `htmlspecialchars` следует вызывать каждый раз, когда необходимо вывести текст, переданный пользователем. Это помогает защитить сайт от хакеров, способных внедрить вредоносный код на страницу.

Возьмем для примера код, используемый для вывода списка шуток, добавленных посетителями (`chapter6/jokes/jokes.html.php`, фрагмент).

```
<?php echo htmlspecialchars($joke['text'], ENT_QUOTES, 'UTF-8'); ?>
```

Кроме того, что у функции `htmlspecialchars` необычайно длинное имя, она еще принимает три аргумента, два из которых остаются неизменными всегда.

Поскольку вывод текста в виде HTML — обычная задача для шаблонов с PHP-кодом, запишем это действие в виде функции (`chapter6/includes/helpers.inc.php`, фрагмент).

```
<?php
function html($text)
{
  return htmlspecialchars($text, ENT_QUOTES, 'UTF-8');
}
```

Данная запись позволяет вызвать `htmlspecialchars`, используя более короткий код.

```
<?php echo html($joke['text']); ?>
```

Пойдем дальше и напишем вторую функцию `htmlout`, которая принимает значение от нашей первой функции и выводит его на страницу (`chapter6/includes/helpers.inc.php`).

```
<?php
function html($text)
{
  return htmlspecialchars($text, ENT_QUOTES, 'UTF-8');
}

function htmlout($text)
{
  echo html($text);
}
```

Такие компактные и удобные функции называют **вспомогательными**, поскольку они упрощают написание шаблонов. Вот как теперь выглядит шаблон для вывода списка шуток с использованием созданных функций (`chapter6/jokes-helpers/jokes.html.php`).

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
```

```
    <title>Список шуток</title>
  </head>
  <body>
    <p><a href="?addjoke">Добавьте свою собственную шутку</a></p>
    <p>Вот все шутки, которые есть в базе данных:</p>
    <?php foreach ($jokes as $joke): ?>
      <form action="?deletejoke" method="post">
        <blockquote>
          <p>
            <?php htmlout($joke['text']); ?>
            <input type="hidden" name="id" value="<?php echo $joke['id'];
            ?>">
            <input type="submit" value="Удалить">
            (автор <a href="mailto:<?php htmlout($joke['email']); ?>">
              <?php htmlout($joke['name']); ?></a>)
          </p>
        </blockquote>
      </form>
    <?php endforeach; ?>
  </body>
</html>
```

ВСПОМОГАТЕЛЬНЫМ ФУНКЦИЯМ МЕСТО В ОБЩЕЙ ДИРЕКТОРИИ INCLUDES

По аналогии с db.inc.php и magicquotes.inc.php файл helpers.inc.php должен находиться в каталоге includes корневой директории вашего сервера (см. подраздел «Разделение подключаемых файлов» этой главы).

По мере того как в ваших шаблонах будет появляться все больше и больше информации, отправленной пользователями, эти фрагменты кода станут по-настоящему полезными.

На завершающем этапе обновите скрипт контроллера, чтобы он использовал разделяемые включения db.inc.php и magicquotes.inc.php (`chapter6/jokes-helpers/index.php`).

```
<?php
include_once $_SERVER['DOCUMENT_ROOT'] . '/includes/magicquotes.inc.php';

if (isset($_GET['addjoke']))
{
  include 'form.html.php';
  exit();
}

if (isset($_POST['joketext']))
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  try
  {
    $sql = 'INSERT INTO joke SET
        joketext = :joketext,
        jokedate = CURDATE()';
    $s = $pdo->prepare($sql);
```

```
    $s->bindValue(':joketext', $_POST['joketext']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при добавлении шутки: ' . $e->getMessage();
    include 'error.html.php';
    exit();
  }

  header('Location: .');
  exit();
}

if (isset($_GET['deletejoke']))
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  try
  {
    $sql = 'DELETE FROM joke WHERE id = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при удалении шутки: ' . $e->getMessage();
    include 'error.html.php';
    exit();
  }

  header('Location: .');
  exit();
}

include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

try
{
  $sql = 'SELECT joke.id, joketext, name, email
      FROM joke INNER JOIN author
        ON authorid = author.id';
  $result = $pdo->query($sql);
}
catch (PDOException $e)
{
  $error = 'Ошибка при извлечении шуток: ' . $e->getMessage();
  include 'error.html.php';
  exit();
}

foreach ($result as $row)
{
  $jokes[] = array(
    'id' => $row['id'],
    'text' => $row['joketext'],
```

```
        'name' => $row['name'],
        'email' => $row['email']
    );
}

include 'jokes.html.php';
```

Самый лучший путь

В этой главе вы вышли за пределы базовых возможностей PHP и приступили к поиску более совершенных решений задач. Конечно, никто не запрещает использовать множество мелких скриптов, которые выполняют определенный набор действий, но когда вы приступите к созданию настоящего сайта и реализации таких задач, как подключение к базе данных, создание общих элементов навигации, ведение статистики посетителей и управление доступом, тщательное структурирование кода окажется как нельзя кстати.

Вы рассмотрели несколько простых, но эффективных механизмов структурирования кода. Так, подключаемые файлы позволили вам многократно использовать один и тот же фрагмент кода на разных страницах сайта, благодаря чему вам стало значительно проще вносить любые изменения в код. Собственные функции, размещенные внутри подключаемых файлов, помогли превратить данные файлы в мощные функциональные библиотеки, способные выполнять необходимые задачи и возвращать значения в скрипт, который их вызвал. Эти новые подходы сослужат вам хорошую службу при изучении оставшейся части книги.

Возможно, вы захотите сделать еще один важный шаг на пути структурирования кода и изучить объектно-ориентированные возможности языка PHP. Некоторую полезную информацию на эту тему вы найдете в разделе «Классы и объекты» руководства PHP (http://www.php.net/oop5). Для получения более исчерпывающих сведений мы рекомендуем ознакомиться с книгой PHP Master: Write Cutting-edge Code (http://www.sitepoint.com/books/phppro1/).

В главе 7 вы найдете применение тем знаниям, которыми овладели на данном этапе (а также освоите несколько новых приемов), и создадите с помощью PHP систему управления содержимым сайта. Главная задача такой системы — предоставить безопасный настраиваемый интерфейс, который позволит управлять содержимым базы данных сайта без использования phpMyAdmin.

Глава 7
СИСТЕМА УПРАВЛЕНИЯ СОДЕРЖИМЫМ

Чтобы выйти на качественно новый уровень и перейти от создания отдельных веб-страниц, которые выводят информацию из базы данных, к созданию сайта, полностью на них основанного, необходимо разработать **систему управления содержимым** (Content Management System, или CMS). Как правило, такая система представляет собой набор веб-страниц, доступ к которым имеют только пользователи, обладающие правом вносить изменения в содержимое и структуру сайта. Эти страницы предоставляют интерфейс для администрирования базы данных — вывода и редактирования хранящейся в ней информации без использования SQL-запросов.

В конце главы 4 у вас уже состоялось небольшое знакомство с CMS, когда вы разрешили посетителям сайта добавлять шутки, используя веб-форму, и удалять их с помощью кнопки Удалить. Сами по себе эти функции замечательные, но в интерфейсе, предназначенном для обычных посетителей, их чаще всего делают недоступными. Вряд ли нужно, чтобы пользователи без вашего ведома размещали на страницах оскорбительный материал или удаляли шутки с вашего сайта. Расположив такие функции на страницах с ограниченным доступом, вы избежите подобного риска и получите возможность управлять содержимым базы данных без применения SQL-запросов.

В этой главе вы расширите возможности системы управления шутками и воспользуетесь теми улучшениями, которые вы внесли в базу данных в главе 5. В частности, вы позволите администратору сайта управлять списком авторов и категориями, привязывая отдельные записи из них к соответствующим шуткам.

Для ограничения доступа к административным страницам необходимо использовать специальный механизм. Один из способов — настроить веб-сервер так, чтобы он защищал соответствующие PHP-файлы, запрашивая у посетителей логин и пароль. В Apache для этого предусмотрен файл .htaccess, в котором перечисляются авторизированные пользователи. Другой способ — использовать возможности PHP. Он более гибкий и дает лучший результат, но потребует больше сил. Подробнее о нем вы узнаете в главе 9.

Пока сосредоточимся на создании страниц, составляющих CMS.

Главная страница

К концу главы 5 ваша база данных содержала таблицы для трех видов сущностей: шуток, авторов и категорий (рис. 7.1). Обратите внимание, что у каждого автора предполагается наличие только одного адреса электронной почты.

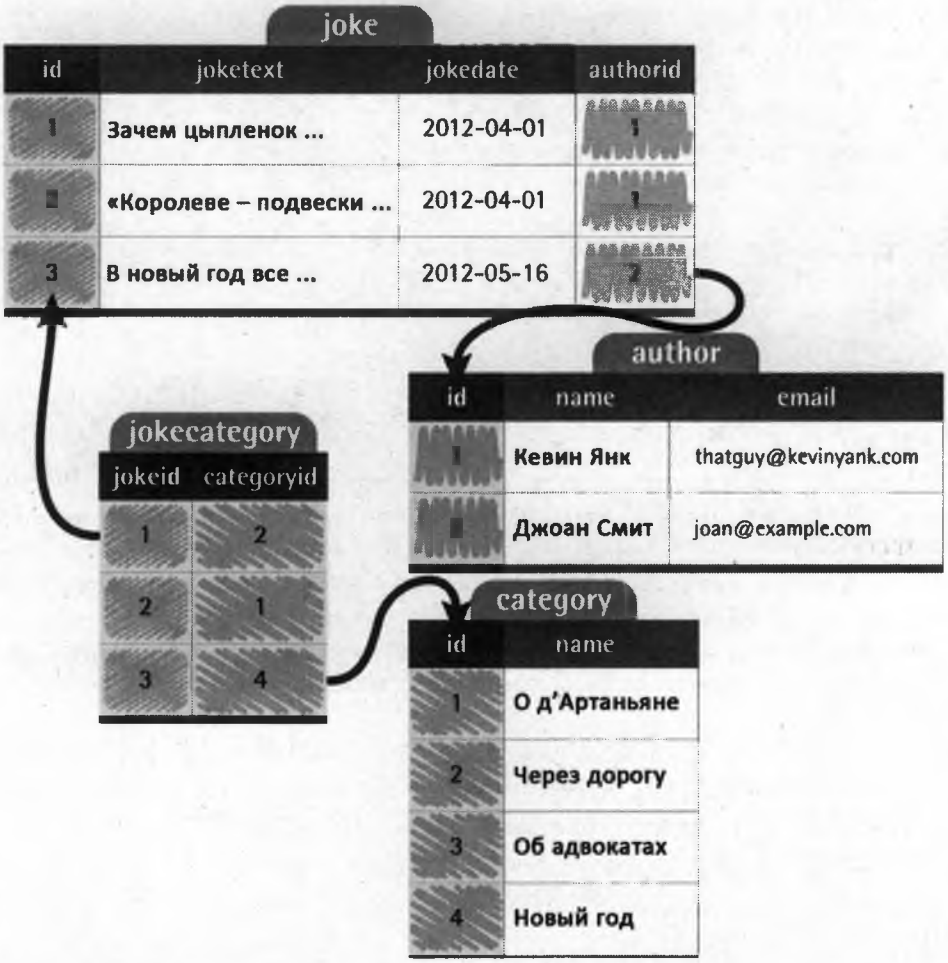


Рис. 7.1. Окончательная структура базы данных `ijdb` включает три вида сущностей

Если вам необходимо создать структуру базы данных с нуля, воспользуйтесь нижеприведенными SQL-запросами (`chapter7/sql/ijdb.sql`).

```
CREATE TABLE joke (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  joketext TEXT,
  jokedate DATE NOT NULL,
  authorid INT
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB;

CREATE TABLE author (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255),
  email VARCHAR(255)
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB;

CREATE TABLE category (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255)
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB;

CREATE TABLE jokecategory (
  jokeid INT NOT NULL,
  categoryid INT NOT NULL,
```

```
  PRIMARY KEY (jokeid, categoryid)
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB;

# Шаблонные данные.
# Используемые идентификаторы позволяют распознать записи
# при создании связей.

INSERT INTO author (id, name, email) VALUES
(1, 'Кевин Янк', 'thatguy@kevinyank.com'),
(2, 'Джоан Смит', 'joan@example.com');

INSERT INTO joke (id, joketext, jokedate, authorid) VALUES
(1, 'Зачем цыпленок перешел дорогу? Чтобы попасть на другую сторону!',
'2012-04-01', 1),
(2, '"Королеве - подвески, Констанции - подвязки, Портосу - подтяжки..." -
повторял, чтобы не перепутать, д\'Артаньян по дороге в Англию.',
'2012-04-01', 1),
(3, ' В Новый год все сбывается, даже то, что в другое время сбыть не
удается.', '2012-04-01', 2),
(4, ' Сколько нужно адвокатов, чтобы вкрутить лампочку? Вопрос в другом:
сколько адвокатов может себе позволить лампочка, чтобы быть вкрученной.',
'2012-04-01', 2);

INSERT INTO category (id, name) VALUES
(1, 'О д\'Артаньяне'),
(2, 'Через дорогу'),
(3, 'Об адвокатах'),
(4, 'Новый год');

INSERT INTO jokecategory (jokeid, categoryid) VALUES
(1, 2),
(2, 1),
(3, 4),
(4, 3);
```

Учитывая вышесказанное, на главной странице CMS необходимо разместить три ссылки, каждая из которых позволит перейти на страницу, предназначенную для управления нужной сущностью (рис. 7.2). Для этого воспользуемся следующим HTML-кодом (`chapter7/admin/index.html`).

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Шуточная CMS</title>
  </head>
  <body>
    <h1>Система управления шутками</h1>
    <ul>
      <li><a href="jokes/">Управление шутками</a></li>
      <li><a href="authors/">Управление авторами</a></li>
      <li><a href="categories/">Управление категориями шуток</a></li>
    </ul>
  </body>
</html>
```

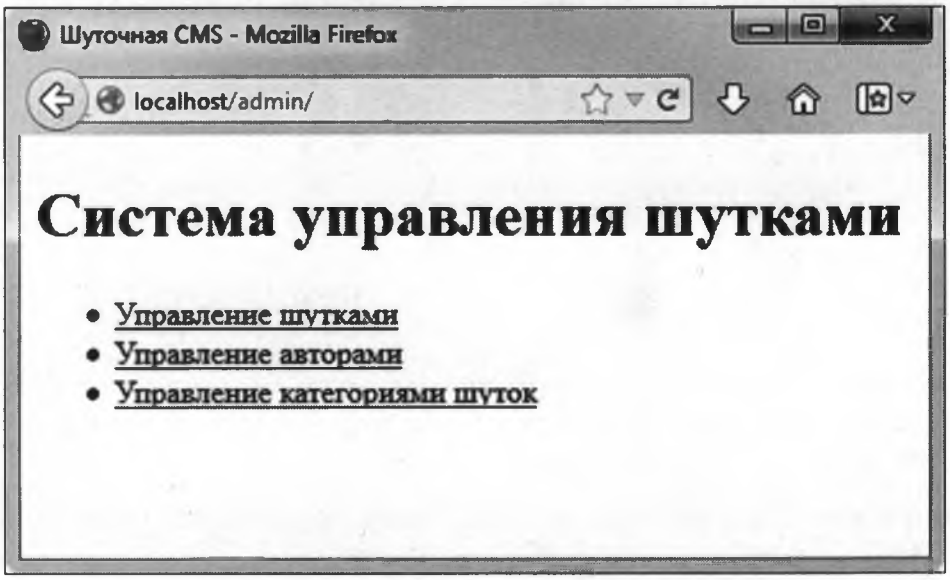


Рис. 7.2. Начальная страница CMS с тремя ссылками

Созданные ссылки указывают на разные подкаталоги вашего проекта: **jokes**, **authors** и **categories**. Каждый из них будет содержать скрипт контроллера (**index.php**) и связанные с ним шаблоны, необходимые для управления записями соответствующего типа в базе данных.

Управление списком авторов

Первым делом напишем скрипты для подкаталога **authors**, которые отвечают за добавление и удаление авторов.

Для начала предоставим администратору список авторов, имена которых содержатся в базе данных в настоящее время. С точки зрения программирования, это то же самое, что вывести перечень имеющихся шуток. Поскольку администратор обладает правом удалять и редактировать записи, рядом с каждой из них добавим кнопки для выполнения указанных действий. По аналогии с кнопкой **Удалить** из главы 4 они отправят идентификатор соответствующей записи контроллеру, чтобы тот мог определить, какую запись требуется отредактировать или удалить. В завершение создадим ссылку **Добавить нового автора**, ведущую на форму. По функциям она будет похожа на ссылку **Добавить новую шутку**, созданную в главе 4.

Вот как выглядит контроллер (`chapter7/admin/authors/index.php`, фрагмент).

```
// Выводим список авторов.
include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

try
{
  $result = $pdo->query('SELECT id, name FROM author');
}
catch (PDOException $e)
{
  $error = 'Ошибка при извлечении авторов из базы данных!';
```

```
    include 'error.html.php';
    exit();
}

foreach ($result as $row)
{
    $authors[] = array('id' => $row['id'], 'name' => $row['name']);
}

include 'authors.html.php';
```

Ничего нового в этом коде нет, отметим только, что соединение с базой данных создается с использованием общего подключаемого файла (**db.inc.php**) из каталога **include** в корневой директории сервера.

Ниже приведен шаблон кода, который выводит список авторов (`chapter7/admin/authors/authors.html.php`, фрагмент).

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?> ❶
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Управление авторами</title>
  </head>
  <body>
    <h1>Управление авторами</h1>
    <p><a href="?add">Добавить нового автора</a></p> ❷
    <ul>
      <?php foreach ($authors as $author): ?>
        <li>
          <form action="" method="post"> ❸
            <div>
              <?php htmlout($author['name']); ?> ❹
              <input type="hidden" name="id" value="<?php
                  echo $author['id']; ?>">
              <input type="submit" name="action" value="Редактировать"> ❺
              <input type="submit" name="action" value="Удалить">
            </div>
          </form>
        </li>
      <?php endforeach; ?>
    </ul>
    <p><a href="..">Вернуться на главную страницу</a></p>
  </body>
</html>
```

Пока понимание кода не должно вызывать у вас затруднений. Остановимся лишь на некоторых моментах.

1. Шаблон использует общий подключаемый файл, созданный в главе 6, что позволяет осуществить безопасный вывод текста с помощью функции `htmlspecialchars` в менее громоздком виде.

2. Ссылка, отправляющая запрос (`?add`) контроллеру, сообщает о намерении пользователя добавить нового автора.
3. Атрибут `action` пуст. При отправке формы контроллеру поступит запрос о том, что необходимо сделать с записью — удалить или отредактировать. В главе 4 в этом атрибуте использовалась строка запроса (`?deletejoke`), указывающая скрипту на действие, которое необходимо выполнить. В данном примере выбор действия зависит от пользователя, поэтому для оповещения контроллера применяется другой способ.
4. Функция `htmlout` помогает отобразить имя каждого автора в безопасном режиме.
5. Форма содержит две кнопки: одну для редактирования записи об авторе, другую для удаления. Обе кнопки имеют одинаковое значение атрибута `name` (`action`), чтобы исходя из отправленных данных (`$_POST['action']`) контроллер мог определить, какая из них была нажата.

На рис. 7.3 показан список авторов, сформированный этим шаблоном.

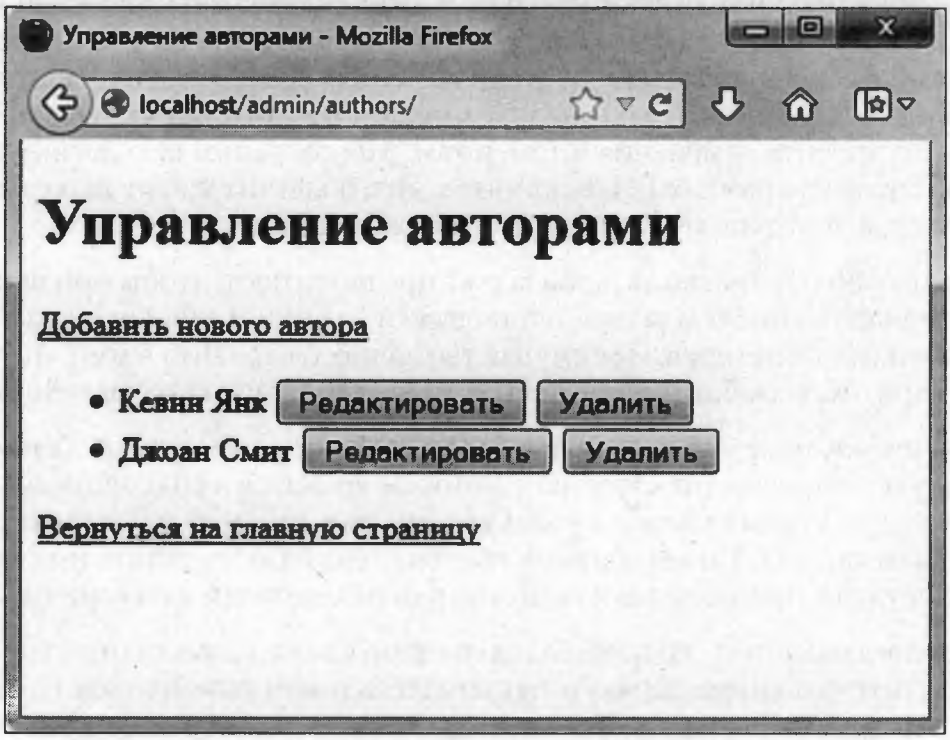


Рис. 7.3. Стартовая страница для управления списком авторов

Удаление имен авторов

После того как пользователь нажмет одну из кнопок **Удалить**, контроллер сотрет соответствующую запись из базы данных, используя идентификатор автора, переданный в форме.

Эта довольно простая задача в нашем примере немного усложнится. Как вы помните, имя автора той или иной шутки указывается в таблице `joke` в столбце `authorid`. Однако, удаляя автора из базы данных, необходимо стереть и все ссылки

на него в оставшихся таблицах. В противном случае в базе данных останутся шутки, связанные с несуществующими авторами.

Есть три способа справиться с этой проблемой:

1) запретить пользователям удалять имена авторов, которые связаны с шутками в базе данных;

2) удалять вместе с именем автора все его шутки;

3) удаляя имя автора, присвоить соответствующим полям `authorid` в таблице `joke` значение `NULL`, обозначая тем самым, что автора у шуток нет.

Подобные меры позволяют сохранить связи между записями в базе данных и тем самым обеспечивают **ссылочную целостность**. Как и большинство других серверов баз данных, MySQL способен справиться со всем самостоятельно, используя **ограничения внешних ключей**. С их помощью MySQL выполнит любое из вышеперечисленных действий и сохранит корректные отношения между данными.

Подробнее ограничения внешних ключей рассмотрены в главе 10. В данном примере они не используются, поскольку в этом случае часть функций CMS выполнялась бы с помощью PHP-кода, а другая осталась бы на уровне структуры базы данных. Если бы позже вы захотели изменить способ удаления авторов (например, запретить пользователю удалять имена тех, у кого уже есть шутки), то вам пришлось бы вносить изменения и там, и там. Мы сохраним всю логику удаления записей об авторах в рамках PHP-скриптов. Это облегчит жизнь вам самим и тем людям, которые в будущем захотят изменить данный код.

Поскольку большинство авторов все же предпочитают, чтобы при использовании их материала на них ссылались, остановимся на втором варианте: удалим шутки вместе с именами их авторов. Преимущество данного варианта в том, что в столбце `authorid` при отображении библиотеки шуток не будет нулевых значений (`NULL`).

Раз уж шутки тоже удаляются, необходимо учесть еще одну особенность. Некоторым шуткам присвоены категории, которые хранятся в виде записей в таблице `jokecategory`. Удаляя шутку, нужно убедиться в том, что эти записи также исчезнут из базы данных. Таким образом, контроллер должен удалить имя автора, все его шутки, а также принадлежность шуток к определенным категориям.

Как вы догадываетесь, код, позволяющий это сделать, выглядит довольно объемно. Прочтите его внимательно и попытайтесь в нем разобраться (`chapter7/admin/authors/index.php`, фрагмент).

```
if (isset($_POST['action']) and $_POST['action'] == 'Удалить')
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  // Получаем шутки, принадлежащие автору.
  try
  {
    $sql = 'SELECT id FROM joke WHERE authorid = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
```

```
{
  $error = 'Ошибка при получении списка шуток, которые нужно удалить.';
  include 'error.html.php';
  exit();
}

$result = $s->fetchAll();

// Удаляем записи о категориях шуток.
try
{
  $sql = 'DELETE FROM jokecategory WHERE jokeid = :id';
  $s = $pdo->prepare($sql);

  // Для каждой шутки
  foreach ($result as $row)
  {
    $jokeId = $row['id'];
    $s->bindValue(':id', $jokeId);
    $s->execute();
  }
}
catch (PDOException $e)
{
  $error = 'Ошибка при удалении записей о категориях шутки.';
  include 'error.html.php';
  exit();
}

// Удаляем шутки, принадлежащие автору.
try
{
  $sql = 'DELETE FROM joke WHERE authorid = :id';
  $s = $pdo->prepare($sql);
  $s->bindValue(':id', $_POST['id']);
  $s->execute();
}
catch (PDOException $e)
{
  $error = 'Ошибка при удалении шуток, принадлежащих автору.';
  include 'error.html.php';
  exit();
}

// Удаляем имя автора.
try
{
  $sql = 'DELETE FROM author WHERE id = :id';
  $s = $pdo->prepare($sql);
  $s->bindValue(':id', $_POST['id']);
  $s->execute();
}
catch (PDOException $e)
{
  $error = 'Ошибка при удалении автора.';
  include 'error.html.php';
  exit();
```

```
    }

    header('Location: .');
    exit();
}
```

По большей части работа кода должна быть вам понятна. Сложные моменты выделены полужирным начертанием.

Первый элемент, который мог показаться вам необычным, — это выражение if, расположенное в начале скрипта (chapter7/admin/authors/index.php, фрагмент).

```
if (isset($_POST['action']) and $_POST['action'] == 'Удалить')
```

Из раздела «Управление списком авторов» этой главы вы знаете, что для удаления имени автора из базы данных нужно нажать кнопку **Удалить** рядом с соответствующей записью. Поскольку эта кнопка имеет атрибут name со значением action, вы можете зафиксировать ее нажатие. Для этого необходимо проверить наличие переменной $_POST['action'], а затем посмотреть, каково ее содержимое (оно должно равняться 'Удалить').

Перейдем к следующему выделенному участку кода (chapter7/admin/authors/index.php, фрагмент).

```
$result = $s->fetchAll();
```

До этой строки выполняется запрос с командой SELECT, который позволяет извлечь все шутки автора, чье имя вы собираетесь удалить. Располагая таким списком и поочередно применяя команду DELETE к каждой шутке, удалить соответствующие записи для категорий не составит труда. Но проблема в том, что как раз этого списка у вас и нет. Рассмотрим, в чем тут дело.

Как правило, при выполнении запроса SELECT используются циклы while или foreach, с помощью которых из результирующего набора извлекается каждая отдельная строка.

```
while ($row = $result->fetch())

foreach ($result as $row)
```

При такой обработке результатов запроса PHP извлекает из базы данных по одной строке за каждую итерацию цикла. При переходе к следующей итерации текущая запись заменяется следующей. Благодаря тому что одновременного хранения всех строк не требуется, PHP экономит место в памяти.

В большинстве случаев разработчикам необязательно знать о том, что PHP использует в своей работе столь тонкую оптимизацию. Однако иногда есть необходимость отправить очередной SQL-запрос до того, как обработаются все результаты предыдущего.

Наш пример — как раз такой случай. Требуется получить список всех шуток конкретного автора по запросу SELECT и, пока идет перебор этого списка, выполнить для каждого элемента команду DELETE. Проблема в том, что нельзя приоста-

новить работу сервера MySQL на полпути и попросить переключиться на команды DELETE, в то время как он возвращает результаты выполнения запроса SELECT. Это приведет к тому, что запросы на удаление завершатся ошибками.

Здесь вам и пригодится метод `fetchAll`. Вызванный из заранее подготовленного выражения `$s`, он извлекает из запроса весь набор результатов и помещает его в массив `$result` (`chapter7/admin/authors/index.php`, фрагмент).

```
$result = $s->fetchAll();
```

Теперь вы можете пройтись по массиву с помощью цикла `foreach`, как в случае с объектом `PDOStatement`, и извлечь каждую строку по очереди. Таким образом, PHP сохраняет все результаты и позволяет отправлять серверу MySQL новые запросы.

И наконец, последний сложный участок кода (`chapter7/admin/authors/index.php`, фрагмент).

```
// Удаляем записи о категориях шуток.
try
{
  $sql = 'DELETE FROM jokecategory WHERE jokeid = :id';
  $s = $pdo->prepare($sql);

  // Для каждой шутки
  foreach ($result as $row)
  {
    $jokeId = $row['id'];
    $s->bindValue(':id', $jokeId);
    $s->execute();
  }
}
```

В данном фрагменте кода вызывается запрос DELETE, который удаляет в базе данных записи из таблицы `jokecategory` для каждой шутки. Казалось бы, самое очевидное решение — начать с цикла `foreach`, но в данном случае создается параметризированный запрос.

Данный код демонстрирует второе существенное преимущество параметризированных запросов (с первым вы познакомились в главе 4[1]). Записав готовое выражение, вы имеете возможность использовать его снова и снова, каждый раз подставляя вместо псевдопеременных новые значения. В этом примере вам необходимо выполнить команду DELETE, указав идентификатор нужной шутки. Употребляя во всех запросах заранее подготовленное выражение, вы избавляете MySQL от необходимости тщательно разбирать каждый фрагмент SQL-кода и предлагаете ему удобный шаблон для выполнения запроса. Сервер MySQL считывает SQL-код всего один раз, затем находит самый эффективный способ для выполнения указанных команд DELETE и применяет его снова и снова, используя переданные идентификаторы.

[1] Если вы забыли: первое преимущество параметризированных запросов заключается в том, что они содержат псевдопеременные, которым можно безопасно присваивать значения и для которых исключается возможность внедрения вредоносного SQL-кода.

Получив необходимые разъяснения, взгляните еще раз на данный фрагмент кода — теперь он должен показаться вам куда более логичным. Для начала вы создаете параметризированный запрос с помощью SQL-кода, содержащего псевдопеременную. Затем посредством цикла `foreach` перебираете результирующий набор, который вернул предыдущий запрос с командой `SELECT`. При этом для каждой шутки выполняется заранее подготовленное выражение `DELETE`, в которое на место псевдопеременной `:id` подставляются идентификаторы, хранящиеся внутри `bindValue`.

Не переживайте, если для понимания кода вам пришлось перечитать его несколько раз. Это один из самых сложных моментов в изучении PHP, с которыми вы столкнетесь в данной книге.

Когда вы почувствуете, что окончательно во всем разобрались, попробуйте удалить одного из авторов. Приложение phpMyAdmin позволит вам убедиться в том, что все шутки выбранного автора и связанные с ними записи о категориях также исчезли. При этом сами категории останутся, даже если в них не будет ни одной шутки.

ПОДТВЕРЖДЕНИЕ УДАЛЕНИЯ

Попробуйте добавить в приведенный код запрос о подтверждении удаления. Возьмите за отправную точку скрипты из архива файлов для данной главы. Измените контроллер таким образом, что в ответ на нажатие кнопки Удалить появлялся шаблон с просьбой подтвердить выполнение действия. После отправки формы с этой страницы должен сработать тот фрагмент кода, который удаляет данные. Для передачи идентификатора имени нужного автора используйте скрытое поле.

Добавление и редактирование имен авторов

Ссылка для добавления имени нового автора очень похожа на ту, которую вы создавали в главе 4 для списка шуток. Только теперь пользователю предлагается ввести не текст шутки, а имя автора и его электронный адрес.

Однако страница для управления списком авторов, кроме добавления записи, предоставляет еще одну похожую функцию — редактирование записи. Поскольку обе функции требуют от пользователя заполнения похожих форм, попробуем их совместить. Создадим код шаблона для формы, которая позволит не только добавлять, но и редактировать информацию об авторах (`chapter7/admin/authors/form.html.php`).

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title><?php htmlout($pageTitle); ?></title>
  </head>
  <body>
    <h1><?php htmlout($pageTitle); ?></h1>
    <form action="?<?php htmlout($action); ?>" method="post">
```

```
      <div>
        <label for="name">Имя: <input type="text" name="name"
            id="name" value="<?php htmlout($name); ?>"></label>
      </div>
      <div>
        <label for="email">Электронная почта: <input type="text"
        name="email"
            id="email" value="<?php htmlout($email); ?>"></label>
      </div>
      <div>
        <input type="hidden" name="id" value="<?php
            htmlout($id); ?>">
        <input type="submit" value="<?php htmlout($button); ?>">
      </div>
    </form>
  </body>
</html>
```

Обратите внимание на шесть переменных:

1) `$pageTitle` задает название страницы и верхний уровень заголовков (`<h1>`);

2) `$action` определяет значение, которое передается вместе со строкой запроса при отправке формы;

3) `$name` устанавливает начальное значение в поле формы для имени автора;

4) `$email` задает начальное значение в поле формы для электронного адреса автора;

5) `$id` устанавливает значение в скрытом поле формы для идентификатора автора в базе данных;

6) `$button` определяет название кнопки, которая отправляет форму.

Эти переменные позволяют использовать одну и ту же форму для нескольких задач: добавления записей и их редактирования. В табл. 7.1 приведены значения, которые присваиваются переменным в каждом из случаев.

Таблица 7.1. Значения переменных для двойной формы

Переменная	Выполняемое действие	
	добавление	**редактирование**
`$pageTitle`	`'Новый автор'`	`'Редактировать автора'`
`$action`	`'addform'`	`'editform'`
`$name`	`''` (пустая строка)	текущее имя
`$email`	`''` (пустая строка)	текущий электронный адрес
`$id`	`''` (пустая строка)	текущий id автора
`$button`	`'Добавить автора'`	`'Обновить информацию об авторе'`

Представленный ниже код контроллера при нажатии соответствующей ссылки загружает форму в режиме добавления записи (chapter7/admin/authors/index.php, фрагмент).

```
<?php
include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/magicquotes.inc.php';

if (isset($_GET['add']))
{
  $pageTitle = 'Новый автор';
  $action = 'addform';
  $name = '';
  $email = '';
  $id = '';
  $button = 'Добавить автора';

  include 'form.html.php';
  exit();
}
```

О том, что форма отправлена именно в режиме добавления записи, позволяет судить наличие переменной $_GET['addform'] (chapter7/admin/authors/index.php, фрагмент).

```
if (isset($_GET['addform']))
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  try
  {
    $sql = 'INSERT INTO author SET
        name = :name,
        email = :email';
    $s = $pdo->prepare($sql);
    $s->bindValue(':name', $_POST['name']);
    $s->bindValue(':email', $_POST['email']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при добавлении автора.';
    include 'error.html.php';
    exit();
  }

  header('Location: .');
  exit();
}
```

Эта форма используется и в том случае, если пользователь нажимает кнопку **Редактировать** в списке авторов, только теперь из базы данных следует загрузить сохраненные сведения об авторе (chapter7/admin/authors/index.php, фрагмент).

```
if (isset($_POST['action']) and $_POST['action'] == 'Редактировать')
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';
  try
  {
    $sql = 'SELECT id, name, email FROM author WHERE id = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при извлечении информации об авторе.';
    include 'error.html.php';
    exit();
  }

  $row = $s->fetch();

  $pageTitle = 'Редактировать автора';
  $action = 'editform';
  $name = $row['name'];
  $email = $row['email'];
  $id = $row['id'];
  $button = 'Обновить информацию об авторе';
  include 'form.html.php';
  exit();
}
```

То, что форма отправлена в режиме редактирования записи, определяется по наличию переменной `$_GET['editform']`. Код, обрабатывающий данные формы, похож на тот, с помощью которого вы добавляли нового автора, только вместо команды `INSERT` используется команда `UPDATE` (`chapter7/admin/authors/index.php`, фрагмент).

```
if (isset($_GET['editform']))
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  try
  {
    $sql = 'UPDATE author SET
        name = :name,
        email = :email
        WHERE id = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->bindValue(':name', $_POST['name']);
    $s->bindValue(':email', $_POST['email']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при обновлении записи об авторе.';
    include 'error.html.php';
```

```
        exit();
      }

      header('Location: .');
      exit();
    }
```

Шаблон формы для редактирования записи показан на рис. 7.4.

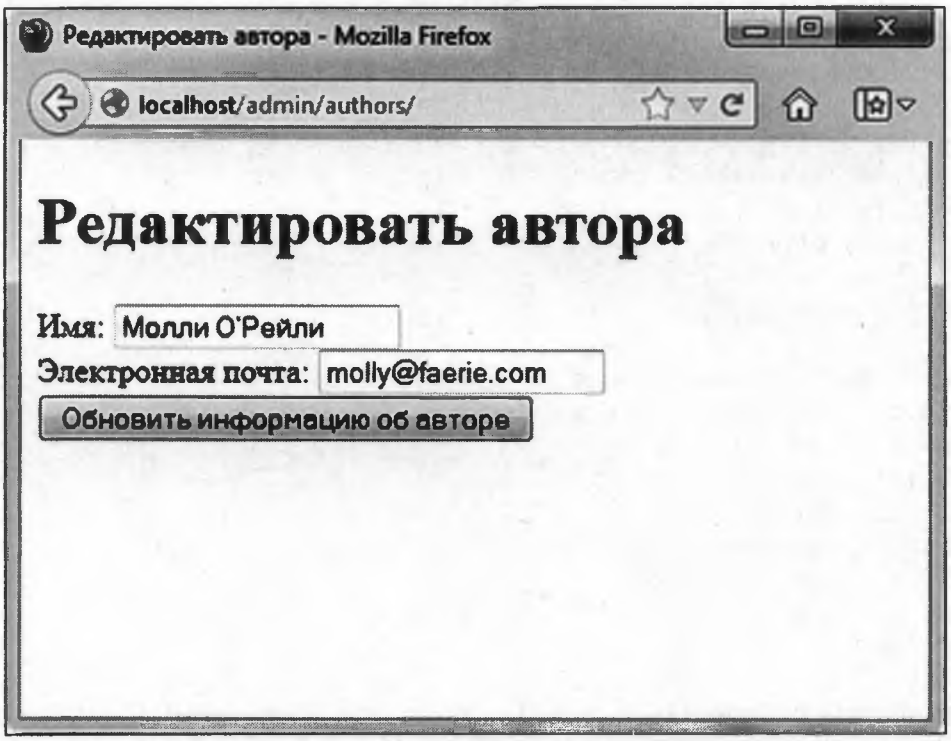


Рис. 7.4. Хочется верить, что этот автор умеет шутить

Теперь вы можете без проблем добавлять имена авторов, редактировать их или удалять. Если при тестировании примера у вас вдруг появятся сообщения об ошибках, вернитесь в начало раздела и проверьте, совпадает ли написанный вами код с тем, который приведен в книге. Если и это не поможет, сравните собственные скрипты со скриптами из архива для этого примера.

Управление списком категорий

Записи об авторах и категориях в базе данных имеют много общего. Каждая из сущностей находится в отдельной таблице и позволяет группировать шутки определенным образом. В связи с этим код для управления категориями очень похож на тот, который вы использовали для работы со списком авторов. Однако есть одно существенное отличие.

При удалении категории принадлежащие ей шутки не удаляются, поскольку вполне вероятно, что они связаны с другими элементами в таблице `category`. Как вариант, можно проверить каждую шутку на вхождение в какую-либо категорию и удалять только в том случае, если вхождений не обнаружено. Однако прежде чем начинать столь трудоемкий процесс, давайте попробуем сделать так, чтобы

для тех шуток, которые добавляются в базу данных, категории не присваивались вовсе. Способ реализации этого зависит от вас. Например, шутки могут оставаться невидимыми для посетителей сайта и находиться в базе данных до тех пор, пока вы не захотите определить их в какую-нибудь категорию.

Итак, чтобы удалить категорию, необходимо стереть также все связанные с ней записи в таблице `jokecategory` (`chapter7/admin/categories/index.php`, фрагмент).

```
// Удаляем все записи, связывающие шутки с этой категорией.
try
{
  $sql = 'DELETE FROM jokecategory WHERE categoryid = :id';
  $s = $pdo->prepare($sql);
  $s->bindValue(':id', $_POST['id']);
  $s->execute();
}
catch (PDOException $e)
{
  $error = 'Ошибка при удалении шуток из категории.';
  include 'error.html.php';
  exit();
}

// Удаляем категорию.
try
{
  $sql = 'DELETE FROM category WHERE id = :id';
  $s = $pdo->prepare($sql);
  $s->bindValue(':id', $_POST['id']);
  $s->execute();
}
catch (PDOException $e)
{
  $error = 'Ошибка при удалении категории.';
  include 'error.html.php';
  exit();
}
```

Помимо указанной особенности управление категориями функционально ничем не отличается от управления списком авторов. Ниже приведены полные версии кода для четырех соответствующих файлов (`chapter7/admin/categories/index.php`, `chapter7/admin/categories/categories.html.php`, `chapter7/admin/categories/form.html.php` и `chapter7/admin/categories/error.html.php`). В них также используются общие подключаемые файлы **db.inc.php**, **magicquotes.inc.php** и **helpers.inc.php** из главы 6.

```
<?php
include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/magicquotes.inc.php';

if (isset($_GET['add']))
{
  $pageTitle = 'Новая категория';
```

```
    $action = 'addform';
    $name = '';
    $id = '';
    $button = 'Добавить категорию';

    include 'form.html.php';
    exit();
}

if (isset($_GET['addform']))
{
    include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

    try
    {
        $sql = 'INSERT INTO category SET
            name = :name';
        $s = $pdo->prepare($sql);
        $s->bindValue(':name', $_POST['name']);
        $s->execute();
    }
    catch (PDOException $e)
    {
        $error = 'Ошибка при добавлении категории.';
        include 'error.html.php';
        exit();
    }

    header('Location: .');
    exit();
}

if (isset($_POST['action']) and $_POST['action'] == 'Редактировать')
{
    include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

    try
    {
        $sql = 'SELECT id, name FROM category WHERE id = :id';
        $s = $pdo->prepare($sql);
        $s->bindValue(':id', $_POST['id']);
        $s->execute();
    }
    catch (PDOException $e)
    {
        $error = 'Ошибка при извлечении информации о категории.';
        include 'error.html.php';
        exit();
    }

    $row = $s->fetch();

    $pageTitle = 'Редактировать категорию';
    $action = 'editform';
    $name = $row['name'];
    $id = $row['id'];
```

```
    $button = 'Обновить категорию';

    include 'form.html.php';
    exit();
}

if (isset($_GET['editform']))
{
    include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

    try
    {
        $sql = 'UPDATE category SET
            name = :name
            WHERE id = :id';
        $s = $pdo->prepare($sql);
        $s->bindValue(':id', $_POST['id']);
        $s->bindValue(':name', $_POST['name']);
        $s->execute();
    }
    catch (PDOException $e)
    {
        $error = 'Ошибка при обновлении категории.';
        include 'error.html.php';
        exit();
    }

    header('Location: .');
    exit();
}

if (isset($_POST['action']) and $_POST['action'] == 'Удалить')
{
    include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

    // Удаляем все записи, связывающие шутки с этой категорией.
    try
    {
        $sql = 'DELETE FROM jokecategory WHERE categoryid = :id';
        $s = $pdo->prepare($sql);
        $s->bindValue(':id', $_POST['id']);
        $s->execute();
    }
    catch (PDOException $e)
    {
        $error = 'Ошибка при удалении шуток из категории.';
        include 'error.html.php';
        exit();
    }

    // Удаляем категорию.
    try
    {
        $sql = 'DELETE FROM category WHERE id = :id';
        $s = $pdo->prepare($sql);
        $s->bindValue(':id', $_POST['id']);
```

```
      $s->execute();
    }
    catch (PDOException $e)
    {
      $error = 'Ошибка при удалении категории.';
      include 'error.html.php';
      exit();
    }

    header('Location: .');
    exit();
  }

  // Выводим список категорий.
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  try
  {
    $result = $pdo->query('SELECT id, name FROM category');
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при извлечении категорий из базы данных!';
    include 'error.html.php';
    exit();
  }

  foreach ($result as $row)
  {
    $categories[] = array('id' => $row['id'], 'name' => $row['name']);
  }

  include 'categories.html.php';

  <?php include_once $_SERVER['DOCUMENT_ROOT'] .
      '/includes/helpers.inc.php'; ?>
  <!DOCTYPE html>
  <html lang="en">
    <head>
      <meta charset="utf-8">
      <title>Управление категориями</title>
    </head>
    <body>
      <h1>Управление категориями</h1>
      <p><a href="?add">Добавить новую категорию</a></p>
      <ul>
        <?php foreach ($categories as $category): ?>
          <li>
            <form action="" method="post">
              <div>
                <?php htmlout($category['name']); ?>
                <input type="hidden" name="id" value="<?php
                    echo $category['id']; ?>">
                <input type="submit" name="action" value="Редактировать">
                <input type="submit" name="action" value="Удалить">
              </div>
```

```
          </form>
        </li>
      <?php endforeach; ?>
    </ul>
    <p><a href="..">Вернуться на главную страницу</a></p>
  </body>
</html>

<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title><?php htmlout($pageTitle); ?></title>
  </head>
  <body>
    <h1><?php htmlout($pageTitle); ?></h1>
    <form action="?<?php htmlout($action); ?>" method="post">
      <div>
        <label for="name">Название: <input type="text" name="name"
            id="name" value="<?php htmlout($name); ?>"></label>
      </div>
      <div>
        <input type="hidden" name="id" value="<?php
            htmlout($id); ?>">
        <input type="submit" value="<?php htmlout($button); ?>">
      </div>
    </form>
  </body>
</html>

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Ошибка скрипта</title>
  </head>
  <body>
    <p>
      <?php echo $error; ?>
    </p>
  </body>
</html>
```

Управление списком шуток

Помимо добавления, удаления и редактирования шуток в базе данных, у вас есть возможность назначать им категорию и автора. Поскольку шуток заведомо больше, чем категорий и авторов, попытка вывести все записи сразу, как это делалось в случае с двумя предыдущими сущностями, приведет к появлению громадного списка. Быстро найти в нем нужную шутку вам вряд ли удастся, поэтому следует придумать более разумный способ навигации по библиотеке записей.

Поиск шуток

Поиск шуток можно осуществлять по категории, имени автора или фрагменту текста. Обеспечим поддержку данных признаков в нашей базе данных. В итоге система станет работать как простой поисковый движок.

Форма, куда администратор заносит информацию при поиске шуток, должна предоставлять список категорий и авторов. Для начала напишем код контроллера, который извлекает эту информацию из базы данных (chapter7/admin/jokes/index.php, фрагмент).

```
// Выводим форму поиска.
include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

try
{
  $result = $pdo->query('SELECT id, name FROM author');
}
catch (PDOException $e)
{
  $error = 'Ошибка при извлечении записей об авторах!';
  include 'error.html.php';
  exit();
}

foreach ($result as $row)
{
  $authors[] = array('id' => $row['id'], 'name' => $row['name']);
}

try
{
  $result = $pdo->query('SELECT id, name FROM category');
}
catch (PDOException $e)
{
  $error = 'Ошибка при извлечении категорий из базы данных!';
  include 'error.html.php';
  exit();
}

foreach ($result as $row)
{
  $categories[] = array('id' => $row['id'], 'name' => $row['name']);
}

include 'searchform.html.php';
```

В этом коде создаются два массива `$authors` и `$categories`, которые в дальнейшем понадобятся в шаблоне **searchform.html.php**. Они помогают вывести в форме поиска два раскрывающихся списка (`chapter7/admin/jokes/searchform.html.php`).

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
```

```
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Управление шутками</title>
  </head>
  <body>
    <h1>Управление шутками</h1>
    <p><a href="?add">Добавить новую шутку</a></p>
    <form action="" method="get">
      <p>Вывести шутки, которые удовлетворяют следующим критериям:</p>
      <div>
        <label for="author">По автору:</label>
        <select name="author" id="author">
          <option value="">Любой автор</option>
          <?php foreach ($authors as $author): ?>
            <option value="<?php htmlout($author['id']); ?>"><?php
                htmlout($author['name']); ?></option>
          <?php endforeach; ?>
        </select>
      </div>
      <div>
        <label for="category">По категории:</label>
        <select name="category" id="category">
          <option value="">Любая категория</option>
          <?php foreach ($categories as $category): ?>
            <option value="<?php htmlout($category['id']); ?>"><?php
                htmlout($category['name']); ?></option>
          <?php endforeach; ?>
        </select>
      </div>
      <div>
        <label for="text">Содержит текст:</label>
        <input type="text" name="text" id="text">
      </div>
      <div>
        <input type="hidden" name="action" value="search">
        <input type="submit" value="Искать">
      </div>
    </form>
    <p><a href="..">Вернуться на главную страницу</a></p>
  </body>
</html>
```

Используя цикл `foreach`, вы генерируете набор элементов `option` в обоих тегах `select`. Атрибут `value` каждого элемента `option` содержит идентификатор из таблицы `author` или `category`, а в текстовых метках хранится имя автора или название категории. Каждый раскрывающийся список начинается элементом `option` с пустым атрибутом `value`, что позволяет исключить поле из критериев поиска.

Стоит отметить, что атрибут `method` в форме имеет значение `get`. Это значит, что у вас есть возможность добавлять результаты поиска в закладки, поскольку значения формы в этом случае будут представлять часть адреса URL в строке запроса. Применение данного подхода к формам поиска — довольно распространенная практика.

Итоговая форма показана на рис. 7.5.

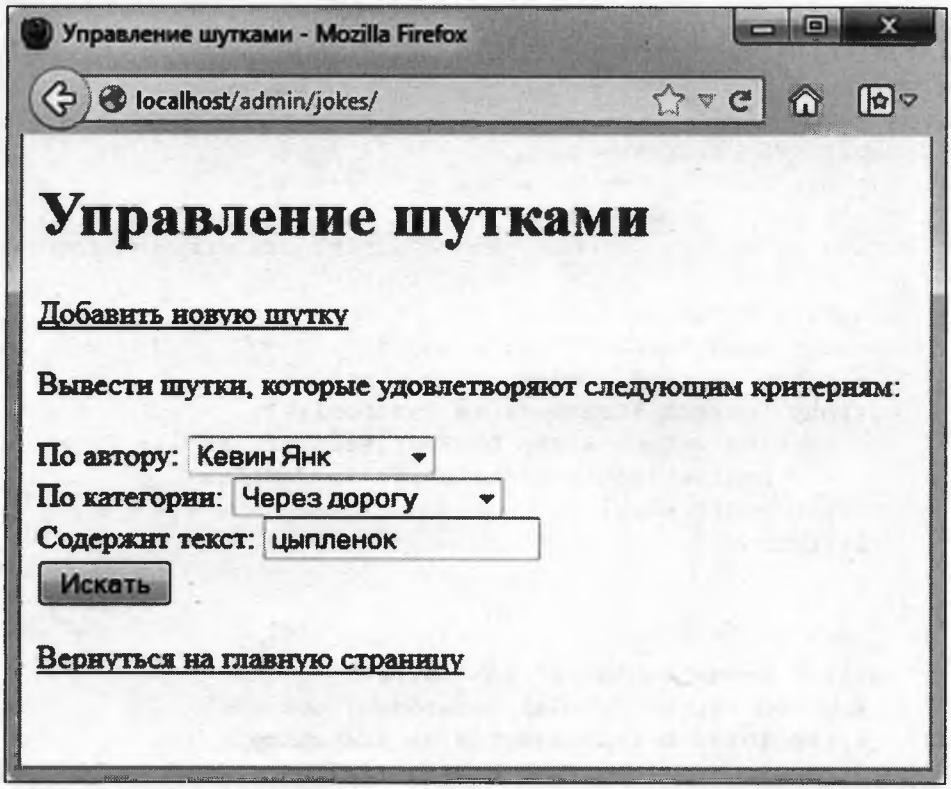


Рис. 7.5. Форма для поиска шуток

Отправленные формой значения контроллер использует для вывода тех шуток, которые удовлетворяют заданным критериям. Очевидно, здесь не обойдется без команды `SELECT`, но конкретный характер запроса будет зависеть от указанных условий. Создание такого запроса — непростая задача, поэтому разберем код контроллера, который отвечает за этот процесс, поэтапно.

Для начала определим несколько строк, объединение которых формирует запрос `SELECT` в том случае, если не выбран ни один критерий (`chapter7/admin/jokes/index.php`, фрагмент).

```
if (isset($_GET['action']) and $_GET['action'] == 'search')
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  // Базовое выражение SELECT.
  $select = 'SELECT id, joketext';
  $from   = ' FROM joke';
  $where  = ' WHERE TRUE';
```

Оператор `WHERE` в этом коде выглядит слегка необычно. Как уже говорилось, вам следует создать базовое выражение `SELECT` исходя из тех критериев, что выбраны в форме. Для этого необходимо добавить в запрос операторы `FROM` и `WHERE`. Если критерии не заданы (то есть администратор хочет получить список всех шуток из базы данных), команда `WHERE` оказывается ненужной. Добавлять выражения к несуществующему оператору проблематично, поэтому придется

подставить такую команду, которая никак не повлияет на результат, если ее опустить. Здесь прекрасно подойдет выражение `WHERE TRUE`, ведь значение `TRUE` всегда истинно[1].

На следующем этапе следует проверить каждый из возможных критериев (автор, категория, искомый текст), который отправляется с помощью формы, и уточнить соответствующие им компоненты в SQL-запросе. Выполним проверку автора. Изначально пустому элементу `option` задан атрибут `value` со значением `""`, поэтому если содержимое поля в переменной `$_GET['author']` не равно пустой строке, то автор указан, и это необходимо учесть в запросе (`chapter7/admin/jokes/index.php`, фрагмент).

```
$placeholders = array();

if ($_GET['author'] != '') // Автор выбран
{
  $where .= " AND authorid = :authorid";
  $placeholders[':authorid'] = $_GET['author'];
}
```

Для добавления новой строки к существующей используется уже известный вам оператор конкатенации (. =). В этом примере к оператору `WHERE` добавляется условие, по которому содержимое поля `authorid` из таблицы `joke` должно совпадать со значением псевдопеременной `:authorid`. Пока с помощью метода `bindValue` установить требуемое общее значение `$_GET['author']` нельзя: у нас не подготовлен объект параметризированного запроса, из которого этот метод вызывается. Вследствие этого запрос все еще разбросан по трем разным строкам: `$select`, `$from` и `$where`. Позже вы объедините данные строки, чтобы создать подготовленное выражение, а пока сохраните параметризированные переменные в массив `$placeholders`, используя их имена в качестве индексов.

Теперь разберемся с категориями для шутки (`chapter7/admin/jokes/index.php`, фрагмент).

```
if ($_GET['category'] != '') // Категория выбрана.
{
  $from  .= ' INNER JOIN jokecategory ON id = jokeid';
  $where .= " AND categoryid = :categoryid";
  $placeholders[':categoryid'] = $_GET['category'];
}
```

Поскольку категории, на которые делятся шутки, хранятся в таблице `jokecategory`, эту таблицу следует добавить в объединяющий запрос. Для этого в конец переменной `$from` достаточно дописать строку `INNER JOIN jokecategory ON id = jokeid`. Она объединит таблицы `joke` и `jokecategory` при условии, что значения соответствующих им столбцов `id` и `jokeid` совпадают.

Объединяющий запрос сформирует как раз то условие, которое определяется с помощью формы (касающееся принадлежности шутки к указанной категории). Добавив его к переменной `$where`, вы можете потребовать, чтобы значение столбца `categoryid` в таблице `jokecategory` совпадало с идентификатором конкретной

[1] Поскольку MySQL считает истинным любое число, это же выражение можно записать как `WHERE 1`.

категории (:categoryid). Значение, которое подставляется на место параметризированной переменной, также хранится в массиве $placeholders.

Благодаря оператору LIKE, с которым вы познакомились в главе 2, работать с полученной строкой довольно просто (chapter7/admin/jokes/index.php, фрагмент).

```
if ($_GET['text'] != '') // Была указана какая-то искомая строка
{
  $where .= " AND joketext LIKE :joketext";
  $placeholders[':joketext'] = '%' . $_GET['text'] . '%';
}
```

Чтобы получить значение параметризированной переменной, разместите содержимое $_GET['text'] между двумя знаками процента (%). Помните, что оператор LIKE воспринимает этот знак как групповой символ, поэтому при поиске строки $_GET['text'] в поле joketext в расчет будут приниматься также строки, где до и после этого значения находится другой текст.

Сформировав все составляющие SQL-запроса, объединим их, чтобы получить и вывести нужные шутки (chapter7/admin/jokes/index.php, фрагмент).

```
  try
  {
    $sql = $select . $from . $where;
    $s = $pdo->prepare($sql);
    $s->execute($placeholders);
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при извлечении шуток.';
    include 'error.html.php';
    exit();
  }

  foreach ($s as $row)
  {
    $jokes[] = array('id' => $row['id'], 'text' => $row['joketext']);
  }

  include 'jokes.html.php';
  exit();
}
```

Обратите внимание на строку, выделенную полужирным начертанием. Поскольку массив $placeholders хранит значения всех псевдопеременных, вы можете передать их в метод execute сразу, а не указывать с помощью bindValue каждое в отдельности.

Рядом с каждой шуткой в шаблоне располагаются кнопки **Редактировать** и **Удалить**. Чтобы страница выглядела аккуратно, систематизируйте результаты внутри HTML-таблицы (chapter7/admin/jokes/jokes.html.php).

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Управление шутками: результаты поиска</title>
  </head>
  <body>
    <h1>Результаты поиска</h1>
    <?php if (isset($jokes)): ?>
      <table>
        <tr><th>Текст шутки</th><th>Действия</th></tr>
        <?php foreach ($jokes as $joke): ?>
        <tr>
          <td><?php htmlout($joke['text']); ?></td>
          <td>
            <form action="?" method="post">
              <div>
                <input type="hidden" name="id" value="<?php
                    htmlout($joke['id']); ?>">
                <input type="submit" name="action" value="Редактировать">
                <input type="submit" name="action" value="Удалить">
              </div>
            </form>
          </td>
        </tr>
        <?php endforeach; ?>
      </table>
    <?php endif; ?>
    <p><a href="?">Искать заново</a></p>
    <p><a href="..">Вернуться на главную страницу</a></p>
  </body>
</html>
```

Результаты поиска отобразятся в виде, представленном на рис. 7.6.

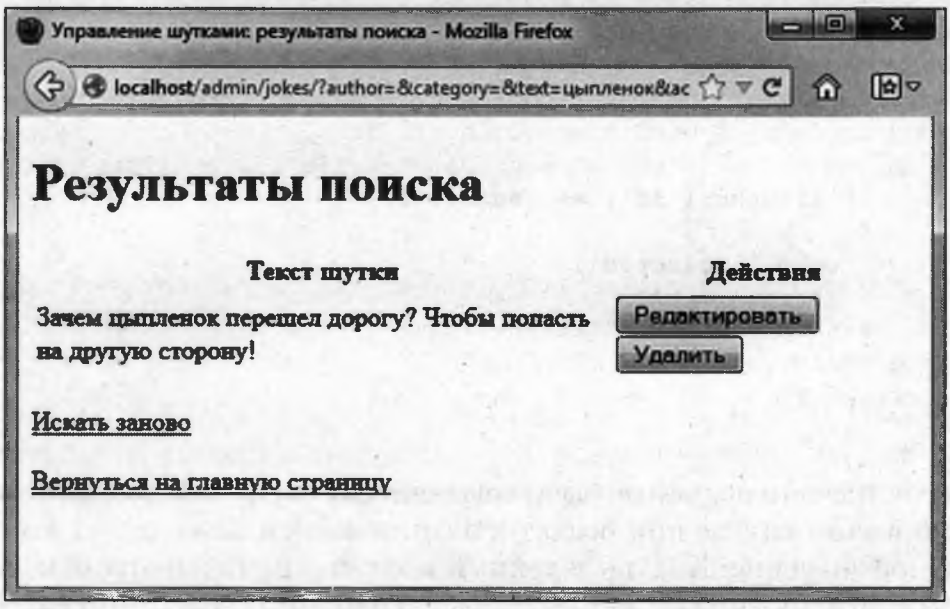


Рис. 7.6. Страница с результатами поиска

НЕЧЕГО ВЫВОДИТЬ

Попробуйте добавить в шаблон немного кода, позволяющего вывести страницу с корректным ответом для случая, если ни одна из шуток не подходит под указанные в форме критерии. На данном этапе на месте таблицы шаблон выведет пустое окно.

Добавление и редактирование шуток

В окне **Управление шутками** над формой поиска располагается ссылка для создания новой шутки (`chapter7/admin/jokes/searchform.html.php`, фрагмент).

```
<p><a href="?add">Добавить новую шутку</a></p>
```

Реализуем эту функцию. В целом ее код похож на тот, что вы использовали для создания новых записей в таблицах `author` и `category`, только, кроме ввода текста шутки, администратор получит возможность назначать автора и категорию.

Как и в предыдущих случаях, для создания и редактирования шуток допустимо использовать один шаблон формы. Рассмотрим его составляющие. Для добавления записи предназначено обычное текстовое поле, если запись редактируется, оно заполняется текстом шутки `$text` (`chapter7/admin/jokes/form.html.php`, фрагмент).

```
<div>
  <label for="text">Введите сюда свою шутку:</label>
  <textarea id="text" name="text" rows="3" cols="40"><?php
      htmlout($text); ?></textarea>
</div>
```

Далее администратору предлагается выбрать имя автора, который создал шутку (`chapter7/admin/jokes/form.html.php`, фрагмент).

```
<div>
  <label for="author">Автор:</label>
  <select name="author" id="author">
    <option value="">Выбрать</option>
    <?php foreach ($authors as $author): ?>
      <option value="<?php htmlout($author['id']); ?>"<?php
          if ($author['id'] == $authorid)
          {
            echo ' selected';
          }
          ?>><?php htmlout($author['name']); ?></option>
    <?php endforeach; ?>
  </select>
</div>
```

Вы уже встречали подобные раскрывающиеся списки в форме для поиска шуток, однако в этом случае при редактировании записи вам следует контролировать исходное значение. Код, выделенный полужирным начертанием, добавляет в тег `option` атрибут `selected`, если идентификатор соответствующего автора (`$author['id']`) совпадает с полем `$authorid` редактируемой шутки.

Далее администратор должен выбрать категории, к которым относится данная шутка. Раскрывающийся список для этого не подходит, поскольку одна и та же запись может находиться сразу в *нескольких* категориях. Здесь лучше использовать набор флажков (`<input type="checkbox">`) — по одному на каждый вариант. Количество нужных элементов заранее неизвестно, поэтому возникает вопрос: каким образом присвоить им атрибут `name`?

В данном случае для флажков выбирается *одна* переменная. Таким образом, название всех элементов становится одинаковым. Чтобы получить несколько значений из одного атрибута `name`, переменная объявляется **массивом**. Из главы 3 вы должны помнить, что массив — это переменная с ячейками, каждая из которых хранит свое значение. Для отправки элемента формы как части массива в конец атрибута `name` добавляются квадратные скобки. В данном примере это будет выглядеть так: `categories[]`.

СПИСОК С МНОЖЕСТВЕННЫМ ВЫБОРОМ

Еще один способ отправить массив — использовать тег `select multiple="multiple"`. В этом случае к значению атрибута `name` также добавляются квадратные скобки, а в качестве массива передаются все значения элементов `option`, выбранные пользователем.

Вы можете использовать данный подход для вывода категорий в виде списка элементов `option`. Однако имейте в виду: не все пользователи знают, что у них есть возможность выбирать несколько пунктов списка, удерживая клавишу Ctrl (или ⌘ в Mac OS X).

Теперь, когда все флажки имеют одинаковое название, необходимо выяснить, как определять выбранные элементы. Назначьте каждому флажку уникальное значение — идентификатор соответствующей категории в базе данных. Таким образом форма передаст массив, содержащий идентификаторы всех категорий, в которые добавляется новая шутка.

Добавим код, позволяющий редактировать шутку. Если шутка входит в соответствующую категорию, он укажет атрибут `selected`. В контроллере о выбранной категории будет сигнализировать значение `TRUE` переменной `$category['selected']` (`chapter7/admin/jokes/form.html.php`, фрагмент).

```
<fieldset>
  <legend>Категории:</legend>
  <?php foreach ($categories as $category): ?>
    <div><label for="category<?php htmlout($category['id']);
        ?>"><input type="checkbox" name="categories[]"
        id="category<?php htmlout($category['id']); ?>"
        value="<?php htmlout($category['id']); ?>"<?php
        if ($category['selected'])
        {
          echo ' checked';
        }
        ?>><?php htmlout($category['name']); ?></label></div>
  <?php endforeach; ?>
</fieldset>
```

В остальном шаблон для шуток работает аналогично тем формам, которые создавались ранее. Ниже приведен код целиком (chapter7/admin/jokes/form.html.php).

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title><?php htmlout($pageTitle); ?></title>
    <style type="text/css">
    textarea {
      display: block;
      width: 100%;
    }
    </style>
  </head>
  <body>
    <h1><?php htmlout($pageTitle); ?></h1>
    <form action="?<?php htmlout($action); ?>" method="post">
      <div>
        <label for="text">Введите сюда свою шутку:</label>
        <textarea id="text" name="text" rows="3" cols="40"><?php
            htmlout($text); ?></textarea>
      </div>
      <div>
        <label for="author">Автор:</label>
        <select name="author" id="author">
          <option value="">Выбрать</option>
          <?php foreach ($authors as $author): ?>
            <option value="<?php htmlout($author['id']); ?>"<?php
                if ($author['id'] == $authorid)
                {
                  echo ' selected';
                }
                ?>><?php htmlout($author['name']); ?></option>
          <?php endforeach; ?>
        </select>
      </div>
      <fieldset>
        <legend>Категории:</legend>
        <?php foreach ($categories as $category): ?>
          <div><label for="category<?php htmlout($category['id']);
              ?>"><input type="checkbox" name="categories[]"
              id="category<?php htmlout($category['id']); ?>"
              value="<?php htmlout($category['id']); ?>"<?php
              if ($category['selected'])
              {
                echo ' checked';
              }
              ?>><?php htmlout($category['name']); ?></label></div>
        <?php endforeach; ?>
      </fieldset>
      <div>
        <input type="hidden" name="id" value="<?php
            htmlout($id); ?>">
```

```
        <input type="submit" value="<?php htmlout($button); ?>">
      </div>
    </form>
  </body>
</html>
```

Внешний вид готовой формы приведен на рис. 7.7.

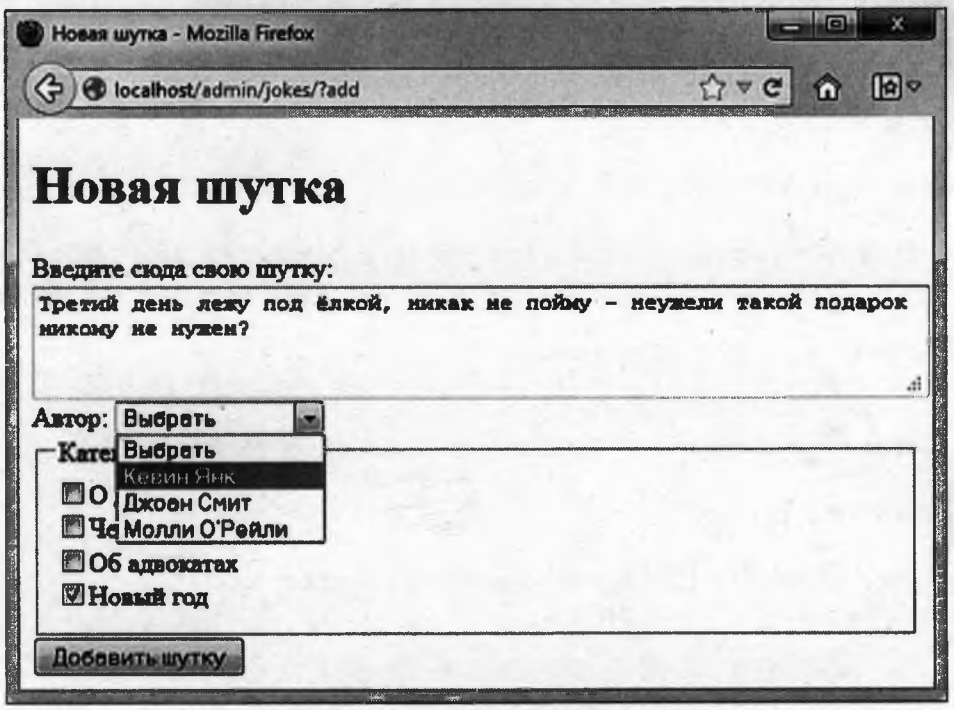


Рис. 7.7. Форма для работы с шутками

Вернемся к контроллеру, который отвечает за отображение и обработку данных формы в режимах добавления и редактирования записей.

После того как пользователь нажмет кнопку **Добавить шутку**, необходимо отобразить форму с пустыми полями. В этом коде для вас не должно быть ничего нового. Рассмотрите его внимательно и убедитесь, что вам все понятно. Если сомневаетесь в назначении той или иной переменной, откройте шаблон формы и узнайте, для чего она используется (`chapter7/admin/jokes/index.php`, фрагмент).

```
<?php
include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/magicquotes.inc.php';

if (isset($_GET['add']))
{
  $pageTitle = 'Новая шутка';
  $action = 'addform';
  $text = '';
  $authorid = '';
  $id = '';
  $button = 'Добавить шутку';
```

```
    include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';
    // Формируем список авторов.
    try
    {
      $result = $pdo->query('SELECT id, name FROM author');
    }
    catch (PDOException $e)
    {
      $error = 'Ошибка при извлечении списка авторов.';
      include 'error.html.php';
      exit();
    }

    foreach ($result as $row)
    {
      $authors[] = array('id' => $row['id'], 'name' => $row['name']);
    }

    // Формируем список категорий.
    try
    {
      $result = $pdo->query('SELECT id, name FROM category');
    }
    catch (PDOException $e)
    {
      $error = 'Ошибка при извлечении списка категорий.';
      include 'error.html.php';
      exit();
    }

    foreach ($result as $row)
    {
      $categories[] = array(
          'id' => $row['id'],
          'name' => $row['name'],
          'selected' => FALSE);
    }

    include 'form.html.php';
    exit();
  }
```

Обратите внимание на то, что каждому элементу `'selected'` в массиве `$categories` присваивается значение `FALSE`. В результате по умолчанию в форме не будет выбрана ни одна из категорий.

После нажатия кнопки **Редактировать** рядом с существующей шуткой контроллеру понадобится загрузить форму с полями, которые уже заполнены сохраненными значениями. По своей структуре данный участок кода похож на тот, который генерирует пустую форму (`chapter7/admin/jokes/index.php`, фрагмент).

```
  if (isset($_POST['action']) and $_POST['action'] == 'Редактировать')
  {
    include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

    try
```

```
{
  $sql = 'SELECT id, joketext, authorid FROM joke WHERE id = :id';
  $s = $pdo->prepare($sql);
  $s->bindValue(':id', $_POST['id']);
  $s->execute();
}
catch (PDOException $e)
{
  $error = 'Ошибка при извлечении информации о шутке.';
  include 'error.html.php';
  exit();
}
$row = $s->fetch();

$pageTitle = 'Редактировать шутку';
$action = 'editform';
$text = $row['joketext'];
$authorid = $row['authorid'];
$id = $row['id'];
$button = 'Обновить шутку';

// Формируем список авторов.
try
{
  $result = $pdo->query('SELECT id, name FROM author');
}
catch (PDOException $e)
{
  $error = 'Ошибка при извлечении списка авторов.';
  include 'error.html.php';
  exit();
}

foreach ($result as $row)
{
  $authors[] = array('id' => $row['id'], 'name' => $row['name']);
}

// Получаем список категорий, к которым принадлежит шутка.
try
{
  $sql = 'SELECT categoryid FROM jokecategory WHERE jokeid = :id'; ❶
  $s = $pdo->prepare($sql);
  $s->bindValue(':id', $id);
  $s->execute();
}
catch (PDOException $e)
{
  $error = 'Ошибка при извлечении списка выбранных категорий.';
  include 'error.html.php';
  exit();
}

foreach ($s as $row)
{
  $selectedCategories[] = $row['categoryid']; ❷
```

```
    }

    // Формируем список всех категорий.
    try
    {
      $result = $pdo->query('SELECT id, name FROM category');
    }
    catch (PDOException $e)
    {
      $error = 'Ошибка при извлечении списка категорий.';
      include 'error.html.php';
      exit();
    }

    foreach ($result as $row)
    {
      $categories[] = array(
          'id' => $row['id'],
          'name' => $row['name'],
          'selected' => in_array($row['id'], $selectedCategories)); ❸
    }

    include 'form.html.php';
    exit();
}
```

Помимо информации о шутке (id, текст, идентификатор автора), код извлекает также список категорий, в которые она входит. Рассмотрим некоторые фрагменты скрипта.

1. Запрос `SELECT` берет записи из промежуточной таблицы `jokecategory`. Он получает идентификаторы всех категорий, связанных с шуткой, которую пользователь хочет отредактировать.
2. В цикле `foreach` идентификаторы всех выбранных категорий сохраняются в массив `$selectedCategories`.
3. Пока формируется список всех категорий, используемых в форме в виде флажков, происходит проверка их идентификаторов, позволяющая определить, упоминаются ли данные категории в массиве `$selectedCategories`. Это делается автоматически с помощью встроенной функции `in_array`. Возвращаемое значение (`TRUE` или `FALSE`) сохраняется в элемент массива `'selected'`, который присутствует в каждой категории. В дальнейшем оно используется в шаблоне формы для выбора соответствующих флажков.

Вопрос с генерированием формы в каждом из двух режимов — добавления и редактирования записи — решен. Теперь рассмотрим тот скрипт контроллера, который обрабатывает переданные данные.

Поскольку массив (список выбранных категорий) вам предстоит передавать впервые, в коде для обработки формы будет содержаться несколько новых приемов. Начинается все довольно просто: вы добавляете шутку в таблицу `joke`. Имя автора указывается обязательно, поэтому вам необходимо проверить, содержит ли значение переменная `$_POST['author']`. Это не позволит администратору вы-

брать первый пункт в списке авторов, который имеет значение " ", то есть пустую строку (chapter7/admin/jokes/index.php, фрагмент).

```
if (isset($_GET['addform']))
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  if ($_POST['author'] == '')
  {
    $error = 'Вы должны выбрать автора для этой шутки.
        Вернитесь назад и попробуйте еще раз.';
    include 'error.html.php';
    exit();
  }

  try
  {
    $sql = 'INSERT INTO joke SET
        joketext = :joketext,
        jokedate = CURDATE(),
        authorid = :authorid';
    $s = $pdo->prepare($sql);
    $s->bindValue(':joketext', $_POST['text']);
    $s->bindValue(':authorid', $_POST['author']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при добавлении шутки.';
    include 'error.html.php';
    exit();
  }

  $jokeid = $pdo->lastInsertId();
```

В самом конце приведенного кода используется новый для вас метод lastInsertId. Он возвращает число, которое сервер MySQL назначил последней добавленной записи с помощью автоинкремента (AUTO_INCREMENT). Иными словами, данный метод получает идентификатор только что добавленной шутки, который вскоре вам пригодится.

Скорее всего, вы пока не совсем ясно представляете себе, какой код нужен для добавления записей в таблицу jokecategory с учетом выбранных флажков. Не удивительно. Вы ведь не знаете, как в PHP-коде значения элементов превращаются в переменные. Вам следует также понимать, что эти конкретные значения помещаются в массив.

Как правило, флажки передают свои значения только в том случае, если они выбраны. Выбранные флажки без атрибута value передают в соответствующие им переменные строку 'on'. В нашем случае значения для флажков указаны — это идентификаторы категорий.

Передача данных с помощью массива имеет определенные преимущества. В результате отправки формы вы получите одно из двух: массив идентификаторов для категорий, в которые добавлена шутка, или вообще ничего, если ни один из флажков не был отмечен.

Во втором случае ничего делать не нужно: поскольку не выбрана ни одна из категорий, то и в таблицу jokecategory добавить нечего. Если массив идентификаторов для категорий есть, то для его обработки воспользуйтесь циклом `foreach` и выполните команду `INSERT` для каждого идентификатора с помощью единого параметризированного запроса.

```
if (isset($_POST['categories']))
{
  try
  {
    $sql = 'INSERT INTO jokecategory SET
        jokeid = :jokeid,
        categoryid = :categoryid';
    $s = $pdo->prepare($sql);

    foreach ($_POST['categories'] as $categoryid)
    {
      $s->bindValue(':jokeid', $jokeid);
      $s->bindValue(':categoryid', $categoryid);
      $s->execute();
    }
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при добавлении шутки в выбранные категории.';
    include 'error.html.php';
    exit();
  }
}

header('Location: .');
exit();
}
```

Обратите внимание на использование переменной `$jokeid`, полученной из метода `lastInsertId`.

Код обработки формы, предназначенной для редактирования шуток, чем-то похож на вышеприведенный. Тем не менее в нем есть два существенных отличия:

1) для сохранения информации о шутке в таблице `joke` вместо команды `INSERT` используется команда `UPDATE`;

2) перед добавлением записей для выбранных в форме флажков из таблицы `jokecategory` удаляются все имеющиеся строки.

Вот как выглядит этот код (`chapter7/admin/jokes/index.php`, фрагмент). Прочитайте его внимательно и убедитесь, что вам все понятно.

```
if (isset($_GET['editform']))
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  if ($_POST['author'] == '')
  {
    $error = 'Вы должны выбрать автора для этой шутки.
        Вернитесь назад и попробуйте еще раз.';
```

```
    include 'error.html.php';
    exit();
  }

  try
  {
    $sql = 'UPDATE joke SET
        joketext = :joketext,
        authorid = :authorid
        WHERE id = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->bindValue(':joketext', $_POST['text']);
    $s->bindValue(':authorid', $_POST['author']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при обновлении шутки.';
    include 'error.html.php';
    exit();
  }

  try
  {
    $sql = 'DELETE FROM jokecategory WHERE jokeid = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при удалении лишних записей относительно категорий
шутки.';
    include 'error.html.php';
    exit();
  }

  if (isset($_POST['categories']))
  {
    try
    {
      $sql = 'INSERT INTO jokecategory SET
          jokeid = :jokeid,
          categoryid = :categoryid';
      $s = $pdo->prepare($sql);

      foreach ($_POST['categories'] as $categoryid)
      {
        $s->bindValue(':jokeid', $_POST['id']);
        $s->bindValue(':categoryid', $categoryid);
        $s->execute();
      }
    }
    catch (PDOException $e)
    {
      $error = 'Ошибка при добавлении шутки в выбранные категории.';
```

```
        include 'error.html.php';
        exit();
      }
    }

    header('Location: .');
    exit();
  }
```

Удаление шуток

Последнее, что осталось сделать, — добавить кнопку удаления рядом с каждой шуткой. Код контроллера, ответственный за эту функцию, почти полностью повторяет код, используемый для удаления авторов и категорий. Однако в нем есть небольшое отличие: вместе с шутками, выбранными в таблице `joke`, удаляются все связанные с ними записи из таблицы `jokecategory`.

Код добавления кнопки представлен ниже (`chapter7/admin/jokes/index.php`, фрагмент).

```
if (isset($_POST['action']) and $_POST['action'] == 'Удалить')
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  // Удаляем записи о категориях для этой шутки.
  try
  {
    $sql = 'DELETE FROM jokecategory WHERE jokeid = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при удалении шутки из категорий.';
    include 'error.html.php';
    exit();
  }

  // Удаляем шутку.
  try
  {
    $sql = 'DELETE FROM joke WHERE id = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при удалении шутки.';
    include 'error.html.php';
    exit();
  }

  header('Location: .');
  exit();
}
```

Подведение итогов

Созданная вами система управления содержимым умеет выполнять далеко не все. Например, она не способна вывести список шуток, которые не входят ни в одну категорию, а это окажется очень полезным, когда количество записей в базе данных начнет расти. Возможно, вы также захотите отсортировать шутки по другим критериям. Для реализации таких функций потребуется несколько более сложных приемов, о которых вы узнаете в главе 11.

НЕКОТОРЫЕ СТРОКИ КОДА ПРОПУЩЕНЫ

Если вы внимательно изучите код в архиве для этой главы, то заметите, что страница для вывода шуток (в каталоге `joke`) немного изменена. В ней отсутствует ссылка для добавления новых записей и кнопка Удалить. Это временная мера, поскольку данные функции не согласуются с новой структурой базы данных, которую вы спроектировали в настоящей главе. В следующей главе в разделе «Новый вызов: модерирование шуток» вы получите уникальную возможность придумать способ для управления шутками, которые добавил пользователь.

Если не принимать в расчет отдельные мелкие недочеты, то ваша система управления позволяет легко администрировать базу данных с шутками даже тем, кто не знаком ни с самой базой данных, ни с SQL. Благодаря набору страниц, написанных с помощью PHP и отображающих шутки на экране, CMS помогла создать полноценный сайт на основе базы данных, которым легко управлять без специальных знаний. Для компаний, занимающихся продажами через Интернет, подобная функциональность сайта окажется весьма кстати.

Фактически, от посетителей требуется лишь одно — уметь форматировать содержимое сайта. Конечно, вы могли позволить администратору сделать это прямо в форме для добавления шуток, используя HTML-код, но чтобы сохранить такое форматирование, потребовалось бы выводить текст шуток в чистом виде без применения функции `htmlout`.

Это неприемлемо по двум причинам. Во-первых, вам пришлось бы запретить пользователям добавлять новые шутки, поскольку злоумышленники получили бы возможность встраивать в них вредоносный код, а сайт выводил бы эти данные без фильтрации, не пропуская через функцию `htmlspecialchars`. Во-вторых, как уже упоминалось во введении к этой книге, одна из наиболее приятных особенностей сайта на основе базы данных заключается в том, что пользователи могут менять его содержимое, не разбираясь в тонкостях HTML. Требование специальных знаний для выполнения таких простых задач, как разделение текста на абзацы или применение наклонного начертания к нескольким словам, отдалило бы вас от этой цели.

В главе 8 вы узнаете, как с помощью PHP осуществить ввод форматированного текста, избегая использования HTML-разметки. Вы также усовершенствуете форму для добавления шуток и обеспечите безопасный прием данных от обычных посетителей сайта.

Глава 8
ФОРМАТИРОВАНИЕ СОДЕРЖИМОГО С ПОМОЩЬЮ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ

Итак, мы почти у цели. Вы спроектировали базу данных для хранения шуток, систематизировали их по категориям и установили авторство. Вы научились создавать веб-страницы для вывода записей по запросу посетителей сайта. Вы даже разработали набор скриптов, с помощью которых администратор способен управлять библиотекой шуток без специальных знаний о базах данных.

В результате у вас получился сайт, который освободил веб-мастера от необходимости держать большое количество связных HTML-шаблонов и постоянно обновлять в них информацию. Разметка HTML теперь отделена от данных, которые отображаются с ее помощью. Если вы захотите изменить внешний вид сайта, вам достаточно отредактировать HTML-код в созданных PHP-шаблонах. Изменения, внесенные в один файл, отвечающий, например, за подвал сайта, незамедлительно отразятся на разметке всех страниц. Нерешенной осталась только одна задача — **форматирование содержимого** без участия HTML.

На любых сайтах, кроме разве что самых простых, используется тот или иной вид форматирования содержимого. В самом простом случае текст разбивается на абзацы. Однако чаще пользователям, решившим добавить информацию на сайт, требуется более широкий набор инструментов: **полужирное** или *курсивное* начертание, гиперссылки и т. д.

Казалось бы, обеспечить поддержку этих требований в коде, который у вас есть на данном этапе, довольно легко. В нескольких последних главах вы использовали функцию `htmlout`, чтобы выводить данные, отправленные пользователями (`chapter7/jokes/jokes.html.php`, фрагмент).

```
<?php htmlout($joke['text']); ?>
```

В вашей власти разрешить администратору форматировать текст шуток с помощью HTML-кода и выводить содержимое базы данных как есть, без обработки.

```
<?php echo $joke['text']; ?>
```

Добавив такую строку в код, вы позволите администратору сайта использовать HTML-теги, чтобы изменять внешний вид текста на странице.

Но нужно ли это делать? Если пользователи вашего сайта получат полную свободу и станут предоставлять информацию с фрагментами HTML-кода, то вашей базе данных может быть причинен непоправимый ущерб. Разрешив простым пользователям, не имеющим достаточной подготовки, формировать содержимое

сайта, вы вскоре обнаружите, что его страницы наполняются некорректным, избыточным и попросту неприемлемым кодом. Действуя из самых лучших побуждений, пользователи способны разрушить разметку вашего сайта с помощью одного лишь случайного тега.

В этой главе вы познакомитесь с несколькими новыми функциями, которые предназначены для поиска и замены текстовых шаблонов внутри сайта. Узнаете, как с их помощью обеспечить поддержку простого языка разметки, который лучше подходит для форматирования текста. К концу этой главы вы получите готовую систему управления содержимым, которой сможет пользоваться любой обладатель браузера. Никаких знаний HTML при этом не потребуется.

Регулярные выражения

Чтобы реализовать язык разметки, необходимо написать PHP-код для поиска тегов в тексте шутки и замены их аналогами из HTML. Для решения подобных задач в PHP предусмотрена полноценная поддержка регулярных выражений.

Регулярное выражение — это небольшой фрагмент кода, который описывает образец текста, содержащийся в ваших данных, например в шутках. Регулярные выражения используются для поиска и замены образцов текста. Они поддерживаются многими системными окружениями, языками программирования, а также языками, предназначенными для веб-разработок, в число которых входит и PHP.

Популярность регулярных выражений основана на их практической пользе, но никак не на легкости применения: простыми их точно назвать нельзя. Для большинства людей, которые впервые сталкиваются с регулярными выражениями, они выглядят так, будто кто-то, не задумываясь, прошелся руками по клавиатуре.

Вот, например, относительно несложное регулярное выражение, которое соответствует любой строке, чье содержимое представляет корректный адрес электронной почты.

```
/^[\w\.\-]+@([\w\-]+\.)+[a-z]+$/i
```

Жутковато, не так ли? Не пугайтесь, прочитав данный раздел, вы сможете во всем разобраться.

Язык регулярных выражений настолько загадочный, что, когда вы им овладеете, вам покажется, будто вы пишите код, используя магические символы. Для начала рассмотрим простейшее выражение. Все, что оно делает, — ищет текст «PHP» (без кавычек):

```
/PHP/
```

Все проще простого: искомый текст окружен парой разделителей. Так сложилось, что в качестве разделителя в регулярных выражениях используется слеш (`/`), довольно часто его заменяют на символ решетки (`#`). На самом деле это может быть какой угодно символ, за исключением букв, цифр и обратного слеша (`\`). В данной главе мы остановились на первом варианте.

ЭКРАНИРОВАНИЕ РАЗДЕЛИТЕЛЕЙ

Если в качестве разделителя выбран слеш, а его требуется добавить в регулярное выражение, экранируйте символ с помощью обратного слеша (`\/`), иначе он будет воспринят как конец шаблона. То же касается и других возможных вариантов: если вы используете для разделения символ решетки, то внутри выражения он также должен экранироваться (`\#`).

Перед тем как приступить к детальному изучению регулярных выражений, вам необходимо познакомиться с соответствующими функциями в PHP. Самая простая из них — `preg_match`. Она помогает определить, **совпадает** ли регулярное выражение с конкретной строкой.

Рассмотрим следующий код (`chapter8/preg_match1/index.php`).

```
<?php
$text = 'PHP рулит!';

if (preg_match('/PHP/', $text))
{
  $output = '$text содержит строку &ldquo;PHP&rdquo;.';
}
else
{
  $output = '$text не содержит строку &ldquo;PHP&rdquo;.';
}

include 'output.html.php';
```

Поскольку строка, хранящаяся в переменной `$text`, содержит «PHP», совпадение будет найдено. В результате код выведет сообщение, показанное на рис. 8.1 (обратите внимание, что благодаря одинарным кавычкам PHP не подставляет вместо переменной `$text` ее значение).

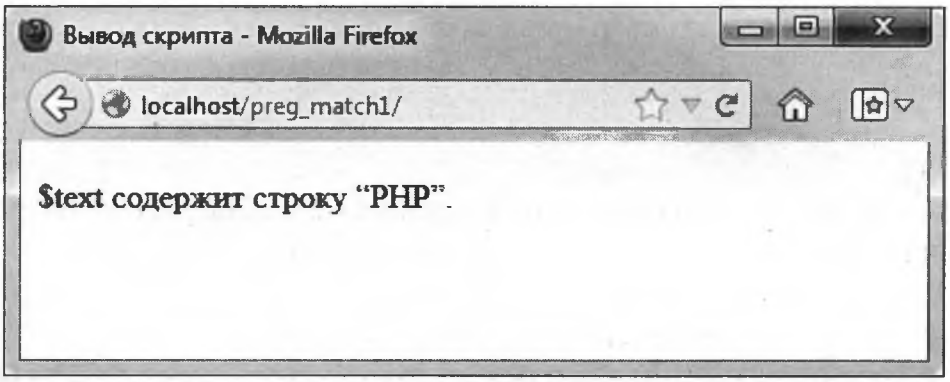


Рис. 8.1. Регулярное выражение обнаружило соответствие

По умолчанию регулярные выражения чувствительны к регистру: строчные символы в выражении соответствуют строчным символам в строке. То же касается и прописных символов. Чтобы игнорировать регистр во время поиска, используют модификатор шаблонов.

Модификаторы шаблонов — это односимвольные метки, которые устанавливают в выражении после закрывающего разделителя. Для поиска без учета регистра

служит модификатор `i`. Таким образом, в результате поиска выражения `/PHP/` будут найдены только те строки, которые содержат «PHP», выражению `/PHP/i` будет соответствовать текст «PHP», «php» и даже «pHp».

Проиллюстрируем это на примере (`chapter8/preg_match2/index.php`).

```
<?php
$text = 'Что такое Php?';

if (preg_match('/PHP/i', $text))
{
  $output = '$text содержит строку &ldquo;PHP&rdquo;.';
}
else
{
  $output = '$text не содержит строку &ldquo;PHP&rdquo;.';
}

include 'output.html.php';
```

Как показано на рис. 8.2, код выведет то же сообщение, что и в предыдущем примере, хотя в действительности текст содержит строку «Php».

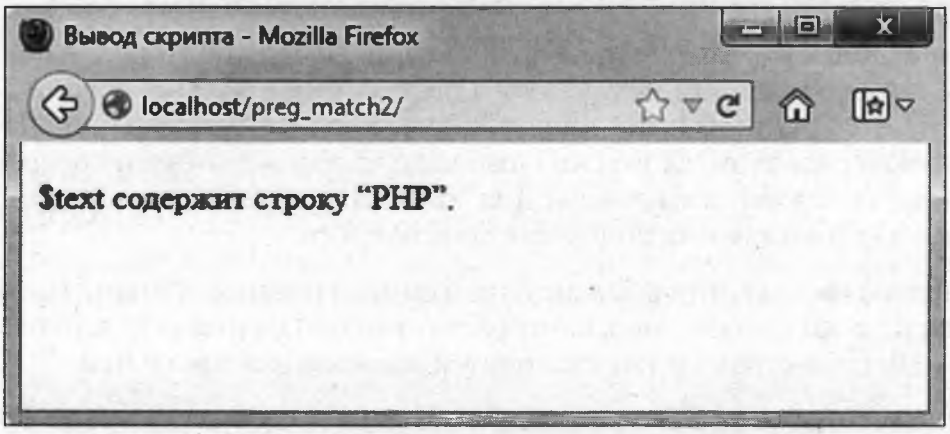


Рис. 8.2. Регулярное выражение не всегда должно быть разборчивым

Регулярные выражения — это почти что полноценный язык программирования. Они могут включать необычайно разнообразные символы, каждый из которых имеет собственное значение. Эти символы позволяют подробнейшим образом описать шаблон строки, поиском которой занимается функция `preg_match`. Чтобы вы поняли, о чем идет речь, рассмотрим более сложное регулярное выражение.

```
/^PH.*/
```

Циркумфлекс (`^`), точка (`.`) и звездочка (`*`) – все это специальные символы, которые имеют особое значение в контексте регулярных выражений. Так, циркумфлекс обозначает начало строки, точка воспринимается как любой символ, а звездочка указывает на ноль или больше символов, соответствующих предыдущему.

Учитывая вышесказанное, шаблон /^PH.*/ соответствует не только строке «PH», но и «PHP», «PHX», «PHP: гипертекстовый препроцессор» и любой другой, начинающейся с «PH».

Поначалу регулярные выражения кажутся запутанными и трудными для запоминания, поэтому вам не помешает хороший справочный материал. Исчерпывающий обзор регулярных выражений вы найдете в руководстве по PHP (http://php.net/manual/en/reference.pcre.pattern.syntax.php). В книге рассмотрены лишь основы.

Ниже представлены некоторые из наиболее востребованных специальных символов для регулярных выражений и примеры их использования.

^ (циркумфлекс) соответствует началу строки, используется для позиционирования и исключает любые символы.

$ (знак доллара) соответствует концу строки, как и в предыдущем случае, используется для позиционирования и исключает любые символы.

```
/PHP/    Соответствует 'PHP рулит!' и 'Что такое PHP?'.
/^PHP/   Соответствует 'PHP рулит!', но не 'Что такое PHP?'.
/PHP$/   Соответствует 'Я люблю PHP', но не 'Что такое PHP?'.
/^PHP$/  Соответствует только 'PHP' и никакой другой строке.
```

. (точка) — групповой знак, соответствует любому символу, кроме перехода на новую строку (`\n`)[1].

```
/^...$/   Соответствует любой трехсимвольной строке (без переносов).
```

*** (звездочка)** требует, чтобы предыдущий символ повторялся ноль и более раз. При совпадении регулярного выражения с частью строки звездочка становится **жадной** и поглощает как можно больше символов. Например, в строке «слово там, слово сям» шаблону `/слово.* /` будет соответствовать фрагмент «слово там, слово ». Для минимального совпадения, когда нужно только «слово там», используйте знак вопроса.

+ (плюс) требует, чтобы предыдущий символ повторялся один раз или более. Если не использовать знак вопроса, то при совпадении регулярного выражения с частью строки плюс становится жадным, как звездочка.

? (знак вопроса) делает предыдущий символ необязательным. Если поместить его после плюса или звездочки, он начнет требовать, чтобы совпадение для предыдущего символа было минимальным и охватывало как можно меньше знаков. Такое совпадение называют **нежадным**, или **ленивым**.

```
/бана?н/      Соответствует строкам 'банан' и 'банн', но не 'банаан'.
/бана+н/      Соответствует строкам 'банан' и 'банаан', но не 'банн'.
/бана*н/      Соответствует строкам 'банн', 'банан' и 'банааан', но не
'бнан'.
/^[a-zA-Z]+$/ Соответствует любой строке, состоящей исключительно из
букв (одной или больше).
```

| (вертикальная черта) обязывает регулярное выражение соответствовать только одному из шаблонов — тому, что находится справа или слева от нее.

(...) (круглые скобки) определяют неразрывную группу символов, к которым можно применять такие модификаторы, как *, + или ? (их указывают после закрывающей скобки). В дальнейшем на эти группы ссылаются в регулярных выражениях, чтобы получать соответствующие фрагменты строки.

[1] Если в конце регулярного выражения указать модификатор S, то точка будет соответствовать символу перехода на новую строку.

```
/^(да|нет)$/ Соответствует исключительно строкам 'да' и 'нет'.
/ба(на)+н/   Соответствует строкам 'банан' и 'банананан', но не 'бана'
или 'банаан'.
/ба(на|ну)+/ Соответствует строкам 'бана' и 'бануна', но не 'нануба'.
```

[...] (квадратные скобки) определяют класс символов, при этом ищется соответствие одному из символов, перечисленных внутри скобок. Такой класс может содержать точный список символов, например `[aqz]`, что эквивалентно `(a|q|z)`, или целый диапазон, например `[a-z]`, что эквивалентно `(a|b|c|...|z)`. Он также может соответствовать любому символу, который *не входит* в класс: для этого сразу после открывающей скобки помещают модификатор `^`. Например, `[^a]` соответствует любому символу, кроме 'a'.

```
/[12345]/     Соответствует строкам '1a' (содержит '1') и '39'
(содержит '3'), но не 'a' или '76'.
/[^12345]/    Соответствует строкам '1a' (содержит 'a') и '39'
(содержит '9'), но не '1' или '54'.
/[1-5]/       Эквивалентно /[12345]/.
/^[a-z]$/     Соответствует любой букве в нижнем регистре.
/^[^a-z]$/    Соответствует любой букве, которая записана не в нижнем
регистре.
/[0-9a-zA-Z]/ Соответствует любой строке, содержащей цифру или букву.
```

Если в регулярном выражении вы хотите использовать приведенные выше специальные символы в качестве литералов, экранируйте их, поместив перед ними обратный слеш (`\`):

```
/1\+1=2/ Соответствует любой строке, содержащей '1+1=2'.
/\$\$\$/ Соответствует любой строке, содержащей '$$$'.
```

Кроме того, существуют так называемые **управляющие последовательности**. Они соответствуют определенному типу символов или символам, которые сложно набрать с клавиатуры. Вот обозначения некоторых из них.

`\n` — переход на новую строку.

`\r` — возврат каретки.

`\t` — символ табуляции.

`\s` — любые пробельные символы, включая переход на новую строку, возврат каретки и табуляцию. То же, что и `[ \n\r\t]`.

`\S` — любой непробельный символ. То же, что и `[^ \n\r\t]`.

`\d` — любая цифра. То же, что и `[0-9]`.

`\D` — любой символ, кроме цифры. То же, что и `[^0-9]`.

`\w` — любой набор символов (слово). То же, что и `[a-zA-Z0-9_]`.

`\W` — любой набор символов (не слово). То же, что и `[^a-zAZ0-9_]`.

`\b` — обозначает границы слова: его начало и конец.

`\B` — обозначает место в строке, которое не является границей слова.

`\\` — соответствует обратному слешу. Регулярное выражение, которое реагирует на строку «\n», должно иметь вид `/\\n/`, но никак не `/\n/` (который соответствует переходу на новую строку). Если вам нужно найти строку «\\», регулярное выражение должно выглядеть как `/\\\\/`.

**\\ ПРЕВРАЩАЕТСЯ В **

Чтобы использовать регулярные выражения в функциях типа `preg_match`, их необходимо записать в виде обычной строки. Однако, как и в регулярных выражениях, в PHP одинарный обратный слеш обозначается с помощью двойного обратного слеша (`\\`). В связи с этим корректная запись регулярного выражения `/\\n/` примет вид `'/\\\\n/'`. PHP воспримет четыре обратные косые черты как две, а это именно то, что вам надо.

Теперь вы обладаете достаточным запасом знаний, чтобы вернуться к регулярному выражению с адресом электронной почты, которое вы видели в начале главы.

```
/^[\w\.\-]+@([\w\-]+\.)+[a-z]+$/i
```

`/` — открывающий разделитель, обозначает начало регулярного выражения.

`^` — начало строки, исключает любые символы перед адресом электронной почты.

`[\w\.\-]+` — фрагмент адреса с логином, состоит из одного или нескольких (знак «плюс») наборов символов (слов), а также точек или дефисов.

`@` — символьное обозначение так называемой собачки.

`([\w\-]+\.)+` — один или несколько (знак «плюс») субдоменов (таких как sitepoint., например), каждый из которых состоит из одного или нескольких наборов символов (слов) или дефисов с точкой в конце `(\.)`.

`[a-z]+` — домен верхнего уровня, например com, состоящий из одной или нескольких (знак «плюс») букв.

`$` — конец строки, исключающий символы после адреса электронной почты.

`/i` — закрывающий разделитель, обозначает конец регулярного выражения. Модификатор `i` после слеша указывает на то, что буквы в шаблоне `[a-z]` должны восприниматься без учета регистра.

Надеемся, что вам все понятно. Теперь вы можете разбивать на составные части шаблоны, которые кто-то уже написал. Но сумеете ли вы самостоятельно составлять подобные выражения? Не волнуйтесь: в оставшейся части данной главы вы познакомитесь со множеством разнообразных регулярных выражений и не заметите, как без труда начнете сами их записывать.

Замена текста с помощью регулярных выражений

Ваша цель — сделать форматирование шуток на сайте доступным для пользователей, которые не разбираются в HTML. Предположим, что слово, помещенное пользователем между двумя звездочками (например, `В *Новый* год...`), нужно выделить с помощью HTML-тегов (`В <em>Новый</em> год...`).

Поиск текстового фрагмента осуществляется с помощью функции `preg_match` и синтаксиса регулярных выражений, который вы только что изучили. Однако вам нужно не просто найти текст, а *определить* его форматирование и *заменить* звездочки на подходящие HTML-теги. Здесь для работы с регулярными выражениями понадобится еще одна функция PHP — `preg_replace`.

Как и `preg_match`, она принимает два параметра — регулярное выражение и строку, после чего пытается найти в указанной строке заданный шаблон. Содержимое третьего параметра используется для замены найденных совпадений.

Вот как выглядит синтаксис этой функции.

```
$newString = preg_replace(regExp, replaceWith, oldString);
```

Здесь `regExp` — это регулярное выражение, а `replaceWith` — строка, которая заменяет собой все совпадения, обнаруженные в `oldString`. В качестве результата функция возвращает новую строку со всеми произведенными заменами, присваивая ее переменной `$newString`.

Теперь вы готовы к тому, чтобы записать функцию для форматирования шуток.

Выделение в тексте

В главе 6 вы создали вспомогательную функцию под названием `htmlout` для вывода произвольного текста в виде HTML. Она находится в общем подключаемом файле **helpers.inc.php**. Поскольку теперь вам понадобится записывать форматированный текст с помощью HTML, добавьте в этот файл новую функцию (`chapter8/includes/helpers.inc.php`, фрагмент).

```
function markdown2html($text)
{
  $text = html($text);
  // Преобразуем форматирование на уровне обычного текста в HTML.
  return $text;
}
```

Синтаксис форматирования, который вы собираетесь поддерживать, — упрощенная версия языка разметки Markdown, созданного Джоном Грубером.

Markdown — это инструмент для преобразования текста в HTML, предназначенный для веб-писателей. Он позволяет считывать и вводить текст в облегченном формате, а затем конвертировать его в XHTML- или HTML-документ с правильной структурой (домашняя страница проекта Markdown, **http://daringfireball.net/projects/markdown/***).*

Исходя из принципа работы назовем нашу вспомогательную функцию `markdown2html`. Первым делом она должна вызвать `htmlout`, чтобы перевести любые элементы разметки в HTML-текст. На выходе не должно оставаться никаких фрагментов HTML, помимо тех, что сгенерируются с учетом форматирования[1].

[1] Формально это идет вразрез с одной из возможностей Markdown, которая позволяет встраивать HTML. Настоящий язык Markdown допускает передачу HTML-кода в браузер в тех случаях, когда требуется выполнить сложное форматирование, которое нельзя обеспечить посредством синтаксиса Markdown. Поскольку в нашем случае эта возможность не используется принципиально, точнее будет сказать, что мы осуществляем форматирование в стиле Markdown.

Для начала рассмотрим, как выполняется форматирование, позволяющее добавить в текст **полужирное** и *курсивное* начертание.

Чтобы добавить курсив в текст, в Markdown его помещают между двумя звездочками (*) или символами подчеркивания (_). Очевидно, что такие пары необходимо заменить тегами `em` и `/em`[1]. Для этого используют два регулярных выражения: одно для обработки звездочек, другое для обработки символов подчеркивания. Начнем с последних.

```
/_[^_]+_/
```

Разберем это выражение:

/ — начало регулярного выражения.

_ — символ, не имеющий особого значения, используется для обозначения начала курсивного текста.

[^_] — последовательность из одного или нескольких символов, не являющихся символами подчеркивания.

_ — еще один символ подчеркивания, обозначает конец курсивного текста.

/ — конец регулярного выражения.

Казалось бы, что данное регулярное выражение можно отправить в функцию `preg_replace`.

```
$text = preg_replace('/_[^_]+_/', '<em>курсив</em>', $text);
```

Однако есть небольшая проблема. В качестве второго аргумента функции выступает текст, который используется для замены совпадений. На данном этапе вы не имеете ни малейшего понятия о том, какой конкретно текст находится между тегами `em` и `/em`. Он часть строки, которая обрабатывается регулярным выражением.

В этом случае на помощь придет еще одна особенность функции `preg_replace`. Если заключить часть регулярного выражения в круглые скобки, то можно **захватить** соответствующий фрагмент в найденном тексте и применить его в строке для замены. Для это в коде необходимо использовать `$n`, где `n` — номер заключенного в скобки фрагмента регулярного выражения (1 — первый, 2 — второй, 99 — девяносто девятый). Рассмотрим следующий пример.

```
$text = 'банан';
$text = preg_replace('/(.*)(нан)/', '$2$1', $text);
echo $text; // выведет 'нанба'
```

В приведенном коде `$1` заменится текстом, который соответствует первой части регулярного выражения, заключенной в круглые скобки (`(.*)` — ноль или более любых символов, за исключением перехода на новую строку). В нашем случае это «ба». Вместо `$2` будет подставлена строка «нан» — это текст, который соответствует

[1] Возможно, вы больше привыкли использовать теги `b` и `i`, чтобы делать текст жирным и наклонным. Самый последний стандарт HTML рекомендует применять более выразительные аналоги – `strong` и `em`. Если жирный и наклонный текст не указывают соответственно на сильный и обычный акцент, то вместо этих тегов вы можете использовать `b` и `i`.

второй части регулярного выражения в круглых скобках. Таким образом, строка-заменитель `'$2$1'` даст значение «нанба».

Используя данный подход, можно сформировать текст, предназначенный для выделения, добавив в регулярное выражение пару круглых скобок.

```
/_([^_]+)_/
```

Эти скобки никак не повлияют на работу регулярного выражения. Они создают **группу** совпавших символов, которая будет использована в строке-заменителе (`chapter8/includes/helpers.inc.php`, фрагмент).

```
$text = preg_replace('/_([^_]+)_/', '<em>$1</em>', $text);
```

Шаблон для сопоставления и замены пар звездочек выглядит практически аналогично. Единственное, следует учесть, что звездочки имеют особое значение в контексте регулярных выражений, поэтому их следует экранировать с помощью обратного слеша (`chapter8/includes/helpers.inc.php`, фрагмент).

```
$text = preg_replace('/\*([^\*]+)\*/', '<em>$1</em>', $text);
```

Помимо выделения текста курсивом, Markdown также поддерживает выделение полужирным начертанием (теги `strong`). Для этого используются двойные звездочки или символы подчеркивания (`**полужирное подчеркивание**` или `__ полужирное подчеркивание __`). Вот регулярное выражение с использованием двойных символов подчеркивания.

```
/__(.+?)__/s
```

Двойные символы подчеркивания в начале и конце выглядят достаточно просто, но что происходит внутри круглых скобок?

Ранее в аналогичном шаблоне для выделения курсивом использовалась последовательность `[^_]+`. Она позволяла обозначить один или нескольких символов, которые не были символами подчеркивания. Такой подход работал, когда завершение выделенного текста обозначалось одним символом подчеркивания. Сейчас ситуация изменилась: одинарное подчеркивание может содержаться внутри выделенного текста (например, «__текст_с_сильным_акцентом__»). Следовательно, необходим другой способ для определения выделенного текста.

Как вариант, можно попытаться использовать комбинацию `.+` (один или более символов любого вида), изменив выражение следующим образом[1].

```
/__(.+)__/s
```

Проблема этого шаблона в том, что знак `+` является жадным. Из-за него фрагмент регулярного выражения поглотит столько символов, сколько сможет. Возьмем следующую шутку.

```
__Зачем__ цыпленок перешел __дорогу__? Чтобы __попасть на другую сторону!__
```

[1] Благодаря модификатору шаблона `S` в конце регулярного выражения, точка будет соответствовать любому символу, включая переход на новую строку.

В данном тексте вышеприведенное регулярное выражение распознает всего одно соответствие: от первого и до последнего двойного прочерка. Остальную часть шутки, включая оставшиеся двойные символы подчеркивания, поглотит жадное выражение . + и сделает выделенной.

Для решения этой проблемы нужно сделать так, чтобы символ + не был жадным, то есть поставить после него знак вопроса. Теперь вместо захвата при совпадении самого длинного отрезка текста, выражение . +? станет искать совпадения с как можно меньшим количеством символов в оставшейся части шаблона. Таким образом вы обеспечите отдельное сопоставление с каждым отрезком, выделенным двойным символом подчеркивания. В результате окончательная версия регулярного выражения будет выглядеть так:

```
/__(.+?)__/s
```

Аналогичный подход применяется для регулярного выражения с двойными звездочками. Ниже представлен итоговый код, с помощью которого текст выделяется полужирным начертанием (chapter8/includes/helpers.inc.php, фрагмент).

```
$text = preg_replace('/__(.+?)__/s', '<strong>$1</strong>', $text);
$text = preg_replace('/\*\*(.+?)\*\*/s', '<strong>$1</strong>', $text);
```

В завершение отметим, что первым делом следует преобразовать двойные звездочки и символы подчеркивания в пары тега strong, а затем их одиночные аналоги в пары тега em. Таким образом, функция markdown2html вначале установит полужирное начертание в тексте, а затем курсивное (chapter8/includes/helpers.inc.php, фрагмент).

```
function markdown2html($text)
{
  $text = html($text);

  // Полужирное начертание
  $text = preg_replace('/__(.+?)__/s', '<strong>$1</strong>', $text);
  $text = preg_replace('/\*\*(.+?)\*\*/s', '<strong>$1</strong>', $text);

  // Курсивное начертание
  $text = preg_replace('/_([^_]+)_/', '<em>$1</em>', $text);
  $text = preg_replace('/\*([^\*]+)\*/', '<em>$1</em>', $text);
  ⋮
  return $text;
}
```

Абзацы

Как и в случае с выделением текста, для обозначения начала и конца абзацев можно было бы подобрать определенные символы, но здесь имеет смысл использовать более простой подход. Пользователи вводят текст в поле формы, создавая абзацы с помощью клавиши Enter. Примем ее одиночное нажатие за разрыв строки (<br>), а двойное — за новый абзац ( ).

Как вы уже знаете, в регулярных выражениях переход на новую строку записывается как `\n`, пробельные символы могут обозначаться по-разному, например `\r` — возврат каретки, `\t` —табуляция. Какой из них добавится при нажатии клавиши Enter, зависит от операционной системы. Как правило, в компьютерах под управлением Windows переход на новую строку представлен сочетанием двух символов — возврата каретки и перехода на новую строку (`\r\n`). В Mac OS раньше применялся символ возврата каретки (`\r`), сейчас, как и в Linux, для этого служит одиночный символ `\n`[1].

Поскольку вы не знаете, какой из этих стилей использует браузер, нужно учесть каждый и выполнить некоторые преобразования:

```
// Преобразуем стиль Windows (\r\n) в Unix (\n).
$text = preg_replace('/\r\n/', "\n", $text);
// Преобразуем стиль Macintosh (\r) в Unix (\n).
$text = preg_replace('/\r/', "\n", $text);
```

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ В ДВОЙНЫХ КАВЫЧКАХ

Все регулярные выражения, которые вы встречали до сих пор, представляли собой строки, заключенные в одинарные кавычки. В большинстве случаев удобно воспользоваться автоматической заменой переменных, предлагаемой PHP, однако для регулярных выражений это чревато возникновением проблем.

Регулярные выражения и строки в двойных кавычках имеют специфическое значение. Так, `"\n"` — это строка, которая содержит символ перехода на новую строку, а `/\n/` — регулярное выражение, которое соответствует любой строке с таким символом. Запись регулярного выражения в виде строки в одинарных кавычках (`'/\n/'`) не создаст проблем, поскольку код `\n` в этом случае не приобретает другого значения, а вот при использовании двойных кавычек пришлось бы добавить еще один обратный слеш (`"/\\n/"`). В противном случае вы бы получили не одинарный слеш и нужное вам выражение `/\n/`, а сочетание слеша с символом `n`, которое обозначает переход на новую строку.

Ввиду сложности, к которой приводят строки в двойных кавычках, при написании регулярных выражений их лучше избегать. В приведенном выше коде двойные кавычки используются для строки-заменителя (`"\n"`), которая передается в качестве второго параметра в функцию `preg_replace`. Здесь они действительно нужны, поскольку нам требуется указать символ перехода на новую строку.

Заменив все разрывы строк символами перехода на новую строку, разделим текст на абзацы (если символы парные) или отдельные строки (если символы одиночные):

```
// Абзацы
$text = '<p>' . preg_replace('/\n\n/', '</p><p>', $text) . '</p>';
// Разрывы строки
$text = preg_replace('/\n/', '<br>', $text);
```

[1] На самом деле обозначение перехода на новую строку зависит не только от операционной системы, но и от программы. Возможно, вы сталкивались с таким неудобством программы Блокнот, когда при открытии в нем текстового файла исчезало разделение на абзацы и все строки сливались в одно целое. Более серьезные текстовые редакторы позволяют указать тип перехода на новую строку при сохранении файла.

Обратите внимание на дополнительные теги p и /p по обе стороны текста. Поскольку шутка может состоять из нескольких абзацев, следует убедиться в том, что она будет набрана в пределах одного текстового блока.

Итак, созданный вами код решил поставленную задачу: символы перехода на новую строку заменились обычными разрывами между строками и абзацами, которые и хотел получить пользователь. Как видите, для выполнения такого простого форматирования не нужны дополнительные знания.

Точно такой же результат можно получить более простым способом, без использования регулярных выражений, с помощью функции str_replace. По принципу работы она похожа на функцию preg_replace, только вместо шаблонов ищет строки.

```
$newString = str_replace(searchFor, replaceWith, oldString);
```

Таким образом, код для разбиения строк примет следующий вид (chapter8/includes/helpers.inc.php, фрагмент).

```
// Преобразуем стиль Windows (\r\n) в Unix (\n).
$text = str_replace("\r\n", "\n", $text);
// Преобразуем стиль Macintosh (\r) в Unix (\n).
$text = str_replace("\r", "\n", $text);

// Абзацы
$text = '<p>' . str_replace("\n\n", '</p><p>', $text) . '</p>';
// Разрывы строк
$text = str_replace("\n", '<br>', $text);
```

Используя функцию str_replace, не нужно придерживаться строгих правил, которые действуют для регулярных выражений. Именно поэтому там, где это возможно, применяйте ее или ее разновидность str_ireplace (для поиска с учетом регистра) вместо preg_replace.

Гиперссылки

Безусловно, поддержка гиперссылок в тексте шуток кажется лишней, но во многих других приложениях она вполне оправданна, поэтому рассмотрим и ее.

Вот как выглядит гиперссылка в Markdown[1].

```
[текст ссылки](URL ссылки)
```

Все просто: текст ссылки помещается в квадратные скобки, за которыми в круглых скобках следует адрес URL.

В действительности вы уже знаете все, что необходимо для поиска таких конструкций и их замены ссылками в формате HTML. Если вы уверены в своих силах, можете отложить на какое-то время книгу и попробовать справиться с задачей самостоятельно.

[1] Markdown поддерживает более сложный синтаксис, который позволяет вставить ссылку в конец документа в виде сноски. Однако в нашей упрощенной реализации Markdown эта возможность не рассматривается.

Для начала нам понадобится регулярное выражение, соответствующее ссылкам подобного вида. Оно выглядит следующим образом.

```
/\[([^\]]+)]\(([-a-z0-9._~:\/?#@!$&'()*+,;=%]+)\)/i
```

Это достаточно сложный шаблон. Присмотритесь к нему повнимательней и попытайтесь понять, как он работает. Если потребуется, возьмите ручку и разделите его на отдельные фрагменты: отметьте две пары круглых скобок (()), которые охватывают отрезки совпавшей строки — текст ссылки ($1) и ее адрес URL ($2).

Итак, разберем выражение по частям.

/ — как и во всех остальных случаях, это начало регулярного выражения.

\[— открывающая квадратная скобка. Поскольку такая скобка имеет особое значение в контексте регулярных выражений, она экранирована с помощью обратного слеша для правильной интерпретации.

([^\]]+) — круглые скобки означают, что текст, совпавший с заключенным в них фрагментом, доступен в строке-заменителе в переменной $1. Внутри скобок ищется текст ссылки. Еще одна закрывающая квадратная скобка говорит о том, что текст включает один или несколько символов, не являющихся закрывающей квадратной скобкой.

]\(— закрывающая квадратная скобка обозначает завершение текста ссылки, следующая за ней открывающая круглая скобка сигнализирует о начале адреса URL. Поскольку круглая скобка не используется для обозначения группы, ее следует экранировать с помощью обратного слеша. Для квадратной скобки в этом нет необходимости, потому что в данном выражении у нее нет неэкранированной пары.

([-a-z0-9._~:\/?#@!$&'()*+,;=%]+) — как и в предыдущем случае, круглые скобки говорят о том, что текст, совпавший с заключенным в них фрагментом, доступен в строке-заменителе, только на этот раз в переменной $2. Фрагмент внутри скобок представляет адрес URL[1]. Квадратные скобки содержат набор символов, которые могут входить в состав URL, а следующий за ними + сигнализирует о том, что таких символов должно быть не меньше одного. Многие специальные символы (., ?, +, *, (и)) внутри квадратных скобок теряют свое особое значение, поэтому экранировать их не нужно. Единственное исключение — слеш (/). Чтобы он не воспринимался как разделитель, закрывающий регулярное выражение, его следует записать в виде \/. Обратите также внимание, что дефис в списке символов идет в самом начале, иначе он станет обозначать диапазон символов, как в выражениях a-z и 0-9.

\) — экранированная закрывающая круглая скобка в конце адреса ссылки.

/i — конец регулярного выражения, за которым следует модификатор i, сигнализирующий о нечувствительности к регистру.

Запишем код для преобразования гиперссылки с помощью PHP (chapter8/includes/helpers.inc.php, фрагмент).

[1] Несмотря на то что используемый шаблон охватывает некоторые строки, которые не являются корректным адресом URL, его точность довольно высока. Если вы заинтересовались регулярными выражениями, ознакомьтесь с официальным стандартом для URL RFC 3986 (http://tools.ietf.org/html/rfc3986#appendix-B). В приложении B эти адреса разбираются с помощью внушительного регулярного выражения.

```
$text = preg_replace(
    '/\[([^\]]+)]\(([-a-z0-9._~:\/?#@!$&\'()*+,;=%]+)\)/i',
    '<a href="$2">$1</a>', $text);
```

Переменные `$1` и `$2` используются для замены распознанного текста и адреса ссылки соответственно. При записи строки с регулярным выражением применяются одинарные кавычки, поэтому, если они появятся в списке допустимых символов, вам придется их экранировать с помощью обратного слеша.

Собираем все воедино

Окончательный вариант вспомогательной функции для преобразования Markdown в HTML выглядит следующим образом (`chapter8/includes/helpers.inc.php`, фрагмент).

```
function markdown2html($text)
{
  $text = html($text);

  // Полужирное начертание
  $text = preg_replace('/__(.+?)__/s', '<strong>$1</strong>', $text);
  $text = preg_replace('/\*\*(.+?)\*\*/s', '<strong>$1</strong>', $text);

  // Курсивное начертание
  $text = preg_replace('/_([^_]+)_/', '<em>$1</em>', $text);
  $text = preg_replace('/\*([^\*]+)\*/', '<em>$1</em>', $text);

  // Преобразуем стиль Windows (\r\n) в Unix (\n).
  $text = str_replace("\r\n", "\n", $text);
  // Преобразуем стиль Macintosh (\r) в Unix (\n).
  $text = str_replace("\r", "\n", $text);

  // Абзацы
  $text = '<p>' . str_replace("\n\n", '</p><p>', $text) . '</p>';
  // Разрывы строк
  $text = str_replace("\n", '<br>', $text);

  // [текст ссылки](адрес URL)
  $text = preg_replace(
      '/\[([^\]]+)]\(([-a-z0-9._~:\/?#@!$&\'()*+,;=%]+)\)/i',
      '<a href="$2">$1</a>', $text);

  return $text;
}
```

Чтобы данным кодом было удобно пользоваться в PHP-шаблонах, введем функцию `markdownout`. Она вызовет `markdown2html` и выведет результат (`chapter8/includes/helpers.inc.php`, фрагмент).

```
function markdownout($text)
{
  echo markdown2html($text);
}
```

Теперь ее можно использовать в шаблонах для вывода шуток. Первый шаблон отображает результаты поиска на странице администратора (`chapter8/admin/jokes/jokes.html.php`).

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Управление шутками: результаты поиска</title>
  </head>
  <body>
    <h1>Результаты поиска</h1>
    <?php if (isset($jokes)): ?>
      <table>
        <tr><th>Текст шутки</th><th>Действия</th></tr>
        <?php foreach ($jokes as $joke): ?>
        <tr valign="top">
          <td><?php markdownout($joke['text']); ?></td>
          <td>
            <form action="?" method="post">
              <div>
                <input type="hidden" name="id" value="<?php
                    htmlout($joke['id']); ?>">
                <input type="submit" name="action" value="Редактировать">
                <input type="submit" name="action" value="Удалить">
              </div>
            </form>
          </td>
        </tr>
        <?php endforeach; ?>
      </table>
    <?php endif; ?>
    <p><a href="?">Искать заново</a></p>
    <p><a href="..">Вернуться на главную страницу</a></p>
  </body>
</html>
```

Второй выводит список шуток на общедоступной странице (`chapter8/jokes/jokes.html.php`).

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Список шуток</title>
  </head>
  <body>
    <p>Вот все шутки, которые есть в базе данных:</p>
    <?php foreach ($jokes as $joke): ?>
      <blockquote>
        <p>
          <?php markdownout($joke['text']); ?>
          (автор <a href="mailto:<?php htmlout($joke['email']); ?>"><?php
              htmlout($joke['name']); ?></a>)
        </p>
      </blockquote>
    <?php endforeach; ?>
  </body>
</html>
```

После того как все изменения внесены, опробуйте текстовое форматирование на практике: отредактируйте несколько шуток, добавив в них синтаксис Markdown, и убедитесь в том, что они корректно отображаются.

ИСПОЛЬЗУЙТЕ БИБЛИОТЕКУ ДЛЯ РАБОТЫ С MARKDOWN

Преимущество от использования синтаксиса форматирования вроде Markdown заключается в том, что для работы с ним существует множество открытого и свободного кода. Навыки создания регулярных выражений, которые вы только что обрели, еще не раз вам пригодятся, но если вы хотите поддерживать на своем сайте форматирование в стиле Markdown, то лучше отказаться от написания собственного кода. Воспользовавшись поиском в Google, вы легко найдете проект PHP Markdown (http://michelf.com/projects/php-markdown/) и отыщите на его страницах файл markdown.php. Загрузите его и поместите в каталог includes. В этом файле находится функция Markdown, которую вы сможете использовать внутри `markdown2html`.

```
function markdown2html($text)
{
  $text = html($text);
  include_once $_SERVER['DOCUMENT_ROOT'] . '/includes/markdown.php';
  return Markdown($text);
}
```

Воспользовавшись библиотекой Markdown, вы убедитесь, что форматирование по-прежнему работает, и, возможно, пожалеете о том, что уделили время изучению регулярных выражений. Однако будьте уверены – вы получили весьма полезные знания.

Передача данных в реальных условиях

Вы потратили немало времени и сил на то, чтобы создать такую систему управления содержимым сайта, использование которой не вызывает затруднений, при этом работать с ней разрешено только администраторам. Кроме того, даже несмотря на всю легкость обновления информации, не требующего знаний HTML, отправленные через форму данные, равно как и любой форматированный текст, все еще нужно преобразовывать в Markdown, что довольно утомительно и скучно.

Что будет, если вы предоставите доступ к форме для добавления шуток обычным посетителям сайта? Если помните, вы уже поступали так в главе 4, когда, изучая команду `INSERT`, размещали ссылку на форму на общедоступной странице со списком шуток. С тех пор вы эту возможность не использовали, поскольку она была несовместима с некоторыми изменениями, внесенными в структуру базы данных. Учитывая легкость форматирования текста, которую обеспечивает Markdown, было бы неплохо вернуть вашим посетителям доступ к форме для отправки шуток.

В следующей главе вы познакомитесь с системой контроля доступа к базе данных, благодаря которой ваш сайт сможет функционировать в реальных условиях. Что еще более важно, вы откроете доступ к его административным страницам только авторизированным пользователям. И пожалуй, самое интересное: вы пересмотрите сам принцип добавления шуток на сайт.

Глава 9
КУКИ, СЕССИИ И КОНТРОЛЬ ДОСТУПА

Куки (cookies) и сессии относятся к разряду таких технологий, которые представляются куда более страшными и сложными, чем они есть на самом деле. Данная глава призвана развеять этот миф. Постараемся объяснить простым языком, в чем суть данных технологий, как они работают и какую пользу из них можно извлечь. Все пояснения сопровождаются практическими примерами.

В конце вы опробуете новые инструменты и используете их для создания замысловатой системы, управляющей доступом к административным возможностям базы данных с шутками.

Куки

Большинство современных приложений при закрытии сохраняют некое *состояние*, например позицию окна на экране или имена последних пяти файлов, с которыми вы работали. Такие данные обычно содержатся в небольшом файле, который считывается при следующем запуске программы. После того как веб-разработчики перешли от статических страниц к полноценным интерактивным интернет-приложениям и вывели архитектуру сайтов на новый уровень, у них возникла необходимость в схожей функциональности для браузеров — так появились куки.

Куки — это пара «ключ — значение», связанная с заданным сайтом и хранящаяся на компьютере, где запущен клиент (браузер). Установленная сайтом, она передается при запросе к каждой его странице до тех пор, пока *не истечет срок ее действия*. Другие веб-ресурсы не могут получить доступ к куки, которые установлены вашим сайтом, равно как и ваш сайт не способен использовать куки других ресурсов. Вопреки распространенному мнению данная технология – относительно безопасное средство для хранения персональных данных. Сами по себе нарушить приватность пользователя куки не могут.

Жизненный цикл куки, сгенерированных с помощью PHP (рис. 9.1), формируется следующим образом.

1. Сначала браузер запрашивает URL, который указывает на PHP-скрипт. Внутри этого скрипта находится вызов функции `setcookie`, встроенной в PHP.
2. Страница, сгенерированная PHP-скриптом, отправляется обратно в браузер вместе с HTTP-заголовком `set-cookie`, содержащим имя (например, `mycookie`) и значение куки, которое следует установить.
3. Получив HTTP-заголовок, браузер создает и сохраняет заданное значение в виде куки с названием `mycookie`.

4. Последующие запросы к страницам сайта включают HTTP-заголовок `cookie`, который передает скриптам пару «ключ — значение» (mycookie=значение).
5. Получив запрос с заголовком `cookie`, PHP автоматически создает запись в массиве `$_COOKIE` с ключом `cookie`.

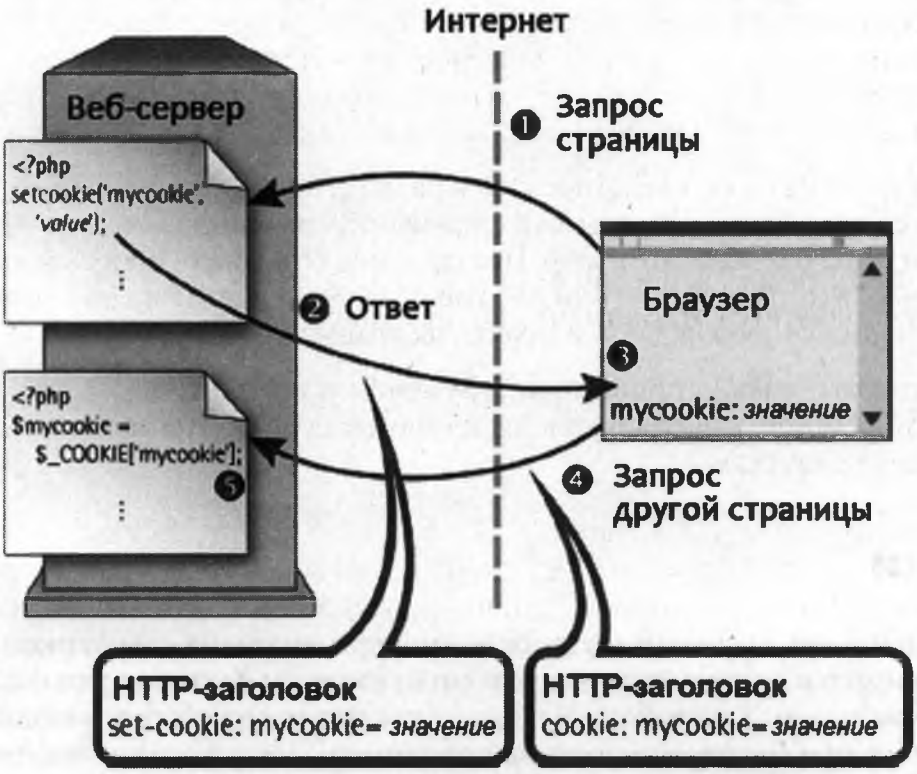


Рис. 9.1. Жизненный цикл куки

Иными словами, функция `setcookie` позволяет создать переменную, которая устанавливается автоматически при последующих запросах из того же браузера. Прежде чем перейти к реальному примеру, давайте рассмотрим эту функцию более подробно

```
setcookie(name[, value[, expiryTime[, path[, domain[, secure[,
httpOnly]]]]]])
```

КВАДРАТНЫЕ СКОБКИ ОБОЗНАЧАЮТ НЕОБЯЗАТЕЛЬНЫЙ КОД

В квадратные скобки (`[...]`) заключают фрагмент кода, который не является обязательным. При использовании данного синтаксиса в своем коде скобки использовать не нужно.

Как и функция `header`, с которой вы познакомились в главе 4, `setcookie` добавляет к странице HTTP-заголовок, поэтому *ее необходимо вызывать до отправки каких-либо данных*. Любая попытка вызвать функцию `setcookie` после отправки содержимого в браузер приведет к сообщению об ошибке. Обычно такие функции используются в скрипте контроллера до того фрагмента кода, где происходит вывод данных (например, с помощью подключения шаблона).

Единственный обязательный параметр `setcookie` — имя куки — `name`. При вызове функции только с этим параметром предыдущий экземпляр куки, хранящийся на стороне браузера, удаляется (если он существовал). Параметр `value` позволяет создать новый экземпляр куки или изменить значение в уже существующем.

По умолчанию куки остаются на клиентской стороне и продолжают передаваться при запросах к страницам, пока пользователь не закроет браузер. Если вы хотите, чтобы куки сохранялись и после завершения текущей сессии браузера, в параметре `expiryTime` необходимо указать количество секунд с 1 января 1970 года до того момента, когда они будут удалены автоматически. Текущее время в этом формате позволяет получить функция `time`. Например, чтобы куки существовали на протяжении одного часа, параметру `expiryTime` следует присвоить значение `time() + 3600`. Чтобы удалить этот экземпляр куки, укажите в качестве конечной даты прошедшее время, предположим, год назад (`time() - 3600 * 24 * 365`). Вот как используется эта методика на практике.

```
// Устанавливаем в качестве срока годности куки один год.
setcookie('mycookie', 'somevalue', time() + 3600 * 24 * 365);

// Удаляем куки.
setcookie('mycookie', '', time() - 3600 * 24 * 365);
```

Параметр `path` позволяет ограничить доступ к куки на вашем сервере определенным образом. Например, если вы установите для куки путь `'/admin/'`, то экземпляр куки будет содержаться в запросах только к тем скриптам, которые находятся в директории **admin** и ее субдиректориях. Обратите внимание на второй слеш: он не дает получить доступ к куки тем скриптам, которые находятся в других директориях и чьи названия также начинаются с `/admin`, например `/adminfake/`. Это полезно, когда приходится делить один сервер с другими пользователями, у каждого из которых есть своя домашняя веб-директория. Таким образом вы получаете возможность устанавливать куки, не открывая данные посетителей вашего сайта для скриптов, принадлежащих другим пользователям вашего сервера.

Параметр `domain` имеет схожее значение: он ограничивает доступ к куки заданным доменом. По умолчанию куки возвращаются только тому серверу, из которого они были отправлены. Однако большие компании часто располагают несколькими доменными именами (например, **www.example.com** и **support.example.com**). Чтобы создать экземпляр куки, доступ к которому получат страницы на обоих серверах, нужно присвоить параметру `domain` значение `'.example.com'`. Обратите внимание на первую точку: она не дает получить доступ к вашим куки другим сайтам, чьи доменные имена также заканчиваются на **example.com** (например, **fakeexample.com**).

Если присвоить параметру `secure` значение `1`, то куки станут отправляться только по защищенному (SSL) соединению (в этом случае URL должен начинаться с **https://**).

Значение `1`, присвоенное параметру `httpOnly`, сообщает браузеру о том, что код JavaScript на вашем сайте не должен видеть установленные куки. Как правило, код на языке JavaScript способен прочитать куки, отправленные сервером для текущей страницы, и в некоторых случаях это бывает полезно. Однако если злоумышленник внедрит на вашу страницу вредоносный JavaScript-код, ваши куки окажутся в опасности. Такой код способен прочесть конфиденциальные данные о ваших пользователях и применить их в нехороших целях. Установленная единица приведет

к тому, что экземпляр куки будет передаваться вашим PHP-скриптам в обычном режиме, но для присутствующего на сайте кода JavaScript он станет невидимым.

Как уже говорилось, все параметры, кроме name, необязательные, однако, указав значение для одного из них, вы должны сделать то же и для остальных. Например, чтобы вызвать функцию setcookie с параметром domain, вам также придется указать значение для expiryTime. Чтобы пропустить параметр, присвойте пустую строку '' для строковых значений (value, path и domain) или 0 для численных (expiryTime и secure).

Рассмотрим пример использования куки. Допустим, вы хотите вывести приветственное сообщение для тех, кто зашел на ваш сайт впервые. Для этого с помощью куки необходимо считать количество посещений, которые выполнил пользователь, и отобразить сообщение только в том случае, если куки не были установлены. Вот как выглядит этот код (chapter9/cookiecounter/index.php).

```
<?php

if (!isset($_COOKIE['visits']))
{
$_COOKIE['visits'] = 0;
}
$visits = $_COOKIE['visits'] + 1;
setcookie('visits', $visits, time() + 3600 * 24 * 365);

include 'welcome.html.php';
```

Сначала проверяется, установлена ли переменная $_COOKIE['visits']. Отрицательный ответ означает, что экземпляр куки visits в браузере пользователя не сохранен. В этом случае $_COOKIE['visits'] присваивается значение 0. Оставшаяся часть кода использует значение переменной $_COOKIE['visits'], которое определяет количество предыдущих посещений сайта пользователем.

Чтобы получить номер этого посещения, к значению $_COOKIE['visits'] добавляется 1. Полученная переменная $visits используется далее в шаблоне PHP. В конце функция setcookie присваивает экземпляру куки visits новую информацию о количестве посещений и устанавливает срок его хранения — один год.

Проделав вышеописанную работу, контроллер подключает шаблон welcome.html.php (chapter9/cookiecounter/welcome.html.php).

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Счетчик cookies</title>
  </head>
  <body>
    <p>
      <?php
      if ($visits > 1)
      {
```

```
        echo "Номер данного посещения: $visits.";
      }
      else
      {
        // Первое посещение
        echo 'Добро пожаловать на мой веб-сайт! Кликните здесь, чтобы
        узнать больше!';
      }
      ?>
    </p>
  </body>
</html>
```

На рис. 9.2 показано, как выглядит страница сайта для пользователя, который зашел на него впервые. При последующем посещении сайта пользователь увидит окно, изображенное на рис. 9.3.

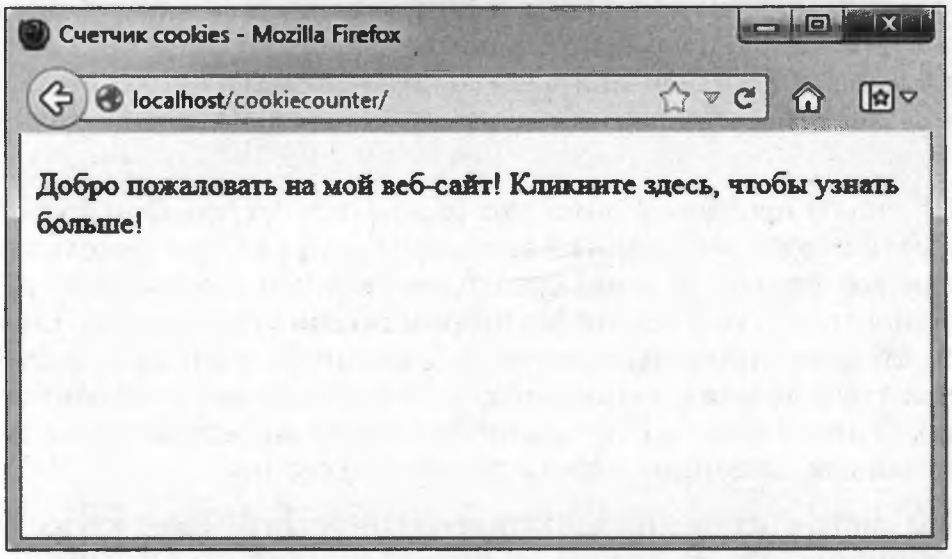


Рис. 9.2. Страница сайта, открывающаяся при первом посещении пользователя

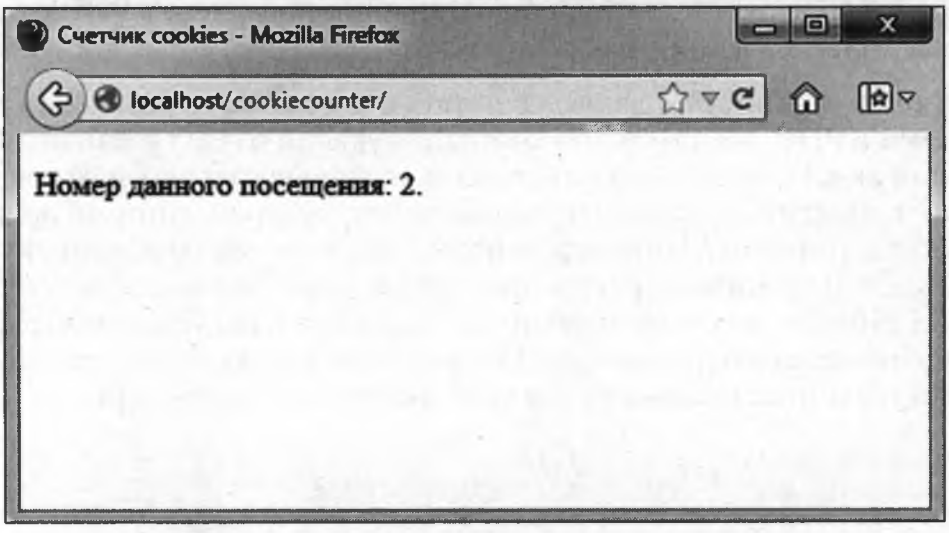


Рис. 9.3. Страница сайта, которую увидит пользователь при втором посещении

Прежде чем вы начнете самостоятельную работу с куки, примите во внимание, что их количество и размер для каждого сайта ограничены. Некоторые браузеры начинают удалять старые куки, чтобы освободить место для новых, когда их становится более 20 для одного сайта. Другие допускают не более 50 куки на ресурс, но прием новых экземпляров прекращают заранее. Кроме того, браузеры накладывают ограничение на размер куки для всех сайтов, поэтому ваши значения могут быть удалены только из-за того, что на другом ресурсе они слишком объемные.

Учитывая все вышесказанное, старайтесь свести к минимуму количество и размер куки, которые создаются вашим сайтом.

Сессии в PHP

В связи с описанными выше ограничениями куки не подходят для хранения больших объемов информации. Это может стать существенной проблемой для интернет-магазина, где куки помогают хранить товары в корзине, пока пользователь просматривает сайт. Чем больший заказ собирается сделать покупатель, тем выше вероятность того, что будут нарушены ограничения, накладываемые браузером на куки.

В PHP с такой проблемой помогают справиться **сессии**. Они позволяют хранить данные (в том числе объемные) не внутри браузера пользователя в виде куки, а на стороне веб-сервера. В этом случае единственным значением на клиентской стороне становятся куки с **идентификатором сессии** пользователя. Он представляет собой длинную строку, состоящую из букв и цифр, которая используется для однозначного определения пользователя в то время, пока он находится на сайте. Данная переменная служит в PHP для отслеживания последовательности запросов и загрузки данных, связанных с соответствующей сессией.

По умолчанию в браузер пользователя сессия устанавливает куки с идентификатором автоматически. Затем браузер отправляет это значение вместе с каждым последующим запросом к страницам сайта, чтобы PHP смог определить, к какой именно из текущих сессий (а их может быть несколько) относится данный запрос. Используя набор временных файлов на стороне веб-сервера[1], PHP отслеживает переменные (и их значения), зарегистрированные в рамках каждой сессии.

Перед тем как двинуться дальше и начать использовать средства для управления сессиями в PHP, убедимся, что соответствующий раздел в файле php.ini имеет надлежащий вид. Если вы используете один из универсальных пакетов (XAMPP или MAMP), описанных в главе 1, или работаете с сервером, который принадлежит вашему веб-хостингу, то велика вероятность того, что у вас присутствуют все нужные настройки. В противном случае файл php.ini необходимо открыть в текстовом редакторе и найти раздел, помеченный как `[Session]`. Вы увидите 20 параметров, которые начинаются со слова session. Большинство из них уже имеют подходящие значения, но есть несколько важных строк, которые стоит проверить:

```
session.save_handler      = files
session.save_path         = "C:\WINDOWS\TEMP"
session.use_cookies       = 1
```

[1] Если вам нужно организовать общий доступ к информации с разных веб-серверов, то PHP можно настроить таким образом, чтобы сессии хранились внутри базы данных MySQL.

Параметр `session.save_path` указывает PHP на место, где следует создавать временные файлы для отслеживания сессий. Лучше использовать директорию, которая присутствует в системе, иначе попытка создать сессию на любой из своих страниц приведет к сообщению об ошибке. В Mac OS X и Linux чаще всего выбирают директорию /tmp. В Windows можно указать C:\WINDOWS\TEMP или любую другую на ваш выбор (например, D:\PHP\SESSIONS). Подкорректировав настройки, перезапустите веб-сервер, чтобы изменения вступили в силу.

Теперь, когда настройки готовы, пройдемся по самым востребованным функциям для работы с сессиями в PHP. Чтобы найти идентификатор существующей сессии или создать новую сессию, необходимо вызвать функцию `session_start`. Отыскав нужный идентификатор, PHP восстановит переменные, которые принадлежат соответствующей сессии. Поскольку данная функция будет пытаться создать куки, ее вызов должен выполняться до того, как браузер получит любые другие данные (аналогично `setcookie`).

```
session_start();
```

Чтобы в рамках сессии создать переменную, используемую на всех страницах сайта для текущего пользователя, добавьте значение в специальный массив `$_SESSION`. Следующий код сохранит в текущей сессии переменную под названием `password`.

```
$_SESSION['password'] = 'mypassword';
```

Чтобы убрать переменную из текущей сессии, используйте функцию `unset`.

```
unset($_SESSION['password']);
```

И наконец, если вы хотите завершить сессию и удалить все переменные, зарегистрированные в процессе ее существования, вызовите функцию `session_destroy` и сотрите все сохраненные значения.

```
$_SESSION = array();
session_destroy();
```

Более подробную информацию об этих и других функциях, предназначенных для управления сессиями, вы найдете в соответствующем разделе руководства PHP (http://www.php.net/session).

Теперь, когда у вас есть под рукой основные функции, рассмотрим их работу на простом примере создания корзины для покупок. В этом примере контроллер наполняет данными два шаблона:

1) каталог товаров, на страницах которого можно добавлять товары в корзину;

2) страницу оформления заказа, которая отображает содержимое корзины пользователя для подтверждения покупки.

Заказ, оформленный на соответствующей странице, передается на обработку в систему, которая берет на себя все вопросы, связанные с приемом платежа и транспортировкой товара. Ее рассмотрение выходит за рамки данной книги,

но если вы хотите познакомиться с одной из таких систем поближе, обратите внимание на сервис PayPal (http://www.paypal.com/). Он довольно прост в настройке, и, чтобы разобраться в его документации, вам вполне хватит тех навыков, которые вы приобрели.

Начнем с той части контроллера, которая подготавливает список товаров для продажи в интернет-магазине. Каждый товар должен сопровождаться описанием и ценой. В нашем примере эта информация будет представлена в виде массива. В реальной системе вам, вероятно, пришлось бы хранить ее в базе данных, но мы воспользуемся упрощенным вариантом, чтобы сосредоточиться на коде, связанном с сессиями. Вы уже обладаете всеми необходимыми знаниями для составления списка товаров, основанного на базе данных, поэтому, если вы достаточно амбициозны, попробуйте написать его самостоятельно.

Вот как выглядит код для списка товаров (`chapter9/shoppingcart/index.php`, фрагмент).

```
<?php
include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/magicquotes.inc.php';

$items = array(
    array('id' => '1', 'desc' => 'Канадско-австралийский словарь',
        'price' => 24.95),
    array('id' => '2', 'desc' => 'Практически новый парашют (никогда не
    раскрывался)',
        'price' => 1000),
    array('id' => '3', 'desc' => 'Песни группы Goldfish (набор из 2 CD)',
        'price' => 19.99),
    array('id' => '4', 'desc' => 'Просто JavaScript (SitePoint)',
        'price' => 39.95));
```

Каждый элемент массива сам по себе является массивом и состоит из трех частей: уникального идентификатора, описания товара и его цены. И то, что этот код похож на массив, который вы могли получить в результате выполнения запроса к базе данных, не случайно.

На следующем этапе необходимо сохранить список товаров, которые пользователь поместил в корзину, в еще один массив. Эта переменная должна сохраняться в перерывах между посещениями сайта пользователем, поэтому вам понадобятся сессии (`chapter9/shoppingcart/index.php`, фрагмент).

```
session_start();
if (!isset($_SESSION['cart']))
{
  $_SESSION['cart'] = array();
}
```

Функция `session_start` начнет новую сессию (и установит куки с соответствующим идентификатором) или восстановит уже существующую (если таковая имеется) со всеми зарегистрированными в ней переменными. Затем код проверит наличие переменной `$_SESSION['cart']` и в случае отрицательного ответа инициализирует ее с помощью пустого массива, который представляет собой незаполненную корзину.

Это все, что вам понадобится для отображения списка товаров с использованием шаблона (`chapter9/shoppingcart/index.php`, фрагмент).

```
include 'catalog.html.php';
```

Теперь рассмотрим непосредственно шаблон (`chapter9/shoppingcart/catalog.html.php`).

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Каталог товаров</title>
    <style>
    table {
      border-collapse: collapse;
    }
    td, th {
      border: 1px solid black;
    }
    </style>
  </head>
  <body>
    <p>Ваша корзина содержит <?php
        echo count($_SESSION['cart']); ?> элементов.</p> ❶
    <p><a href="?cart">Просмотреть корзину</a></p> ❷
    <table border="1">
      <thead>
        <tr>
          <th>Описание товара</th>
          <th>Цена</th>
        </tr>
      </thead>
      <tbody>
        <?php foreach ($items as $item): ?>
          <tr>
            <td><?php htmlout($item['desc']); ?></td>
            <td>
              $<?php echo number_format($item['price'], 2); ?> ❸
            </td>
            <td>
              <form action="" method="post"> ❹
                <div>
                  <input type="hidden" name="id" value="<?php
                      htmlout($item['id']); ?>">
                  <input type="submit" name="action" value="Купить">
                </div>
              </form>
            </td>
          </tr>
        <?php endforeach; ?>
      </tbody>
    </table>
```

```
    <p>Все цены указаны в тугриках.</p>
  </body>
</html>
```

Остановимся на фрагментах, обозначенных цифрами.

1. Встроенная в PHP функция `count` позволяет вывести количество элементов в массиве, который хранится в переменной `$_SESSION['cart']`.
2. Ссылка, отображающая содержимое корзины пользователя. В системе, которая поддерживает работу с платежами, вы могли бы назвать ее **Перейти к оформлению заказа**.
3. Встроенная функция `number_format` помогает вывести цену с двумя знаками после запятой. Информацию об этой функции вы найдете в руководстве PHP (http://www.php.net/number_format).
4. Форма с кнопкой **Купить**, представленная для каждого товара из списка, отправляет уникальный идентификатор соответствующего элемента.

Список товаров, сформированный на основе представленного шаблона, показан на рис. 9.4.

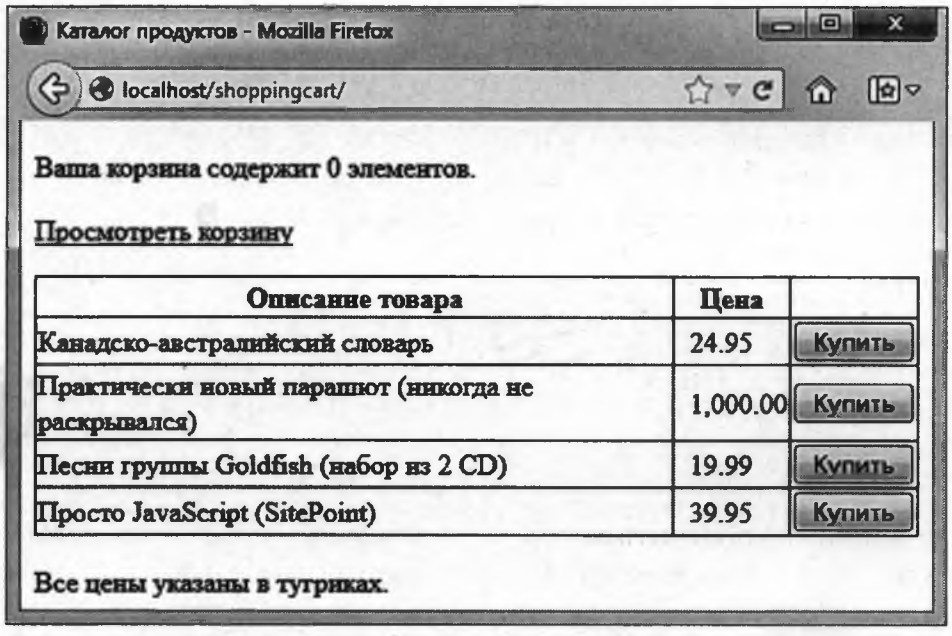


Рис. 9.4. Список товаров

Теперь, когда пользователь нажмет одну из кнопок **Купить**, в контроллер отправится форма с переменной `$_POST['action']`, получившей значение `'Купить'`. Вот как обрабатывается данное действие (`chapter9/shoppingcart/index.php`, фрагмент).

```
if (isset($_POST['action']) and $_POST['action'] == 'Купить')
{
  // Добавляем элемент в конец массива $_SESSION['cart'].
  $_SESSION['cart'][] = $_POST['id'];
  header('Location: .');
  exit();
}
```

Идентификатор товара добавляется в массив `$_SESSION['cart']` и отсылается браузеру на ту же страницу, но уже без данных, отправленных через форму. Таким образом, обновление страницы пользователем не приведет к повторному добавлению товара в корзину.

Когда пользователь нажмет ссылку **Просмотреть корзину**, контроллер получит запрос с переменной `$_GET['cart']`. Ниже приведен код для обработки этого действия (`chapter9/shoppingcart/index.php`, фрагмент).

```
if (isset($_GET['cart']))
{
  $cart = array();
  $total = 0;
  foreach ($_SESSION['cart'] as $id)
  {
    foreach ($items as $product)
    {
      if ($product['id'] == $id)
      {
        $cart[] = $product;
        $total += $product['price'];
        break;
      }
    }
  }

  include 'cart.html.php';
  exit();
}
```

Созданный массив `$cart` похож на `$items`, только в нем представлены товары, которые пользователь добавил в корзину. Для этого используются два вложенных цикла `foreach`. Первый цикл перебирает идентификаторы внутри `$_SESSION['cart']`. Второй запускается для каждого идентификатора и ищет товар с тем же id (`$product['id']`), но уже в массиве `$items`. Найденный товар помещается в массив `$cart`.

Одновременно код подсчитывает общую стоимость товаров в корзине. Каждый раз, когда второй цикл `foreach` находит товар в корзине, он добавляет значение его цены (`$product['price']`) к переменной `$total`. Команда `break` останавливает выполнение второго цикла `foreach`, когда искомый товар найден.

Как только массив `$cart` заполнится, загружается второй шаблон **cart.html.php** (`chapter9/shoppingcart/cart.html.php`). Его код очень похож на код шаблона для списка товаров, только элементы для перечисления берутся из массива `$cart`, а не `$items`. Внизу таблицы выводится общая сумма.

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Корзина</title>
    <style>
```

```
      table {
        border-collapse: collapse;
      }
      td, th {
        border: 1px solid black;
      }
      </style>
    </head>
    <body>
      <h1>Ваша корзина</h1>
      <?php if (count($cart) > 0): ?>
      <table>
        <thead>
          <tr>
            <th>Описание товара</th>
            <th>Цена</th>
          </tr>
        </thead>
        <tfoot>
          <tr>
            <td>Итого:</td>
            <td>$<?php echo number_format($total, 2); ?></td>
          </tr>
        </tfoot>
        <tbody>
          <?php foreach ($cart as $item): ?>
            <tr>
              <td><?php htmlout($item['desc']); ?></td>
              <td>
                $<?php echo number_format($item['price'], 2); ?>
              </td>
            </tr>
          <?php endforeach; ?>
        </tbody>
      </table>
      <?php else: ?>
      <p>Ваша корзина пуста!</p>
      <?php endif; ?>
      <form action="?" method="post">
        <p>
          <a href="?">Продолжить покупки</a> или
          <input type="submit" name="action" value="Очистить корзину">
        </p>
      </form>
    </body>
  </html>
```

Нажатие кнопки **Очистить корзину** приведет к удалению переменной `$_SESSION['cart']` (chapter9/shoppingcart/index.php, фрагмент).

```
if (isset($_POST['action']) and $_POST['action'] == 'Очистить корзину')
{
  // Опустошаем массив $_SESSION['cart'].
  unset($_SESSION['cart']);
  header('Location: ?cart');
  exit();
}
```

На рис. 9.5 показана страница сайта, которая выводится после того, как корзина заполнится товаром.

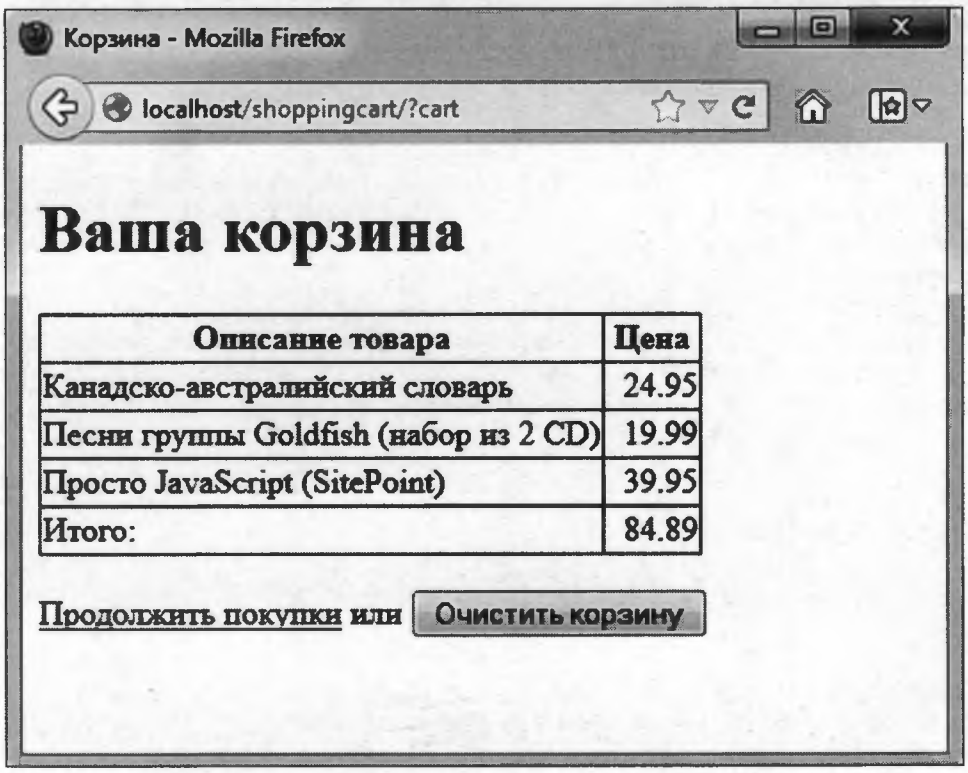


Рис. 9.5. Так выглядит наполненная корзина

Вот как выглядит пустая корзина (рис. 9.6).

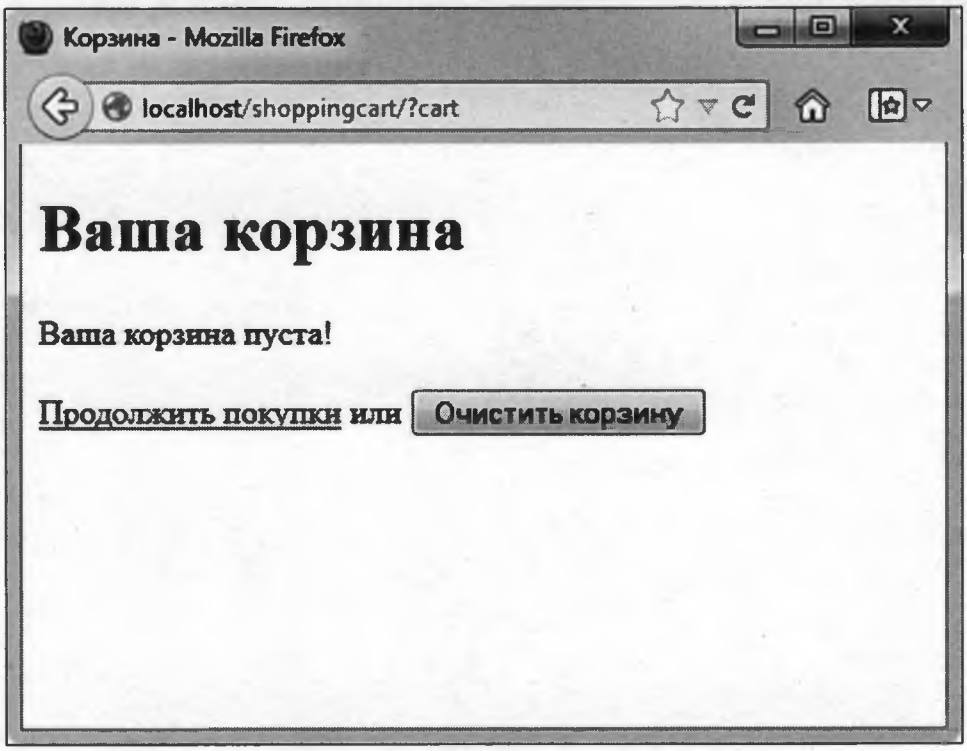


Рис. 9.6. Страница, сообщающая об очистке корзины

Ниже представлен код контроллера со всеми его составляющими (chapter9/shoppingcart/index.php).

```
<?php
include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/magicquotes.inc.php';

$items = array(
    array('id' => '1', 'desc' => 'Канадско-австралийский словарь',
        'price' => 24.95),
    array('id' => '2', 'desc' => 'Практически новый парашют (никогда не
раскрывался)',
        'price' => 1000),
    array('id' => '3', 'desc' => 'Песни группы Goldfish (набор из 2 CD)',
        'price' => 19.99),
    array('id' => '4', 'desc' => 'Просто JavaScript (SitePoint)',
        'price' => 39.95));

session_start();
if (!isset($_SESSION['cart']))
{
  $_SESSION['cart'] = array();
}

if (isset($_POST['action']) and $_POST['action'] == 'Купить')
{
  // Добавляем элемент в конец массива $_SESSION['cart'].
  $_SESSION['cart'][] = $_POST['id'];
  header('Location: .');
  exit();
}

if (isset($_POST['action']) and $_POST['action'] == 'Очистить корзину')
{
  // Опустошаем массив $_SESSION['cart'].
  unset($_SESSION['cart']);
  header('Location: ?cart');
  exit();
}

if (isset($_GET['cart']))
{
  $cart = array();
  $total = 0;
  foreach ($_SESSION['cart'] as $id)
  {
    foreach ($items as $product)
    {
      if ($product['id'] == $id)
      {
        $cart[] = $product;
        $total += $product['price'];
        break;
      }
    }
  }
```

```
    include 'cart.html.php';
    exit();
}

include 'catalog.html.php';
```

Контроль доступа

Еще одно важное преимущество сайтов, основанных на базе данных, заключается в том, что такие сайты позволяют своим владельцам обновлять содержимое откуда угодно и с помощью какого-угодно браузера. Однако в реальном мире, наполненном хакерами, которые только и ждут того, чтобы наполнить ваш сайт вирусами и порнографией, необходимо серьезно задуматься над вопросами безопасности администрирования.

Чтобы получить доступ к администрируемой части сайта, посетитель должен как минимум ввести логин и пароль. Это можно реализовать двумя способами:

1) настроить веб-сервер так, чтобы он требовал аутентификацию для открытия соответствующих страниц;

2) запросить у пользователя при необходимости данные учетной записи и затем проверить их с помощью PHP.

Если у вас есть доступ к конфигурации веб-сервера, то первый вариант, скорее всего, окажется наиболее простым в настройке. Второй вариант намного гибче: PHP позволяет создать форму для входа на сайт и внедрить ее на страницы, что упрощает изменение данных, необходимых для получения доступа, а также облегчает управление базой данных с авторизированными пользователями, где каждый имеет свои полномочия и привилегии.

В этом разделе вы защитите сайт с шутками с помощью аутентификации на основе логина и пароля, а также создадите сложную **ролевую систему контроля доступа**, позволяющую распределять полномочия среди пользователей.

«Какое отношение это все имеет к куки и сессиям?» — спросите вы. Ответ прост: вместо того чтобы запрашивать у пользователей логин и пароль каждый раз, когда они хотят просмотреть администрируемую страницу или выполнить важное действие, вы сохраните их полномочия, пока они находятся на сайте, используя механизм сессий.

Структура базы данных

В зависимости от типа приложения, над которым вы работаете, у вас может возникнуть необходимость в создании новой базы данных для хранения списка авторизированных пользователей и их паролей. В нашем случае, когда речь идет о сайте с шутками, соответствующая таблица уже есть — это таблица `author` (рис. 9.7).

Вместо того чтобы хранить авторов и пользователей отдельно, расширим уже существующую таблицу и позволим авторам входить в систему самостоятельно. Некоторые из авторов в базе данных могут ни разу не воспользоваться этой возможностью и будут существовать только для того, чтобы шутки сохранили авторство.

Другие, напротив, могут не добавлять записи и отвечать только за администрирование сайта. Но информацию о тех пользователях, которые выполняют обе задачи, целесообразно хранить в одной таблице, а не разбрасывать по разным местам.

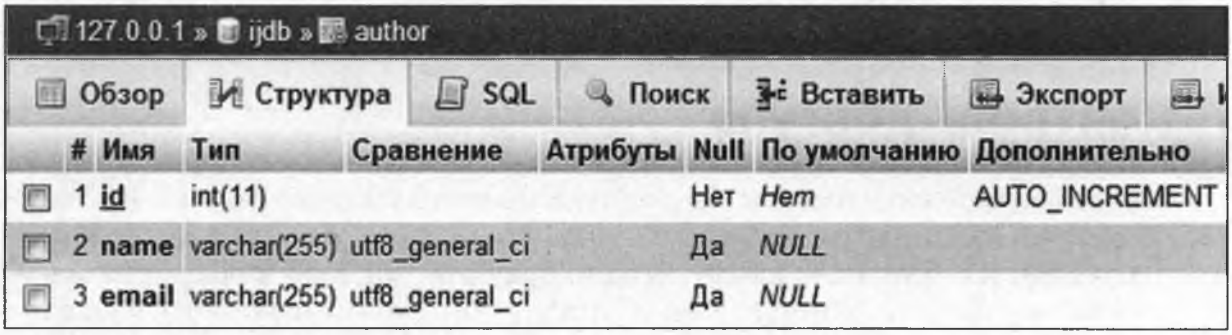


Рис. 9.7. Текущая структура таблицы `author`

В качестве логина можно использовать почтовый адрес. Для этого следует предварительно убедиться, что все авторы в базе данных имеют уникальные адреса. Выполните команду `ALTER TABLE ADD UNIQUE`[1], используя окно запросов в приложении phpMyAdmin.

```
ALTER TABLE author ADD UNIQUE (email)
```

Благодаря сделанным изменениям MySQL сгенерирует ошибку, если вы попытаетесь внести в базу данных имя автора с тем адресом электронной почты, который уже указан для другой записи.

Теперь осталось добавить в таблицу столбец для хранения паролей.

```
ALTER TABLE author ADD COLUMN password CHAR(32)
```

Обратите внимание, что модификатор `NOT NULL` для этого столбца не используется, поэтому у некоторых авторов строка с паролем может оставаться пустой. При написании PHP-кода такие авторы не получат разрешения входить в систему.

Еще один важный момент: для столбца указан тип `CHAR(32)`. В базе данных существует строгий запрет на хранение паролей в открытом виде. Многие имеют плохую привычку использовать один и тот же пароль на разных сайтах. Поэтому администраторы, как правило, *шифруют* предоставляемые им пароли, чтобы в случае взлома вашей базы данных злоумышленники не смогли воспользоваться полученной информацией для доступа к учетным записям пользователя на других ресурсах.

Типичный метод шифрования паролей заключается в использовании встроенной в PHP функции `md5`.

```
$scrambled = md5($password . 'ijdb');
```

[1] В этой главе вам понадобится уже готовая база данных со всеми изменениями, внесенными к настоящему моменту. Если вам нужно пересоздать ее, воспользуйтесь командами из файла ijdb.sql, который находится в архиве с кодом для этой главы.

Перед тем как зашифровать пароль к нему добавляется строка `'ijdb'`, благодаря чему вы получаете уверенность, что зашифрованные версии одного и того же пароля в разных базах данных будут выглядеть по-разному. Эксперты в области безопасности называют это **«солью»**, по аналогии с процессом приготовления: перед тем как взбивать яйца, добавьте щепотку соли.

ЗАМЕЧАНИЕ ОТ ЭКСПЕРТОВ В ОБЛАСТИ БЕЗОПАСНОСТИ

Любой эксперт подтвердит, что использование одной и той же «соли» для всех паролей в базе данных заведомо опасно. Злоумышленник, способный эту «соль» разгадать (например, при получении копии кода от вашего сайта), становится на шаг ближе к подбору оригинальных паролей на основе их зашифрованных версий. Конечно, те же эксперты посоветуют вам не писать код для обработки паролей, а воспользоваться проверенными решениями, которые они же и разработали. В рассматриваемом примере вы ознакомитесь только с базовым уровнем безопасности, требующим дальнейшего значительного улучшения (если вы вдруг этим заинтересуетесь).

Функция `md5` создает строку длиной 32 символа, состоящую, по всей видимости, из случайных букв и цифр. Несмотря на то что для каждого пароля всегда генерируется одна и та же строка, угадать те 32 символа, из которых она составлена, практически невозможно. Сохраняя в базе данных только подобные строки, вы сможете проверить, ввел пользователь правильный пароль или нет.

В отличие от `VARCHAR` тип `CHAR(32)` позволяет хранить только те значения, которые имеют длину 32 символа. Постоянство любого рода позволяет базе данных работать быстрее. Поскольку функция `md5` всегда генерирует 32-символьные строки, вы получите прирост в скорости выполняемых базой операций.

В MySQL есть своя функция `MD5`, которая выполняет ту же задачу. Чтобы убедиться в этом, попробуйте сохранить пароль для своей учетной записи (или добавить нового автора).

```
UPDATE author SET password = MD5('пароль ijdb')
WHERE id = 1
```

Заметьте, что к своему паролю вы должны добавить тот же суффикс, что используется в PHP-коде (в нашем случае это `'ijdb'`).

Теперь стоит подумать о том, где хранить список с действиями, дозволенными для выполнения каждому автору. Конечно, любому вошедшему в систему пользователю можно предоставить полную свободу действий, однако для большинства сайтов целесообразнее иметь дифференцированный контроль полномочий.

Итак, создадим новую таблицу со списком **ролей**, которые назначаются авторам. Каждый из них может играть несколько ролей. Например, автор, которому назначена роль редактора, способен редактировать шутки в вашей CMS. Такой механизм называется **ролевой системой контроля доступа**.

```
CREATE TABLE role (
  id VARCHAR(255) NOT NULL PRIMARY KEY,
  description VARCHAR(255)
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB
```

Каждая роль имеет короткий строковой идентификатор и более развернутое описание. Вот некоторые из них.

```
INSERT INTO role (id, description) VALUES
('Редактор', 'Добавление, удаление и редактирование шуток'),
('Администратор учетных записей', 'Добавление, удаление и редактирование
авторов'),
('Администратор сайта', 'Добавление, удаление и редактирование
категорий')
```

И наконец, вам понадобится промежуточная таблица, которая свяжет пользователей с их ролями по принципу отношения «многие ко многим».

```
CREATE TABLE authorrole (
  authorid INT NOT NULL,
  roleid VARCHAR(255) NOT NULL,
  PRIMARY KEY (authorid, roleid)
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB
```

Записав этот код, присвойте себе роль администратора учетных записей.

```
INSERT INTO authorrole (authorid, roleid) VALUES
(1, ' Администратор учетных записей')
```

На этом работу с базой данных можно считать законченной. Теперь сосредоточим внимание на PHP-коде, который использует новую структуру базы данных.

Код контроллера

Очевидно, что контроль доступа пригодится во многих других проектах, написанных на языке PHP. Поэтому, как и в случае с кодами для подключения к базе данных и вывода текста, логичнее выделить как можно большую часть этой функции в общий подключаемый файл, чтобы использовать ее в будущем.

Не станем гадать, какие именно функции должны быть общими, а сразу приступим к редактированию контроллера, будто подключаемый файл уже написан.

Как вы помните, стартовая страница панели администрирования представляет собой обычный HTML-файл, который отображает меню (рис. 9.8).

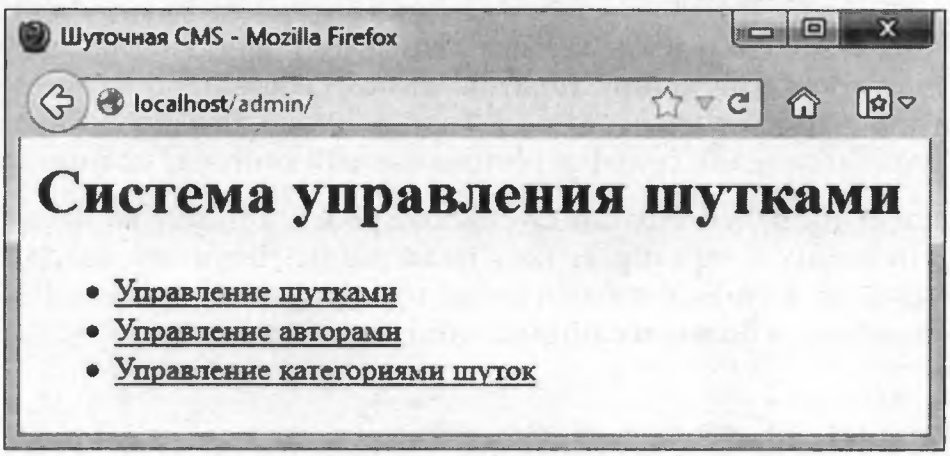


Рис. 9.8. Стартовая страница панели администрирования в защите не нуждается

По сути эта страница не содержит никакой конфиденциальной информации, поэтому, скорее всего, ваша интуиция подсказывает, что ее можно оставить без изменений. Однако все три ссылки указывают на скрипты контроллеров, которые отвечают за выполнение важных операций.

1. **/admin/jokes/index.php** ищет, отображает, добавляет, редактирует и удаляет шутки. Эти действия разрешено выполнять пользователям-редакторам.
2. **/admin/authors/index.php** отображает список авторов, добавляет имена новых, редактирует их данные и удаляет из системы. К выполнению этих действий допускаются пользователи, имеющие роль администратора учетных записей.
3. **/admin/categories/index.php** отображает, добавляет, редактирует и удаляет категории. Доступ к этим действиям имеют пользователи, обладающие ролью администратора сайта.

Таким образом, прежде чем приступить к выполнению задачи, каждый из этих контроллеров должен проверить, вошел ли текущий пользователь в систему и назначена ли ему необходимая роль. Если пользователь пока не прошел аутентификацию, следует отобразить форму для входа в систему, если прошел, но не обладает необходимой ролью, нужно вывести соответствующее сообщение об ошибке.

Допустим, у вас уже есть функции для выполнения всех обозначенных действий, тогда код примет следующий вид (`chapter9/admin/authors/index.php`, фрагмент).

```
 <?php
include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/magicquotes.inc.php';

require_once $_SERVER['DOCUMENT_ROOT'] . '/includes/access.inc.php';

if (!userIsLoggedIn())
{
  include '../login.html.php';
  exit();
}

if (!userHasRole('Администратор учетных записей'))
{
  $error = 'Доступ к этой странице имеет только администратор учетных
записей.';
  include '../accessdenied.html.php';
  exit();
}

: Остальная часть контроллера остается неизменной.
```

Похожий код добавляется и в два других контроллера, в каждом из которых указана соответствующая роль (`chapter9/admin/categories/index.php`, фрагмент, и `chapter9/admin/jokes/index.php`, фрагмент).

```
<?php
include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/magicquotes.inc.php';
```

```
require_once $_SERVER['DOCUMENT_ROOT'] . '/includes/access.inc.php';

if (!userIsLoggedIn())
{
  include '../login.html.php';
  exit();
}

if (!userHasRole('Администратор сайта'))
{
  $error = 'Доступ к этой странице имеет только администратор сайта.';
  include '../accessdenied.html.php';
  exit();
}

: Остальная часть контроллера остается неизменной.
```

```
<?php
include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/magicquotes.inc.php';

require_once $_SERVER['DOCUMENT_ROOT'] . '/includes/access.inc.php';

if (!userIsLoggedIn())
{
  include '../login.html.php';
  exit();
}

if (!userHasRole('Редактор'))
{
  $error = 'Доступ к этой странице имеет только редактор.';
  include '../accessdenied.html.php';
  exit();
}

: Остальная часть контроллера остается неизменной.
```

Проанализировав каждый из блоков, вы увидите, что вам предстоит написать три кода:

1) форму для входа **login.html.php**;

2) страницу для вывода сообщения о закрытом доступе **accessdenied.html.php**;

3) общий подключаемый файл **access.inc.php** с функциями:

- `userIsLoggedIn` для проверки входа пользователя в систему и правильности адреса электронной почты и пароля, если он только что отправил соответствующую форму;
- `userHasRole` для проверки роли вошедшего в систему пользователя.

Поскольку форма для входа и страница для вывода ошибки общие для всех трех контроллеров, их следует разместить в директории **admin** рядом с файлом **index.html**.

Код страницы для вывода ошибки прост. Все, что он делает, — выводит переменную `$error`, назначенную контроллером (`chapter9/admin/accessdenied.html.php`).

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Доступ запрещен</title>
  </head>
  <body>
    <h1>Доступ запрещен</h1>
    <p><?php htmlout($error); ?></p>
  </body>
</html>
```

Код, отвечающий за форму для входа, немного сложнее (`chapter9/admin/login.html.php`).

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Вход</title>
  </head>
  <body>
    <h1>Вход</h1>
    <p>Пожалуйста, войдите в систему, чтобы просмотреть страницу,
к которой вы обратились.</p>
    <?php if (isset($loginError)): ?> ❶
      <p><?php htmlout($loginError); ?></p>
    <?php endif; ?>
    <form action="" method="post"> ❷
      <div>
        <label for="email">Email: <input type="text" name="email"
            id="email"></label>
      </div>
      <div>
        <label for="password">Пароль: <input type="password" ❸
            name="password" id="password"></label>
      </div>
      <div>
        <input type="hidden" name="action" value="login"> ❹
        <input type="submit" value="Войти">
      </div>
    </form>
    <p><a href="/admin/">Вернуться на главную страницу</a></p> ❺
  </body>
</html>
```

Как и предполагалось, данная форма позволяет ввести адрес электронной почты и пароль. Остановимся на некоторых моментах.

1. Если пользователь отправил неверный адрес или пароль, ему будет отказано в доступе и он снова увидит формудля входа. Чтобы сообщить ему, что именно пошло не так, шаблон проверит наличие переменной `$loginError` и, если она существует, выведет ее содержимое над формой.
2. Тег `form` содержит пустой атрибут `action`, поэтому форма отправится по тому же адресу, по которому она была сгенерирована. Таким образом, если пользователь успешно вошел в систему, контроллер отобразит изначально запрошенную страницу.
3. Атрибут `type` во втором теге `input` имеет значение `password`. Благодаря этому браузер скроет текст пароля, который вводится в данное поле, и защитит его от стороннего наблюдателя.
4. Скрытое поле отправляется вместе с формой и сообщает функции `userIsLoggedIn` о том, что пользователь передал данные и попытался войти в систему. Как вариант, можно было указать атрибут `name="action"` для кнопки **Войти** прямо в теге `input` и затем отследить это значение. Однако в случае, когда пользователь отправляет форму нажатием клавиши **Enter** при редактировании одного из текстовых полей, данные, находящиеся в кнопке, не пересылаются. Использование скрытого поля гарантирует, что переменная `action` будет передана вне зависимости от того, как произошла отправка формы.
5. Если пользователь обратился к защищенной странице случайно или не знал, что доступ к ней ограничен, он не догадается об этом, пока не увидит форму для входа. На этот случай добавляется ссылка, которая позволяет вернуться в общедоступную часть сайта.

Эта форма отвечает за вход. Теперь необходимо предоставить посетителям возможность выйти из системы. Функция `userIsLoggedIn` определяет данные для аутентификации пользователей, однако точно так же она может применяться и в форме для выхода. Разместим такую форму внизу каждой защищенной страницы (`chapter9/admin/logout.inc.html.php`).

```
<form action="" method="post">
  <div>
    <input type="hidden" name="action" value="logout">
    <input type="hidden" name="goto" value="/admin/">
    <input type="submit" value="Выйти">
  </div>
</form>
```

Здесь снова используется скрытое поле с именем `action`, которое сообщает о намерениях пользователя. Поле `goto` указывает на то, куда необходимо перенаправить пользователя, который только что вышел из системы.

Добавьте эту форму на защищенные страницы, указав в конце каждого шаблона соответствующую команду `include` (`chapter9/admin/authors/authors.html.php`, фрагмент; `chapter9/admin/categories/categories.html.php`, фрагмент; `chapter9/admin/jokes/jokes.html.php`, фрагмент; `chapter9/admin/jokes/searchform.html.php`, фрагмент).

```
    ⋮
    <p><a href="..">Вернуться на главную страницу</a></p>
    <?php include '../logout.inc.html.php'; ?>
  </body>
</html>
```

```
    ⋮
    <p><a href="..">Вернуться на главную страницу</a></p>
    <?php include '../logout.inc.html.php'; ?>
  </body>
</html>
```

```
    ⋮
    <p><a href="..">Вернуться на главную страницу</a></p>
    <?php include '../logout.inc.html.php'; ?>
  </body>
</html>
```

```
    ⋮
    <p><a href="..">Вернуться на главную страницу</a></p>
    <?php include '../logout.inc.html.php'; ?>
  </body>
</html>
```

Библиотека функций

В этом разделе вы напишите общий подключаемый файл access.inc.php, который часто упоминается в коде. Теперь, когда остальные скрипты готовы, вы имеете довольно хорошее представление о том, что именно он должен делать.

Проанализируем его содержимое. В этом файле нужно определить две нестандартные функции.

`userIsLoggedIn`. Ее задача — вернуть `TRUE`, если пользователь вошел в систему, и `FALSE` — если нет. Она также необходима для обработки нескольких особых ситуаций.

- Если текущий запрос содержит данные, отправленные через форму входа (скрытое поле присвоит переменной `$_POST['action']` значение `'login'`), эта функция проверит правильность введенных логина и пароля. При положительном ответе она выполнит аутентификацию пользователя и вернет `TRUE`. В противном случае присвоит глобальной переменной `$loginError` соответствующее сообщение об ошибке и вернет `FASLE`.
- Если текущий запрос содержит данные, отправленные через форму выхода (скрытое поле присвоит переменной `$_POST['action']` значение `'logout'`), эта функция отменит аутентификацию пользователя и перенаправит браузер по адресу, указанному в переменной `$_POST['goto']`.

`userHasRole`. Подключается к базе данных и проверяет, назначена ли текущему пользователю та роль, которая передана в качестве параметра. Если роль назначена, функция вернет `TRUE`, если нет — `FALSE`.

Пройдемся по этим двум функциям и начнем сразу с нескольких строк кода (chapter9/includes/access.inc.php, фрагмент).

```
<?php

function userIsLoggedIn()
{
  if (isset($_POST['action']) and $_POST['action'] == 'login')
  {
```

Первым делом функция userIsLoggedIn проверит, отправлена ли форма для входа в систему (chapter9/includes/access.inc.php, фрагмент).

```
    if (!isset($_POST['email']) or $_POST['email'] == '' or
      !isset($_POST['password']) or $_POST['password'] == '')
    {
      $GLOBALS['loginError'] = 'Пожалуйста, заполните оба поля';
      return FALSE;
    }
```

На следующем этапе, прежде чем подключиться к базе данных, убедитесь, что пользователь ввел значения как для почтового адреса, так и для пароля. Если хотя бы одно из них не передано или является пустой строкой, то для глобальной переменной $loginError следует установить нужное значение с помощью специального массива $GLOBALS (он рассматривался в главе 6) и вернуть FALSE.

Проверив наличие адреса и пароля, приступайте к поиску соответствующего автора в базе данных. Первым делом зашифруйте переданный пароль, чтобы он совпадал с тем значением, которое хранится в базе данных (chapter9/includes/access.inc.php, фрагмент).

```
    $password = md5($_POST['password'] . 'ijdb');
```

Затем выполните запрос к базе данных, чтобы найти запись о соответствующем авторе. Поскольку это действие выполняется более одного раза, лучше написать для него отдельную функцию (chapter9/includes/access.inc.php, фрагмент).

```
function databaseContainsAuthor($email, $password)
{
  include 'db.inc.php';

  try
  {
    $sql = 'SELECT COUNT(*) FROM author
        WHERE email = :email AND password = :password';
    $s = $pdo->prepare($sql);
    $s->bindValue(':email', $email);
    $s->bindValue(':password', $password);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при поиске автора.';
    include 'error.html.php';
    exit();
```

```
    }

    $row = $s->fetch();

    if ($row[0] > 0)
    {
      return TRUE;
    }
    else
    {
      return FALSE;
    }
  }
```

Написание дальнейшей части кода не должно вас затруднить. Сначала вы соединяетесь с базой данных с помощью общего подключаемого файла db.inc.php и команды `include`[1]. Затем используете обычный подход для выполнения параметризированного запроса `SELECT`, который содержит переданные значения — почтовый адрес и зашифрованный пароль. Этот запрос посчитает количество записей в таблице `author`, у которых указанный адрес и пароль совпадают. Если результат окажется больше ноля, следует вернуть значение `TRUE`, в противном случае — значение `FALSE`.

Теперь пришло время вернуться к функции `userIsLoggedIn` и вызвать внутри нее новую функцию `databaseContainsAuthor` (`chapter9/includes/access.inc.php`, фрагмент).

```
    if (databaseContainsAuthor($_POST['email'], $password))
    {
```

Если база данных содержит автора, с указанным электронным адресом и паролем, значит, пользователь заполнил форму для входа верно и его следует впустить в систему. Существует два способа сделать это, и оба связаны с сессиями.

1. Переменная сессии используется в качестве флага (например, `$_SESSION['loggedIn'] = TRUE`). При последующих запросах проверяется наличие этой переменной: если она установлена, это говорит о том, что пользователь вошел в систему и функции `isUserLoggedIn` следует вернуть `TRUE`.
2. Флаг сохраняется вместе с переданными адресом и паролем в двух дополнительных сессионных переменных, наличие которых проверяется в следующих запросах. Если переменные есть, нужно удостовериться, что они по-прежнему совпадают с той информацией об авторе, которая хранится в базе данных с помощью функции `databaseContainsAuthor`. Если и эта проверка прошла успешно, то функция `isUserLoggedIn` может вернуть `TRUE`.

Первый способ обеспечивает более высокую скорость работы базы данных, поскольку информация о пользователе проверяется всего один раз при отправке формы для входа. Второй делает больший упор на безопасность: информация о пользователе сверяется с записью в базе данных при запросе к любой защищенной странице.

[1] Команда `include` используется вместо `include_once` потому, что переменная `$pdo`, которая создается внутри db.inc.php, недоступна за пределами функции `databaseContainsAuthor`. Если на каком-то другом участке кода нам понадобится соединение с базой данных, мы опять должны будем подключать db.inc.php.

В целом безопасный подход более предпочтительный. Он позволяет удалить автора, даже если тот находится в системе. Если используется первый способ, пользователь, вошедший в систему, остается там до тех пор, пока сессия остается активной. Это слишком высокая цена за незначительное улучшение производительности.

Вот код для второго способа (`chapter9/includes/access.inc.php`, фрагмент).

```
    session_start();
    $_SESSION['loggedIn'] = TRUE;
    $_SESSION['email'] = $_POST['email'];
    $_SESSION['password'] = $password;
    return TRUE;
  }
```

И наконец, в том случае, когда пользователь отправит форму с неверными значениями, нужно сделать так, чтобы он вышел из системы, установить соответствующее сообщение об ошибке и вернуть `FALSE` (`chapter9/includes/access.inc.php`, фрагмент).

```
  else
  {
    session_start();
    unset($_SESSION['loggedIn']);
    unset($_SESSION['email']);
    unset($_SESSION['password']);
    $GLOBALS['loginError'] =
        'Указан неверный адрес электронной почты или пароль.';
    return FALSE;
  }
}
```

На этом обработка аутентификации закончена. Теперь следует заняться формой для выхода из системы. Здесь все настолько просто, что код говорит сам за себя (`chapter9/includes/access.inc.php`, фрагмент).

```
if (isset($_POST['action']) and $_POST['action'] == 'logout')
{
  session_start();
  unset($_SESSION['loggedIn']);
  unset($_SESSION['email']);
  unset($_SESSION['password']);
  header('Location: ' . $_POST['goto']);
  exit();
}
```

В завершение, если ни одно из двух условий не выполняется, возьмите сессионные переменные, о которых говорилось ранее, и проверьте, находится ли пользователь в системе (`chapter9/includes/access.inc.php`, фрагмент).

```
session_start();
if (isset($_SESSION['loggedIn']))
{
  return databaseContainsAuthor($_SESSION['email'],
  $_SESSION['password']);
}
}
```

С функцией `userIsLoggedIn` вы разобрались. Теперь рассмотрим функцию `userHasRole`. В действительности она выполняет длинный запрос к базе данных, чтобы определить, назначена ли автору, чей почтовый адрес хранится в сессии, та роль, идентификатор которой передан в функцию. Этот запрос затрагивает сразу три таблицы, поэтому разберем его код отдельно.

```
SELECT COUNT(*) FROM author
INNER JOIN authorrole ON author.id = authorid
INNER JOIN role ON roleid = role.id
WHERE email = :email AND role.id = :roleId
```

Вначале объединяются таблицы `author` и `authorrole`, при этом сопоставляются поля `id` и `authorid` соответственно. Затем полученный результат объединяется с таблицей `role`, на этот раз ее столбец `id` сопоставляется с полем `roleid` таблицы `authorrole`. Когда все три таблицы объединены, оператор `WHERE` позволяет найти запись с нужным почтовым адресом и идентификатором роли.

Теперь напишите PHP-код, который выполнит данный запрос и интерпретирует полученный результат (`chapter9/includes/access.inc.php`, фрагмент).

```
function userHasRole($role)
{
  include 'db.inc.php';

  try
  {
    $sql = "SELECT COUNT(*) FROM author
        INNER JOIN authorrole ON author.id = authorid
        INNER JOIN role ON roleid = role.id
        WHERE email = :email AND role.id = :roleId";
    $s = $pdo->prepare($sql);
    $s->bindValue(':email', $_SESSION['email']);
    $s->bindValue(':roleId', $role);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при поиске ролей, назначенных автору.';
    include 'error.html.php';
    exit();
  }

  $row = $s->fetch();

  if ($row[0] > 0)
  {
    return TRUE;
  }
  else
  {
    return FALSE;
  }
}
```

Сохраните внесенные изменения и откройте одну из защищенных страниц. Если вы назначили себе роль администратора учетных записей, как рекомендовалось, то вы получите возможность перейти в раздел «Управление авторами» в панели администрирования. При попытке открыть другие разделы вы, как и положено, получите сообщение «Доступ запрещен». Нажмите кнопку **Выйти**. В результате вы окажитесь на главной странице панели управления и при желании перейти в любой из защищенных разделов увидите предложение снова войти в систему.

Если у вас возникли какие-либо проблемы, проверьте свой код, используя в качестве ориентира сообщения об ошибках. Ниже представлена полная версия файла **access.inc.php** (`chapter9/includes/access.inc.php`).

```
<?php

function userIsLoggedIn()
{
  if (isset($_POST['action']) and $_POST['action'] == 'login')
  {
    if (!isset($_POST['email']) or $_POST['email'] == '' or
      !isset($_POST['password']) or $_POST['password'] == '')
    {
      $GLOBALS['loginError'] = 'Пожалуйста, заполните оба поля';
      return FALSE;
    }

    $password = md5($_POST['password'] . 'ijdb');

    if (databaseContainsAuthor($_POST['email'], $password))
    {
      session_start();
      $_SESSION['loggedIn'] = TRUE;
      $_SESSION['email'] = $_POST['email'];
      $_SESSION['password'] = $password;
      return TRUE;
    }
    else
    {
      session_start();
      unset($_SESSION['loggedIn']);
      unset($_SESSION['email']);
      unset($_SESSION['password']);
      $GLOBALS['loginError'] =
          'Указан неверный адрес электронной почты или пароль.';
      return FALSE;
    }
  }

  if (isset($_POST['action']) and $_POST['action'] == 'logout')
  {
    session_start();
    unset($_SESSION['loggedIn']);
    unset($_SESSION['email']);
    unset($_SESSION['password']);
    header('Location: ' . $_POST['goto']);
    exit();
  }
```

```
  session_start();
  if (isset($_SESSION['loggedIn']))
  {
    return databaseContainsAuthor($_SESSION['email'],
    $_SESSION['password']);
  }
}

function databaseContainsAuthor($email, $password)
{
  include 'db.inc.php';

  try
  {
    $sql = 'SELECT COUNT(*) FROM author
        WHERE email = :email AND password = :password';
    $s = $pdo->prepare($sql);
    $s->bindValue(':email', $email);
    $s->bindValue(':password', $password);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при поиске автора.';
    include 'error.html.php';
    exit();
  }

  $row = $s->fetch();

  if ($row[0] > 0)
  {
    return TRUE;
  }
  else
  {
    return FALSE;
  }
}

function userHasRole($role)
{
  include 'db.inc.php';

  try
  {
    $sql = "SELECT COUNT(*) FROM author
        INNER JOIN authorrole ON author.id = authorid
        INNER JOIN role ON roleid = role.id
        WHERE email = :email AND role.id = :roleId";
    $s = $pdo->prepare($sql);
    $s->bindValue(':email', $_SESSION['email']);
    $s->bindValue(':roleId', $role);
    $s->execute();
  }
  catch (PDOException $e)
```

```
    {
      $error = 'Ошибка при поиске ролей, назначенных автору.';
      include 'error.html.php';
      exit();
    }

    $row = $s->fetch();

    if ($row[0] > 0)
    {
      return TRUE;
    }
    else
    {
      return FALSE;
    }
}
```

Управление паролями и ролями

Теперь, когда вы добавили в базу данных пароли и роли, нужно обновить страницы для управления авторами, чтобы иметь возможность изменять эти новые значения.

Начнем с того, что добавим в форму для редактирования информации об авторе поле Задать пароль и флажки для выбора назначаемых ролей (`chapter9/admin/authors/form.html.php`).

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title><?php htmlout($pageTitle); ?></title>
  </head>
  <body>
    <h1><?php htmlout($pageTitle); ?></h1>
    <form action="?<?php htmlout($action); ?>" method="post">
      <div>
        <label for="name">Имя: <input type="text" name="name"
            id="name" value="<?php htmlout($name); ?>"></label>
      </div>
      <div>
        <label for="email">Email: <input type="text" name="email"
            id="email" value="<?php htmlout($email); ?>"></label>
      </div>
      <div>
        <label for="password">Задать пароль: <input type="password"
            name="password" id="password"></label>
      </div>
      <fieldset>
        <legend>Roles:</legend>
        <?php for ($i = 0; $i < count($roles); $i++): ?>
          <div>
            <label for="role<?php echo $i; ?>"><input type="checkbox"
```

```
            name="roles[]" id="role<?php echo $i; ?>"
            value="<?php htmlout($roles[$i]['id']); ?>"<?php
            if ($roles[$i]['selected'])
            {
              echo ' checked';
            }
            ?>><?php htmlout($roles[$i]['id']); ?></label>:
            <?php htmlout($roles[$i]['description']); ?>
          </div>
        <?php endfor; ?>
      </fieldset>
      <div>
        <input type="hidden" name="id" value="<?php
            htmlout($id); ?>">
        <input type="submit" value="<?php htmlout($button); ?>">
      </div>
    </form>
  </body>
</html>
```

Поле **Задать пароль** немного отличается от остальных: если оно пустое, контроллер оставит текущий пароль пользователя неизменным. Помните, что в базе данных пароль хранится в зашифрованном виде, поэтому вы не можете вывести текущую версию записи и разрешить ее редактирование пользователю.

Флажки для выбора ролей во многом похожи на те, что вы использовали для обозначения категорий в форме, предназначенной для добавления и редактирования шуток (см. главу 7). Однако здесь есть некоторое отличие. Поскольку в базе данных для идентификаторов ролей вместо чисел используются строки, вы не можете с их помощью сгенерировать атрибуты `id` для тегов `input`: атрибут `id` не должен содержать пробелы. Чтобы сгенерировать уникальное число для каждой роли, придется применить немного другой подход. Для этого переберите значения массива ролей, воспользовавшись старой доброй конструкцией `for` вместо цикла `foreach` (`chapter9/admin/authors/form.html.php`, фрагмент).

```
<?php for ($i = 0; $i < count($roles); $i++): ?>
```

Счетчик `$i` имеет начальное значение `0` и с каждой итерацией цикла увеличивается на единицу. Таким образом, доступ к любой роли внутри цикла записывается как `$roles[$i]`, а переменная `$i` формирует уникальные атрибуты `id` (`chapter9/admin/authors/form.html.php`, фрагмент).

```
id="role<?php echo $i; ?>"
```

Теперь обновите контроллер, чтобы учесть новые поля (`chapter9/admin/authors/index.php`). Процесс обработки поля с паролем вам уже знаком, а код для флажков с ролями почти ничем не отличается от того, что вы использовали для работы с категориями шуток (все изменения в нем выделены полужирным начертанием).

```
<?php
include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/magicquotes.inc.php';
```

```
require_once $_SERVER['DOCUMENT_ROOT'] . '/includes/access.inc.php';

if (!userIsLoggedIn())
{
  include '../login.html.php';
  exit();
}

if (!userHasRole('Администратор учетных записей'))
{
  $error = 'Доступ к этой странице имеет только администратор учетных
записей.';
  include '../accessdenied.html.php';
  exit();
}

if (isset($_GET['add']))
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  $pageTitle = 'Новый автор';
  $action = 'addform';
  $name = '';
  $email = '';
  $id = '';
  $button = 'Добавить автора';

  // Формируем список ролей.
  try
  {
    $result = $pdo->query('SELECT id, description FROM role');
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при получении списка ролей.';
    include 'error.html.php';
    exit();
  }

  foreach ($result as $row)
  {
    $roles[] = array(
      'id' => $row['id'],
      'description' => $row['description'],
      'selected' => FALSE);
  }

  include 'form.html.php';
  exit();
}

if (isset($_GET['addform']))
{
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  try
  {
```

```
    $sql = 'INSERT INTO author SET
        name = :name,
        email = :email';
    $s = $pdo->prepare($sql);
    $s->bindValue(':name', $_POST['name']);
    $s->bindValue(':email', $_POST['email']);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при добавлении автора.';
    include 'error.html.php';
    exit();
  }

  $authorid = $pdo->lastInsertId();

  if ($_POST['password'] != '')
  {
    $password = md5($_POST['password'] . 'ijdb');

    try
    {
      $sql = 'UPDATE author SET
          password = :password
          WHERE id = :id';
      $s = $pdo->prepare($sql);
      $s->bindValue(':password', $password);
      $s->bindValue(':id', $authorid);
      $s->execute();
    }
    catch (PDOException $e)
    {
      $error = 'Ошибка при назначении пароля для автора.';
      include 'error.html.php';
      exit();
    }
  }

  if (isset($_POST['roles']))
  {
    foreach ($_POST['roles'] as $role)
    {
      try
      {
        $sql = 'INSERT INTO authorrole SET
            authorid = :authorid,
            roleid = :roleid';
        $s = $pdo->prepare($sql);
        $s->bindValue(':authorid', $authorid);
        $s->bindValue(':roleid', $role);
        $s->execute();
      }
      catch (PDOException $e)
      {
        $error = 'Ошибка при назначении роли для автора.';
```

```
            include 'error.html.php';
            exit();
          }
        }
      }

      header('Location: .');
      exit();
    }

    if (isset($_POST['action']) and $_POST['action'] == 'Редактировать')
    {
      include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

      try
      {
        $sql = 'SELECT id, name, email FROM author WHERE id = :id';
        $s = $pdo->prepare($sql);
        $s->bindValue(':id', $_POST['id']);
        $s->execute();
      }
      catch (PDOException $e)
      {
        $error = 'Ошибка при извлечении информации об авторе.';
        include 'error.html.php';
        exit();
      }

      $row = $s->fetch();

      $pageTitle = 'Изменение информации об авторе';
      $action = 'editform';
      $name = $row['name'];
      $email = $row['email'];
      $id = $row['id'];
      $button = 'Обновить информацию об авторе';

      // Получаем список ролей, назначенных для данного автора.
      try
      {
        $sql = 'SELECT roleid FROM authorrole WHERE authorid = :id';
        $s = $pdo->prepare($sql);
        $s->bindValue(':id', $id);
        $s->execute();
      }
      catch (PDOException $e)
      {
        $error = 'Ошибка при получении списка назначенных ролей.';
        include 'error.html.php';
        exit();
      }

      $selectedRoles = array();
      foreach ($s as $row)
      {
        $selectedRoles[] = $row['roleid'];
      }
```

```
    // Формируем список всех ролей.
    try
    {
      $result = $pdo->query( SELECT id, description FROM role );
    }
    catch (PDOException $e)
    {
      $error = 'Ошибка при получении списка ролей.';
      include 'error.html.php';
      exit();
    }

    foreach ($result as $row)
    {
      $roles[] = array(
        'id' => $row['id'],
        'description' => $row['description'],
        'selected' => in_array($row['id'], $selectedRoles));
    }

    include 'form.html.php';
    exit();
  }

  if (isset($_GET['editform']))
  {
    include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

    try
    {
      $sql = 'UPDATE author SET
          name = :name,
          email = :email
          WHERE id = :id';
      $s = $pdo->prepare($sql);
      $s->bindValue(':id', $_POST['id']);
      $s->bindValue(':name', $_POST['name']);
      $s->bindValue(':email', $_POST['email']);
      $s->execute();
    }
    catch (PDOException $e)
    {
      $error = 'Ошибка при обновлении информации об авторе.';
      include 'error.html.php';
      exit();
    }

    if ($_POST['password'] != '')
    {
      $password = md5($_POST['password'] . 'ijdb');

      try
      {
        $sql = 'UPDATE author SET
            password = :password
            WHERE id = :id';
```

```
        $s = $pdo->prepare($sql);
        $s->bindValue(':password', $password);
        $s->bindValue(':id', $_POST['id']);
        $s->execute();
      }
      catch (PDOException $e)
      {
        $error = 'Ошибка при назначении пароля автору.';
        include 'error.html.php';
        exit();
      }
    }

    try
    {
      $sql = 'DELETE FROM authorrole WHERE authorid = :id';
      $s = $pdo->prepare($sql);
      $s->bindValue(':id', $_POST['id']);
      $s->execute();
    }
    catch (PDOException $e)
    {
      $error = 'Ошибка при удалении неактуальных записей о ролях автора.';
      include 'error.html.php';
      exit();
    }

    if (isset($_POST['roles']))
    {
      foreach ($_POST['roles'] as $role)
      {
        try
        {
          $sql = 'INSERT INTO authorrole SET
              authorid = :authorid,
              roleid = :roleid';
          $s = $pdo->prepare($sql);
          $s->bindValue(':authorid', $_POST['id']);
          $s->bindValue(':roleid', $role);
          $s->execute();
        }
        catch (PDOException $e)
        {
          $error = 'Ошибка при назначении автору заданных ролей.';
          include 'error.html.php';
          exit();
        }
      }
    }

    header('Location: .');
    exit();
  }

  if (isset($_POST['action']) and $_POST['action'] == 'Удалить')
  {
    include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';
```

```
// Удаляем записи о ролях, назначенных для данного автора.
try
{
  $sql = 'DELETE FROM authorrole WHERE authorid = :id';
  $s = $pdo->prepare($sql);
  $s->bindValue(':id', $_POST['id']);
  $s->execute();
}
catch (PDOException $e)
{
  $error = 'Ошибка при удалении ролей автора.';
  include 'error.html.php';
  exit();
}

// Получаем шутки, принадлежащие автору.
try
{
  $sql = 'SELECT id FROM joke WHERE authorid = :id';
  $s = $pdo->prepare($sql);
  $s->bindValue(':id', $_POST['id']);
  $s->execute();
}
catch (PDOException $e)
{
  $error = 'Ошибка при получении списка шуток, которые нужно удалить.';
  include 'error.html.php';
  exit();
}

$result = $s->fetchAll();

// Удаляем записи о категориях для шутки.
try
{
  $sql = 'DELETE FROM jokecategory WHERE jokeid = :id';
  $s = $pdo->prepare($sql);

  // Для каждой шутки
  foreach ($result as $row)
  {
    $jokeId = $row['id'];
    $s->bindValue(':id', $jokeId);
    $s->execute();
  }
}
catch (PDOException $e)
{
  $error = 'Ошибка при удалении записей о категориях для шутки.';
  include 'error.html.php';
  exit();
}

// Удаляем шутки, принадлежащие автору.
try
{
```

```
      $sql = 'DELETE FROM joke WHERE authorid = :id';
      $s = $pdo->prepare($sql);
      $s->bindValue(':id', $_POST['id']);
      $s->execute();
    }
    catch (PDOException $e)
    {
      $error = 'Ошибка при удалении шуток автора.';
      include 'error.html.php';
      exit();
    }

    // Удаляем автора.
    try
    {
      $sql = 'DELETE FROM author WHERE id = :id';
      $s = $pdo->prepare($sql);
      $s->bindValue(':id', $_POST['id']);
      $s->execute();
    }
    catch (PDOException $e)
    {
      $error = 'Ошибка при удалении автора.';
      include 'error.html.php';
      exit();
    }

    header('Location: .');
    exit();
  }

  // Выводим список авторов.
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

  try
  {
    $result = $pdo->query('SELECT id, name FROM author');
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при извлечении информации об авторах из базы данных!';
    include 'error.html.php';
    exit();
  }

  foreach ($result as $row)
  {
    $authors[] = array('id' => $row['id'], 'name' => $row['name']);
  }

  include 'authors.html.php';
```

Вот и все. Протестируйте улучшенную версию кода и почувствуйте неограниченную власть, присвоив себе все роли. Убедитесь, что все работает должным образом. Находясь в системе, попробуйте ради интереса изменить собственный пароль. При следующем нажатии ссылки или кнопки код должен перенаправить вас на страницу с формой для входа, где вам потребуется ввести новый пароль, чтобы снова вернуться в систему.

Новый вызов: модерирование шуток

Конечно, сверяться с уже готовым предоставленным кодом легко и просто, совсем другое дело — самостоятельно реализовать новые и важные функции. Пришло время попробовать себя в проектировании и воплотить в жизнь одну из основных возможностей сайта с шутками.

В последних нескольких главах вы настолько сосредоточились на страницах администрирования, что почти не продвинулись с работой над общедоступной частью сайта. Более того, вы успели сделать несколько шагов назад. До того как вы убрали ссылку для добавления шуток и кнопку **Удалить**, главная страница вашего сайта выглядела, как на рис. 9.9.

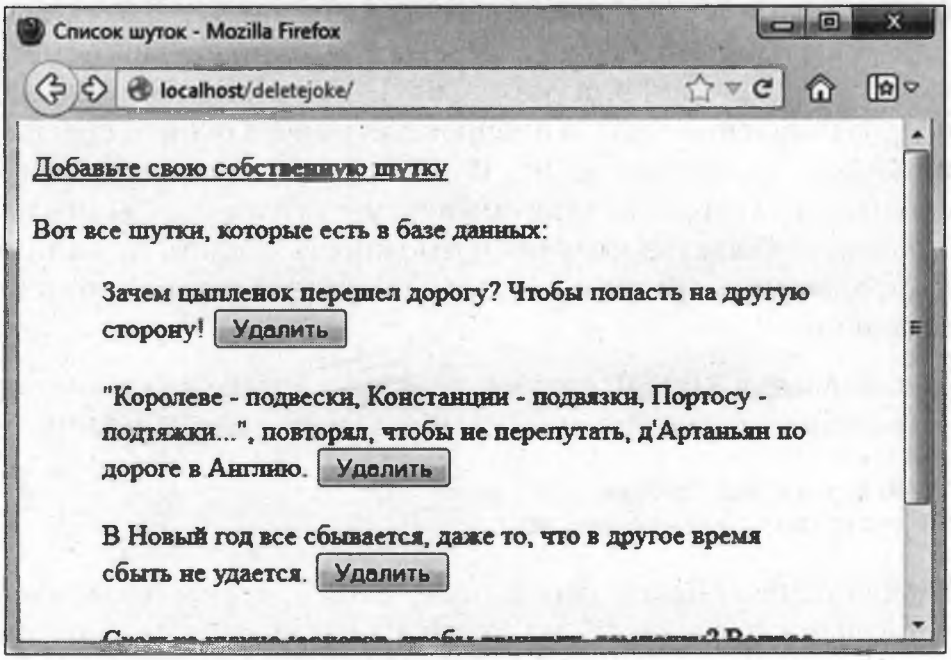


Рис. 9.9. Прежний список шуток

Очевидно, что эти кнопки больше не нужны. А что насчет ссылки для добавления новых шуток? Изначально она вела на страницу с формой (рис. 9.10).

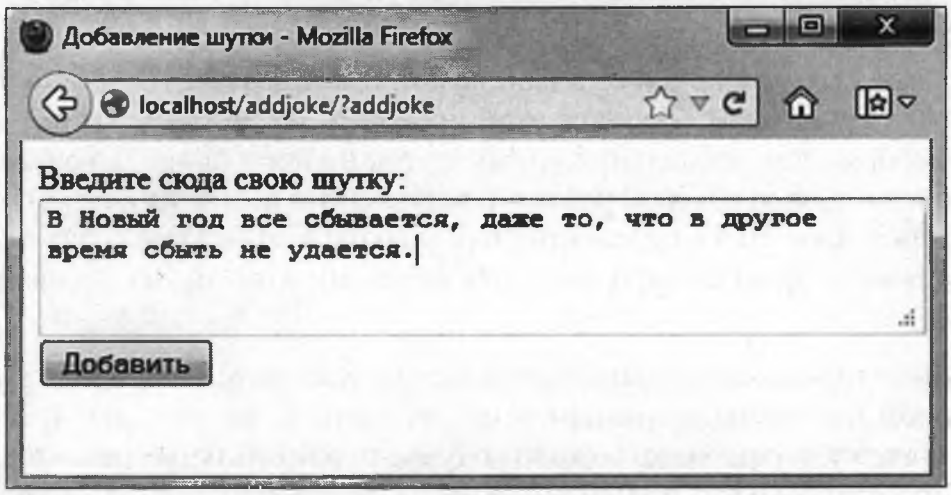


Рис. 9.10. В базу данных добавляется еще одна шутка

При отправке форма добавляла в базу данных новую шутку, не связанную ни с автором, ни с какой-либо категорией (на тот момент база данных не поддерживала таких возможностей). Исправить это достаточно просто, и вы вполне сможете справиться самостоятельно. Однако существует проблема, решить которую не так легко, — возможные злоупотребления. Запустив сайт с такой доступной формой для добавления шуток, вы сразу же столкнетесь со спамерами, которые мгновенно наполнят вашу базу данных всяким мусором.

Как же быть? Убрать функцию? Заставить авторов отправлять шутки редактору по электронной почте? Подумайте, ведь способ обеспечить безопасное добавление шуток на сайт, скорее всего, есть.

Должны ли новые шутки появляться на сайте сразу? Что если создать в таблице `joke` еще один столбец `visible`, который станет принимать одно из двух значений — `'YES'` (да) или `'NO'` (нет)? Тогда все новые шутки автоматически получат значение `'NO'`, а в запрос для вывода общего списка добавится условие `WHERE visible='YES'`. В этом случае только что отправленные шутки не появятся на страницах сайта, а переместятся в базу данных, где будут ожидать проверку. Редактор получит возможность сделать их видимыми (изменить текст, назначить автора и включить в нужную категорию) или удалить как нежелательные.

Чтобы созданный в MySQL столбец принимал только два значения, одно из которых устанавливается по умолчанию, ему следует задать тип `ENUM`.

```
ALTER TABLE joke ADD COLUMN
visible ENUM('NO', 'YES') NOT NULL
```

Поскольку столбец обязательный (`NOT NULL`), первое заданное в скобках значение (в нашем случае это `'NO'`) станет присваиваться по умолчанию, если при добавлении новой записи в команде `INSERT` не указано ничего другого. Теперь осталось лишь модифицировать систему администрирования, чтобы позволить редактору делать скрытые шутки видимыми. Для этого в форму добавления и редактирования шуток достаточно включить обычный флажок. Можно также изменить форму поиска, чтобы редактор осуществлял поиск только видимых или только скрытых записей.

Если в качестве отправной точки вы возьмете код из главы 6, то у новых шуток не будет связи с автором. Решение этой проблемы остается за вами. Вы можете дополнить форму для добавления новых шуток полем для ввода контактной информации, с помощью которой редактор определит и назначит авторство отправленной записи. Еще более перспективный подход — предложить автору сначала зарегистрироваться, задать пароль и войти в систему, а потом уже добавлять новые шутки.

Здесь нет единственно правильного решения. Выберите свой вариант для устранения данной проблемы и примените его на сайте. У вас есть все необходимые средства, поэтому не спешите и хорошо подумайте, как его лучше реализовать. Если у вас что-то не получается, поищите ответы на свои вопросы на форуме SitePoint, посвященном PHP (http://www.sitepoint.com/forums/forumdisplay.php?34-PHP).

Нет предела совершенству

В этой главе вы познакомились с двумя главными способами, которые позволяют создать постоянные переменные, способные сохраняться между открытиями веб-страниц. Первый использует браузер посетителя и сохраняет значения в виде куки. По умолчанию куки удаляются после закрытия браузера, но если задать им срок годности, то они могут существовать столько, сколько потребуется. К сожалению, куки довольно ненадежное средство: вы не можете быть уверены, что их не удалит браузер или сами пользователи, беспокоясь о защищенности своих данных.

Сессии устраняют все ограничения, связанные с куки, и позволяют хранить бесконечное количество потенциально объемных переменных. Они являются неотъемлемым компонентом современных приложений для интернет-торговли, в чем вы убедились на примере простого проекта с корзиной для покупок. Сессии также играют важную роль в системах, обеспечивающих контроль доступа. Одну из них вы реализовали для CMS, позволяющей управлять базой данных с шутками.

На этом этапе вы усвоили все важнейшие концепции и теперь обладаете достаточными навыками, чтобы написать собственный сайт на основе базы данных. Неудивительно, если у вас возникнет соблазн пропустить создание полноценной системы для безопасного приема информации на общедоступных страницах. Однако лучше этого не делать. Нет опыта ценнее, чем ошибки, на которых можно поучиться. В крайнем случае отложите задачу и вернитесь к ней, когда закончите читать книгу.

Если вы уверенно справились с поставленной задачей, попробуйте решить и другие. Сделайте так, чтобы пользователи оценивали шутки на сайте, или позвольте авторам редактировать шутки таким образом, чтобы внесенные изменения отображались на сайте только после того, как их одобрит администратор. Возможности и сложность созданной вами системы ограничены исключительно вашим воображением.

Оставшаяся часть книги посвящена более сложным темам, которые помогут оптимизировать производительность вашего сайта и решить некоторые трудные задачи, сократив объем кода. Вы познакомитесь с еще более захватывающими возможностями PHP и MySQL.

В главе 10 вы ненадолго отложите список шуток в сторону и подробно рассмотрите процесс поддержки и администрирования MySQL-сервера. Вы научитесь создавать резервные копии базы данных (важнейшая задача для любого интернет-проекта), управлять пользователями и их паролями, а также проходить аутентификацию в том случае, если вы вдруг забыли пароль.

Глава 10
АДМИНИСТРИРОВАНИЕ MySQL

В основе большинства сайтов содержательного характера, имеющих удачную архитектуру, лежит реляционная база данных. Материал данной книги построен на изучении системы управления реляционными базами данных MySQL. Свободное распространение и простота настроек обеспечили ей популярность в среде веб-разработчиков. Как вы уже смогли убедиться в главе 1, имея под рукой подходящие инструкции, даже новичок способен получить рабочий MySQL-сервер менее чем за пять минут, а если немного попрактиковаться, то и за две.

Если MySQL-сервер вам нужен лишь для того, чтобы разобраться с несколькими примерами и немного поэкспериментировать, то, вероятно, вам хватит базовых установок, которые рассмотрены в главе 1. Если же вы хотите подготовить базу данных для использования в реальном проекте, например для создания сайта, от которого зависит успешная деятельность компании, то, прежде чем полностью положиться на MySQL-сервер, вам придется изучить еще несколько основополагающих вещей.

Для начала вы познакомитесь с резервным копированием баз данных. Создание резервных копий для информации, которая важна лично для вас или вашего бизнеса, входит в список приоритетных задач любого администратора. Однако поскольку у администраторов хватает и других куда более важных занятий, процедура резервного копирования часто выполняется только один раз по необходимости. Такой подход считается вполне подходящим для любого рода приложений. Если вы до сих пор думали, что делать резервную копию базы данных необязательно, потому что она сохранится со всей остальной информацией, то вам следует обязательно прочесть данную главу. Вы узнаете, почему универсальные решения для резервного копирования файлов не подходят для многих конфигураций MySQL-сервера, и научитесь *правильно* архивировать и восстанавливать базу данных MySQL.

Вы также более подробно рассмотрите, как осуществляется контроль доступа к базам данных в MySQL. Ранее вы уже познакомились с основами этого процесса, но есть кое-какие неочевидные моменты, непонимание которых значительно усложнит вам жизнь. Кроме того, вы узнаете, как восстановить контроль над MySQL-сервером, если забыт пароль.

Далее вы перейдете к вопросам производительности, в частности к тому, как ускорить выполнение запросов, содержащих команду `SELECT`. Правильное применение индексов (удивительно, но этого навыка порой не хватает многим PHP-разработчикам) позволит вам поддерживать высокую скорость работы базы данных, даже если она содержит тысячи (и даже сотни тысяч) строк.

В конце главы 10 вы узнаете об относительно новой возможности MySQL — внешних ключах, используемых для описания структуры базы данных. Вы также разберетесь, каким образом связаны между собой таблицы, имеющие такие ключи.

Как видите, данная глава содержит достаточно разнообразной информации, но, дочитав ее до конца, вы станете понимать MySQL значительно лучше.

Резервное копирование баз данных в MySQL

Как и веб-сервер, MySQL должен быть доступен 24 часа в сутки семь дней в неделю. Это делает резервное копирование файлов базы данных несколько проблематичным. Чтобы повысить эффективность обновления данных, хранящихся на диске, MySQL-сервер помещает информацию в кеш или буфер оперативной памяти, поэтому говорить о полноте данных в любой момент времени не приходится. Стандартная процедура архивации заключается в обычном копировании файлов, принадлежащих системе и базе данных. Однако в случае с MySQL она не является надежной, потому что нет никакой гарантии, что копируемые файлы находятся в состоянии, подходящем для замены в аварийной ситуации.

Кроме того, многие базы данных, на основе которых работают сайты, получают новую информацию сутки напролет, а стандартная процедура архивирования обеспечивает только периодическое копирование содержимого базы. Информация, изменившаяся с момента выполнения последней копии, пропадет, если файлы с данными в MySQL повредятся или станут непригодными для использования. Во многих ситуациях (например, когда MySQL-сервер используется для отслеживания заказов в интернет-магазине) такая потеря очень нежелательна.

Для поддержания актуальности резервных копий, которые в момент архивации почти не зависят от активности самого сервера, в MySQL существуют специальные инструменты. Чтобы получить к ним доступ, придется настроить отдельную систему архивации, отвечающую исключительно за данные внутри MySQL и никак не взаимодействующую со средствами для резервного копирования остальной информации. Зато потом, когда придет время ею воспользоваться, вы оцените ее по достоинству.

Резервное копирование базы данных с помощью phpMyAdmin

В ходе чтения книги вы регулярно использовали phpMyAdmin — инструмент для администрирования MySQL, который работает в браузере. Он также предоставляет удобные средства для получения резервных копий базы данных сайта. Выбрав подходящую базу данный, перейдите на вкладку **Экспорт** (рис. 10.1).

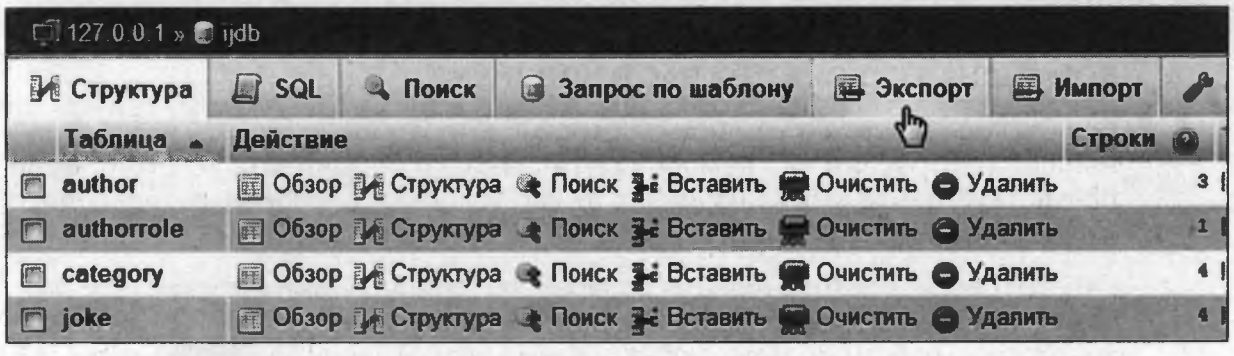


Рис. 10.1. Вкладка **Экспорт** позволяет сохранить резервную копию базы данных

Быстрый экспорт в формат SQL, который используется по умолчанию, идеально подходит для архивирования. Нажмите кнопку **OK**, чтобы браузер загрузил соответствующий файл. В нашем случае это **ijdb.sql**, поскольку база данных называется `ijdb`.

Открыв данный файл в текстовом редакторе, вы увидите набор SQL-команд, таких как `CREATE TABLE` и `INSERT`. Их выполнение позволит воспроизвести содержимое базы данных с нуля. В действительности резервная копия базы данных в MySQL — это всего лишь последовательность команд на языке SQL.

Попробуйте восстановить базу данных из резервного файла. Для начала убедитесь, что ваша база данных пуста (выделите все таблицы во вкладке **Структура** и в меню **С отмеченными** выберите пункт **Удалить**). Затем перейдите на вкладку **Импорт** и выберите на своем компьютере файл с резервной копией (здесь, как и прежде, подойдут стандартные настройки). Через мгновение содержимое вашей базы данных примет прежний вид.

Таким образом, приложение phpMyAdmin при необходимости подойдет для создания резервных копий базы данных. Оно подключается к MySQL-серверу, но при этом не имеет прямого доступа к файлам с информацией. В результате вы можете быть уверены, что полученная копия базы данных корректна, чего нельзя сказать о тех архивных файлах базы данных, которые регулярно сохраняются на диск и постоянно меняются на протяжении всей работы MySQL-сервера.

Резервное копирование базы данных с помощью mysqldump

Приложение phpMyAdmin позволяет получать резервные копии базы данных при любом удобном случае, однако лучше всего, когда этот процесс автоматизирован. К сожалению, здесь phpMyAdmin бессилен.

Если вы когда-либо имели дело с MySQL в Linux, то, скорее всего, знаете, что этот сервер баз данных поставляется с набором полезных утилит, предназначенных для работы в командной строке. Одна из таких утилит — `mysqldump`.

При запуске `mysqldump` подключается к MySQL-серверу (подобно тому как делает PHP) и загружает все содержимое одной или нескольких баз данных, которые вы указали. Затем эта программа выводит набор SQL-команд, необходимых для создания баз данных с точно таким содержимым. Сохранив вывод `mysqldump` в файл, вы получите резервную копию, подобную той, что сгенерирована с помощью phpMyAdmin.

КОМАНДНАЯ СТРОКА ДЛЯ ВАС В НОВИНКУ?

Если вы не знакомы с командной строкой операционной системы, то дальнейшие инструкции вас немало смутят. Однако переживать не стоит. Если почувствуете, что смысл начинает от вас ускользать, спокойно переходите к следующему разделу.

Строго говоря, в автоматизации резервного копирования на компьютере разработчика необходимости нет. После того как вы закроете XAMPP, MySQL прекратит свою работу, поэтому любая система для создания резервных копий (вы ведь применяете резервное копирование, не так ли?) сохранит файлы вашей

базы данных. Что касается реального MySQL-сервера, на основе которого работает общедоступный сайт, то любой приличный веб-хостинг все заботы, связанные с резервным копированием, берет на себя.

Настройкой автоматического создания резервных копий вам придется заниматься только в том случае, если вы собираетесь запустить собственный сервер баз данных промышленного уровня. Смеем предположить, что если вы возьметесь за такое, то у вас уже будет опыт использования командной строки в той операционной системе, которую вы сами выберите. Тем не менее для тех, кто пользуется Windows и все-таки решит изучить основы работы с командной строкой, рекомендуем ознакомиться со статьей «Kev's Command Prompt Cheat Sheet» («Шпаргалка по командной строке от Кева»), которая была написана автором данной книги в 2002 году (http://www.sitepoint.com/command-prompt-cheat-sheet/).

Следующая команда позволит вам подключиться к MySQL-серверу, запущенному на локальном компьютере, представиться пользователем `root` с паролем `password` и сохранить резервную копию базы данных `ijdb` в файл **ijdb.sql**[1].

```
mysqldump -u root -ppassword ijdb > ijdb.sql
```

Чтобы восстановить базу данных после аварийной ситуации на сервере, отправьте полученный SQL-файл программе phpMyAdmin[2] или воспользуйтесь утилитой `mysql`.

```
mysql -u root -ppassword ijdb < ijdb.sql
```

Данная команда подключается к MySQL-серверу, выбирает базу данных `ijdb` и передает файл с резервной копией в виде списка команд на языке SQL, требующих исполнения.

Осталось подумать над тем, как сделать так, чтобы резервные копии всегда оставались актуальными.

Инкрементальное резервное копирование с помощью бинарного журнала изменений

Ранее уже упоминалось, что во многих ситуациях, когда используется MySQL, потеря любых данных неприемлема. В связи с этим необходимо свести к минимуму перерывы между получением резервных копий с помощью phpMyAdmin или `mysqldump`.

Чтобы решить эту проблему, следует сконфигурировать MySQL-сервер таким образом, чтобы он вел **бинарный журнал**, записывая все SQL-запросы, которые

[1] Чтобы запустить `mysqldump`, равно как и любую другую утилиту из состава MySQL, нужно перейти в директорию **bin** в то место, куда установлен ваш сервер (для Mac OS X и MAMP это каталог **/Applications/MAMP/Library/bin**, для XAMPP — поддиректория **mysql**), или добавить указанную директорию в список системных путей.

[2] В отличие от SQL-файлов, созданных с помощью меню **Экспорт** в программе phpMyAdmin, резервные копии в исполнении `mysqldump` включают в себя команды, которые перед созданием новых таблиц удаляют их старые версии. Поэтому проверять, пуста ли ваша база данных, не нужно.

получены базой данных и так или иначе изменили ее содержимое. Особенно это касается команд INSERT, UPDATE и DELETE, исключение составляет команда SELECT.

Основная идея бинарного журнала заключается в том, что он позволяет восстанавливать содержимое базы данных, имеющееся на тот момент, когда произошла катастрофа. Такое восстановление подразумевает использование резервной копии, выполненной с помощью phpMyAdmin или mysqldump, и добавление содержимого бинарных журналов, сгенерированных после того, как произошло архивирование.

Вы также имеете возможность редактировать бинарные журналы, чтобы устранить вероятные ошибки. Например, если ваш коллега случайно выполнил команду DROP TABLE, вам достаточно экспортировать его бинарный журнал в текстовый файл, внести в него необходимые изменения (убрать этот запрос) и затем восстановить базу данных с помощью последней резервной копии и измененного журнала. Вы также можете сохранить все изменения, которые произошли *после* выполнения ошибочного запроса, хотя, вероятно, в целях безопасности лучше всего закрыть своему коллеге доступ к команде DROP.

Чтобы MySQL-сервер вел бинарный журнал, нужно отредактировать его конфигурационный файл (my.ini в Windows или my.cnf в Mac OS X или Linux). Внутри этого файла содержится обычный текст со списком параметров, которые определяют различные нюансы в работе сервера. В большинстве случаев MySQL устанавливается без конфигурационного файла и работает со стандартными настройками. Но в этой ситуации вам необходимо создать новый файл и внести в него соответствующие изменения.

ГДЕ НАХОДИТСЯ MY.INI И MY.CNF?

XAMPP для Windows поставляется с заранее подготовленным файлом my.ini, который находится в директории C:\xampp\mysql\bin\my.ini. Вы можете открыть и отредактировать его в Блокноте.

Сервер MySQL, встроенный в MAMP (в Mac OS X), при запуске станет искать файл /etc/my.cnf, создание которого связано с некоторыми трудностями. Как правило, программа Finder строго следит за порядком в таких важных директориях, поэтому вам придется создать данный файл в любом другом месте, а затем переместить его в каталог /etc посредством консольной команды, воспользовавшись привилегиями администратора.

```
sudo mv my.cnf /etc/
```

Чтобы включить ведение бинарного журнала, необходимо добавить параметр log-bin в раздел [mysqld] своего конфигурационного файла. Если вы создали файл с нуля, то вам нужно самостоятельно вписать раздел [mysqld] в самом верху и только потом добавлять параметр в новой строке.

Параметр log-bin сообщает MySQL-серверу, где хранить файлы бинарного журнала и какие у них должны быть имена. Например, в Windows их лучше хранить в каталоге с данными MySQL[1].

```
[mysqld]
log-bin="C:/xampp/mysql/data/binlog"
```

[1] Стоит заметить, что конфигурационные файлы MySQL используют в путях Windows обычный слеш (/), а не обратный (\).

В Mac OS X для этого подойдет каталог MAMP.

```
[mysqld]
log-bin=/Applications/MAMP/logs/binlog
```

В обоих примерах MySQL сгенерирует файлы с названиями binlog.000001, binlog.000002 и т. д. Создание новые файлов происходит при каждом сбрасывании журнала на диск, а в реальных условиях — при каждом перезапуске сервера.

СОХРАНЯЙТЕ БИНАРНЫЙ ЖУРНАЛ НА ДРУГОМ ЖЕСТКОМ ДИСКЕ

Старайтесь не держать бинарный журнал на том винчестере, где находятся файлы вашей базы данных MySQL. Если жесткий диск сломается, вы избежите одновременной потери базы данных и ее резервных копий.

Подготовив конфигурационный файл, перезапустите MySQL-сервер. С этого момента он начнет создавать файлы бинарного журнала. На всякий случай убедитесь, что они записываются при каждом запуске сервера в указанной вами директории.

По понятным причинам на активном сервере бинарный журнал занимает довольно много места. Поэтому важно, чтобы MySQL удалял устаревшие журнальные файлы при каждом создании полноценных резервных копий. Этого легко добиться, если для резервного копирования используется `mysqldump`.

```
mysqldump -u root -ppassword --flush-logs --master-data=2 --delete-
master-logs ijdb > ijdb.sql
```

Параметр `--flush-logs` сообщает MySQL-серверу о том, что следует закрыть текущий файл бинарного журнала и начать новый (как при перезапуске). Параметр `--masterdata=2` отвечает за то, чтобы утилита `mysqldump` добавляла в конец файла ijdb.sql название нового бинарного журнала, в котором будут содержаться изменения, внесенные в базу данных сразу после создания полноценной резервной копии. И наконец, благодаря параметру `--delete-master-logs` утилита `mysqldump` узнает, что файлы бинарного журнала, которые оказались ненужными после выполнения полного резервного копирования, необходимо удалить.

Если в случае непредвиденной ситуации у вас на руках окажется полноценная резервная копия и файлы бинарного журнала, сгенерированные после выполнения архивации, то восстановление базы данных не составит труда. Вам понадобится настроить новый пустой MySQL-сервер и поработать с полноценной резервной копией базы данных, как описано в предыдущем разделе. После чего останется воспользоваться бинарным журналом с помощью утилиты `mysqlbinlog`.

Задача `mysqlbinlog` — преобразовать бинарный журнал формата MySQL в команды на языке SQL, выполнимые в контексте базы данных. Допустим, у вас есть два файла бинарного журнала, которые нужно использовать после восстановления базы данных из последней резервной копии. С помощью `mysqlbinlog` вы имеете возможность сгенерировать из них один текстовый SQL-файл, а затем передать его на выполнение MySQL-серверу, как в случае с файлом, полученным посредством команды `mysqldump`.

```
mysqlbinlog binlog.000041 binlog.000042 > binlog.sql
mysql -u root -ppassword < binlog.sql
```

Советы по управлению доступом к MySQL

Из главы 2 вы узнали, что база данных mysql, которая присутствует на любом MySQL-сервере, используется для хранения информации о пользователях, их паролях и полномочиях. В главе 4 вы рассмотрели, как с помощью phpMyAdmin создать еще одну пользовательскую учетную запись, которая открывает доступ только к базе данных сайта.

Система управления доступом к MySQL полностью документирована, вы найдете ее в соответствующем справочном руководстве (http://dev.mysql.com/doc/refman/5.5/en/privilege-system.html). В сущности, полномочия пользователя определяются содержимым пяти таблиц, расположенных в базе данных mysql: `user`, `db`, `host`, `tables_priv` и `columns_priv`. Если вы пожелаете отредактировать перечисленные таблицы вручную с помощью команд `INSERT`, `UPDATE` и `DELETE`, ознакомьтесь с соответствующим разделом руководства MySQL. В остальном вам вполне хватит средств для управления доступом к MySQL-серверу, которые предоставляет phpMyAdmin.

Система управления доступом к MySQL имеет несколько особенностей, о которых вы обязаны знать, если планируете ответственно относиться к назначению полномочий пользователям на сервере баз данных.

Проблемы, связанные с именем сервера

Добавляя в phpMyAdmin новых пользователей, которые проходят аутентификацию на MySQL-сервере, только на том компьютере, на котором этот сервер работает (чтобы, например, разрешить им запускать внутри системы утилиту командной строки mysql или выполнять подключение с помощью таких серверных скриптов, как PHP), вы можете задаться вопросом: что именно вводить в поле Хост? Представьте, что сервер работает на домене www.example.com. Что нужно указать: www.example.com или localhost?

На самом деле, когда речь идет о создании каких бы то ни было соединений, ни один из этих вариантов не надежен. Теоретически, если пользователь при подключении указал имя сервера (с помощью консольной утилиты mysql либо в PHP-скрипте в классе PDO), это имя должно совпасть с записью в системе управления доступом. Однако вряд ли вам захочется, чтобы пользователи указывали имя сервера каким-то определенным образом (и на самом деле вряд ли у них есть желание вникать в такие тонкости), поэтому лучше пойти другим путем.

Для тех пользователей, кому нужно подключаться к MySQL-серверу через компьютер, на котором он работает, в системе управления доступом лучше создать две записи: одну непосредственно с именем сервера, например www.example.com, другую со значением localhost. Конечно, вам придется назначать и снимать привилегии для каждой записи отдельно, но это действительно единственный надежный способ.

Еще одна проблема, с которой часто сталкиваются администраторы MySQL, заключается в том, что записи пользователей, где в именах серверов содержатся заполнители (например, %.example.com), иногда не работают. Причины непредсказуемого поведения системы управления доступом чаще всего связаны с тем, как MySQL-сервер выставляет приоритеты для учетных записей. В частности, он сортирует пользователей таким образом, что более конкретные имена серверов идут в начале списка (к примеру, имя www.example.com абсолютно определенное, %.example.com менее конкретное, а % совершенно неопределенное).

По умолчанию после установки[1] система управления доступом к MySQL содержит две анонимные учетные записи (они позволяют подключиться с локального компьютера, используя любой логин, а также имя сервера или значение localhost, как описано ранее) и две администраторские. Вышеописанная проблема возникает, когда анонимные записи имеют более конкретное имя сервера и получают повышенный приоритет по сравнению с новым пользователем.

Давайте рассмотрим фрагмент таблицы, куда включены пользователи домена www.example.com — вымышленного MySQL-сервера, для которого добавлена новая учетная запись пользователя с именем `jess`. Строки показаны в том порядке, в каком их рассматривает сервер при проверке подключения.

Имя сервера	Пользователь	Пароль
localhost	`root`	*`зашифрованное значение`*
www.example.com	`root`	*`зашифрованное значение`*
localhost		
www.example.com		
%.example.com	`jess`	*`зашифрованное значение`*

Как видите, запись `jess` содержит наименее конкретное имя сервера, поэтому она находится в конце списка. Когда пользователь с данным логином попытается подключиться с адреса www.example.com, MySQL-сервер сопоставит его с одной из анонимных учетных записей (пустой логин соответствует любому пользователю). Пользователь `jess`, скорее всего, введет свой пароль (для анонимных записей это делать необязательно), и в результате MySQL-сервер отклонит попытку подключения. Но даже если пользователь `jess` подключится без пароля, он получит очень ограниченные привилегии, свойственные анонимным учетным записям, которые вряд ли соответствуют правам, назначенным системой управления доступом для его логина.

Эту проблему можно решить двумя способами: либо удалить анонимные учетные записи с самого начала (`DELETE FROM mysql.user WHERE User=""`), либо выдать по две дополнительные записи всем пользователям, которым для работы необходимо подключаться на локальном компьютере (по одной для localhost и настоящего имени сервера).

Имя сервера	Пользователь	Пароль
localhost	`root`	*`зашифрованное значение`*
www.example.com	`root`	*`зашифрованное значение`*
localhost	`jess`	*`зашифрованное значение`*
www.example.com	`jess`	*`зашифрованное значение`*
localhost		
www.example.com		
%.example.com	`jess`	*`зашифрованное значение`*

[1] В единых установочных пакетах, таких как XAMPP и MAMP, как правило, все устроено немного иначе.

Поскольку для каждого логина поддерживать три учетные записи (с тремя наборами привилегий) довольно обременительно, лучше удалить анонимных пользователей, особенно если в них нет необходимости.

Имя сервера	Пользователь	Пароль
localhost	root	*зашифрованное значение*
www.example.com	root	*зашифрованное значение*
%.example.com	jess	*зашифрованное значение*

Забыли пароль?

Забыть пароль от MySQL-сервера после того, как потрачен целый час на его установку и настройку, — это почти то же самое, что закрыть автомобиль вместе с ключами внутри. К счастью, если у вас есть административный доступ к компьютеру, на котором работает MySQL, или возможность зайти в систему как пользователь, обладающий полномочиями для управления MySQL-сервером, то все не так уж плохо. Ознакомившись с этим разделом, вы узнаете, как восстановить контроль над сервером.

ПРЕДПОЛАГАЕТСЯ ИСПОЛЬЗОВАНИЕ КОМАНДНОЙ СТРОКИ

В этом разделе мы будем исходить из того, что раз уж вы забыли пароль от MySQL, то у вас хотя бы есть навыки работы с системной командной строкой. Если вы используете MySQL-сервер из состава XAMPP или MAMP, то, скорее всего, в вашей базе данных отсутствует некоторая очень важная информация, позволяющая восстановить утерянный пароль `root`. В таком случае сервер проще переустановить. Если вы пользователь Windows, можете почерпнуть всю необходимую информацию из вышеупомянутой статьи автора данной книги «Kev's Command Prompt Cheat Sheet» («Шпаргалка по командной строке от Кева», http://www.sitepoint.com/command-prompt-cheat-sheet/).

Первым делом следует остановить MySQL-сервер. В обычных условиях вы делали это с помощью консольной утилиты `mysqladmin`, которая требовала пароль. Теперь вам придется «убить» все процессы на сервере. Если вы работаете в Windows, используйте Диспетчер задач: завершите процесс MySQL или остановите MySQL-сервис. В Mac OS X или Linux найти идентификатор нужного процесса поможет команда `ps`. Еще один способ — заглянуть в PID-файл, который находится в директории с данными вашего MySQL-сервера, и завершить процесс с помощью следующей команды.

```
kill pid
```

pid — идентификатор процесса MySQL-сервера.

Чтобы остановить сервер, этого достаточно. *Не используйте* команду принудительного завершения `kill -9`, если в этом нет крайней необходимости: она способна повредить файлы с данными. На тот случай, если ситуация вынудила вас поступить именно так, в следующем разделе вы найдете инструкции по проверке и восстановлению файлов.

Теперь, когда сервер остановлен, запустите его снова, используя параметр `skip-grant-tables`. Вы также можете добавить его в конфигурационный файл MySQL-сервера — my.ini или my.cnf (инструкции по настройке файла ищите в разделе «Инкрементальное резервное копирование с помощью бинарного журнала изменений»).

```
[mysqld]
skip-grant-tables
```

Так сервер MySQL узнает о том, что необходимо открыть свободный доступ для любого пользователя. Поскольку такой режим работы несет потенциальную угрозу с точки зрения безопасности, сервер должен находиться в нем как можно меньше.

Подключившись к серверу (с помощью phpMyAdmin или консольной утилиты `mysql`), укажите новый пароль администратора.

```
UPDATE mysql.user SET Password=PASSWORD("newpassword")
WHERE User="root"
```

Снова остановите MySQL-сервер и удалите параметр `skip-granttables`.

Индексы

Подобно тому как алфавитный указатель в данной книге упрощает поиск конкретной информации, так и индекс базы данных помогает MySQL находить записи, которые вы запрашиваете посредством команды `SELECT`. Рассмотрим пример.

По мере развития вашего проекта со списком шуток размер таблицы `joke` способен увеличиться до тысяч, если не до сотен тысяч записей. Теперь представим, что PHP-скрипт запрашивает текст конкретной шутки.

```
SELECT joketext FROM joke WHERE id = 1234
```

Если бы база данных не имела индекса, MySQL-серверу пришлось бы просмотреть одно за другим значение столбца `id` в каждой из строк таблицы `joke`, пока не нашлась бы та самая строка с идентификатором 1234. Что еще хуже, без индекса MySQL-сервер не узнал бы о том, что таким значением обладает только одна строка, поэтому ему пришлось бы перебрать оставшееся содержимое таблицы, чтобы гарантированно вернуть все совпадения.

Компьютеры имеют высокую производительность и хорошо справляются с подобной работой, но в индустрии веб-разработки, где счет идет на микросекунды, сочетание больших таблиц и сложных операторов `WHERE` легко приведет к задержкам поиска в полминуты и более.

Ваш запрос сработает быстро. Все благодаря тому, что столбец `id` в таблице `joke` имеет индекс. Чтобы убедиться в этом, откройте phpMyAdmin, выберите таблицу `joke` и перейдите на вкладку **Структура**, после списка столбцов вы увидите таблицу с индексами (рис. 10.2).

Действие	Имя индекса	Тип	Уникальный	Упакован	Поле	Уникальных элементов	Сравнение	Null	Комментарий
Изменить Удалить	PRIMARY	BTREE	Да	Нет	id	4	A	Нет	

Рис. 10.2. Во всех таблицах уже содержится по одному индексу

Обратите внимание на колонку **Поле**: этот индекс перечисляет значения столбца `id`. По содержимому колонки **Имя индекса** (`PRIMARY`) несложно догадаться, как появился данный индекс: он создан автоматически, когда вы приняли столбец `id` за первичный ключ для таблицы `joke`.

Вспомним, как объявлялся столбец `id`.

```
CREATE TABLE joke (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  joketext TEXT,
  jokedate DATE NOT NULL,
  authorid INT
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB;
```

На языке базы данных `KEY` фактически означает «индекс», а `PRIMARY` — это первичный ключ, или индекс, который требует, чтобы все значения в заданном столбце были уникальными. На рис. 10.2 видно, что в колонке **Уникальный** стоит значение `Да`.

Все вышесказанное сводится к тому, что созданные в базе данных таблицы индексируются по столбцу `id`. Таким образом, любая команда `WHERE`, имея доступ к индексу, получает возможность быстро найти запись с определенным идентификатором.

Чтобы убедиться в этом, попросите MySQL-сервер объяснить, как он выполняет конкретный запрос с командой `SELECT`: добавьте в начало запроса оператор `EXPLAIN`.

```
EXPLAIN SELECT joketext FROM joke WHERE id = 1
```

РЕАЛЬНЫЕ ЗНАЧЕНИЯ ДАЮТ РЕАЛЬНЫЕ РЕЗУЛЬТАТЫ

Стоит заметить, что в данном запросе в качестве идентификатора шутки указано значение 1, которое на самом деле существует в вашей базе данных. Если бы использовался вымышленный вариант, например 1234, то MySQL-сервер сразу бы догадался, что такого идентификатора в таблице `joke` не существует, и даже не пытался бы его извлечь.

Результат выполнения запроса с командой `EXPLAIN` в phpMyAdmin будет выглядеть так, как показано на рис. 10.3.

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	joke	const	PRIMARY	PRIMARY	4	const	1	

Рис. 10.3. Команда `SELECT` использует индекс `PRIMARY`

Тот же механизм применяется для того, чтобы ускорить выполнение SQL-запросов, которые объединяют две таблицы, используя значения `id` (например, при поиске автора шутки по значению столбца `authorid`).

Рассмотрим запрос с командой SELECT, который извлекает все шутки заданного автора.

```
SELECT * FROM joke WHERE authorid = 2
```

Попросите MySQL объяснить запрос с помощью оператора EXPLAIN (рис. 10.4).

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	joke	ALL	*NULL*	*NULL*	*NULL*	*NULL*	4	Using where

Рис. 10.4. Значения NULL свидетельствуют о низкой скорости поиска

Как видите, MySQL-сервер не смог воспользоваться индексом для ускорения запроса и ему пришлось выполнять сложный перебор содержимого таблицы. Чтобы повысить производительность работы сервера для выполнения такого рода запросов, следует добавить индекс для столбца authorid.

ИДЕНТИФИКАТОРЫ АВТОРОВ УЖЕ ПРОИНДЕКСИРОВАНЫ!

Несмотря на то что столбец id таблицы author служит первичным ключом и содержит индекс, в данном запросе это индекс бесполезен, поскольку таблица author в нем не фигурирует. Оператор WHERE ищет значение внутри таблицы joke в поле authorid, которое индекса не содержит.

Выберите в phpMyAdmin таблицу joke и перейдите на вкладку Структура. В разделе Индексы перейдите в форму Создать индекс для 1 столбца/ов, перед вами появится панель для создания нового индекса (рис. 10.5).

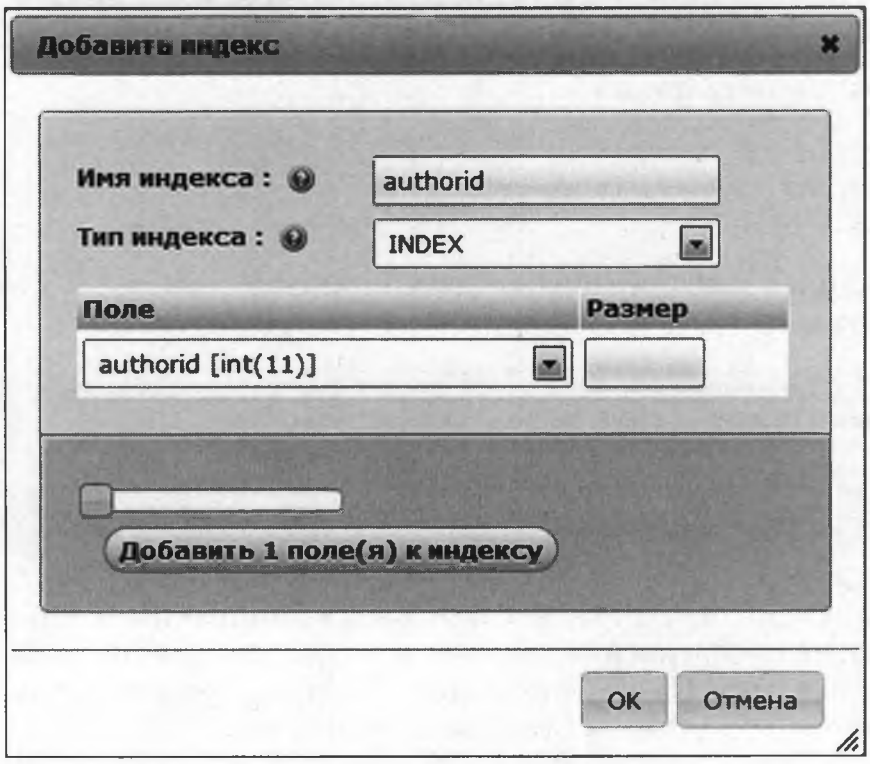


Рис. 10.5. Создание нового индекса для столбца authorid

Заполните форму следующим образом.

- **Имя индекса**. Установите название индексируемого столбца (в действительности вы можете назвать его как угодно).
- **Тип индекса**. Поскольку таблица уже имеет первичный ключ, вместо `PRIMARY` выберите `INDEX`. Кроме того, здесь есть значение `UNIQUE` (уникальный) и `FULLTEXT` (полнотекстовый). Первый нам не требуется, поскольку шутка не обязана иметь уникального автора, второй используется для поиска больших объемов текста.
- **Поле**. В качестве столбца для индексирования выберите `authorid`.
- **Размер**. Это поле оставьте пустым, чтобы значение столбца индексировалось целиком (а не, скажем, первые пять символов).

Нажав кнопку **OK**, вы увидите, что рядом с `PRIMARY` появится еще один индекс. Снова попросите MySQL объяснить работу команды `SELECT`, используя оператор `EXPLAIN`, и убедитесь в том, что на этот раз при запросе используется индекс `authorid`.

Если идея индексации всех полей в базе данных кажется вам заманчивой, то реализовывать ее настоятельно не рекомендуется. И не только потому, что индексы требуют дополнительного дискового пространства. Каждый раз, когда вы изменяете содержимое базы данных (например, с помощью команд `INSERT` или `UPDATE`), MySQL-серверу приходится тратить время на то, чтобы пересмотреть все индексы, которых коснулись эти изменения. По этой причине необходимо добавлять только те индексы, которые потребуются, чтобы ускорить выполнение запросов `SELECT` на вашем сайте, и не более того.

Если вы обратили внимание, то столбец `id` есть *не у всех* таблиц, которые вы создали на данный момент. Возьмем, например, таблицу `jokecategory`.

```
CREATE TABLE jokecategory (
  jokeid INT NOT NULL,
  categoryid INT NOT NULL,
  PRIMARY KEY (jokeid, categoryid)
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB;
```

Ее первичный ключ состоит из двух столбцов: `jokeid` и `categoryid`. На рис. 10.6 показано, как выглядит этот индекс в phpMyAdmin.

Действие	Имя индекса	Тип	Уникальный	Упакован	Поле	Уникальных элементов	Сравнение	Null
Изменить Удалить	PRIMARY	BTREE	Да	Нет	jokeid	4	A	Нет
					categoryid	4	A	Нет

Рис. 10.6. Индексы могут содержать несколько полей

Такой многостолбцовый индекс называется **композитным**. Он существенно повышает скорость выполнения запросов, которые связаны с несколькими индексируемыми столбцами. Следующий запрос, например, проверяет, входит ли шутка с id 3 в категорию с id 4 (что на самом деле так и есть).

```
SELECT * FROM jokecategory WHERE jokeid = 3 AND categoryid = 4
```

Такой двухстолбцовый индекс можно использовать как одностолбцовый для первого поля в списке (в нашем случае это столбец `jokeid`). Таким образом, при запросе категорий, к которым принадлежит шутка с id 1, также будет использован индекс.

```
SELECT name
FROM jokecategory
  INNER JOIN category ON categoryid = category.id
WHERE jokeid = 1
```

Внешние ключи

Надеемся, что к этому моменты вы уже уяснили, что для представления связей между двумя таблицами необходимо, чтобы столбец одной из них ссылался на поле `id` другой. Например, чтобы обозначить авторство шутки, столбец `authorid` в таблице `joke` должен указывать на поле `id` в таблице `author`.

В реляционных базах данных столбец, чьи значения совпадают с содержимым столбца из другой таблицы, называется **внешним ключом**. Таким образом, `authorid` — это внешний ключ, который ссылается на столбец `id` в таблице `author`.

До этого момента при проектировании таблиц вы держали все связи между ними в голове, на уровне MySQL они зафиксированы не были. Вы сами следили за тем, чтобы значения в столбце `authorid` соответствовали записям в таблице `author`. Но если бы вы по неосторожности вставили значение `authorid`, которое не совпадает ни с одной строкой внутри `author`, то MySQL-сервер не стал бы противиться: для него `authorid` — это обычное поле с целыми числами.

MySQL поддерживает функцию под названием **ограничения внешнего ключа**, которая позволяет записывать подобные связи между таблицами конкретно и помогает серверу следить за их соблюдением. Чтобы указать эти ограничения, воспользуйтесь командой `CREATE TABLE` или добавьте их в уже существующую таблицу с помощью запроса `ALTER TABLE` (`chapter10/sql/ijdb.sql`, фрагмент).

```
CREATE TABLE joke (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  joketext TEXT,
  jokedate DATE NOT NULL,
  authorid INT,
  FOREIGN KEY (authorid) REFERENCES author (id)
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB

ALTER TABLE joke
ADD FOREIGN KEY (authorid) REFERENCES author (id)
```

Приложение phpMyAdmin также позволяет создавать ограничения внешнего ключа. Для этого убедитесь сначала в том, что ключевой столбец (в данном случае это `authorid`) содержит индекс. Если вы выполните один из двух вышеприведенных запросов, MySQL создаст этот индекс автоматически, в phpMyAdmin вам придется действовать самостоятельно. Индекс для `authorid` у вас уже есть, поэтому перейдите на вкладку **Структура** для таблицы `joke` и нажмите ссылку **Связи** (рис. 10.7).

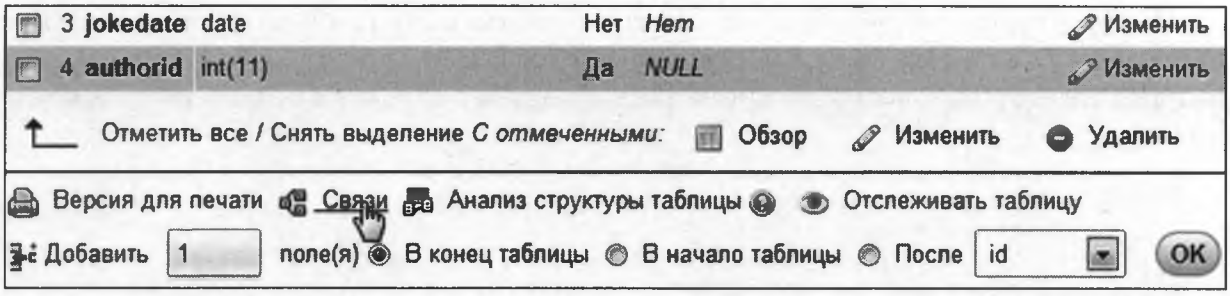


Рис. 10.7. Переход по ссылке **Связи** позволит отредактировать внешние ключи в таблице

Открывшаяся страница предоставит список всех полей таблицы `joke` и позволит сконфигурировать ограничения внешнего ключа для тех из них, у которых есть индекс. В колонке **Ограничение внешнего ключа (INNODB)** напротив столбца `authorid` выберите пункт `'ijdb'.'author'.'id'` (`ijdb` — название вашей базы данных). Поле в колонке **Внутренняя связь** оставьте пустым[1], также оставьте выбранные по умолчанию значения **RESTRICT** для полей **ON DELETE** и **ON UPDATE** (рис. 10.8). Нажмите кнопку **Сохранить**.

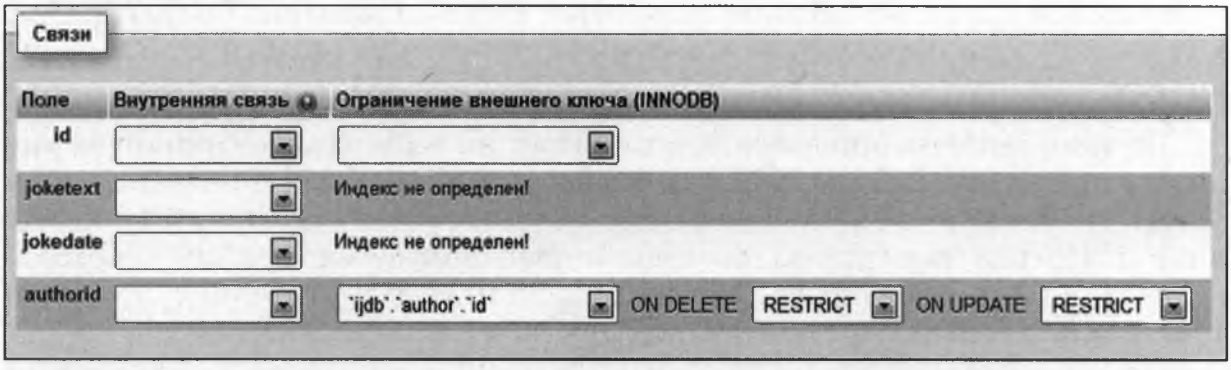


Рис. 10.8. Настройка внешнего ключа в phpMyAdmin

Учитывая заданные ограничения внешнего ключа, MySQL отвергнет любые попытки добавить в таблицы `joke` и `authorid` значения, которые нельзя сопоставить ни с одной записью в таблице `author`. Более того, чтобы удалить запись в таблице `author`, вам придется убрать любые связанные с ней элементы в таблице `joke`.

Еще одно преимущество, которое вы получите, заключается в том, что, просматривая таблицу `joke` в phpMyAdmin, вы можете щелкнуть на значении в столбце `authorid` и перейти к соответствующей строке в таблице `author`. Все благодаря тому, что теперь MySQL и, следовательно, phpMyAdmin понимают структуру вашей базы данных и потому способны помочь вам в ней ориентироваться.

Для закрепления пройденного материала попробуйте создать ограничения внешнего ключа, которые описывают все связи между вашими таблицами. Попытайтесь сделать это самостоятельно, не заглядывая в список команд `ALTER TABLE`, приведенный ниже (подсказка: помимо представленных связей, есть еще четыре).

[1] В phpMyAdmin в колонке **Внутренняя связь** можно установить связь между таблицами без создания ограничений внешнего ключа (и даже без индекса для столбца). Такой способ позволяет быстрее переходить между таблицами в интерфейсе phpMyAdmin, но MySQL следить за соблюдением такой связи не будет.

```
ALTER TABLE jokecategory
ADD FOREIGN KEY (jokeid) REFERENCES joke (id)

ALTER TABLE jokecategory
ADD FOREIGN KEY (categoryid) REFERENCES category (id)

ALTER TABLE authorrole
ADD FOREIGN KEY (authorid) REFERENCES author (id)

ALTER TABLE authorrole
ADD FOREIGN KEY (roleid) REFERENCES role (id)
```

О ССЫЛОЧНЫХ ДЕЙСТВИЯХ ЗАМОЛВИТЕ СЛОВО

Вместо того чтобы отвергать попытки удалить или обновить записи, на которые ссылаются внешние ключи (например, попытку удалить авторов, у которых остались шутки), воспользуйтесь **ссылочными действиями**. Особая конфигурация внешних ключей позволит MySQL автоматически разрешать подобные конфликты. Вы можете использовать два способа: выстроить цепочки операций (то есть вместе с автором удалить все его шутки) или присвоить всем затрагиваемым столбцам, которые имеют внешний ключ, значение `NULL` (обнулить столбцы `authorid` соответствующих шуток). Вот для чего нужны параметры `ON RESTRICT` и `ON UPDATE` в интерфейсе настройки ограничений внешнего ключа в phpMyAdmin.

Если у вас появился соблазн перепоручить MySQL-серверу заботу о том, что делать с шутками, для которых пользователь удаляет автора или категорию, то мы обязаны вас предостеречь. Конечно, выбрать параметр в phpMyAdmin проще, чем написать соответствующий PHP-код, который, перед тем как убрать имя автора, автоматически сотрет связанные с ним шутки. Однако проблема в том, что бизнес-логика вашего сайта разделится на две части: PHP-код и ограничения внешнего ключа. Вы больше не сможете заглянуть в содержимое контроллера и увидеть все, что происходит при удалении шутки. По этой причине большинство опытных PHP-разработчиков предпочитают не использовать ссылочные действия, а некоторые стараются не прибегать даже к ограничениям внешних ключей.

Лучше перестраховаться, чем потом жалеть

Данная глава не была похожа на предыдущие, где было очень много кода. Однако, изучив рассмотренные в ней темы (резервное копирование и восстановление данных в MySQL, работу системы управления доступом к базе данных, повышение скорости выполнения запросов с помощью индексов, выстраивание структуры базы данных с использованием внешних ключей), вы получили в свое распоряжение инструменты, которые вам, несомненно, понадобятся, чтобы организовать стабильную работу MySQL-сервера, способного выдержать постоянный наплыв посетителей на ваш сайт.

В главе 11 вы снова вернетесь к привычному изложению материала и изучите некоторые расширенные возможности SQL, с помощью которых сервер реляционных баз данных сможет выполнять то, что вам даже не снились.

Глава 11
РАСШИРЕННЫЕ SQL-ЗАПРОСЫ

Работая над сайтом с каталогом шуток, вы познакомились с разными аспектами языка SQL и рассмотрели множество его команд, начиная с простейших версий запроса `CREATE TABLE` и заканчивая двумя видами синтаксиса для запроса `INSERT`. В завершение этой темы вы ознакомитесь с еще несколькими приемами работы со структурированным языком запросов и остановитесь более детально на тех моментах, для изучения которых вам не хватало накопленных знаний. Большинство представленных в этой главе примеров основаны на непростой и даже слегка замысловатой команде `SELECT`.

Сортировка результатов выполнения запроса SELECT

Большие объемы информации легче использовать, если они представлены в виде списка, организованного в определенном порядке. В противном случае поиск конкретного автора среди множества остальных, зарегистрированных в базе данных, превратится в сущий кошмар.

Формулируя запрос `SELECT`, вы имеете возможность уточнить его и указать столбец, по которому следует сортировать полученный результат. Допустим, вы хотите вывести для дальнейшего использования содержимое таблицы `author`.

```
SELECT id, name, email FROM author
```

В результате вы увидите список, краткий вариант которого представлен на рис. 11.1.

id	name	email
1	Кевин Янк	thatguy@kevinyank.com
2	Джоан Смит	joan@example.com
3	Молли О'Рейли	molly@faerie.com
4	Эми Мэтьюсон	amym@example.com

Рис. 11.1. Неотсортированный список авторов

Записи не отсортированы, но для такого незначительного списка это не проблема. В том случае, когда количество записей исчисляется сотнями, найти адрес электронной почты конкретного автора, например Эми Мэтьюсон, куда проще, если список отсортирован по именам в алфавитном порядке. Вот как выглядит этот запрос.

```
SELECT id, name, email FROM author ORDER BY name
```

Результат его выполнения показан на рис. 11.2.

id	name	email
2	Джоан Смит	joan@example.com
1	Кевин Янк	thatguy@kevinyank.com
3	Молли О'Рейли	molly@faerie.com
4	Эми Мэтьюсон	amym@example.com

Рис. 11.2. Список авторов, отсортированный по именам в алфавитном порядке

Помимо оператора `WHERE`, конкретизирующего список результатов, в команде `SELECT` используется оператор `ORDER BY`. С его помощью указывается столбец, по которому выполняется сортировка результирующего набора. Ключевое слово `DESC`, добавленное после имени столбца, изменит порядок сортировки записей на противоположный (рис. 11.3).

```
SELECT id, name, email FROM author ORDER BY name DESC
```

id	name	email
4	Эми Мэтьюсон	amym@example.com
3	Молли О'Рейли	molly@faerie.com
1	Кевин Янк	thatguy@kevinyank.com
2	Джоан Смит	joan@example.com

Рис. 11.3. Список авторов, отсортированный по именам в порядке убывания

В действительности оператор `ORDER BY` позволяет осуществлять упорядочивание данных сразу по нескольким столбцам, которые вводятся в запрос через запятую. В этом случае MySQL сначала отсортирует результаты запроса в первом столбце, а затем выстроит в нужном порядке записи в связанных наборах во втором и т. д. Во всех столбцах, перечисленных в операторе `ORDER BY`, допустимо использовать ключевое слово `DESC`, чтобы изменить порядок расположения данных на противоположный.

Очевидно, что, если потребуется отсортировать результирующий набор в большой таблице, MySQL придется проделать довольно сложную работу. Чтобы облегчить ему задачу, задайте индексы для столбцов (или наборов столбцов), которые по вашему замыслу должны участвовать в данном запросе (о том, как это сделать, рассказано в главе 10).

Установка лимитов

Часто бывает так, что в большой таблице вас интересуют всего несколько записей. Допустим, вы захотели отслеживать на сайте популярность шуток. Для этого добавьте в таблицу `joke` столбец `timesviewed` и задайте в нем для каждой новой

шутки значение, равное нулю, а при каждом просмотре увеличьте это значение на единицу. Таким образом вы получите информацию о том, сколько раз пользователи прочитали ту или иную шутку, находящуюся в вашей базе данных. Вот как выглядит код, добавляющий единицу к значению из столбца `timesviewed`, которое принадлежит шутке с заданным идентификатором.

```
try
{
  $sql = 'UPDATE joke SET
      timesviewed = timesviewed + 1
      WHERE id = :id';
  $s = $pdo->prepare($sql);
  $s->bindValue(':id', $id);
  $s->execute();
}
catch (PDOException $e)
{
  $error = 'Ошибка при обновлении информации о количестве просмотров
шутки.';
  include 'error.html.php';
  exit();
}
```

Этот счетчик пригодится для вывода десяти самых популярных шуток на стартовой странице вашего сайта. Отсортировав шутки по популярности в порядке убывания значений из столбца `timesviewed (ORDER BY timesviewed DESC)`, вы возьмете только первые десять записей в списке. Предположим, что в базе данных находятся около тысячи шуток — извлекать их все только ради десяти результатов довольно расточительно с точки зрения времени на обработку запроса и рационального использования системных ресурсов сервера, таких как загрузка памяти и центрального процессора.

Воспользуйтесь оператором `LIMIT`, чтобы указать точное количество возвращаемых результатов. В нашем примере нужно только десять первых.

```
$sql = 'SELECT joke.id, joketext, name, email
    FROM joke INNER JOIN author
      ON authorid = author.id
    ORDER BY timesviewed DESC
    LIMIT 10';
```

Аналогичным образом можно получить десять наименее популярных, если убрать из запроса ключевое слово `DESC`.

Иногда длинные списки, полученные в результате поиска, удобнее просматривать, если одновременно отображается только определенное количество записей[1]. Вспомните, как вы в последний раз пытались найти нужный сайт, пролистывая страницы с ссылками, выданными поисковой системой. Для подобных действий используется оператор `LIMIT`: просто укажите место, с которого станут отсчитываться записи, и максимальное количество отображаемых результатов. Представленный ниже запрос выводит список самых популярных шуток в базе данных, начиная с 21-й и заканчивая 25-й.

[1] Более подробно этот подход рассматривается в статье «Объектно-ориентированный PHP: постраничный вывод результирующих наборов» (ObjectOriented PHP:PagingResultSets; http://www.sitepoint.com/php-paging-result-sets/).

```
$sql = 'SELECT joke.id, joketext, name, email
    FROM joke INNER JOIN author
      ON authorid = author.id
    ORDER BY timesviewed DESC
    LIMIT 20, 5';
```

Обратите внимание, что нумерация записей в списки начинается с нуля, поэтому 21-я запись в списке идет под номером 20.

Транзакции в базе данных

Иногда требуется выполнить серию SQL-запросов таким образом, чтобы множество команд сработало одновременно. Это возможно благодаря **транзакциям**.

Представим, что в базе данных вашего магазина находится две таблицы: в одной представлен список товаров, выставленных на продажу, и их количество, а в другой — заказы, сделанные посетителями сайта. После того как посетитель оформит заказ, необходимо обновить данные о наличии товара и пометить товары, помещенные в корзину, как заказанные. Если одна из этих операций выполнится раньше другой, вы рискуете попасть в ситуацию, когда остальные посетители, просматривая информацию на вашем сайте, столкнутся с противоречивой информацией. Так, они могут увидеть, что из десяти якобы проданных виджетов, на самом деле заказано только девять.

Транзакции позволяют выполнять сложные операции, состоящие из множества запросов, которые вступают в действие одновременно. Чтобы инициировать транзакцию, достаточно отправить SQL-команду `START TRANSACTION`.

```
START TRANSACTION
```

После чего необходимо дописать нужный набор SQL-запросов и отправить команду `COMMIT`.

```
COMMIT
```

В ходе выполнения транзакции вы имеете возможность передумать и, решив, что отправляемую группу запросов выполнять не нужно, воспользоваться командой `ROLLBACK`.

```
ROLLBACK
```

Команда `ROLLBACK` особенно полезна при возникновении неполадок. Если первый запрос в транзакции прошел успешно, а второй выдал ошибку, то `ROLLBACK` позволит отменить результаты выполнения первого запроса и пересмотреть дальнейшие действия.

В PHP-скриптах объект PDO также предоставляет довольно удобные методы работы с транзакциями: метод `beginTransaction` инициирует транзакцию, `commit` — ее применяет, а `rollBack` — отменяет.

```
try
{
  $pdo->beginTransaction();
```

```
    ⋮ выполняем последовательность запросов…

    $pdo->commit();
  }
  catch (PDOException $e)
  {
    $pdo->rollBack();

    $error = 'Ошибка при выполнении транзакции.';
    include 'error.html.php';
    exit();
  }
```

ТРАНЗАКЦИИ ТРЕБУЮТ ФОРМАТ InnoDB

Одна из причин, по которой все таблицы данной книги созданы в формате InnoDB (`ENGINE=InnoDB`), — поддержка транзакций. Более старый формат MyISAM транзакции не поддерживает.

Псевдонимы для столбцов и таблиц

В некоторых ситуациях к столбцам и таблицам MySQL гораздо удобнее обращаться по другим именам. Рассмотрим это действие на примере базы данных для заказа авиабилетов, которая была разработана на страницах форума SitePoint еще при написании первого издания книги. Если вы пожелаете протестировать все на практике, соответствующую структуру базы данных вы найдете в файле airline.sql внутри архива с кодом.

Информация о предлагаемых авиакомпанией перелетах представлена в виде двух таблиц — `flight` и `city`. Каждая запись в таблице `flight` отражает сведения об отдельном авиарейсе: место вылета (столбец `origincityid`), пункт назначения (столбец `destinationcityid`), а также дата и время вылета, тип самолета, номер рейса и стоимость билетов (остальные столбцы). В таблице `city` находится список всех городов, куда совершаются полеты. Таким образом, в столбцах `origincityid` и `destinationcityid` таблицы `flight` будут находиться только идентификаторы, ссылающиеся на записи внутри таблицы `city`.

Чтобы получить список перелетов с пунктами вылета (рис. 11.4), необходимо выполнить следующий запрос.

```
SELECT flight.number, city.name
FROM flight INNER JOIN city
  ON flight.origincityid = city.id
```

Запрос для получения списка перелетов с пунктами назначения (рис. 11.5) похож на предыдущий.

```
SELECT flight.number, city.name
FROM flight INNER JOIN city
  ON flight.destinationcityid = city.id
```

number	name
CP110	Монреаль
QF2026	Мельбурн
CP226	Сидней
QF2027	Сидней

Рис. 11.4. Список рейсов с пунктами вылета

number	name
CP226	Монреаль
QF2027	Мельбурн
CP110	Сидней
QF2026	Сидней

Рис. 11.5. Список рейсов с пунктами назначения

Логично предположить, что для вывода рейсов с обоими пунктами — вылета и назначения — нужно осуществить запрос, представленный ниже.

```
SELECT flight.number, city.name, city.name
FROM flight INNER JOIN city
  ON flight.origincityid = city.id
INNER JOIN city
  ON flight.destinationcityid = city.id
```

Однако, как только вы попытаетесь это сделать, phpMyAdmin выведет ошибку: `#1066 – Not unique table/alias: 'city'`.

Почему так происходит? Отбросьте в сторону свои ожидания и попробуйте разобраться в том, что на самом деле означает данный запрос. Он просит MySQL-сервер объединить таблицы `flight`, `city` и `city`. Попытка дважды присоединить одну и ту же таблицу и приводит к выводу сообщения об ошибке.

Кроме того, в запросе чувствуется нехватка логики. Он пытается вывести номер рейса, название города и название города (опять же дважды) для всех полученных записей, сопоставляя с `id` таблицы `city` столбцы `origincityid` и `destinationcityid`. Другими словами, `id` таблицы `city` и столбцы `origincityid` и `destinationcityid` должны быть одинаковыми. Даже если бы запрос сработал, его результатом стал бы список всех рейсов, в котором пункты вылета и пункты назначения совпадают. Вряд ли найдется хоть один рейс, который будет соответствовать такому описанию, если речь идет не об авиакомпании, предлагающей обзорные полеты над городами.

Нужно придумать другой способ для повторного использования таблицы city, позволяющий MySQL избавиться от путаницы. Вам следует вернуть из таблицы две разные записи для каждого результата: одну для места вылета, а другую для пункта назначения. Значительно упростить ситуацию помогли бы две копии таблицы `city` (одна по имени `origin`, а другая — `destination`). Но зачем заниматься поддержкой двух разных списков с одинаковыми городами? Решить проблему можно, указав для таблицы `city` два уникальных **псевдонима** (временных имени) внутри запроса.

Если после имени таблицы во фрагменте запроса `SELECT`, начинающегося с ключевого слова `FROM`, написать `AS псевдоним`, то вы получите временное имя, на которое можно ссылаться в любом другом месте запроса. Обратимся к первому коду, выдающему номера рейсов и места вылета, и укажем для таблицы `city` псевдоним `origin`.

```
SELECT flight.number, origin.name
FROM flight INNER JOIN city AS origin
  ON flight.origincityid = origin.id
```

Запрос сработает, как и раньше, и его результаты останутся прежними, только теперь вы имеете возможность заменить длинные имена таблиц более короткими. Если бы вы воспользовались псевдонимами `f` и `o` вместо `flight` и `origin` соответственно, то запись сократилась бы уже существенно.

Вернемся к проблемному запросу. Теперь, дважды сославшись на таблицу `city` с помощью разных псевдонимов, вы сможете выполнить тройное объединение (в котором две таблицы на самом деле являются одной) и получить желаемый результат (рис. 11.6).

```
SELECT flight.number, origin.name, destination.name
FROM flight INNER JOIN city AS origin
  ON flight.origincityid = origin.id
INNER JOIN city AS destination
  ON flight.destinationcityid = destination.id
```

Аналогичным образом определяются псевдонимы для столбцов. В нашем случае это поможет различить столбцы `name` в итоговой таблице (рис. 11.7).

```
SELECT f.number, o.name AS origin, d.name AS destination
FROM flight AS f INNER JOIN city AS o
  ON f.origincityid = o.id
INNER JOIN city AS d
  ON f.destinationcityid = d.id
```

number	name	name
CP110	Монреаль	Сидней
CP226	Сидней	Монреаль
QF2026	Мельбурн	Сидней
QF2027	Сидней	Мельбурн

Рис. 11.6. Список рейсов с пунктами вылета и назначения

number	origin	destination
CP110	Монреаль	Сидней
CP226	Сидней	Монреаль
QF2026	Мельбурн	Сидней
QF2027	Сидней	Мельбурн

Рис. 11.7. Список рейсов с пунктами вылета и назначения с обозначенными столбцами

Используйте данный подход, чтобы добавить в свой проект с шутками систему для обмена сообщениями между авторами. В списке отправленных сообщений таблицу `author` стоит упомянуть дважды: один раз для отправителя, другой — для получателя. Попробуйте реализовать эту идею самостоятельно.

Группирование результатов

В главе 2, чтобы узнать, сколько шуток хранится в таблице `joke`, вы использовали следующий запрос.

```
SELECT COUNT(*) FROM joke
```

Функция `COUNT` принадлежит к особому виду функций, которые называются **агрегирующими**, или **функциями группирования**. Их полный список приведен

в руководстве по MySQL (глава 11; http://dev.mysql.com/doc/mysql/en/group-by-functions.html) и в приложении В к данной книге. В отличие от других функций, которые работают с отдельными записями, полученными при выполнении команды SELECT, они группируют все результаты и возвращают единый ответ. В вышеприведенном примере функция COUNT возвращает общее количество полученных строк.

Предположим, вы захотели вывести список авторов и количество принадлежащих им шуток. Скорее всего, первое, что придет вам в голову, — это извлечь имена и идентификаторы всех авторов, получить списки всех шуток для каждого автора, используя соответствующие id, а затем применить к полученным результатам функцию COUNT. Вот как примерно выглядел бы ваш PHP-код (для упрощения записи обработка ошибок не использовалась).

```
// Получаем список всех авторов.
$result = $pdo->query('SELECT id, name FROM author');

// Считываем всех авторов.
foreach ($result as $row)
{
  $authors[] = array(
      'id' => $row['id'],
      'name' => $row['name']
  );
}

// Получаем количество шуток, принадлежащих автору.
$sql = 'SELECT COUNT(*) AS numjokes FROM joke WHERE authorid = :id';
$s = $pdo->prepare($sql);

// Перебираем список авторов.
foreach ($authors as $author)
{
  $s->bindValue(':id', $author['id']);
  $s->execute();
  $row = $s->fetch();
  $numjokes = $row['numjokes'];

  // Выводим имя автора и количество шуток.
  $output .= htmlspecialchars($author['name'], ENT_QUOTES, 'UTF-8')
      . " ($numjokes jokes)<br>";
}
```

Обратите внимание, что во втором запросе использовалось ключевое слово AS, которое позволило дать понятное имя результатам вычисления COUNT(*).

Код сработает, но повлечет выполнение $n + 1$ дополнительных запросов, где n — количество авторов в базе данных. Зависимости между количеством запросов и записей в базе данных всегда лучше избегать, поскольку, если авторов будет огромное множество, ваш скрипт станет медленно работать и окажется требователен к ресурсам. Здесь пригодится еще одна особенность оператора SELECT.

Добавив к команде SELECT оператор GROUP BY, вы укажите, что MySQL необходимо сгруппировать результаты в наборы. Причем для заданного столбца результаты могут быть общими для двух таблиц. В этом случае функции агрегирования, такие как COUNT, получат на обработку не весь результирующий набор, а лишь

отдельные группы. Вот как станет выглядеть запрос, вычисляющий количество шуток, принадлежащих каждому автору.

```
SELECT author.name, COUNT(*) AS numjokes
FROM joke INNER JOIN author
  ON authorid = author.id
GROUP BY authorid
```

Результат, показанный на рис. 11.8, подтверждает, что вы получили нужную вам информацию с помощью единственного запроса SELECT.

name	numjokes
Кевин Янк	3
Джессика Грэм	1

Рис. 11.8. Список авторов и количество принадлежащих им шуток

Тот же результат можно получить, если использовать в запросе выражение GROUP BY author.id (при условии, что столбцы, указанные в операторе FROM совпадают). Выражение GROUP BY author.name в большинстве случаев тоже сработает. Однако существует вероятность (хоть и небольшая), что у двух разных авторов имена одинаковы, и их значения сольются в один результат. Учитывая эту особенность, лучше привязываться к столбцам с идентификаторами, уникальность которых для каждого автора гарантирована.

Оператор левого объединения

По результатам предыдущего запроса вы узнали, что у Кевина Янка на счету три шутки, а у Джессики Грэм одна. Однако вы не увидели здесь имена еще двух авторов — Эми Мэтьюсон и Майкла Йейтса, которые не добавили пока в базу данных ни одной шутки. В таблице joke не нашлось записей, значения которых в поле authorid совпадали бы с идентификаторами этих авторов, поэтому данные результаты не прошли проверку оператором ON и были изъяты из результирующего набора.

Единственный способ преодолеть эту проблему, используя уже знакомые вам средства, — добавить в таблицу author еще один столбец для хранения количества шуток каждого автора. Однако поддерживать этот столбец в актуальном состоянии — настоящее мучение: его придется обновлять каждый раз, когда в таблице joke происходит какое-либо добавление, удаление или изменение информации (например, изменение значения authorid). Для синхронизации всех этих действий понадобятся транзакции, что довольно неприятно, если не сказать больше.

Помимо оператора INNER JOIN (внутреннее объединение), которым вы пользовались до сих пор, в MySQL существует еще **оператор левого объединения**. Чтобы понять, в чем заключается его особенность, вспомните, как работает INNER JOIN.

Для стандартного объединения двух таблиц MySQL перебирает все возможные комбинации строк (рис. 11.9). В простейшем случае, когда две таблицы имеют по две записи каждая, объединение выполняется следующим образом: строка 1 табли-

цы 1 со строкой 1 таблицы 2; строка 1 таблицы 1 со строкой 2 таблицы 2; строка 2 таблицы 1 со строкой 1 таблицы 2 и, наконец, строка 2 таблицы 1 со строкой 2 таблицы 2. Получив результат, MySQL смотрит на условие ON и решает, какие строки нужно оставить, например те, в которых столбец id первой таблицы совпадает со столбцом authorid второй.

В нашем случае это решение не подходит потому, что в результирующий набор необходимо включить строки из таблицы 1 (то есть author), которые не имеют соответствий в таблице 2 (joke). Оператор левого объединения добавит в список результатов все строки первой таблицы (расположенной слева), даже если для них не удастся найти связанных записей во второй (расположенной справа). Всем столбцам таких строк, взятым из правой таблицы, будут присвоены значения NULL.

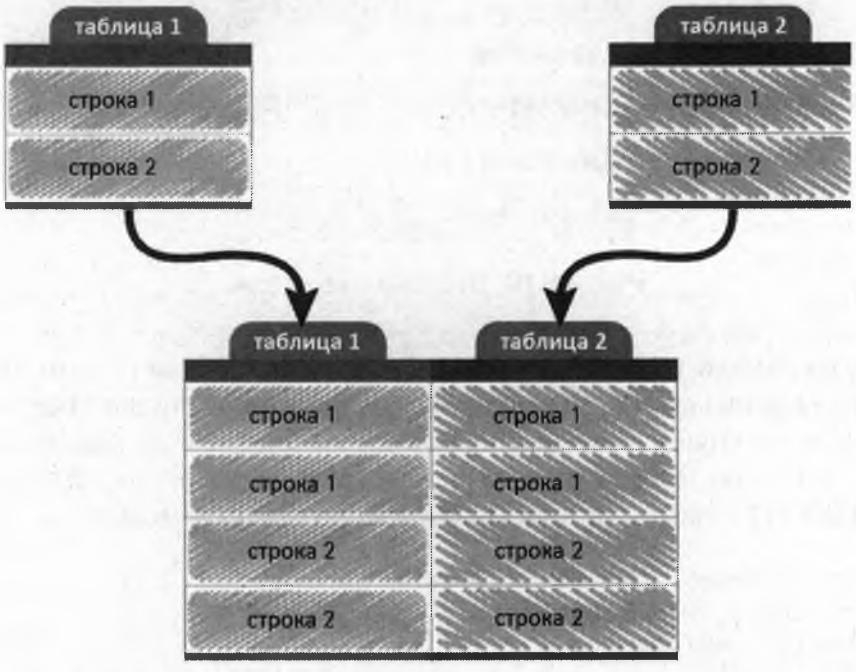


Рис. 11.9. Внутренние объединения перебирают все возможные комбинации строк

Чтобы выполнить в MySQL левое объединение двух таблиц, достаточно использовать команду LEFT JOIN вместо INNER JOIN в операторе FROM. Вот скорректированный запрос, который перечисляет авторов и количество их шуток.

```
SELECT author.name, COUNT(*) AS numjokes
FROM author LEFT JOIN joke
  ON authorid = author.id
GROUP BY author.id
```

В данном запросе стоит отметить два важных момента.

1. Необходимо писать author LEFT JOIN joke, а не joke LEFT JOIN author. Порядок, в котором перечисляются таблицы для объединения, очень важен. Запись LEFT JOIN приведет к отображению в результатах запроса всех строк из таблицы, которая находится слева (author).
2. Вместо GROUP BY authored следует использовать GROUP BY author.id. author.id — это столбец id в таблице author, а authored — это столбец

в таблице joke. В предыдущих запросах SELECT объединение основывалось на том, что значения этих столбцов совпадают. Однако когда оператор LEFT JOIN создает результирующую строку, основанную на строке из таблицы author, не имеющей соответствия в таблице joke, он присваивает NULL всем полям, которые взяты из таблицы joke. Это касается и поля authorid. Если бы вы написали GROUP BY authorid, запрос сгруппировал бы всех авторов, у которых нет ни одной шутки, поскольку в таком случае после выполнения оператора LEFT JOIN соответствующий им столбец authorid был бы равен NULL.

Выполнив запрос, вы получите результат, представленный на рис. 11.10.

name	numjokes
Кевин Янк	3
Эми Мэтьюсон	1
Джессика Грэм	1
Майкл Йейтс	1

Рис. 11.10. Что-то здесь не так...

Получилось, будто у Эми Мэтьюсон и Майкла Йейтса есть по одной шутке, хотя на самом деле это не так. Все потому, что запрос составлен неверно. Функция COUNT(*) подсчитывает количество строк, возвращаемых для каждого автора. Если вы внимательно посмотрите на результаты до того, как они группируются с помощью LEFT JOIN, то увидите, что происходит на самом деле.

```
SELECT author.name, joke.id AS jokeid
FROM author LEFT JOIN joke
  ON authorid = author.id
```

Строки, соответствующие авторам Эми Мэтьюсон и Майклу Йейтсу (рис. 11.11), включены в результат, хотя и не имеют совпадений в таблице joke, которая указана справа от LEFT JOIN. Тот факт, что идентификатор шутки равен NULL, никак не влияет на функцию COUNT(*) — функция засчитывает такую запись как обычно.

name	jokeid
Кевин Янк	1
Кевин Янк	2
Кевин Янк	3
Эми Мэтьюсон	*NULL*
Джессика Грэм	4
Майкл Йейтс	*NULL*

Рис. 11.11. Результаты запроса до выполнения группировки

Если вместо символа * указать имя конкретного столбца (предположим, joke.id), то для него функция COUNT проигнорирует значения NULL и выдаст то количество записей, которое нужно (рис. 11.2).

```
SELECT author.name, COUNT(joke.id) AS numjokes
FROM author LEFT JOIN joke
  ON authorid = author.id
GROUP BY author.id
```

name	numjokes
Кевин Янк	3
Эми Мэтьюсон	0
Джессика Грэм	1
Майкл Йейтс	0

Рис. 11.12. Правильный результат выполнения запроса

Ограничение результатов с помощью оператора HAVING

А что если вам понадобится получить список *только* тех авторов, у которых нет шуток? Рассмотрим запрос, который многие разработчики использовали бы в первую очередь.

```
SELECT author.name, COUNT(joke.id) AS numjokes
FROM author LEFT JOIN joke
  ON authorid = author.id
WHERE numjokes = 0
GROUP BY author.id
```

Выполнение данного кода приведет к тому, что phpMyAdmin выдаст ошибку `#1054 - Unknown column 'numjokes' in 'where clause'`.

Причина, по которой выражение `WHERE numjokes = 0` приводит к ошибке, связана с тем, как MySQL-сервер обрабатывает результирующий набор. Сначала он формирует черновой комбинированный список авторов и шуток, взятых из таблиц author и joke соответственно. Затем переходит к ключевому слову ON, которое следует за оператором FROM, и отбирает только релевантные строки (в нашем случае это строки, которые связывают автора и количество шуток, где столбец numjokes имеет значение 0). В завершение MySQL обращается к оператору GROUP BY: группирует результаты по столбцу authorid, подсчитывает в каждой группе с помощью функции COUNT количество записей, которые не содержат NULL в столбце joke.id, и выводит содержимое numjokes.

Обратите внимание, что столбец numjokes создан уже после того, как отработал оператор GROUP BY, который, в свою очередь, запускался после оператора WHERE. Следовательно, сообщение об ошибке возникло из-за того, что оператор WHERE искал столбец numjokes, который еще не создан.

Если вы не хотите учитывать шутки, которые содержат слово «цыпленок», вы можете спокойно использовать оператор WHERE, поскольку это исключение не зависит от результатов выполнения оператора GROUP BY. Однако условия, которые затрагивают результат уже *после* группирования, должны перечисляться в рамках специального выражения HAVING. Ниже представлен исправленный запрос и его результат (рис. 11.13).

```
SELECT author.name, COUNT(joke.id) AS numjokes
FROM author LEFT JOIN joke
  ON authorid = author.id
GROUP BY author.id
HAVING numjokes = 0
```

name	numjokes
Эми Мэтьюсон	0
Майкл Йейтс	0

Рис. 11.13. Список авторов, у которых нет шуток в базе данных

Некоторые условия срабатывают внутри обоих операторов, как HAVING, так и WHERE. Например, чтобы исключить из списка результаты по имени конкретного автора, условие `author.name != 'Имя Автора'` можно записать в любом из них. Результат останется неизменным вне зависимости от того, где произойдет исключение — до группирования результатов или после него. В таких случаях всегда лучше использовать оператор WHERE, поскольку он лучше оптимизируется внутри MySQL и поэтому работает быстрее.

Дополнительные источники информации

В этой главе вы расширили свои знания, касающиеся поддерживаемого в MySQL языка структурированных запросов. Большую часть времени вы посвятили изучению наиболее гибкой и мощной команды SELECT, которая позволяет выводить информацию, хранящуюся в базе данных. Используя ее преимущества, вы сможете возложить на MySQL-сервер часть выполняемых функций и существенно упростить код PHP.

Тем не менее есть еще несколько видов запросов, с которыми вы пока не сталкивались. MySQL предоставляет целую библиотеку встроенных функций, позволяющих выполнять такие задачи, как вычисление дат или форматирование текста (см. приложение В). Чтобы стать настоящим специалистом по MySQL, вам также необходимо хорошо разбираться в различных типах столбцов, которые поддерживаются этим сервером. Например, тип TIMESTAMP позволит вам сэкономить время. Более подробную информацию вы сможете найти в руководстве по MySQL (http://dev.mysql.com/doc/refman/5.5/en/datetime.html), в приложении Г эта же информация изложена более кратко.

Чтобы познакомиться поближе с возможностями SQL, рассмотренными в этой главе (и со множеством тех, о которых здесь еще не упоминалось), прочтите книгу Руди Лаймбека Simply SQL («Просто об SQL»; http://www.sitepoint.com/books/sql1/).

Глава 12
БИНАРНЫЕ ДАННЫЕ

Изучая базы данных, вы до сих пор работали с сайтами, основу которых составляла текстовая информация: шутки, авторы, категории... Все эти элементы можно представить в виде строк текста. Но что, если вы решите создать фотогалерею и предоставить пользователям возможность помещать туда собственные снимки? Чтобы реализовать подобную идею, вам потребуется предусмотреть передачу фотографий на сайт и отслеживание связанных с ней действий.

В этой главе вы спроектируете систему, которая позволит загружать бинарные файлы (изображения или документы), сохранять их на веб-сервере и просматривать на сайте. В ходе ее разработки вы освоите несколько новых приемов: работу с файлами в PHP, их загрузку и обработку, а также хранение и извлечение бинарных данных посредством MySQL. Знакомясь с особенностями перемещения файлов с использованием PHP, вы научитесь создавать полудинамические страницы и снижать нагрузку на веб-сервер.

Полудинамические страницы

Как владельцу (потенциально) успешного веб-проекта, вам, вероятно, хотелось бы, чтобы он имел высокую посещаемость. К сожалению, большой наплыв пользователей для системного администратора иногда превращается в настоящий кошмар, особенно если сайт состоит преимущественно из динамических страниц, которые генерируются на основе содержимого базы данных. По сравнению со старыми добрыми HTML-файлами, такие страницы требуют значительно больше вычислительных ресурсов компьютера, на котором работает программное обеспечение веб-сервера, поскольку каждое обращение к странице эквивалентно запуску миниатюрной программы.

Не все страницы сайта, основанного на базе данных, должны обновлять информацию ежесекундно. Возьмем, например, главную страницу сайта sitepoint.com, на которой представлен анонс новых материалов. Насколько часто этот список изменяется: раз в час, раз в день? Насколько важно, чтобы посетители увидели изменения сразу же после того, как они внесены? Пострадает ли сайт от того, что изменения вступят в силу с небольшой задержкой?

Статические веб-страницы, которые регулярно генерируются заново, обновляя содержимое, называются полудинамическими. Если вы используете эту модель для высоконагруженных страниц своего сайта, то сможете значительно снизить требования к ресурсам веб-сервера со стороны компонентов, основанных на базе данных.

Допустим, у вас есть скрипт контроллера **index.php**, который на основе шаблона генерирует главную страницу сайта со сводкой свежих материалов. Исследовав журнальные записи сервера, вы, скорее всего, обнаружите, что это одна из самых посещаемых страниц вашего ресурса. Задавшись упомянутыми выше вопросами, вы поймете, что нет необходимости динамически генерировать ее при каждом посещении. Достаточно обновить страницу лишь тогда, когда на сайте появится новая информация. Вместо того чтобы обрабатывать каждый запрос с помощью контроллера, используйте PHP-код, чтобы сгенерировать статическую копию того результата, который выдает шаблон **index.html**, и заменить динамическую версию.

Чтобы реализовать эту маленькую хитрость, вам придется заняться чтением и записью данных, а также некоторыми манипуляциями с файлами. PHP идеально подходит для выполнения данной задачи, однако, прежде чем приступить к делу, вам следует ознакомиться с еще несколькими функциями.

- `file_get_contents`. Открывает файл, считывает содержимое и возвращает его в виде строки. Файл хранится на жестком диске сервера или, как в случае с браузером, загружается по URL с помощью PHP. В случае ошибки функция возвращает `FALSE`.
- `file_put_contents`. Открывает файл и записывает в него указанные данные. Допускает использование различных параметров[1]. Например, вы можете сделать так, чтобы файл не перезаписывался целиком (как это происходит по умолчанию), а только добавлялись данные в его конец.
- `file_exists`. Проверяет, существует ли файл с заданным именем. Если такого файла нет, функция возвращает `FALSE`.
- `copy`. Копирует файл.
- `unlink`. Удаляет файл с жесткого диска.

Начнем с простейших примеров контроллера и шаблона (`chapter12/recentjokes/controller.php` и `chapter12/recentjokes/jokes.html.php`), которые выводят список из трех последних шуток, добавленных в базу данных на вашем сайте (в последний раз вы работали с этим проектом в главе 10).

```
<?php

include_once $_SERVER['DOCUMENT_ROOT'] . '/includes/db.inc.php';

try
{
  $sql = 'SELECT id, joketext FROM joke
      ORDER BY jokedate DESC
      LIMIT 3';
  $result = $pdo->query($sql);
}
catch (PDOException $e)
{
  $error = 'Ошибка при получении шуток.';
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
  exit();
```

[1] Более подробную информацию о доступных параметрах вы найдете в руководстве по PHP (**http://php.net/file_put_contents**).

```
}

foreach ($result as $row)
{
  $jokes[] = array('text' => $row['joketext']);
}

include 'jokes.html.php';
```

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Последние шутки</title>
    <link rel="canonical" href="/recentjokes/">
  </head>
  <body>
    <p>Вот самые свежие шутки в базе данных:</p>
    <?php foreach ($jokes as $joke): ?>
      <?php markdownout($joke['text']); ?>
    <?php endforeach; ?>
  </body>
</html>
```

Скрипт контроллера стоило бы назвать **index.php**, чтобы он запустился при запросе по адресу **http://www.example.com/recentjokes/** и налету сформировал список шуток. Однако в нашем примере контроллер называется **controller.php**. Зная имя файла, браузер все равно сможет к нему обращаться, но, как видно из тега `link rel="canonical"`[1] в шаблоне **jokes.html.php**, все еще ожидается, что посетители придут на страницу **http://www.example.com/recentjokes/**. Браузер, запросивший этот URL, получит вместо контроллера статическую версию подготовленной страницы.

Для генерирования статической версии следует написать еще один скрипт **generate.php**. Его задача — загрузить контроллер **controller.php** (динамическую версию главной страницы), как это делает браузер, и записать на диск статическую копию в виде файла **index.html**. При этом вам нужно сохранить существующую версию **index.html** на случай, если что-то пойдет неправильно. Для этого необходимо, чтобы скрипт записывал новую статическую копию во временный файл **tempindex.html**, и лишь после того, как не будет выявлено проблем, замещал ею **index.html**.

Начните с установки переменных, чтобы сконфигурировать следующие значения: адрес PHP-скрипта, который вам нужно загрузить, имя временного файла и имя статической страницы, которую вы хотите создать (`chapter12/recentjokes/generate.php`, фрагмент).

```
<?php
$srcurl = 'http://localhost/recentjokes/controller.php';
$tempfilename = $_SERVER['DOCUMENT_ROOT'] .
    '/recentjokes/tempindex.html';
$targetfilename = $_SERVER['DOCUMENT_ROOT'] .
    '/recentjokes/index.html';
```

[1] Полное описание этого тега вы найдете в блоге GoogleWebmasterCentralBlog (**http://googlewebmastercentral.blogspot.com/2009/02/specify-your-canonical.html**).

ПЕРЕМЕННАЯ $SRCURL ДОЛЖНА ХРАНИТЬ URL

Если возник соблазн присвоить переменной `$srcurl` путь к файлу controller.php на вашем веб-сервере, лучше ему не поддаваться. Чтобы получить сгенерированную страницу, вам следует запросить скрипт controller.php с помощью пути URL, который ссылается на веб-сервер. Если вы укажите непосредственно путь к файлу, то генератор получит код скрипта controller.php, а не возвращаемый им результат.

На следующем этапе требуется удалить временный файл, который мог остаться после неудачного запуска этого же скрипта. Проверьте его наличие с помощью функции `file_exists` и в случае, если он есть, удалите файл, используя `unlink` (`chapter12/recentjokes/generate.php`, фрагмент).

```
if (file_exists($tempfilename))
{
  unlink($tempfilename);
}
```

Теперь перейдите к загрузке динамической страницы `controller.php`, запросив ее адрес с помощью функции `file_get_contents`. Поскольку используется не прямой путь к файлу, а URL, то до того, как вы получите нужный скрипт, его обработает веб-сервер и вернет в виде обычной статической HTML-страницы (`chapter12/recentjokes/generate.php`, фрагмент).

```
$html = file_get_contents($srcurl);
if (!$html)
{
  $error = "Не удалось загрузить $srcurl. Обновление статической страницы
прервано!";
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
  exit();
}
```

Чтобы записать содержимое страницы, хранящееся в переменной `$html`, в статический HTML-файл, воспользуйтесь функцией `file_put_contents` (`chapter12/recentjokes/generate.php`, фрагмент).

```
if (!file_put_contents($tempfilename, $html))
{
  $error = "Не удалось произвести запись в $tempfilename. Обновление
статической страницы прервано!";
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
  exit();
}
```

Итак, статическая страница записана во временный файл. Скопируйте его, чтобы заменить им предыдущую статическую версию, после чего удалите с помощью `unlink` (`chapter12/recentjokes/generate.php`, фрагмент).

```
copy($tempfilename, $targetfilename);
unlink($tempfilename);
```

Таким образом, при каждом выполнении скрипта generate.php будет генерироваться свежая копия index.html на основе controller.php. Запросите скрипт

generate.php в своем браузере, а затем загрузите директорию recentjokes (например, http://localhost/recentjokes/). Вы должны увидеть содержимое сгенерированного файла index.html.

ОШИБКИ, СВЯЗАННЫЕ С ПРАВАМИ ДОСТУПА К ФАЙЛАМ

В системах Mac OS X и Linux скрипт завершиться неудачно, если он обладает недостаточными правами на копирование и удаление файлов в конкретной указанной директории на вашем сервере. В том случае, когда полученная ошибка сообщает именно об этой проблеме, нужно сделать так, чтобы сервер мог записывать файлы в эту директорию. Как правило, для этого служит команда chmod.

```
chmod 777 /путь/к/recentjokes/
```

Если вам нужна помощь в установке привилегий, которые позволили бы выполнять PHP-скриптам запись в директорию сайта, свяжитесь с технической поддержкой своего веб-хостинга.

Конечно, запрашивать вручную скрипт generate.php при каждом изменении на сайте не очень удобно. Самый простой способ автоматизировать этот процесс — подключить данный скрипт в коде, который отвечает за добавление, удаление или обновление шуток в вашей системе управления содержимым.

Если страница достаточно сложная, найти в ней все участки кода, где следует выполнять генерирование статической версии, будет довольно затруднительно. Одно из решений — настроить свой сервер таким образом, чтобы он запускал скрипт generate.php через определенные промежутки времени, например через каждый час. В Windows вы можете запускать php.exe (автономную версию PHP, которая поставляется вместе с XAMPP и другими дистрибутивами для Windows) ежечасно с помощью Планировщика задач. Для этого создайте файл под названием generate.bat и пропишите в нем следующую строку (chapter12/recentjokes/generate.bat).

```
@C:\xampp\php\php.exe generate.php
```

Исправьте путь и имя файла, если это необходимо, и настройте Планировщик задач так, чтобы он вызывал файл generate.bat каждый час.

В Mac OS X или Linux к аналогичному результату приведет использование системной утилиты cron, которая определяет задачи для регулярного выполнения. Наберите команду man crontab в системном терминале и ознакомьтесь с тем, как подготовить задачу для данной программы.

Задание, которое необходимо назначить для выполнения в Mac OS X или Linux, похоже на задание для Windows. В MAMP также есть автономная версия PHP, которая запускается с помощью cron (в нашем случае она находится в директории /Applications/MAMP/bin/php/php5.3.6/bin/php). Чтобы сработал скрипт generate.php, добавьте в файл crontab следующую строку.

```
0 0-23 * * * /Applications/MAMP/bin/php/php5.3.6/bin/php /путь/к/
generate.php > /dev/null
```

Обеспечение загрузки файлов

Итак, вы уже умеете управляться с теми файлами, что создали сами. Теперь необходимо научиться искусно обращаться с файлами, которые загружают пользователи сайта.

Для начала создадим основу и напишем HTML-форму, позволяющую посетителям размещать документы на сайте. Это легко сделать с помощью тега `input type="file"`. Однако по умолчанию такая форма отправит только имя выбранного файла. Чтобы вместе с данными формы передавался и сам файл, необходимо добавить в тег `form` атрибут `enctype="multipart/form-data"`.

```
<form action="index.php" method="post"
    enctype="multipart/form-data">
  <div><label for="upload">Select file to upload:
    <input type="file" id="upload" name="upload"></label></div>
  <div>
    <input type="hidden" name="action" value="upload">
    <input type="submit" value="Submit">
  </div>
</form>
```

Как вы видите, PHP-скрипт (в нашем случае это index.php) обрабатывает данные, отправленные формой. Информация о загруженных файлах размещается в массиве `$_FILES`, который создается автоматически. Скорее всего, вы полагаете, что информация о файле хранится в элементе `$_FILES['upload']` (ключ которого совпадает с названием атрибута в теге `input`), но это не совсем так. Внутри `$_FILES['upload']` находится не содержимое файла, а еще один массив. Чтобы получить нужную информацию, вам придется использовать вторую пару квадратных скобок.

`$_FILES['upload']['tmp_name']`. Предоставляет имя файла, который хранится на жестком диске веб-сервера во временной директории (или в директории, указанной посредством параметра `upload_tmp_dir` в файле php.ini). При этом файл доступен ровно до тех пор, пока выполняется PHP-скрипт, ответственный за его обработку. Если вы хотите использовать документ в дальнейшем (например, отображать на страницах сайта), вам придется скопировать его в другое место, используя функцию `copy`, описанную в разделе «Полудинамические страницы».

`$_FILES['upload']['name']`. Содержит имя файла, которое тот носил, будучи на клиентском компьютере, до отправления. При создании постоянной копии у вас есть возможность присвоить это первоначальное имя вместо имени временного файла, которое сгенерировано автоматически.

`$_FILES['upload']['size']`. Сообщает размер файла (в байтах).

`$_FILES['upload']['type']`. Содержит MIME-тип файла. Его еще называют **типом файла**, или **типом содержимого**. Это идентификатор, который описывает формат файла, например `text/plain`, `image/png` и т. д.

Помните, что `'upload'` — это всего лишь атрибут в теге `input`, с помощью которого отправлен файл. Вместе с изменением атрибута изменится и индекс массива.

Использование данных переменных помогает решить, принимать загружаемый файл или нет. Например, если у вас фотогалерея, то, скорее всего, вас интересуют файлы формата JPEG, а также, возможно, GIF или PNG. Такие файлы имеют MIME-типы `image/jpeg`, `image/gif` и `image/png` соответственно. Из-за различий между браузерами[1] для проверки типов понадобятся регулярные выражения (если вы забыли их синтаксис, вернитесь к главе 8).

```
if (preg_match('/^image\/p?jpeg$/i', $_FILES['upload']['type']) or
    preg_match('/^image\/gif$/i', $_FILES['upload']['type']) or
    preg_match('/^image\/(x-)?png$/i', $_FILES['upload']['type']))
{
  : Обрабатываем файл.
}
else
{
  $error = 'Пожалуйста, отправьте изображение в формате JPEG, GIF или
PNG.';
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
  exit();
}
```

Аналогичным способом вы могли бы запретить загрузку слишком объемных файлов, проверяя переменную `$_FILES['upload']['size']`, но лучше этого не делать. Еще до того, как вы сможете проверить данное значение, файл уже будет загружен и сохранен во временной директории. Если вы хотите таким образом бороться с ограниченным местом на диске или недостаточной пропускной способностью канала, то у вас все равно возникнут проблемы: несмотря на то что вы удалите файл почти сразу, он так или иначе будет получен.

Более подходящий вариант решения этой проблемы — сообщить интерпретатору о максимально допустимом размере файла. Существуют два способа. Первый заключается в использовании параметра `upload_max_filesize` в файле **php.ini**. По умолчанию загружаются файлы размером до 2 Мбайт. Если вы планируете принимать более объемные загрузки, вам следует изменить это значение[2].

Второй способ предполагает использование формы. В этом случае вы добавляете скрытое поле `input` с именем `MAX_FILE_SIZE` и в качестве значения указываете максимальный размер файлов, которые эта форма может принимать. Из соображений безопасности значение не должно превышать объем, указанный для параметра `upload_max_filesize` в файле **php.ini**. Такой подход позволяет варьировать размеры загрузок в зависимости от страницы. Приведенная ниже форма допускает загрузку данных размером не более 1 Кбайта (1024 байта).

```
<form action="upload.php" method="post"
      enctype="multipart/form-data">
  <p><label id="upload">Выберите файл для загрузки:
```

[1] Конкретное значение MIME-типа зависит от используемого браузера. В Internet Explorer для JPEG и PNG выделены типы `image/pjpeg` и `image/x-png`, а в Firefox и других браузерах применяются обозначения `image/jpeg` и `image/png`.

[2] Файл **php.ini** допускает еще одно ограничение `post_max_size`, которое устанавливается на общий объем данных, отправляемых с помощью формы. Значение этого параметра по умолчанию равно 8 Мбайтам, поэтому если вы хотите принимать по-настоящему большие файлы, то следует изменить и его.

```
    <input type="hidden" name="MAX_FILE_SIZE" value="1024">
    <input type="file" id="upload" name="upload"></label></p>
  <p>
    <input type="hidden" name="action" value="upload">
    <input type="submit" value="Отправить">
  </p>
</form>
```

Обратите внимание, что скрытое поле `MAX_FILE_SIZE` следует указать до тегов `input type="file"`, чтобы интерпретатор узнал об ограничении еще до того, как будут получены какие-либо файлы. Еще одни важный момент заключается в том, что злоумышленники способны легко обойти указанное ограничение, написав собственную форму без поля `MAX_FILE_SIZE`. Чтобы обезопасить себя от загрузки объемных файлов, лучше все-таки воспользоваться параметром `upload_max_filesize` в **php.ini**.

Чтобы сохранить загруженные файлы, их нужно скопировать в другую директорию. И хотя у вас есть доступ к имени файла через переменную `$_FILE['upload']['name']`, нет никакой гарантии, что вам не загрузят два документа с одинаковыми названиями. В таком случае использование оригинальных имен приведет к перезаписи старых файлов вновь загруженными.

Чтобы избежать подобной проблемы, применяют специальную схему, позволяющую присваивать всем загружаемым файлам уникальные имена. Используя системное время (доступ к которому обеспечивает функция `time`), вы можете легко сгенерировать название исходя из количества секунд, прошедших с 1 января 1970 года. Но что если два файла окажутся загруженными в одну и ту же секунду? Чтобы застраховаться от такой ситуации, в имя файла добавляется IP-адрес клиента, который автоматически сохраняется в переменной `$_SERVER['REMOTE_ADDR']`. В большинстве случаев этого достаточно, поскольку вряд ли вы в течение одной секунды получите два файла одинаковым именем с одного и того же IP.

```
// Извлекаем расширение файла.
if (preg_match('/^image\/p?jpeg$/i', $_FILES['upload']['type']))
{
  $ext = '.jpg';
}
else if (preg_match('/^image\/gif$/i', $_FILES['upload']['type']))
{
  $ext = '.gif';
}
else if (preg_match('/^image\/(x-)?png$/i',
    $_FILES['upload']['type']))
{
  $ext = '.png';
}
else
{
  $ext = '.unknown';
}

// Имя файла и полный путь к нему
$filename = 'C:/uploads/' . time() . $_SERVER['REMOTE_ADDR'] . $ext;

// Копируем файл (если это безопасно).
```

```
if (!is_uploaded_file($_FILES['upload']['tmp_name']) or
    !copy($_FILES['upload']['tmp_name'], $filename))
{
  $error = "Не удалось сохранить файл под именем $filename!";
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
  exit();
}
```

В данном коде стоит обратить внимание на использование функции `is_uploaded_file`, которая проверяет, безопасен ли файл. Она возвращает `TRUE` при условии, что имя файла, переданное ей в качестве параметра (в нашем случае это `$_FILES['upload']['tmp_name']`), загружено вместе с формой. Если злоумышленник откроет скрипт и вручную укажет вместо имени файла **/etc/passwd** (системное хранилище паролей на серверах под управлением Linux), а вы не проверите с помощью функции `is_uploaded_file`, что переменная `$_FILES['upload']` действительно ссылается на загруженный файл, то скрипт позволит скопировать важные файлы с сервера в публичную директорию, доступную в Интернете. Именно поэтому, прежде чем доверять переменной, которая, по вашему мнению, содержит имя загруженного файла, проверьте ее с помощью `is_uploaded_file`.

В вышеприведенном коде есть еще один интересный прием — сочетание функций `is_uploaded_file` и `copy` внутри условия для оператора `if`. Если выражение `is_uploaded_file($_FILES['upload']['tmp_name'])` вернет `FALSE` (при этом `!is_uploaded_file($_FILES['upload']['tmp_name'])` будет равно `TRUE`), то PHP сразу узнает, что условие выполняется целиком (поскольку вызов двух функций разделяет оператор `or`), и, чтобы сэкономить время, не станет запускать функцию `copy`. Таким образом, если вызов `is_uploaded_file` вернет `FALSE`, то файл не скопируется. Если же `is_uploaded_file` вернет `TRUE`, PHP пойдет дальше и выполнит копирование, о результатах которого вы сможете судить по тому, появится ли сообщение об ошибке или нет. Если вместо оператора `or` использовать `and`, то результат `FALSE` в первой части условия приведет к пропуску и во второй части. Это свойство конструкции `if` называется **вычислением по короткой схеме**, или **ленивым вычислением.** Оно работает и в других условных структурах, таких как циклы `while` и `for`.

И наконец, последняя особенность: несмотря на то что код написан для Windows, для составления пути применяется слеш (/) в стиле Unix. Если бы использовался обратный слеш, то его пришлось бы дублировать (\\), чтобы он не был воспринят PHP как символ для экранирования. Интерпретатор способен преобразовать обычный слеш в обратный, если скрипт работает в Windows. Аналогичным образом можно составлять пути к файлам с помощью обычного слеша (/) и в других системах, делая тем самым скрипты более переносимыми.

Запись загруженных файлов в базу данных

К этому моменту созданная вами система позволяет загружать изображения в форматах `JPEG`, `GIF` и `PNG` и сохранять их на сервер. Если использовать ее в нынешнем виде, то загружаемые изображения сначала потребуется собрать в отдельном каталоге, где они сохраняются, а затем добавить на сайт вручную. Если вы теперь вспомните конец главы 9, где вам предлагалось разработать систему, позволяющую посетителям сайта отправлять шутки в базу данных для последующего одобрения редактором, то вы поймете, что эта задача требует более разумного решения.

В MySQL бинарную информацию позволяют хранить столбцы нескольких типов. Если использовать терминологию, то можно сказать, что эти типы поддерживают большие бинарные объекты (Binary Large Objects, или **BLOB**). Однако размещать в базе данных потенциально объемные файлы — все-таки плохая идея. Несмотря на то, что держать всю информацию в одном месте довольно удобно, большие файлы приводят к увеличению размеров базы данных, снижая ее производительность и раздувая до неприличных объемов резервные копии.

Лучше всего хранить в базе данных только *имена* файлов. Если при удалении записей вы не забудете удалить файлы, которые с ними связаны, все будет работать корректно. Не станем вдаваться в подробности, поскольку вы уже видели SQL-код, необходимый для решения этой задачи. Если вдруг вам понадобится помощь, вы сможете получить ее на форумах сообщества SitePoint.

Когда речь идет об относительно небольших файлах (например, о портретах сотрудников компании), их размещение в MySQL вполне оправданно. В оставшейся части главы вы узнаете, как использовать PHP для сохранения в MySQL бинарной информации, загруженной с помощью Интернета, и научитесь извлекать файлы для скачивания или отображения.

Типы бинарных столбцов

Первое, что стоит определить при создании веб-приложения, — структуру базы данных. Чтобы отделить настоящий пример от сайта с шутками, создайте еще одну базу данных.

```
CREATE DATABASE filestore
```

Если это невозможно (ваш удаленный MySQL-сервер допускает использование всего одной базы данных), продолжайте работать с прежней базой данных.

Для каждого файла необходимо сохранить имя, MIME-тип (например, `image/jpeg` для изображений в формате JPEG), краткое описание и собственно бинарные данные. Вот выражение `CREATE TABLE` для создания соответствующей таблицы (`chapter12/sql/filestore.sql`, фрагмент).

```
CREATE TABLE filestore (
  id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
  filename VARCHAR(255) NOT NULL,
  mimetype VARCHAR(50) NOT NULL,
  description VARCHAR(255) NOT NULL,
  filedata MEDIUMBLOB
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB
```

По большей части синтаксис вам знаком. Новым для вас является тип столбца `MEDIUMBLOB`. Заглянув в приложение Г, вы узнаете, что тип `MEDIUMBLOB` отличается от `MEDIUMTEXT` только тем, что поиск и сортировка в нем происходят с учетом регистра. С точки зрения MySQL между бинарными данными и блоками текста нет никакой разницы: и то и другое — это всего лишь последовательность байтов, хранящаяся в базе данных. Просто MySQL применяет к текстовым типам множество дополнительных правил, чтобы сортировка строк и преобразования между кодировками символов выполнялись прозрачно.

Таким образом, используя тип MEDIUMBLOB, вы повышаете производительность базы данных, игнорируя дополнительные правила. Кроме того, этот тип позволяет сравнивать содержимое двух бинарных файлов с учетом регистра, используя последовательности байтов. В отличие от него MEDIUMTEX не способен различить прописную и строчную букву А, несмотря на то, что они состоят из разных байтов.

MEDIUMBLOB — это один из нескольких типов BLOB-столбцов, которые предназначены для хранения данных переменной длины. Друг от друга эти типы отличаются только двумя параметрами: максимальным объемом данных, который допустимо содержать в столбце, и количеством байтов, которые используются для хранения длины каждого значения (табл. 12.1).

Таблица 12.1. Бинарные столбцы, используемые в MySQL

Тип столбца	Максимальный размер	Место, занимаемое каждой записью
TINYBLOB	255 байт	Размер данных + 1 байт
BLOB	65 Кбайт	Размер данных + 2 байта
MEDIUMBLOB	16,7 Мбайт	Размер данных + 3 байта
LONGBLOB	4,3 Гбайт	Размер данных + 4 байта

Как видите, созданная вами таблица способна хранить файлы размером не более 16,7 Мбайт. Если вы думаете, что вам понадобится работать с файлами большего объема, нужно поменять тип столбца `filedata` на LONGBLOB. В этом случае каждая строка станет занимать в базе данных немного больше места (дополнительный байт нужен MySQL, чтобы записать размер большого файла), а вы получите возможность сохранять файлы размером до 4,3 Гбайт (если, конечно, их поддерживает ваша система).

Если вы создали эту таблицу в отдельной базе данных, то, чтобы ваши скрипты смогли подключиться к серверу, вам понадобится новый файл **db.inc.php** (`chapter12/filestore/db.inc.php`).

```
<?php
try
{
  $pdo = new PDO('mysql:host=localhost;dbname=filestore',
      'filestoreuser', 'mypassword');
  $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $pdo->exec('SET NAMES "utf8"');
}
catch (PDOException $e)
{
  $error = 'Не удалось подключиться к серверу баз данных.';
  include 'error.html.php';
  exit();
}
```

Сохранение файлов

Теперь, когда база данных готова, следующим шагом станет создание контроллера и шаблона, с помощью которых пользователи получат возможность загружать файлы и сохранять их в базе данных. Не спешите копировать коды в следующих двух разделах: полное содержимое скриптов представлено в конце главы. Код для

формы не должен вызвать у вас затруднений (chapter12/filestore/files.html.php, фрагмент).

```
<form action="" method="post" enctype="multipart/form-data">
  <div>
    <label for="upload">Загрузить файл:
    <input type="file" id="upload" name="upload"></label>
  </div>
  <div>
    <label for="desc">Описание файла:
    <input type="text" id="desc" name="desc"
        maxlength="255"></label>
  </div>
  <div>
    <input type="hidden" name="action" value="upload">
    <input type="submit" value="Загрузить">
  </div>
</form>
```

Как вы уже знаете из этой главы, форма создает на сервере временный файл и сохраняет его имя в переменную $_FILES['upload']['tmp_name']. Она также устанавливает переменные $_FILES['upload']['name'] (первоначальное имя файла), $_FILES['upload']['size'] (размер файла в байтах) и $_FILES['upload']['type'] (MIME-тип файла).

Добавить файл в базу данных довольно просто: нужно считать информацию из временного файла в переменную, а затем использовать эту переменную в обычном запросе с командой INSERT. Однако прежде чем это сделать, не забудьте применить функцию is_uploaded_file и убедиться, что переданному имени соответствует загруженный файл (chapter12/filestore/index.php, фрагмент).

```
<?php
include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/magicquotes.inc.php';

if (isset($_POST['action']) and $_POST['action'] == 'upload')
{
  // Останавливаем скрипт, если этот файл не был загружен
  if (!is_uploaded_file($_FILES['upload']['tmp_name']))
  {
    $error = 'Файл не был загружен!';
    include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
    exit();
  }
  $uploadfile = $_FILES['upload']['tmp_name'];
  $uploadname = $_FILES['upload']['name'];
  $uploadtype = $_FILES['upload']['type'];
  $uploaddesc = $_POST['desc'];
  $uploaddata = file_get_contents($uploadfile);

  include 'db.inc.php';

  try
  {
    $sql = 'INSERT INTO filestore SET
```

```
            filename = :filename,
            mimetype = :mimetype,
            description = :description,
            filedata = :filedata';
        $s = $pdo->prepare($sql);
        $s->bindValue(':filename', $uploadname);
        $s->bindValue(':mimetype', $uploadtype);
        $s->bindValue(':description', $uploaddesc);
        $s->bindValue(':filedata', $uploaddata);
        $s->execute();
      }
      catch (PDOException $e)
      {
        $error = 'Ошибка при сохранении файла в базе данных!';
        include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
        exit();
      }

      header('Location: .');
      exit();
    }
```

Отображение сохраненных файлов

Итак, полдела вы уже сделали: написали код, который принимает загруженные файлы и сохраняет их в базу данных. Теперь для дальнейшего использования информации ее нужно каким-то образом извлечь из базы данных. В нашем случае запрашиваемые файлы следует передать браузеру.

Данный процесс также не сложен: вы получаете из базы данных содержимое запрашиваемого файла и отправляете его в браузер. Единственная трудность заключается в том, как передать информацию о файле:

- **размер** — браузер должен показывать пользователю точную информацию о выполнении загрузки;
- **тип** — браузеру необходимо знать, что делать с полученными данными: отобразить их как веб-страницу, текстовый файл, изображение или же предложить сохранить файл на диск;
- **имя** — если не указать его, браузер посчитает, что загруженный файл имеет то же название, что и скрипт контроллера.

Вся эта информация отправляется браузеру посредством **HTTP-заголовков**, которые предваряют передачу содержимого файла. Как вы уже знаете, передача HTTP-заголовков в PHP выполняется с помощью функции `header`. Единственный нюанс — процесс должен быть завершен до отправки содержимого файла, поэтому данную функцию следует вызвать до того, как скрипт выведет какую-либо информацию.

Размер файла указывается в заголовке `Content-length` (`chapter12/filestore/index.php`, фрагмент).

```
header('Content-length: ' . strlen($filedata));
```

`strlen` — это встроенная функция, которая возвращает длину заданной строки. С ее помощью получают размер бинарных данных (в байтах), поскольку с точки зрения PHP это всего лишь последовательность байтов.

Тип файла задается в заголовке `Content-type` (`chapter12/filestore/index.php`, фрагмент).

```
header("Content-type: $mimetype");
```

И наконец, имя файла передается с помощью заголовка `Content-disposition`.

```
header("Content-disposition: inline; filename=$filename");
```

Следующий код позволяет извлечь из базы данных файл с указанным идентификатором и отправить его в браузер.

```
include 'db.inc.php';

try
{
  $sql = 'SELECT filename, mimetype, filedata
      FROM filestore
      WHERE id = :id';
  $s = $pdo->prepare($sql);
  $s->bindValue(':id', $_GET['id']);
  $s->execute();
}
catch (PDOException $e)
{
  $error = 'Ошибка при извлечении файла из базы данных.';
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
  exit();
}

$file = $s->fetch();
if (!$file)
{
  $error = 'Файл с указанным id не найден в базе данных!';
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
  exit();
}

$filename = $file['filename'];
$mimetype = $file['mimetype'];
$filedata = $file['filedata'];

header('Content-length: ' . strlen($filedata));
header("Content-type: $mimetype");
header("Content-disposition: inline; filename=$filename");

echo $filedata;
exit();
```

В завершение необходимо сделать так, чтобы файлы можно было не только просматривать, но и скачивать. Согласно веб-стандартам для этого в заголовке

Content-disposition вместо значения inline нужно указать attachment. Ниже представлена модифицированная версия кода. Она проверяет, содержит ли переменная $_GET['action'] значение 'download': положительный ответ говорит о том, что необходимо отправить особый заголовок.

```
include 'db.inc.php';

try
{
  $sql = 'SELECT filename, mimetype, filedata
      FROM filestore
      WHERE id = :id';
  $s = $pdo->prepare($sql);
  $s->bindValue(':id', $_GET['id']);
  $s->execute();
}
catch (PDOException $e)
{
  $error = 'Ошибка при извлечении файла из базы данных.';
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
  exit();
}

$file = $s->fetch();
if (!$file)
{
  $error = 'Файл с указанным id не найден в базе данных!';
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
  exit();
}

$filename = $file['filename'];
$mimetype = $file['mimetype'];
$filedata = $file['filedata'];
$disposition = 'inline';

if ($_GET['action'] == 'download')
{
  $disposition = 'attachment';
}

header('Content-length: ' . strlen($filedata));
header("Content-type: $mimetype");
header("Content-disposition: $disposition; filename=$filename");

echo $filedata;
exit();
```

Однако, старые версии браузеров склонны игнорировать заголовок Content-disposition. Чтобы определить, что именно делать с файлом, они используют для анализа заголовок Content-type, особенно если он идет после Content-disposition.

Чтобы как можно больше браузеров вели себя при загрузке должным образом, убедитесь, что заголовок Content-type идет перед Content-disposition и укажите вместо реального MIME-типа файла универсальное значение application/

octet-stream, которое укажет старым версиям браузеров на то, что требуется начать загрузку (chapter12/filestore/index.php, фрагмент).

```
include 'db.inc.php';

try
{
  $sql = 'SELECT filename, mimetype, filedata
      FROM filestore
      WHERE id = :id';
  $s = $pdo->prepare($sql);
  $s->bindValue(':id', $_GET['id']);
  $s->execute();
}
catch (PDOException $e)
{
  $error = 'Ошибка при извлечении файла из базы данных.';
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
  exit();
}

$file = $s->fetch();
if (!$file)
{
  $error = 'Файл с указанным id не найден в базе данных!';
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
  exit();
}

$filename = $file['filename'];
$mimetype = $file['mimetype'];
$filedata = $file['filedata'];
$disposition = 'inline';

if ($_GET['action'] == 'download')
{
  $mimetype = 'application/octet-stream';
  $disposition = 'attachment';
}

// Заголовок Content-type должен идти перед Content-disposition
header('Content-length: ' . strlen($filedata));
header("Content-type: $mimetype");
header("Content-disposition: $disposition; filename=$filename");

echo $filedata;
exit();
```

Собираем все воедино

Ниже представлена полная версия файлового хранилища. Помимо тех элементов, которые вы рассмотрели в предыдущем разделе, она содержит простой код для отображения списка файлов в базе данных, позволяющий просмотреть файлы, скачать их или удалить.

Вот как выглядит скрипт контроллера (chapter12/filestore/index.php).

```
<?php
include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/magicquotes.inc.php';

if (isset($_POST['action']) and $_POST['action'] == 'upload')
{
  // Останавливаем скрипт, если этот файл не был загружен.
  if (!is_uploaded_file($_FILES['upload']['tmp_name']))
  {
    $error = 'Файл не был загружен!';
    include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
    exit();
  }
  $uploadfile = $_FILES['upload']['tmp_name'];
  $uploadname = $_FILES['upload']['name'];
  $uploadtype = $_FILES['upload']['type'];
  $uploaddesc = $_POST['desc'];
  $uploaddata = file_get_contents($uploadfile);

  include 'db.inc.php';

  try
  {
    $sql = 'INSERT INTO filestore SET
        filename = :filename,
        mimetype = :mimetype,
        description = :description,
        filedata = :filedata';
    $s = $pdo->prepare($sql);
    $s->bindValue(':filename', $uploadname);
    $s->bindValue(':mimetype', $uploadtype);
    $s->bindValue(':description', $uploaddesc);
    $s->bindValue(':filedata', $uploaddata);
    $s->execute();
  }
  catch (PDOException $e)
  {
    $error = 'Ошибка при сохранении файла в базе данных!';
    include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
    exit();
  }

  header('Location: .');
  exit();
}

if (isset($_GET['action']) and
    ($_GET['action'] == 'view' or $_GET['action'] == 'download') and
    isset($_GET['id']))
{
  include 'db.inc.php';

  try
  {
    $sql = 'SELECT filename, mimetype, filedata
```

```
            FROM filestore
            WHERE id = :id';
        $s = $pdo->prepare($sql);
        $s->bindValue(':id', $_GET['id']);
        $s->execute();
      }
      catch (PDOException $e)
      {
        $error = 'Ошибка при извлечении файла из базы данных.';
        include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
        exit();
      }

      $file = $s->fetch();
      if (!$file)
      {
        $error = 'Файл с указанным id не найден в базе данных!';
        include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
        exit();
      }

      $filename = $file['filename'];
      $mimetype = $file['mimetype'];
      $filedata = $file['filedata'];
      $disposition = 'inline';

      if ($_GET['action'] == 'download')
      {
        $mimetype = 'application/octet-stream';
        $disposition = 'attachment';
      }

      // Заголовок Content-type должен идти перед Content-disposition.
      header('Content-length: ' . strlen($filedata));
      header("Content-type: $mimetype");
      header("Content-disposition: $disposition; filename=$filename");

      echo $filedata;
      exit();
    }

    if (isset($_POST['action']) and $_POST['action'] == 'delete' and
        isset($_POST['id']))
    {
      include 'db.inc.php';

      try
      {
        $sql = 'DELETE FROM filestore
            WHERE id = :id';
        $s = $pdo->prepare($sql);
        $s->bindValue(':id', $_POST['id']);
        $s->execute();
      }
      catch (PDOException $e)
      {
        $error = 'Ошибка при удалении файла из базы данных.';
```

```
      include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
      exit();
    }

    header('Location: .');
    exit();
}

include 'db.inc.php';

try
{
  $result = $pdo->query(
      'SELECT id, filename, mimetype, description
      FROM filestore');
}
catch (PDOException $e)
{
  $error = 'Ошибка при извлечении файла из базы данных.';
  include $_SERVER['DOCUMENT_ROOT'] . '/includes/error.html.php';
  exit();
}

$files = array();
foreach ($result as $row)
{
  $files[] = array(
      'id' => $row['id'],
      'filename' => $row['filename'],
      'mimetype' => $row['mimetype'],
      'description' => $row['description']);
}

include 'files.html.php';
```

Теперь рассмотрим шаблон, который содержит форму и выводит список файлов (`chapter12/filestore/files.html.php`).

```
<?php include_once $_SERVER['DOCUMENT_ROOT'] .
    '/includes/helpers.inc.php'; ?>
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Репозиторий файлов на основе PHP/MySQL</title>
  </head>
  <body>
    <h1>Репозиторий файлов на основе PHP/MySQL</h1>

    <form action="" method="post" enctype="multipart/form-data">
      <div>
        <label for="upload">Загрузить файл:
        <input type="file" id="upload" name="upload"></label>
      </div>
      <div>
        <label for="desc">Описание файла:
        <input type="text" id="desc" name="desc"
```

```
            maxlength="255"></label>
        </div>
        <div>
          <input type="hidden" name="action" value="upload">
          <input type="submit" value="Загрузить">
        </div>
      </form>

      <?php if (count($files) > 0): ?>

      <p>В базе данных хранятся следующие файлы:</p>

      <table>
        <thead>
          <tr>
            <th>Имя файла</th>
            <th>Тип</th>
            <th>Описание</th>
          </tr>
        </thead>
        <tbody>
          <?php foreach($files as $f): ?>
          <tr>
            <td>
              <a href="?action=view&amp;id=<?php htmlout($f['id']); ?>"
                 ><?php htmlout($f['filename']); ?></a>
            </td>
            <td><?php htmlout($f['mimetype']); ?></td>
            <td><?php htmlout($f['description']); ?></td>
            <td>
              <form action="" method="get">
                <div>
                  <input type="hidden" name="action" value="download"/>
                  <input type="hidden" name="id" value="<?php
                  htmlout($f['id']); ?>"/>
                  <input type="submit" value="Скачать"/>
                </div>
              </form>
            </td>
            <td>
              <form action="" method="post">
                <div>
                  <input type="hidden" name="action" value="delete"/>
                  <input type="hidden" name="id" value="<?php
                  htmlout($f['id']); ?>"/>
                  <input type="submit" value="Удалить"/>
                </div>
              </form>
            </td>
          </tr>
          <?php endforeach; ?>
        </tbody>
      </table>

      <?php endif; ?>
    </body>
  </html>
```

И наконец, подключаемый файл для соединения с базой данных (`chapter12/filestore/db.inc.php`).

```
<?php
try
{
  $pdo = new PDO('mysql:host=localhost;dbname=filestore',
'filestoreuser', 'mypassword');
  $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $pdo->exec('SET NAMES "utf8"');
}
catch (PDOException $e)
{
  $error = 'Не удалось подключиться к серверу баз данных.';
  include 'error.html.php';
  exit();
}
```

Обратите внимание, что база данных (`filestore`) и пользователь (`filestoreuser`) иные, чем для проекта со списком шуток. Если вы решили разместить таблицу `filestore` внутри базы данных `ijdb` вместе с другими данными, которые в ней находятся, то для соединения можно использовать общий подключаемый файл **db.inc.php**.

После того как вы допишите все файлы и подготовите базу данных, запустите браузер и посмотрите, что у вас получилось. Страница с пустым репозиторием представлена на рис. 12.1.

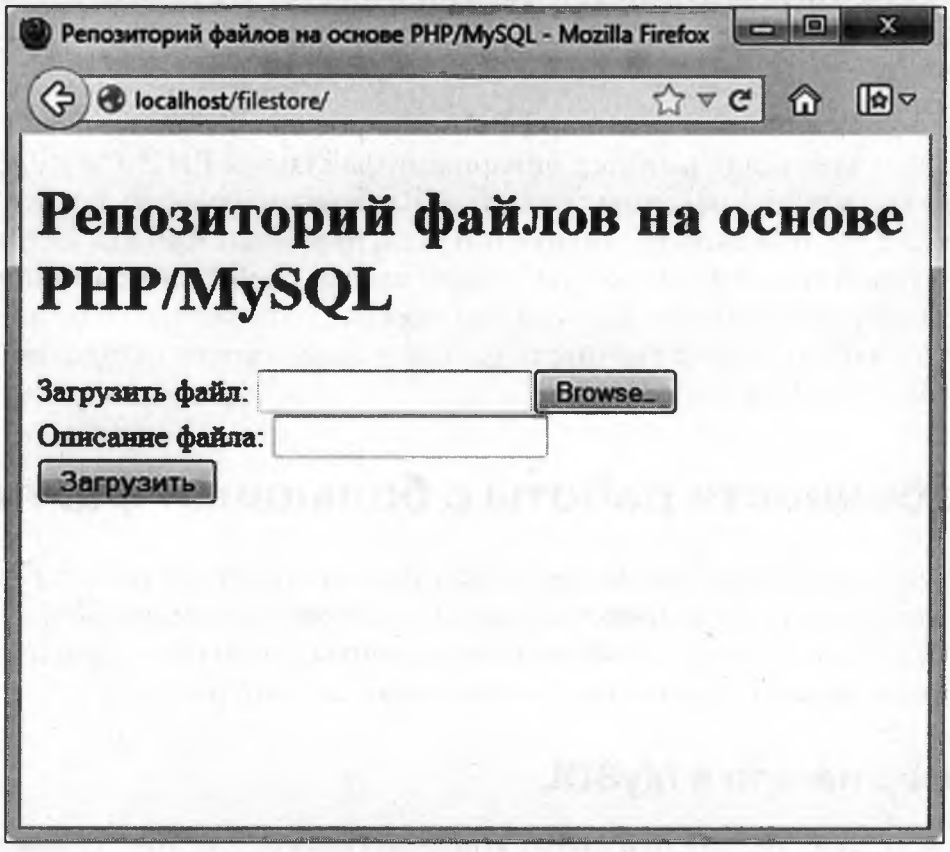


Рис. 12.1. Пустой репозиторий

Загружаемые вами файлы появятся в таблице (рис. 12.2).

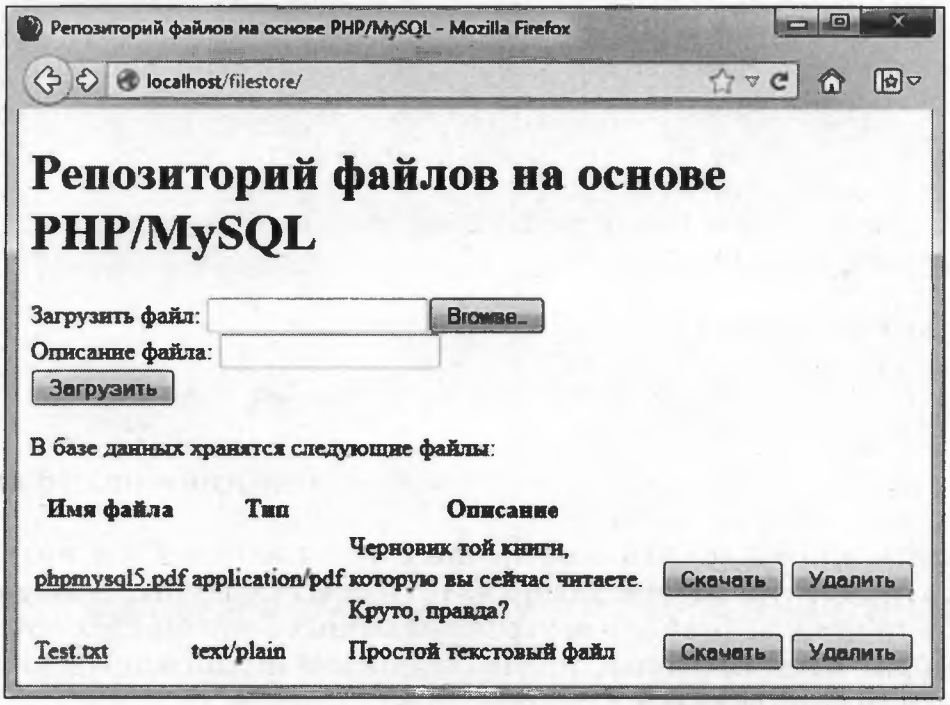


Рис. 12.2. Несколько загруженных файлов

Нажмите на имени файла, чтобы открыть его в браузере (при условии, что браузер поддерживает данный тип файлов). Попробуйте нажать кнопки **Скачать** и **Удалить**, которые находятся напротив каждой записи. Они должны сработать так, как вы того ожидаете.

На примере данного проекта продемонстрированы все технические приемы, которые необходимы для работы с бинарными файлами в PHP и MySQL. Подойдите творчески к использованию этого кода. Разберите ситуацию, когда файловый архив требует логин и пароль для просмотра загруженных файлов, а если пользователь не проходит аутентификацию, скрипт выдает сообщение об ошибке вместо того, чтобы отправить данные. Еще одно интересное усовершенствование — создать скрипт, который позволяет скачивать файлы в зависимости от информации, отправленной с помощью формы.

Особенности работы с большими файлами

В системах, подобных той, которую вы только что создали, работа с большими файлами вынуждает программистов решать некоторые весьма специфические задачи. Их анализ выходит за рамки рассматриваемых в этой книге тем, но получить краткую информацию о некоторых из них вам будет интересно.

Размер пакета в MySQL

По умолчанию MySQL отклоняет команды (пакеты), размер которых превышает 1 Мбайт. Это вводит разумное, хотя и строгое ограничение относительно

максимального размера хранящихся файлов. Тем не менее вы можете его обойти, если разобьете содержимое файла на фрагменты по 1 Мбайту и добавите их посредством нескольких команд UPDATE, идущих за запросом INSERT. Чтобы увеличить максимальный размер пакета, нужно установить параметр max_allowed_packet в конфигурационных файлах my.cnf или my.ini. Более подробно об этой проблеме вы прочтете в руководстве по MySQL (http://dev.mysql.com/doc/refman/5.1/en/packet-too-large.html).

Ограничение потребляемой памяти в PHP

По умолчанию интерпретатор PHP сконфигурирован так, чтобы задействовать для обработки любого отдельного запроса не более 8 Мбайт памяти. Если скрипт пытается прочитать целиком файл, чей размер близок к этому лимиту или превышает его, то браузер, скорее всего, выдаст сообщение о том, что PHP не удалось выделить память. Чтобы решить эту проблему, внесите изменения в файл php.ini на своем сервере, увеличив значение memory_limit.

Ограничение времени выполнения скрипта в PHP

По умолчанию PHP завершает работу скрипта, если она длится более 30 секунд. Очевидно, что приблизится к этому ограничению довольно легко, если попробовать загрузить большой файл при медленном соединении. Используйте функцию set_time_limit, чтобы указать подходящий лимит для загрузки, или укажите для нее значение, равное нулю, — это позволит скрипту выполнить команды до конца вне зависимости от того, сколько времени для этого потребуется. Поступайте так только в том случае, если уверены, что скрипт обязательно завершится и не будет работать вечно.

Подводя итоги

Рассмотрев в этой главе реальный пример с загрузкой файлов и хранением бинарной информации в базе данных, вы закончили изучение PHP и MySQL. Следует признать, что книга заканчивается довольно неожиданно: есть еще много других аспектов PHP и MySQL, достойных внимания и не менее важных и фундаментальных, чем бинарные данные.

Одному только PHP (с его философией «все включено», подразумевающей добавление различного рода возможностей в виде встроенных функций) можно посвятить десятки книг не меньшего объема, чем эта. В зависимости от особенностей проекта вам могут пригодиться совершенно разные аспекты языка. Даже автор данной книги, с многолетним стажем в области PHP-разработок, вынужден признать, что не знаком и с *половиной* тех возможностей, которые предоставляет этот язык. Тема для исследования слишком обширна. Именно поэтому так мало людей решились распечатать руководство по PHP целиком (http://php.net/docs.php).

Самый лучший способ закрепить приобретенные знания о PHP и MySQL — применить их на практике: создать с нуля собственный сайт на основе базы данных, используя описанные в книге приемы; выложить его в Интернете и попросить

реальных пользователей оценить вашу работу. Скорее всего, вы получите стимул для внесения дальнейших улучшений, которые вы не смогли реализовать сразу из-за нехватки опыта. Требования, которые диктует реальный мир, должны подтолкнуть вас к углубленному изучению PHP и MySQ. А изучать там есть что!

Отличным помощником в совершенствовании знаний может стать для вас книга издательства SitePoint «Специалист в PHP: пишем современный код» (PHP Master: Write Cutting-edge Code; http://www.sitepoint.com/books/phppro1/). В ней демонстрируются эффективные способы решения многих проблем, в том числе и тех, которые вы здесь встречали, но при этом за основу берутся объектно-ориентированные возможности языка PHP.

Разработав несколько самостоятельных проектов, вы заметите, что некоторые фрагменты кода повторяются снова и снова. Вместо того чтобы тратить время на пополнение собственной коллекции общих подключаемых файлов, вам следует обратить внимание на фреймворки, такие как Zend Framework (http://framework.zend.com/), CakePHP (http://cakephp.org/) или Symfony (http://www.symfony-project.org/).

Каждый из этих фреймворков — результат долгой работы специалистов в области PHP. В них предусмотрены готовые решения для большинства проблем, с которыми часто сталкиваются PHP-программисты. Применяя эти решения в собственных проектах, вы перестанете изобретать велосипед и сможете сосредоточиться на написании кода, который выполняет уникальные задачи. Любой фреймворк имеет свою философию, сильные и слабые стороны. Чтобы найти тот, который подходит именно вам, придется приложить некоторые усилия. Однако, если вы планируете стать профессиональным разработчиком на языке PHP, это время вы потратите не зря.

Находясь даже в самом начале своей карьеры в области программирования, вы обладаете серьезными базовыми знания, чего нельзя сказать о многих современных разработчиках. Не премините воспользоваться этим преимуществом.

Так чего же вы ждете? Идите и напишите какой-нибудь код!

Приложение А
РУЧНАЯ УСТАНОВКА PHP И MySQL

В главе 1 для получения на компьютере веб-сервера с поддержкой PHP и MySQL рекомендовалось использовать такие пакеты, как XAMPP или MAMP. Эти среды разработки уже содержат все необходимое, их легко включить и выключить, полностью удалить и установить заново даже в том случае, если вы новичок.

Тем не менее может наступить момент, когда вам захочется установить PHP и MySQL самостоятельно, хотя бы для того, чтобы понять, как их компоненты взаимодействуют друг с другом. В этом приложении вы рассмотрите процесс их установки на каждой из основных платформ — Windows, Mac OS X и Linux.

Windows

Установка MySQL

Программа MySQL совершенно бесплатна. Чтобы установить ее, откройте на сайте MySQL страницу с загрузками (http://dev.mysql.com/downloads/) и в разделе MySQL Community Server перейдите по ссылке Download. Появится страница со списком ссылок для скачивания рекомендованной версии MySQL (на момент написания данной книги это версия 5.5.22).

Вверху списка находятся ссылки на 32- и 64-битные версии Windows. Если вы уверены, что у вас 64-битная система, нажмите кнопку Download рядом с названием пакета Windows (x86, 64-bit), MSI Installer (около 33 Мбайт). Если же вы сомневаетесь или у вас установлена 32-битная версия Windows, используйте пакет Windows (x86, 32-bit), MSI Installer (около 31 Мб), он будет работать даже в 64-битной системе.

После того как файл загрузился, дважды щелкните на его названии и пройдите через процесс установки аналогично любой другой программе. При выборе типа настроек укажите пункт Typical (если только у вас нет особых предпочтений относительно директории, в которую будет произведена установка). По окончании ответьте утвердительно на вопрос, нужно ли запустить мастер конфигурации MySQL (Instance Configuration Wizard), и выберите пункт Detailed Configuration. С его помощью задаются параметры, обеспечивающие совместимость с PHP. Выберите следующие варианты.

- **Тип сервера (Server type).** Исходя из того, что вы устанавливаете MySQL на настольном компьютере, выберите пункт Developer Machine.
- **Использование базы данных (Database usage).** Хотя любой из предложенных вариантов подойдет для примеров, рассмотренных в данной книге, лучше оставить параметр по умолчанию — Multifunctional Database.
- **Настройки табличной области InnoDB (InnoDB tablespace settings).** Здесь указано место для хранения файлов базы данных. По умолчанию выбрана та директория, в которую установлен сервер MySQL. Это идеально подходит для нужд разработки, поэтому ничего менять не нужно.
- **Максимальное количество соединений (Connection limit).** Выберите пункт Decision Support (DSS)/OLAP, чтобы оптимизировать MySQL для относительно небольшого количества соединений.
- **Сетевые настройки (Networking options).** Отмените выбор параметра Enable Strict Mode, чтобы обеспечить совместимость сервера MySQL с более старыми версиями PHP, которые могут понадобиться для работы.
- **Кодировка по умолчанию (Default character set).** Выберите пункт Best Support For Multilingualism. Так сервер MySQL поймет, что вы хотите использовать текст, набранный в кодировке UTF-8, которая поддерживает весь спектр символов, используемых в современных интернет-проектах.
- **Настройки Windows (Windows options).** Позволяет установить программу MySQL в виде службы Windows, которая запускается автоматически. Выберите пункт Include Bin Directory in Windows PATH, чтобы использовать администраторские инструменты MySQL в командной строке.
- **Настройки безопасности (Security options).** Укажите пароль `root` для учетной записи администратора, которая предоставляет доступ ко всем базам данных, хранящимся на сервере MySQL. Остальные параметры оставьте без изменений.

Как только вы закончите работу с мастером конфигурации, система полностью готова к запуску сервера MySQL.

Убедитесь, что сервер MySQL работает должным образом. Для этого нажмите клавиши Ctrl+Alt+Del и откройте Диспетчер задач. На вкладке Процессы нажмите кнопку Показать процессы всех пользователей (если она не была нажата). Если все функционирует, как положено, в списке будет значиться программа сервера mysqld.exe, запущенная вместе с системой.

Установка PHP

Следующий шаг — установка PHP. Перейдите на страницу http://windows.php.net/download/ для его загрузки. Вы увидите две версии PHP 5.4.x для Windows — VC9 Non Thread Safe и VC9 Thread Safe. Вам нужна вторая, потокобезопасная. Первая версия не подходит при использовании в качестве подключаемого модуля для Apache.

Загрузите версию VC9 Thread Safe в виде пакета в формате zip.

Интерпретатор PHP спроектирован как дополнение к уже имеющемуся серверному программному обеспечению, такому как Apache или IIS, поэтому перед установкой интерпретатора необходимо предварительно настроить веб-сервер.

Многие версии Windows поставляются вместе со встроенным веб-сервером IIS. Исключение составляют Windows XP Home, Windows Vista Home, Windows 7 Home Basic и некоторые другие. Если вы хотите разрабатывать сайты на основе баз данных в указанных системах, вам придется установить веб-сервер самостоятельно. Кроме того, встроенный сервер IIS также имеет несколько версий и процесс их настройки для обеспечения поддержки PHP иногда существенно отличается.

Еще одна особенность IIS, о которой вам следует знать, заключается в том, что этот веб-сервер довольно редко используется при создании сайтов на PHP. Обычно более дешевым и надежным решением считается операционная система Linux, где установлен Apache. Использование IIS в сочетании с PHP может быть оправдано лишь тем, что ваша компания уже вложила деньги в серверы на базе Windows, чтобы запускать программы, написанные с помощью ASP.NET — технологии Microsoft, встроенной в IIS, и вы хотите воспользоваться этой инфраструктурой для размещения PHP-приложений.

Проще всего настроить среду разработки таким образом, чтобы она максимально соответствовала серверу, на котором будет работать публичный сайт (хотя это совсем не обязательно). По этой причине лучше использовать веб-сервер Apache, даже если вы ведете разработку на компьютере с Windows. Если все же вы или компания, в которой вы работаете, решили выбрать для своего PHP-проекта сервер IIS, то все необходимые инструкции по его установке вы найдете в файле **install.txt** внутри **zip**-архива, который вы загрузили на официальном сайте PHP.

Чтобы установить Apache, зайдите на сайт Apache Lounge (**http://www.apachelounge.com/**) и загрузите последнюю версию программы (на момент написания данной книги это версия 2.4.1).

Загрузив **zip**-архив, щелкните на его названии правой кнопкой мыши и выберите пункт **Извлечь все**, чтобы распаковать содержимое в нужную директорию. Внутри директории вы найдете два текстовых файла, содержащих инструкции по установке. Один из них содержит напоминание о том, чтобы вы загрузили и установили последнюю версию пакета Microsoft Visual C++ Redistributable Package. Сделайте это прямо сейчас, после чего поместите загруженный вами файл с расширением **exe** в **Корзину**.

Следуя дальнейшим инструкциям, переместите каталог **Apache24**, который вы извлекли из **zip**-файла, в корневую директорию диска **C:**, чтобы сервер Apache находился по адресу **C:\Apache24**. При желании вы можете изменить этот путь, но вслед за этим вам понадобится отредактировать конфигурационные файлы, в которых он упоминается, поэтому лучше оставить все как есть.

Чтобы запустить впервые установленный сервер Apache и сконфигурировать его так, чтобы он загружался автоматически вместе с системой, воспользуйтесь командной строкой. Зайдите в меню **Пуск**, выполните команды **Все программы ▸ Стандартные**, найдите пункт **Командная строка** и, щелкнув на нем правой кнопкой мыши, выберите **Запуск от имени администратора**. В результате откроется окно, в котором отобразится текущая директория.

```
C:\Windows\system32>
```

В конце строки запроса вы увидите мигающий курсор. Наберите `C:` и нажмите клавишу **Enter**, чтобы удостовериться в том, что вы работаете с диском **C**. Затем

введите команду `cd \Apache24\bin` и снова нажмите Enter, чтобы перейти в каталог C:\Apache24\bin.

```
C:\Windows\system32>C:

C:\Windows\system32>cd \Apache24\bin

C:\Apache24\bin>
```

Теперь вы можете запустить Apache, для этого наберите httpd.exe и нажмите Enter.

```
C:\Apache24\bin>httpd.exe
```

Если все работает, как следует, то вы не увидите на экране никаких изменений: новая строка не появится, а просто начнет выполняться команда. В противном случае вы получите сообщение об ошибке вроде того, что приведено ниже.

```
C:\Apache24\bin>httpd.exe
(OS 10013)An attempt was made to access a socket in a way forbidden
by its access permissions. : AH00072: make_sock: could not bind to
address [::]:80(OS 10013)An attempt was made to access a socket in a way
forbidden by its access permissions. : AH00072: make_sock: could not bind
to address 0.0.0.0:80AH00451: no listening sockets available, shutting
downAH00015: Unable to open logs

C:\Apache24\bin>
```

Так Apache пытается сообщить о том, что на вашем компьютере уже работает веб-сервер, который отслеживает подключения на 80-м порту — стандартном для веб-серверов. Проверьте, не запущен ли у вас IIS или другая копия Apache, например из пакета XAMPP. Закройте их, если таковые имеются, и попробуйте снова запустить Apache из командной строки.

Устранив проблему, откройте браузер, наберите в адресной строке http://localhost и нажмите Enter. На рис. А.1 показана страница, которая подтверждает правильную работу Apache.

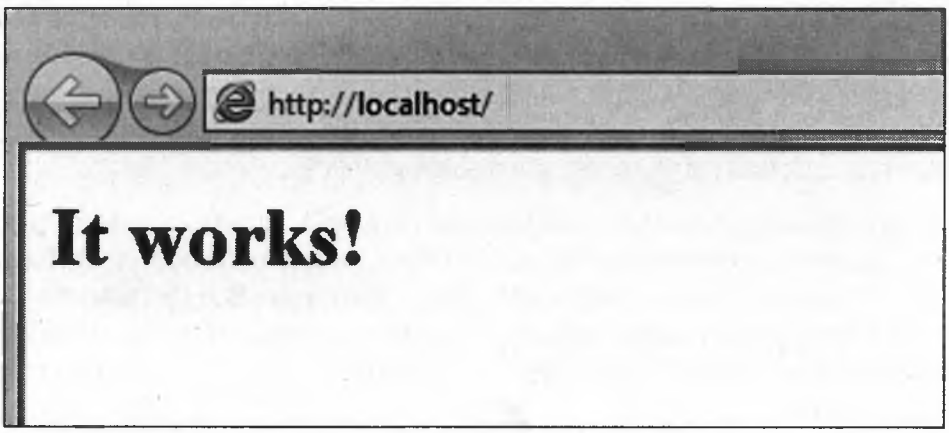


Рис. А.1. Сервер Apache работает правильно

Закройте браузер и вернитесь к командной строке. Остановите Apache, нажав Ctrl+C. Через мгновение появится новая строка запроса.

```
C:\Apache24\bin>httpd.exe

C:\Apache24\bin>
```

Скорее всего, вам не захочется открывать командную строку каждый раз, чтобы запустить Apache. В таком случае введите команду `httpd.exe -k install`, чтобы установить Apache в качестве системной службы. В ответ вы получите следующее сообщение.

```
C:\Apache24\bin>httpd.exe -k install
Installing the Apache2.4 service
The Apache2.4 service is successfully installed.
Testing httpd.conf....
Errors reported here must be corrected before the service can be started.
```

Вполне вероятно, что вы увидите несколько другой текст.

```
C:\Apache24\bin>httpd.exe -k install
Installing the Apache2.4 service
(OS 5)Access is denied. : AH00369: Failed to open the WinNT service
manager, perhaps you forgot to log in as Adminstrator?yoss
```

Это значит, что при запуске командной строки вы не щелкнули на ней правой кнопкой мыши и не выбрали пункт **Запуск от имени администратора**, как было сказано выше. Попробуйте повторить все в командной строке, запущенной от имени администратора.

Сконфигурировав Apache как системную службу, закройте командную строку и перейдите в каталог **C:\Apache24\bin**, используя **Проводник**. Там вы найдете файл под названием **ApacheMonitor.exe**. Эта программа поможет отслеживать и контролировать веб-сервер, если он работает в режиме системной службы. Закрепите ее в меню **Пуск** (или создайте ярлык и поместите его в любой участок меню **Все программы**) и запустите. В системном лотке появится значок Apache Monitor. Чтобы он оставался на экране, вам понадобится настроить свой системный лоток.

Щелкните на этом значке, и вы увидите в списке системную службу Apache2.4. Если навести на нее курсор, появится всплывающее меню с пунктами **Start**, **Stop** и **Restart**. Чтобы запустить Apache, выберите пункт **Start**. Небольшая зеленая стрелка внутри значка сигнализирует о том, что веб-сервер работает (рис. А.2).

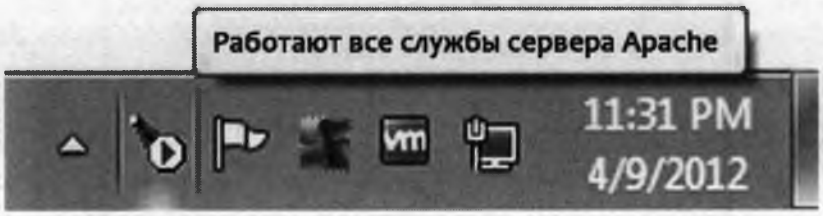


Рис. А.2. Индикатор в виде стрелки зеленого цвета означает, что сервер Apache запущен и работает

Закончив работу над веб-проектом, остановите Apache, используя тот же значок.

Теперь, когда Apache настроен, вы можете установить PHP. Для этого выполните следующие шаги.

1. Распакуйте файл, который вы загрузили на сайте PHP. Лучше использовать каталог C:\php, в дальнейшем мы будем ссылаться именно на него, но вы можете выбрать и любую другую директорию, если хотите.
2. Найдите в каталоге файл под названием php.ini-development и создайте его копию. Самый простой путь — щелкнув правой кнопкой мыши на файле, перетащить его в то же окно Проводника и выбрать во всплывающем меню пункт Копировать сюда. Появится новый файл с именем вроде php - Copy.ini-development (название зависит от версии Windows, которую вы используете). Присвойте файлу новое имя — php.ini. При запросе «Действительно ли вы хотите изменить расширение файла (с .ini-dist на .ini)?» нажмите Да.

ПО УМОЛЧАНИЮ WINDOWS СКРЫВАЕТ ИЗВЕСТНЫЕ РАСШИРЕНИЯ ФАЙЛОВ

Переименовав файл в php.ini, вы можете заметить, что теперь рядом со значком появилось имя php. Так происходит потому, что ваша версия Windows по умолчанию скрывает знакомые расширения. Операционная система знает, что расширение .ini принадлежит конфигурационным файлам, поэтому она его не показывает.

Это может привести к определенной путанице. Если вы снова захотите отредактировать файл php.ini, вам будет проще отличить его от файлов php.gif и php.exe, которые находятся в той же директории, если вы сможете видеть их расширения.

Чтобы система перестала скрывать расширения файлов, откройте Панель управления Windows и поищите раздел Параметры папок. Перейдите на вкладку Вид и в разделе Файлы и папки снимите флажок в пункте Скрывать расширения для зарегистрированных типов файлов (рис. А.3).

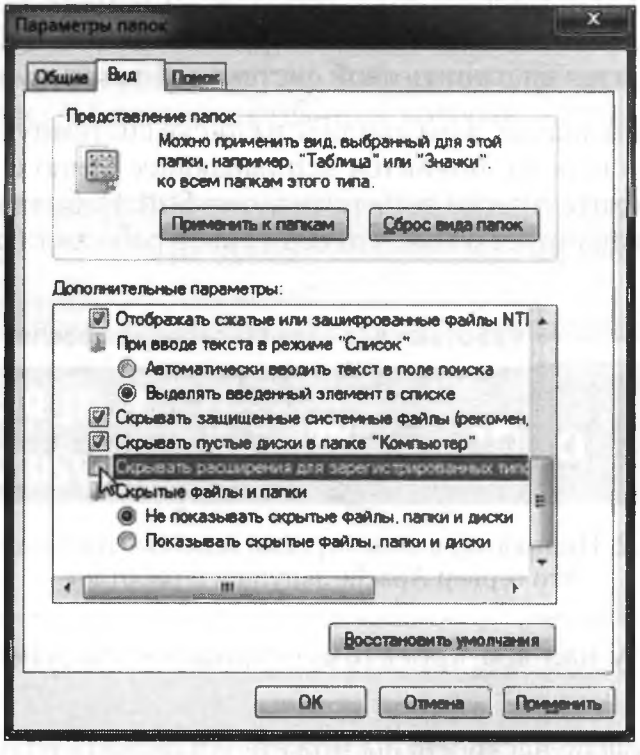


Рис. А.3. Сделайте так, чтобы расширение отображалось для всех типов файлов

3. Откройте в любом текстовом редакторе файл php.ini. Если у вас нет особых предпочтений, щелкните на имени файла дважды, и он откроется в Блокноте. Внутри документа содержится объемный текст со множеством параметров. Вам следует найти строку, которая начинается с `doc_root` (воспользуйтесь командой меню Правка ▸ Найти).

```
doc_root =
```

Добавьте в конец строки путь к корневой директории вашего веб-сервера. В случае с Apache это подкаталог htdocs, который находится в главном каталоге, куда произведена установка. Если вы не меняли стандартное местоположение, то путь будет выглядеть как C:\Apache24\htdoc. Если вы выполнили установку в другое место, то найдите там каталог htdocs и наберите путь к нему.

```
doc_root = "C:\Apache24\htdocs"
```

Чуть далее по тексту вы увидите строку, которая начинается выражением `;extension_dir`. Уберите точку с запятой в начале строки и укажите путь к подкаталогу ext вашего PHP-каталога.

```
extension_dir = "C:\php\ext"
```

Спуститесь еще немного вниз, и вы увидите множество строк, которые начинаются с выражения `;extension=`. Это опциональные расширения для PHP, выключенные по умолчанию. Вам следует включить расширение, с помощью которого PHP сможет взаимодействовать с MySQL. Для этого уберите точку с запятой в строке `php_mysqli.dll`.

```
extension=php_mysqli.dll
```

PHP_MYSQLI НЕ PHP_MYSQL

Прямо над `php_mysqli.dll` вы можете увидеть строку `php_mysql.dll`. «Лишняя» буква *i* означает *improved* (улучшенное). Именно это улучшенное расширение и нужно включить. Расширение без буквы *i* является устаревшим, и некоторые из его возможностей несовместимы с текущими версиями MySQL.

В завершение найдите строку, которая начинается выражением `;session.save_path`. Снова уберите точку с запятой, чтобы активировать этот параметр, и присвойте ему в качестве значения путь к временному каталогу Windows.

```
session.save_path = "C:\Windows\Temp"
```

Сохраните внесенные изменения и закройте текстовый редактор.

На этом настройка PHP завершена. Теперь вы можете сконфигурировать сервер Apache таким образом, чтобы он использовал PHP в качестве подключаемого модуля.

1. Откройте Блокнот (или любой другой текстовый редактор) и выберите пункт меню Файл ▸ Открыть. Перейдите в подкаталог conf внутри каталога, в который вы

установили Apache (по умолчанию это C:\Apache24\conf), и выберите файл httpd.conf. Чтобы этот файл отображался в списке, отметьте внизу окна в раскрывающемся меню для типов файлов пункт Все файлы (*.*).

2. Найдите в файле строку, которая начинается с `DirectoryIndex`.

```
<IfModule dir_module>
    DirectoryIndex index.html
</IfModule>
```

Благодаря этой строке Apache знает, какие имена файлов нужно использовать при поиске страницы по умолчанию для заданной директории. Добавьте в конец index.php.

```
<IfModule dir_module>
    DirectoryIndex index.html index.php
</IfModule>
```

3. Остальные настройки в этом длинном конфигурационном файле можно оставить без изменений. Добавьте лишь в конце файла следующие строки.

```
LoadModule php5_module "C:/php/php5apache2_4.dll"
AddType application/x-httpd-php .php
PHPIniDir "C:/php"
```

Убедитесь, что значения `LoadModule` и `PHPIniDir` указывают на директорию, в которую установлен PHP, и обратите внимание на то, что при записи пути используется обычный слеш, а не обратный.

ВЕРСИИ PHP И APACHE

Так сложилось, что основные новые версии Apache требуют наличия новых версий .dll-файла, на который вы ссылаетесь в строке `LoadModule`, а PHP иногда за новыми релизами Apache не успевает. Например, на момент написания данной книги PHP для Windows поставляется *без* файла php5apache2_4.dll.

Если вы еще раз взглянете на директорию, куда установлен PHP, то вы, вероятно, увидите файлы с названиями php5apache2_2.dll и php5apache2_3.dll. Они предназначены для версий Apache 2.2 и 2.3 соответственно. На тот момент, когда вы будете читать эту книгу, у Apache уже может выйти новая основная версия, для которой понадобится новый .dll-файл. К примеру, для Apache 2.5 может потребоваться библиотека php5apache2_5.dll.

Если у вас нет файла, версия которого совпадала бы с версией установленного вами сервера Apache, поищите его на странице с загрузками Apache Lounge (http://www.apachelounge.com/download/). Это может быть, к примеру, php5apache2_4.dll-php-5.4-win32.zip. Загрузите zip-архив, распакуйте его и поместите `.dll`-файл в каталог, в который вы установили PHP. Apache станет искать его по тому пути, который вы указали в строке `LoadModule`.

4. Сохраните изменения и закройте Блокнот.

5. Перезапустите Apache с помощью значка Apache Monitor в системном лотке. Если все прошло успешно, Apache загрузится без каких-либо проблем. В противном случае попробуйте выполнить запуск из командной строки, как вы делали это ранее. Так вы сможете получить более подробное сообщение

об ошибке. Например, вы могли допустить неточность при редактировании httpd.conf.

6. Дважды щелкните на значке Apache Monitor, чтобы открыть окно Apache Service Monitor. Если PHP установлен правильно, на панели состояния отобразится версия вашего интерпретатора (рис. А.4).

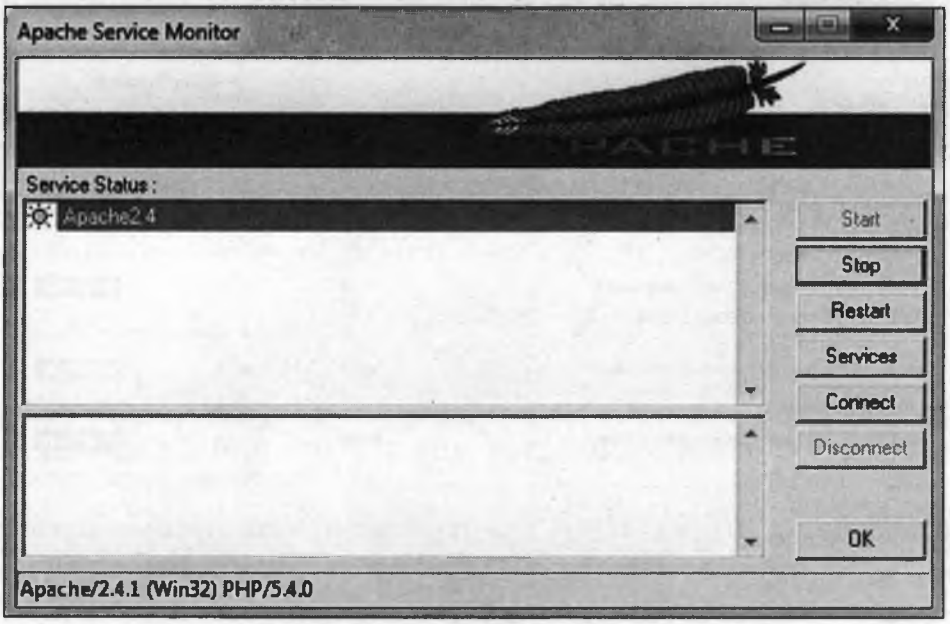


Рис. А4. Версия PHP свидетельствует о том, что сервер Apache сконфигурирован для поддержки данного языка

7. Закройте окно Apache Service Monitor, нажав кнопку OK.

Теперь, имея в распоряжении MySQL, Apache и PHP, вы можете приступить к работе с новым веб-сервером.

Mac OS X

При составлении приведенных ниже инструкций предполагалось, что вы работаете с Mac OS X 10.6 (Snow Leopard) или более поздней версией. Если у вас установлена старая версия OS X, вам лучше воспользоваться вариантом с единым инсталляционным пакетом.

Установка MySQL

Первым делом зайдите на страницу для загрузки MySQL (http://dev.mysql.com/downloads/) и перейдите по ссылке Download в разделе MySQL Community Server. Появится страница со списком ссылок для загрузки текущей рекомендованной версии MySQL (на момент написания книги это версия 5.5.22), как показано на рис. А.5.

Выбор подходящей ссылки зависит от версии вашей операционной системы и используемой вами архитектуры. Если вы точно знаете, что у вашего Mac 64-битный процессор, выбирайте пакет Mac OS X ver. 10.6 (x86, 64-bit), DMG Archive. Если вы не уверены, вам лучше остановиться на версии Mac OS X ver. 10.6

(x86, 32-bit), DMG Archive, единственное требование которой — наличие Intel-совместимой архитектуры. (Чтобы перестраховаться, проверьте информацию о процессоре в окне **Об этом компьютере**, которое открывается в меню Apple.) 32-битная версия является универсальным выбором, поскольку способна работать и в 64-битных системах.

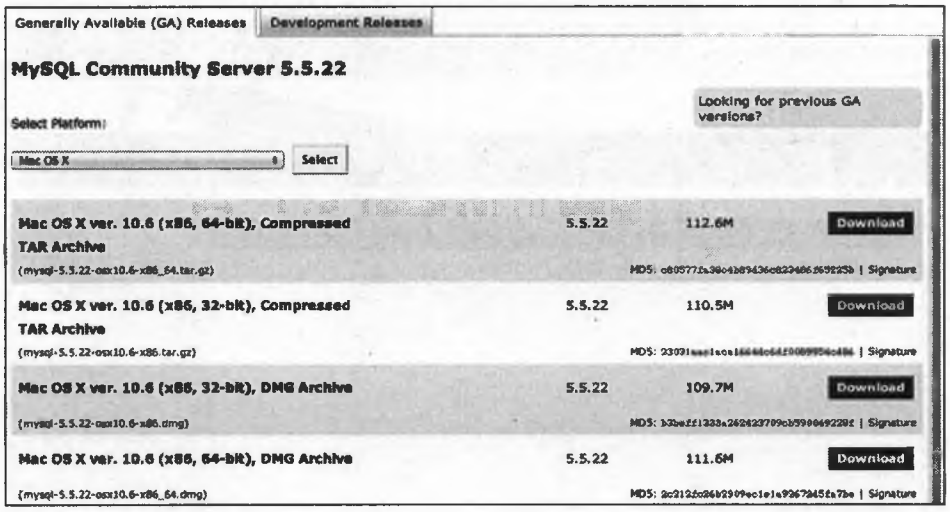


Рис. А.5. 32-битная версия MySQL для Intel-совместимых процессоров подойдет для большинства современных компьютеров Mac

Скачав файл **mysql-*версияosx-архитектура*.dmg**, щелкните на нем дважды, чтобы подключить образ диска. Как видно на рис. А.6, он содержит установщик в формате **.pkg**, а также файл **MySQLStartupItem.pkg**. Выполните двойной щелчок на инсталляторе, чтобы начать процесс установки MySQL.

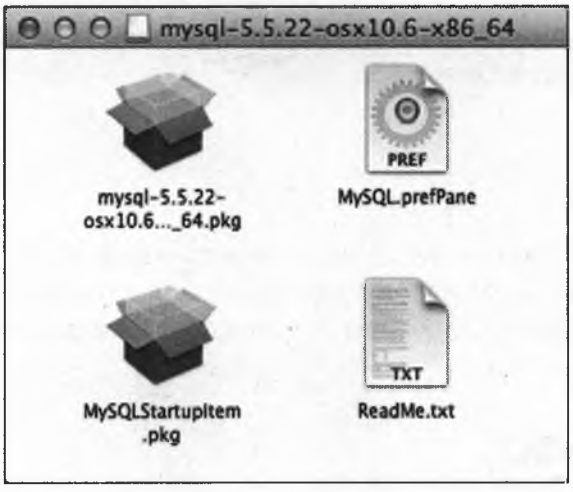


Рис. А.6. Содержимое пакета MySQL для Mac OS X

Завершив установку, запустите сервер MySQL. Откройте окно **Терминала**[1] и наберите следующую команду.

```
Machine:~ user$ sudo /usr/local/mysql/bin/mysqld_safe
```

[1] Чтобы открыть окно **Терминала**, запустите приложение **Терминал**, которое находится в подкаталоге **Утилиты** каталога **Программы**.

ЧТО НУЖНО НАБИРАТЬ

Machine:~ user$, где *Machine* — имя вашего компьютера, — это та часть командной строки, которая уже отображается. Вам нужно набрать только команду, которая выделена полужирным начертанием.

Закончив набор, нажмите клавишу Enter.

Команда запустит скрипт mysqld_safe, используя права администратора, поэтому вам придется ввести пароль. Сообщение о состоянии сервера подтвердит, что MySQL работает.

Запущенный сервер MySQL можно перевести в фоновый режим. Для этого нажмите комбинацию клавиш Ctrl+Z, чтобы остановить процесс, а затем наберите следующую команду, чтобы сервер продолжил работу в фоне:

```
Machine:~ user$ bg
```

После того как вы закроете Терминал, MySQL продолжит работать в вашей системе в качестве сервера. Чтобы остановить сервер MySQL, откройте новое окно Терминала и наберите следующую команду.

```
Machine:~ user$ sudo /usr/local/mysql/bin/mysqladmin shutdown
```

Эти команды для управления MySQL-сервером запоминать вовсе не обязательно. Вернувшись к установочному образу (см. рис. А.6), вы увидите файл с именем MySQL.prefPane. Щелкните на нем дважды, чтобы установить новую панель в окне Системных настроек (рис. А.7).

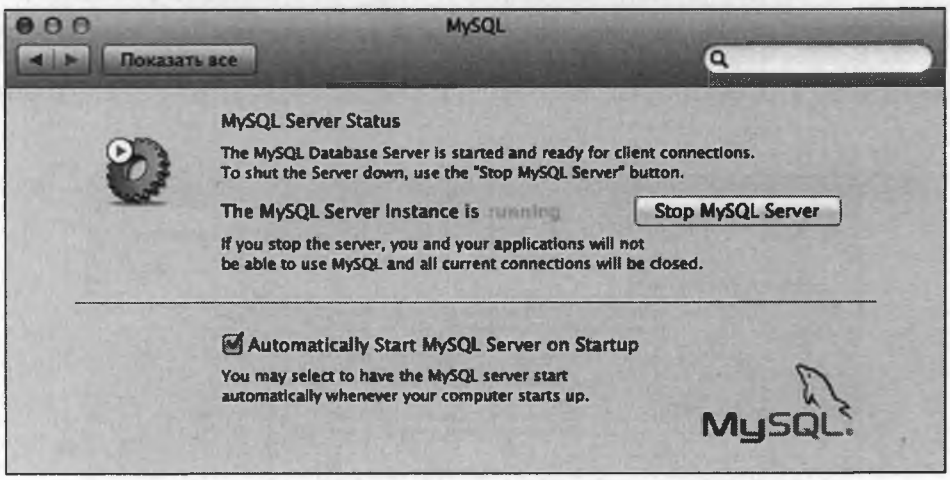


Рис. А.7. Панель MySQL в Системных настройках

С помощью этого окна вы сможете узнать, работает ли ваш MySQL-сервер, а также запускать и выключать его одним нажатием кнопки.

Если вы не хотите повторять вышеописанный процесс при каждой перезагрузке, можно сделать так, чтобы сервер MySQL запускался автоматически вместе с системой. Для этого на панели системных настроек предусмотрен специальный флажок. Однако, прежде чем вы сможете им воспользоваться, необходимо установить пакет MySQLStartupItem.pkg из инсталляционного образа.

После того как все настроено должным образом, значок установочного диска MySQL можно переместить с Рабочего стола в Корзину и удалить загруженный ранее файл .dmg.

Последнее, что нужно сделать, — добавить директорию /usr/local/mysql/bin в список системных путей. Это позволит запускать в Терминале такие программы, как mysqladmin и mysql без необходимости вводить их полный путь. Откройте новое окно Терминала и наберите следующие команды.

```
Machine:~ user$ sudo su
Password: (введите пароль)
sh-3.2# echo ‹/usr/local/mysql/bin› >> /etc/paths.d/mysql
sh-3.2# exit
```

Чтобы внесенные изменения вступили в силу, закройте текущее окно Терминала и откройте новое. Затем во время работы сервера MySQL попробуйте запустить программу mysqladmin из домашней директории.

```
Machine:~ user$ mysqladmin status
```

Если все работает как положено, вы увидите короткий перечень статистических данных о сервере MySQL.

Установка PHP

Mac OS X 10.5 (Leopard) поставляется с предустановленными копиями Apache 2.2 и PHP 5. Все, что вам нужно сделать, — это включить их.

1. Откройте окно Системные настройки в меню Apple.
2. Выберите пункт Общий доступ в разделе Интернет и сеть.
3. Убедитесь, что напротив параметра Общий веб-доступ установлен флажок (рис. А.8).

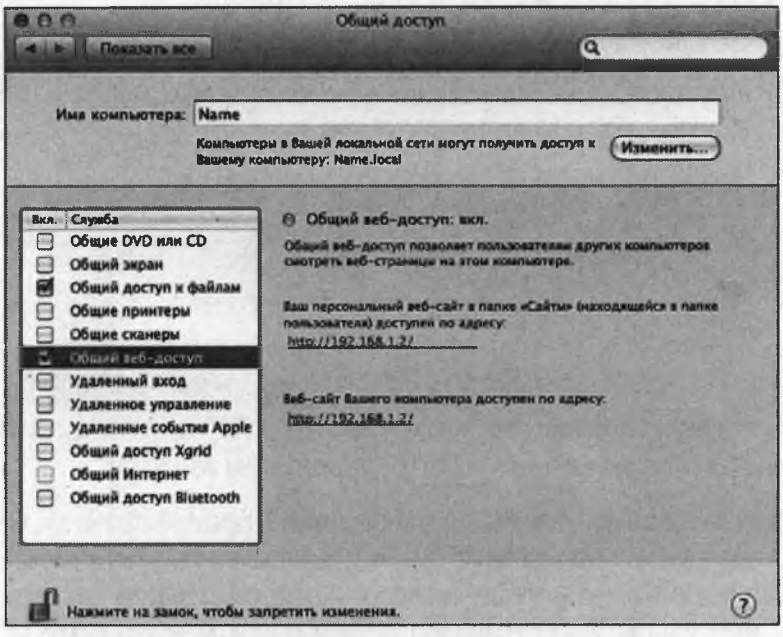


Рис. А.8. Активируйте Общий веб-доступ в Mac OS X

4. Закройте **Системные настройки**.
5. Откройте браузер, наберите в адресной строке **http://localhost** и нажмите клавишу **Enter**. Появится страница со стандартным приветствием сервера Apache (рис. А.9).

Рис. А.9. Стандартная приветственная страница сервера Apache

После того как все настроено, Apache станет запускаться автоматически вместе с вашей системой. Теперь вы можете усовершенствовать работу сервера, добавив к нему поддержку PHP.

1. Выберите в меню **Finder** пункт **Переход ▸ Переход к папке** (⇧+⌘+G), введите `/private/etc/apache2/` и нажмите кнопку **Перейти**.
2. В открывшемся окне **Finder** вы увидите файл **httpd.conf**, который используется для конфигурирования Apache. По умолчанию он доступен только для чтения. Щелкните на нем правой кнопкой мыши и выберите в контекстном меню пункт **Свойства** (⌘+I). В открывшемся окне найдите раздел **Общий доступ и права доступа**.

   По умолчанию все настройки в этом разделе неактивны. Чтобы их активировать, щелкните на небольшом значке с изображением замка́ (рис. А.10) и введите пароль.

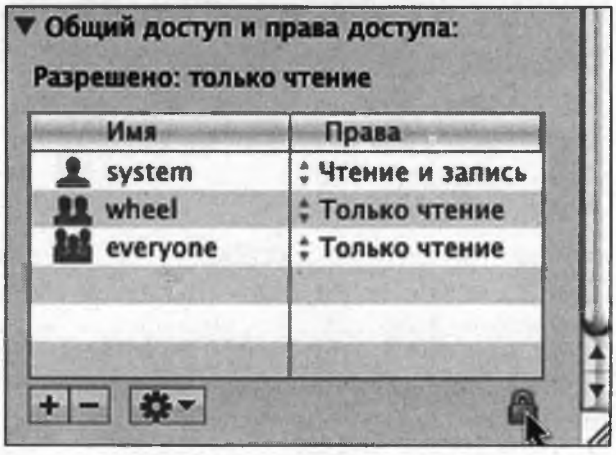


Рис. А.10. Чтобы изменить настройки, щелкните на изображении замка́

Чтобы разрешить редактирование файла **httpd.conf**, измените значение в столбце **Права** для строки **everyone** на **Чтение и запись** (рис. А.11).

Рис. А.11. Укажите для строки **everyone** права **Чтение и запись**

3. Вернитесь в окно **Finder**, где открыт каталог **apache2**. Щелкните правой кнопкой мыши на пустом месте окна и выберите в появившемся контекстном меню пункт **Свойства**. По аналогии с предыдущим шагом откройте раздел **Общий доступ и права доступа** и установите для строки **everyone** значение **Чтение и запись**.
4. И наконец, дважды щелкните на названии файла **httpd.conf**, чтобы открыть его в программе TextEdit.
5. Найдите в открывшемся файле следующую строку.

```
#LoadModule php5_module        libexec/apache2/libphp5.so
```

   Активируйте команду, удалив символ решетки (#) в начале строки.
6. Сохраните внесенные изменения и закройте файл.
7. Теперь вы можете вернуться назад и восстановить первоначальные права на доступ к файлу **httpd.conf** и каталогу **apache2**. Это не позволит другим пользователям вашего компьютера изменять конфигурацию Apache.
8. Перезапустите Apache. Для этого откройте окно терминала и наберите следующую команду.

```
Machine:~ user$ sudo /usr/sbin/apachectl restart
```

   Затем введите пароль.
9. Загрузите в браузере страницу **http://localhost** и убедитесь в том, что Apache все еще работает. Подождите немного, если он еще не успел запуститься.

   Если сервер так и не заработал, откройте **Консоль** в подкаталоге **Утилиты** каталога **Программы**. Выберите в боковой панели в разделе **/var/log** пункт **apache2 ▸ error_log**, чтобы просмотреть журнал ошибок Apache и разобраться с причиной возникшей проблемы. Если вы так и не нашли решение, попробуйте обратиться за помощью на форум SitePoint.

Теперь ваш компьютер оснащен веб-сервером Apache с поддержкой PHP. Из вышеприведенных инструкций вы узнали, как изменить конфигурацию сервера с помощью файла **httpd.conf**. Однако свой конфигурационный файл есть и у подключаемого модуля PHP, он называется **php.ini**. Его также следует отредактировать, чтобы интерпретатор знал, как именно подключаться к серверу MySQL.

Версия PHP, поставляемая с Mac OS X, по умолчанию не содержит файла **php.ini**. PHP работает со стандартными настройками. Чтобы эти настройки изменить, нужно открыть **Терминал** и скопировать файл **/private/etc/php.ini.default** в **/private/etc/php.ini**.

```
Machine:~ user$ cd /private/etc
Machine:etc user$ sudo cp php.ini.default php.ini
Password: (введите свой пароль)
```

Чтобы пользователи вроде вас могли редактировать новый файл, повторите процедуру, описанную выше для файла **httpd.conf**. Выберите в окне **Finder** пункт меню **Переход ▸ Переход к папке**, укажите каталог **/private/etc**, поменяйте права для файла **php.ini** и для каталога, в котором он находится, а затем откройте файл в текстовом редакторе TextEdit.

Воспользуйтесь пунктом меню **Правка ▸ Найти ▸ Найти** (⌘+F) для поиска параметра `mysql.default_socket`. Отредактируйте найденную строку, она должна выглядеть следующим образом.

```
mysql.default_socket = /tmp/mysql.sock
```

Вам нужно добавить только ту часть, которая выделена полужирным начертанием.

Спуститесь ниже, чтобы найти параметр `mysqli.default_socket` (`mysqli`, а не `mysql`), и проделайте с этой строкой то же самое.

```
mysqli.default_socket = /tmp/mysql.sock
```

Сохраните внесенные изменения, закройте TextEdit и при желании восстановите права на файл и директорию. В завершение откройте окно **Терминала** и снова запустите Apache с помощью следующей команды.

```
Machine:~ user$ sudo /usr/sbin/apachectl restart
```

Введите пароль. После того как сервер Apache запустится, введите в адресной строке браузера `http://localhost` и убедитесь в корректной работе сервера.

Вот и все. Теперь, имея в своем распоряжении MySQL, Apache и PHP, вы можете приступить к работе.

Linux

В этом разделе описан процесс ручной установки Apache, PHP и MySQL для наиболее актуальных дистрибутивов Linux. Приведенные инструкции проверялись на Ubuntu 10.04.4 (**http://www.ubuntu.com**), однако они должны подойти и для других

дистрибутивов, таких как Fedora (http://fedoraproject.org), Debian (http://www.debian.org), openSUSE (http://www.opensuse.org) и Gentoo (http://www.gentoo.org). Возможны несущественные отличия.

Большинство дистрибутивов Linux поставляются вместе с **пакетным менеджером**. Программа Synaptic (https://help.ubuntu.com/community/SynapticHowto), которая входит в состав Ubuntu, является графической надстройкой для APT (http://www.debian.org/doc/user-manuals#apt-howto) и пакетным менеджером Debian. В других дистрибутивах распространена система RPM. Однако, независимо от того, какую версию Linux вы используете, вам доступны заранее упакованные копии Apache, PHP и MySQL. Программы, поставляемые в таком виде, очень легко устанавливаются. К сожалению, за это приходится платить ограниченным выбором настроек конфигурации. По этой причине, а также потому, что любая попытка описать процесс установки пакетов во всех популярных дистрибутивах Linux обречена на провал, мы разберем установку указанных программ вручную.

Если у вас уже установлены пакетные версии Apache, PHP и MySQL, можете пользоваться ими. При возникновении проблем всегда есть возможность удалить их и вернуться к ручной установке.

Установка MySQL

Для начала загрузите MySQL. Для этого зайдите на страницу загрузок (http://dev.mysql.com/downloads/) и перейдите по ссылке Download в разделе MySQL Community Server. Появится список ссылок для загрузки текущей рекомендованной версии MySQL (на момент написания книги это версия 5.5.22).

Убедитесь, что в меню над списком файлов для загрузки выбран пункт Linux – Generic. Теперь следует подобрать пакет, который соответствует архитектуре вашей системы. Если вы уверены, что у вас 64-битная версия Linux, загружайте пакет Linux — Generic 2.6 (x86, 64-bit), Compressed TAR Archive (размером около 177 Мбайт). Для 32-битной версии скачайте пакет Linux — Generic 2.6 (x86, 32-bit), Compressed TAR Archive (около 171 Мбайт), он работает даже в том случае, если у вас 64-битная архитектура. Нажмите кнопку Download рядом с той версией, которая вам подходит.

Загрузив файл, откройте Терминал и войдите в систему от имени администратора.

```
user@machine:~$ sudo su
```

Естественно, вам потребуется ввести пароль.

Перейдите в директорию /usr/local и распакуйте загруженный файл.

```
root@machine:/home/user# cd /usr/local
root@machine:/usr/local# tar xfz ~user/Desktop/mysql-version-linux2.
6-platform.tar.gz
```

Вторая команда написана с учетом того, что вы сохранили загруженный файл на Рабочем столе, то есть в каталоге Desktop домашней директории. Вместо `user` нужно вписать имя пользователя, вместо `version` — выбранную вами версию MySQL, а вместо `platform` — архитектуру и версию компилятора для той копии, которую

вы загрузили. В результате в записи команды должны использоваться соответствующие имя пакета и путь к нему. Например, данная команда может выглядеть следующим образом.

```
root@mythril:/usr/local# tar xfz ~kyank/Desktop/mysql-5.5.
22-linux2.6-x86_64.tar.gz
```

Через некоторое время вы снова увидите сообщение в командной строке. Команда `ls` поможет удостовериться в том, что у вас есть директория под названием mysql-version-linux-platform. Вот как это может выглядеть.

```
root@mythril:/usr/local# ls
bin  games    lib  mysql-5.5.22-linux2.6-x86_64 share
etc  include  man  sbin                         src
```

Чтобы упростить доступ к этой директории, создайте символьную ссылку на нее и назовите ее `mysql`. Затем перейдите туда.

```
root@machine:/usr/local# ln -s mysql-version-linux-platform mysql
root@machine:/usr/local# cd mysql
```

Сервер MySQL можно запускать от имени администратора или даже от своего имени (например, если вы установили MySQL в домашнюю директорию), но, как правило, для запуска сервера в системе создают отдельную учетную запись. Это исключит возможность взлома системы, используя уязвимости в MySQL. Чтобы создать специального пользователя, выполните следующие команды (оставаясь в системе в качестве администратора).

```
root@machine:/usr/local/mysql# groupadd mysql
root@machine:/usr/local/mysql# useradd -g mysql mysql
```

Присвойте новому пользователю права на владение директорией mysql.

```
root@machine:/usr/local/mysql# chown -R mysql .
root@machine:/usr/local/mysql# chgrp -R mysql .
```

Теперь у вас есть MySQL-сервер. Но, прежде чем вы сможете им воспользоваться, необходимо установить для него файлы базы данных. Введите следующую команду, оставаясь в директории mysql.

```
root@machine:/usr/local/mysql# scripts/mysql_install_db --user=mysql
```

Если команда вернет ошибку, связанную с файлом libaio.so, попробуйте установить указанную библиотеку. В Ubuntu Linux это делается довольно просто с помощью утилиты apt-get.

```
root@machine:/usr/local/mysql# apt-get install libaio1
root@machine:/usr/local/mysql# scripts/mysql_install_db --user=mysql
```

Теперь MySQL-сервер полностью готов к первому запуску. Находясь в той же директории, введите следующую команду.

```
root@machine:/usr/local/mysql# bin/mysqld_safe --user=mysql &
```

Если вы увидите сообщение `mysql daemon ended`, это означает, что сервер MySQL запустить не удалось. Текст ошибки вносится в файл hostname.err в подкаталоге data, где hostname — имя вашего компьютера. Обычно подобного рода ошибка случается из-за того, что на данном компьютере уже работает другой экземпляр MySQL.

Если сервер MySQL запустился без проблем, он станет работать так же, как FTP или веб-сервер, — до тех пор, пока не выключат компьютер. Чтобы проверить, действительно ли сервер MySQL работает как положено, выполните следующую команду.

```
root@machine:/usr/local/mysql# bin/mysqladmin -u root status
```

Должна появиться небольшая сводка статистической информации о сервере MySQL. После получения сообщения об ошибке следует поискать причину возникшей проблемы в файле hostname.err. Если вы перепроверили все выполненные действия и убедились в том, что правильно следовали вышеприведенным инструкциям, но так и не смогли решить проблему, обратитесь на форум SitePoint (http://www.sitepoint.com/forums/). Там вам помогут справиться с возникшими трудностями.

Если вы хотите, чтобы сервер MySQL запускался автоматически вместе с системой, его необходимо настроить соответствующим образом. В подкаталоге support-files каталога mysql находится скрипт под названием mysql.server, который можно сделать частью механизма запуска системы. В большинстве дистрибутивов Linux для этого в директории /etc/init.d нужно создать ссылку на скрипт mysql.server, а затем сослаться на нее еще два раза в директориях /etc/rc2.d/S99mysql и /etc/rc0.d/K01mysql.

```
root@machine:/usr/local/mysql# cd /etc
root@machine:/etc# ln -s /usr/local/mysql/support-files/mysql.server
init.d/
root@machine:/etc# ln -s /etc/init.d/mysql.server rc2.d/S99mysql
root@machine:/etc# ln -s /etc/init.d/mysql.server rc0.d/K01mysql
```

Вот и все. Чтобы убедиться в том, что все работает правильно, перезагрузите систему и запросите состояние сервера с помощью утилиты mysqladmin, как вы это делали ранее.

Теперь необходимо уделить внимание созданию пароля для учетной записи `root`. Для этого запустите программу bin/mysql_secure_installation.

```
root@machine:/usr/local/mysql# ./bin/mysql_secure_installation
```

НЕ ПОЛУЧАЕТСЯ УДАЛИТЬ БАЗУ ДАННЫХ TEST?

Не волнуйтесь, если программа mysql_secure_installation не удалит базу данных `test`. Эта проблема связана со способом упаковки MySQL в Linux, а именно с файлом .empty в каталоге data/test, где хранится база данных `test`. MySQL не способен распознать этот файл и, следовательно, не может с ним ничего сделать. Если вы удалите данный файл вручную, MySQL сотрет указанную базу данных.

Для более удобной работы можно прописать для MySQL-клиента, с помощью которого вы затем будете администрировать сервер, системный путь. Для этого поместите символьные ссылки на mysql, mysqladmin и mysqldump в директорию /usr/local/bin.

```
root@machine:/etc# cd /usr/local/bin
root@machine:/usr/local/bin# ln -s /usr/local/mysql/bin/mysql .
root@machine:/usr/local/bin# ln -s /usr/local/mysql/bin/mysqladmin .
root@machine:/usr/local/bin# ln -s /usr/local/mysql/bin/mysqldump .
```

Выполнив все необходимое, вы можете завершить работу с сервером в качестве администратора. Теперь вы способны управлять MySQL-сервером из любой директории в вашей системе.

```
root@machine:/usr/local/bin# exit
user@machine:~$ mysqladmin -u root -p status
```

Установка PHP

Как уже упоминалось ранее, PHP является скорее подключаемым модулем для веб-сервера, нежели полноценной программой. В случае с Apache этот модуль можно установить тремя разными способами:

- в виде CGI-приложения, которое запускается сервером Apache каждый раз, когда необходимо обработать вызов PHP-страницы;
- в качестве модуля, скомпилированного с программой Apache;
- в качестве модуля, который загружается при каждом запуске сервера Apache.

Первый вариант наиболее прост в установке и настройке, но подразумевает, что интерпретатор запускается при каждом вызове PHP-страницы. Такой подход способен значительно увеличить время отклика вашего веб-сервера, особенно если необходимо выполнить более одного запроса одновременно.

Второй и третий варианты почти ничем не отличаются в плане производительности. Последний является чуть более гибким, поскольку позволяет добавлять и удалять модули для Apache без необходимости каждый раз выполнять перекомпиляцию. Остановим свой выбор на нем.

Если вы не установили на своем компьютере веб-сервер и не хотите сделать это автоматически с помощью специальной команды, например `sudo apt-get install apache2` в Ubuntu Linux, зайдите на страницу проекта Apache HTTP Server (http://httpd.apache.org/) и найдите там последнюю доступную версию Apache (на момент написания данной книги это версия 2.4.4).

После того как вы перейдете на страницу загрузки, спуститесь вниз к списку доступных вариантов. Вам понадобятся исходные тексты Unix Source (рис. А.12). Архивы .tar.gz и .tar.bz2 ничем не отличаются, выберите тот формат, который вам более привычен.

То, что вы загрузили, на самом деле является исходным кодом сервера Apache. Первым делом следует скомпилировать его в исполняемый бинарный установщик. Откройте терминал, перейдите в директорию с загруженным файлом, распакуйте его и войдите в созданный каталог.

```
user@machine:~$ cd Desktop
user@machine:~/Desktop$ tar xfz httpd-version.tar.gz
user@machine:~/Desktop$ cd httpd-version
```

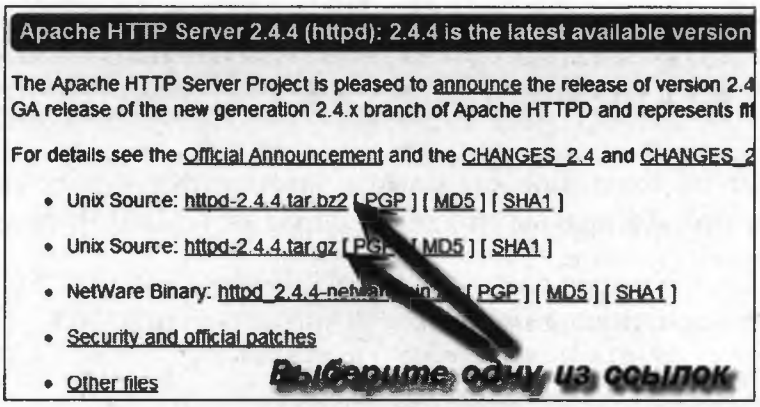


Рис. А.12. Архивы, которые вам необходимы для установки PHP

Перед тем как скомпилировать сервер Apache, необходимо изменить его конфигурацию согласно вашим требованиям. Большую часть стандартных параметров можно оставить без изменений, активировав лишь механизм динамической загрузки модулей, таких как PHP, который по умолчанию выключен. Кроме того, вам, вероятно, стоит включить возможность переопределения адресов URL, которую используют многие написанные на PHP приложения, хотя для примеров из данной книги это делать не обязательно. Чтобы внести в конфигурацию указанные изменения, введите следующую команду.

```
user@machine:~/Desktop/httpd-version$ ./configure --enable-so --enable-
rewrite
```

На экране появится множество сообщений. Если процесс завершился ошибкой, это значит, что в вашей системе не хватает важных компонентов, которые необходимы для компиляции Apache. В состав некоторых дистрибутивов Linux не входят основные библиотеки и даже компилятор для языка си. Установив необходимые компоненты, вы сможете успешно выполнить команду. Текущие версии Ubuntu, как правило, поставляются со всем необходимым.

Спустя несколько минут поток сообщений прекратится.

```
⋮
config.status: creating build/rules.mk
config.status: creating build/pkg/pkginfo
config.status: creating build/config_vars.sh
config.status: creating include/ap_config_auto.h
config.status: executing default commands
user@machine:~/Desktop/httpd-version$
```

Теперь все готово для компиляции Apache. Достаточно ввести одну короткую команду `make`.

```
user@machine:~/Desktop/httpd-version$ make
```

Этот процесс также займет несколько минут и завершится следующим сообщением.

```
⋮
make[1]: Leaving directory `/home/user/Desktop/httpd-version'
user@machine:~/Desktop/httpd-version$
```

Чтобы установить только что скомпилированную версию Apache, введите команду `sudo make install` (`sudo` необходимо указывать потому, что для выполнения записи в установочной директории нужны права администратора).

```
user@machine:~/Desktop/httpd-version$ sudo make install
```

После этого введите пароль.

Установка Apache завершится, как только команда закончит копировать файлы. Перейдите в установочную директорию и запустите Apache с помощью скрипта apachectl:

```
user@machine:~/Desktop/httpd-version$ cd /usr/local/apache2
user@machine:/usr/local/apache2$ sudo bin/apachectl -k start
```

Скорее всего, вы увидите предупреждение, что Apache не может определить полное доменное имя сервера. Не обращайте на это внимания: у большинства персональных компьютеров такого имени просто не существует.

Если вы решили установить Apache с помощью пакетного менеджера своего дистрибутива, то запуск можно выполнить, используя следующую команду (в Ubuntu).

```
user@machine:~$ sudo service apache2 start
```

Откройте браузер и наберите в адресной строке `http://localhost`. Если Apache запущен и работает, вы увидите приветственное сообщение (рис. А.13).

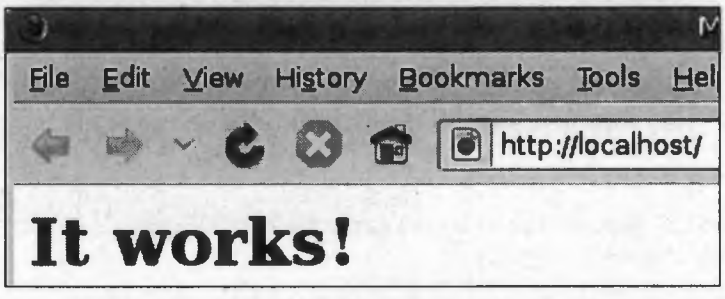


Рис. А.13. Нет никаких сомнений, что Apache работает

Как и в случае с сервером MySQL, вероятно, вам захочется сконфигурировать Apache таким образом, чтобы он автоматически запускался вместе с системой. Если Apache установлен из пакета, то, скорее всего, все уже настроено за вас.

Данная процедура аналогична той, которую вы выполняли для MySQL. Просто скопируйте ссылку на скрипт apachectl из установленной копии Apache.

```
user@machine:/usr/local/apache2$ sudo su
root@machine:/usr/local/apache2# cd /etc
root@machine:/etc# ln -s /usr/local/apache2/bin/apachectl init.d/
root@machine:/etc# ln -s /etc/init.d/apachectl rc2.d/S99httpd
root@machine:/etc# ln -s /etc/init.d/apachectl rc0.d/K01httpd
```

Чтобы убедиться в том, что все работает, перезагрузите систему и снова откройте в браузере страницу http://localhost.

Теперь вы можете добавить поддержку PHP в новую копию Apache. Для начала загрузите на сайте PHP (http://www.php.net/downloads.php) пакет с исходным кодом

Complete Source Code. Как и в предыдущем случае, архивы .tar.gz и .tar.bz2 ничем не отличаются, поэтому выбирайте тот, с которым вам легче работать.

Файл, который вы загрузили, должен называться php-*version*.tar.gz (или .bz2). Откройте новое окно Терминала, перейдите в директорию с загруженным файлом, распакуйте его и войдите в созданный каталог.

```
user@machine:~$ cd Desktop
user@machine:~/Desktop$ tar xfz php-version.tar.gz
user@machine:~/Desktop$ cd php-version
```

Чтобы установить PHP в качестве модуля для веб-сервера, необходимо воспользоваться программой apxs из состава Apache. Если вы следовали всем инструкциям во время компиляции, то она уже установлена вместе с сервером. Если же вы использовали упакованную версию, вам понадобится дополнительный пакет Apache для разработчиков. Он доступен в том же пакетном менеджере, с помощью которого вы установили Apache. Например, в Ubuntu можно воспользоваться командой `apt-get`:

```
user@machine:~$ sudo apt-get install apache2-dev
```

Чтобы установить PHP, вы должны войти в систему как администратор:

```
user@machine:~/Desktop/php-version$ sudo su
[sudo] password for user: (введите пароль)
root@machine:/home/user/Desktop/php-version#
```

Первым делом необходимо сконфигурировать установщик PHP, включив требуемые параметры и указав, где искать нужные ему программы, такие как MySQL и apxs из состава Apache. Вот как выглядит эта команда (она набирается в одну строку).

```
root@machine:/home/user/Desktop/php-version# ./configure --prefix=/usr/
local/php --with-apxs2=/usr/local/apache2/bin/apxs --with-mysqli=/usr/
local/mysql/bin/mysql_config
```

Параметр `--prefix` сообщает инсталлятору о том, куда вы хотите установить PHP (/usr/local/php — хороший выбор).

Параметр `--with-apxs2` указывает, где искать вышеупомянутую утилиту apxs. Если использовался пакетный менеджер, она, как правило, размещается в каталоге /usr/bin/. Если вы скомпилировали и установили Apache самостоятельно, как было описано выше, то ищите ее в директории Apache, предназначенной для бинарных файлов (/usr/local/apache2/bin/).

Благодаря параметру `--with-mysqli` инсталлятор узнает, куда установлен сервер MySQL. Точнее, она указывает на программу mysql_config внутри директории bin (/usr/local/mysql/bin/mysql_config).

Экран снова заполнится вереницей сообщений. По окончании конфигурации ознакомьтесь с ошибками и установите любые недостающие файлы. К примеру, в стандартной поставке Ubuntu 10.04.4 вы, скорее всего, увидите ошибку, связанную с неполной установкой библиотеки libxml2. Чтобы это исправить, откройте пакетный менеджер Synaptic, найдите и установите с его помощью пакет libxml2-dev (пакет libxml2 должен присутствовать в системе). Как вариант, наберите в Терминале

команду `apt-get install libxml2-dev`. Завершив необходимые действия, попробуйте снова запустить скрипт **configure**.

После того как на экране промелькнет несколько тестовых страниц, вы увидите строку с приглашением и сообщение с благодарностью за использование PHP (Thank you for using PHP). Следующие две команды скомпилируют и установят PHP.

```
root@machine:/home/user/Desktop/php-version# make
root@machine:/home/user/Desktop/php-version# make install
```

Это займет некоторое время. После того как команда `make install` завершит работу, вы найдете установленную копию PHP в директории **/usr/local/php**, если, конечно, вы не указали другой путь с помощью параметра `--prefix` скрипта **configure**. Теперь необходимо заняться настройкой.

Конфигурационный файл для PHP называется **php.ini**. В комплекте находятся два образца этого файла — **php.ini-development** и **php.ini-production**. Скопируйте их из рабочей установочной директории в каталог **/usr/local/php/lib**, затем создайте копию файла **php.ini-development** и назовите ее **php.ini**.

```
root@machine:/home/user/Desktop/php-version# cp php.ini* /usr/local/php/lib/
root@machine:/home/user/Desktop/php-version# cd /usr/local/php/lib
root@machine:/usr/local/php/lib# cp php.ini-development php.ini
```

Теперь вы можете удалить директорию, в которой компилировали PHP, — она вам больше не понадобится.

Редактированием **php.ini** мы займемся чуть позже. Для начала необходимо изменить конфигурацию Apache, сделав ее дружественной по отношению к PHP. Найдите конфигурационный файл, который содержится в подкаталоге **conf** установочного каталога Apache (**/usr/local/apache2/conf/httpd.conf**) или в **/etc/apache2/apache2.conf**, если вы использовали для установки пакетный менеджер.

Чтобы отредактировать данный файл, следует войти в систему в качестве администратора. Запустите текстовый редактор из **Терминала**, в котором вы все еще находитесь под учетной записью `root`.

```
root@machine:/usr/local/php/lib# cd /usr/local/apache2/conf
root@machine:/usr/local/apache2/conf# gedit httpd.conf
```

Перейдите в конец файла и добавьте туда строки, приведенные ниже. Благодаря этому Apache узнает, что файлы с расширением **.php** нужно воспринимать как PHP-скрипты.

```
<FilesMatch \.php$>
  SetHandler application/x-httpd-php
</FilesMatch>
```

Сделанных изменений должно быть достаточно. Сохраните их и перезапустите сервер Apache.

```
root@machine:/usr/local/apache2/conf# /usr/local/apache2/bin/apachectl -k restart
```

Если все выполнено верно, Apache запустится без каких-либо сообщений об ошибках. При возникновении проблем, участники форума SitePoint (**http://www.sitepoint.com/forums/**), в том числе и автор данной книги, с радостью придут вам на помощь.

Приложение Б
СПРАВОЧНИК ПО СИНТАКСИСУ MySQL

В этом приложении вы найдете синтаксис SQL-выражений, которые чаще всего используются в MySQL версии 5.5.22 — самой последней на момент написания книги.

Справочник построен по следующим правилам:

- команды перечислены в алфавитном порядке;
- необязательный участок любой команды заключен в квадратные скобки ([ ]);
- список элементов, среди которых следует выбрать один, заключен в фигурные скобки ({ }), а сами элементы разделены вертикальной чертой (|);
- многоточие означает, что предыдущий элемент может повторяться.

Синтаксис запросов, задокументированный в этом приложении, в некоторых случаях упрощен. Например, здесь не рассматривается альтернативная запись команд и ключевые слова, которые изначально добавлены для совместимости с другими базами данных, но не выполняют никаких функций. Кроме того, опущены некоторые расширенные возможности, такие как транзакции. Полную и наиболее актуальную информацию о синтаксисе, который поддерживает MySQL, ищите в соответствующем справочном руководстве (http://dev.mysql.com/doc/mysql/en/).

ALTER TABLE

```
ALTER [IGNORE] TABLE tbl_name action[, action ...]
```

В этом коде `action` обозначает определенное соответствующим образом действие.

Запросы `ALTER TABLE` используются для изменения структуры таблицы без потери хранящихся в ней данных (если не считать очевидных случаев, таких как удаление столбца). Ниже приведены основные действия, которые выполняют данный запрос.

- `ADD [COLUMN] create_definition [FIRST | AFTER column_name]`. Добавляет в таблицу новый столбец. Для `create_definition` используется такой же синтаксис, как и для запроса `CREATE TABLE`, которому посвящен отдельный раздел. По умолчанию столбец добавляется в конец таблицы, но при желании вы можете задать ему любую позицию, используя ключевые слова `FIRST` и `AFTER column_name`. Параметр `COLUMN` не играет никакой роли и используется исключительно для наглядности записи.

- ADD INDEX [*index_name*] (*index_col_name*, ...). Создает новый индекс для ускорения поисковых запросов, в которых фигурируют указанные столбцы. Вы можете указать имя индекса с помощью *index_name*. Если вы этого не сделаете, ему будет присвоено имя по умолчанию, основанное на имени первого столбца, который в нем используется. Создавая индекс на основе столбцов с типами CHAR и/или VARCHAR, вы вправе определять количество символов, которое необходимо индексировать, используя параметр *index_col_name*. Например, myColumn(5) проиндексирует первых пять символов столбца myColumn. Этот параметр обязателен при индексировании столбцов типа BLOB и TEXT.

- ADD FULLTEXT [*index_name*] (*index_col_name*, ...). Создает полнотекстовый индекс для указанных столбцов. Особый вид индекса позволяет выполнять сложный поиск по столбцам типа CHAR, VARCHAR и TEXT, используя функцию MATCH из состава MySQL. Подробности вы найдете в справочном руководстве (http://dev.mysql.com/doc/mysql/en/fulltext-search.html).

- ADD FOREIGN KEY [*index_name*] (*index_col_name*, ...) *reference_definition*. Создает ограничение внешнего ключа в таблицах формата InnoDB, которое требует, чтобы данный индекс соответствовал записям в другой таблице.

  Параметр *reference_definition* указывает таблицу и столбцы, которые фигурируют в ограничении.

```
REFERENCES tbl_name (index_col_name, ...)
        [ON DELETE { RESTRICT | CASCADE | SET NULL | NO ACTION }]
        [ON UPDATE { RESTRICT | CASCADE | SET NULL | NO ACTION }]
```

  У *reference_definition* есть необязательные участки с ключевыми словами ON DELETE и ON UPDATE, которые определяют, что должно происходить с записями, когда удаляются или изменяются соответствующие строки в другой таблице. Более подробно с данным действием вы можете ознакомиться в справочном руководстве по MySQL (http://dev.mysql.com/doc/mysql/en/innodb-foreign-key-constraints.html).

- ADD PRIMARY KEY (*index_col_name*, ...). Создает индекс с именем PRIMARY для заданных строк, объявляя его первичным ключом для всей таблицы. Все значения (или их комбинации) должны быть уникальными, как описано в следующем действии ADD UNIQUE. Если таблица уже содержит первичный ключ, возникнет ошибка. Параметр *index_col_name* определяется так же, как в рассмотренной ранее команде ADD INDEX.

- ADD UNIQUE [*index_name*] (*index_col_name*, ...). Создает индекс для заданных столбцов с условием, что все значения в столбце (или комбинации значений, если столбцов несколько) должны быть уникальными. Параметры *index_name* и *index_col_name* играют ту же роль, что и в действии ADD INDEX.

- ALTER [COLUMN] *col_name* {SET DEFAULT *value* | DROP DEFAULT}. Создает для столбца новое значение по умолчанию (SET DEFAULT) или удаляет старое (DROP DEFAULT). Слово COLUMN роли не играет и является необязательным.

- CHANGE [COLUMN] *col_name create_definition*. Заменяет существующий столбец *col_name* на новый, описанный в параметре *create_definition* (его синтаксис совпадает с тем, что описан в разделе CREATE TABLE). При необходимости имеющиеся данные преобразуются и помещаются в новый столбец. Стоит заметить, что внутри *create_definition* содержится имя нового столбца, поэтому данное действие используется также для переименования. Если вы хотите оставить старое имя, не забудьте указать его дважды: один раз для *col_name*, а второй — для *create_definition*, или воспользуйтесь описанным ниже действием MODIFY.

- DISABLE KEYS и ENABLE KEYS. При добавлении в таблицу большого количества записей MySQL затрачивает много времени на обновление индексов. Если перед этим выполнить команду ALTER TABLE ... DISABLE KEYS, MySQL отложит обновление. Как только записи будут добавлены, выполните ALTER TABLE ... ENABLE KEYS, чтобы обновить индексы для всех новых строк сразу. Такой подход позволяет сэкономить время по сравнению с одиночными обновлениями.

- DROP [COLUMN] *col_name*. Назначение этого действия понятно исходя из его синтаксиса. Оно полностью удаляет столбец из таблицы. Хранящиеся в нем данные уже не подлежат восстановлению, поэтому будьте внимательны, когда указываете имя столбца. Как и в предыдущих случаях, слово COLUMN можно опустить.

- DROP PRIMARY KEY, DROP INDEX *index_name* и DROP FOREIGN KEY *index_name*. Суть этих действий понятна без дополнительных объяснений: они удаляют из таблицы первичный ключ, индекс и ограничение внешнего ключа соответственно.

- MODIFY [COLUMN] *create_definition*. Почти ничем не отличается от вышеупомянутого действия CHANGE. Позволяет задать новое определение для столбца в таблице, но по умолчанию оставляет ему старое имя. Таким образом, нет необходимости переопределять столбец с тем же именем в параметре *create_definition*, чей синтаксис описан в разделе CREATE TABLE. Как и ранее, слово COLUMN является необязательным. Хотя это довольно удобное действие, оно не входит в стандартный синтаксис языка SQL и добавлено для совместимости с базами данных Oracle, имеющими такое же расширение.

- ORDER BY *col_name*. Сортирует записи по конкретному столбцу. Тем не менее это не гарантирует упорядоченность записей, добавленных или измененных после выполнения запроса. Единственная практическая выгода от использования такого действия заключается в повышении производительности таблицы, данные из которой сортируются в приложении в рамках запросов SELECT. В некоторых случаях сортировка происходит быстрее, если строки уже выстроены в (почти) правильном порядке.

- RENAME [TO] *new_tbl_name*. Переименовывает таблицу. Ключевое слово TO роли не играет и является необязательным.

- table_options. Используя тот же синтаксис, что и в запросе CREATE TABLE, данное действие позволяет добавлять и изменять расширенные

свойства таблицы, которые задокументированы в справочном руководстве по MySQL (http://dev.mysql.com/doc/refman/5.5/en/create-table.html).

ANALYZE TABLE

```
ANALYZE TABLE tbl_name[, tbl_name ...]
```

Функция обновляет информацию, с помощью которой оптимизируются запросы SELECT, использующие табличные индексы. Периодический запуск такого запроса повышает производительность таблиц, чье содержимое со временем претерпевает значительные изменения. Во время проведения анализа все вовлеченные таблицы переводятся в режим «только для чтения».

BEGIN

```
BEGIN
```

Выполняет то же действие, что и START TRANSACTION.

COMMIT

```
COMMIT
```

После того как выполнена команда START TRANSACTION и начинается транзакция (то есть когда режим автоматического выполнения отключен), MySQL начинает накапливать изменения, вносимые в базу данных, чтобы применить их одновременно. Это делается с помощью команды COMMIT, которая к тому же завершает транзакцию.

CREATE DATABASE

```
CREATE DATABASE [IF NOT EXISTS] db_name
```

Действие создает новую базу данных с заданным именем *db_name*. Если недостает необходимых полномочий или указанная база данных уже существует, запрос выполниться не сможет.

CREATE INDEX

```
CREATE [UNIQUE | FULLTEXT] INDEX index_name ON tbl_name (col_
name[(length)], ...)
```

Запрос создает новый индекс для уже существующей таблицы. Работает точно так же, как команда ALTER TABLE ADD {INDEX | UNIQUE | FULLTEXT}, описанная в разделе ALTER TABLE.

CREATE TABLE

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] [db_name.]tbl_name
    {   [(create_definition, ...)]
        [table_options] [[IGNORE | REPLACE] select_statement]

      | LIKE [db_name.]old_tbl_name }
```

Параметр *create_definition* представляет собой следующее.

```
{   col_name type [NOT NULL] [DEFAULT default_value]
          [AUTO_INCREMENT] [PRIMARY KEY]

      | PRIMARY KEY (index_col_name, ...)

      | INDEX [index_name] (index_col_name, ...)

      | UNIQUE [INDEX] [index_name] (index_col_name, ...)

      | FULLTEXT [index_name] (index_col_name, ...)

      | FOREIGN KEY [index_name] (index_col_name, ...)
        REFERENCES tbl_name (index_col_name, ...)
        [ON DELETE { RESTRICT | CASCADE | SET NULL | NO ACTION }]
        [ON UPDATE { RESTRICT | CASCADE | SET NULL | NO ACTION }]}
```

В этом коде *type* обозначает тип столбца (см. приложение Г), а *index_col_name* делает то же, что и команда ALTER TABLE ADD INDEX, описанная в разделе ALTER TABLE.

Запрос CREATE TABLE используется для создания таблицы с именем *tbl_name* в текущей базе данных или в той, что была указана с помощью параметра *db_name*. Если ввести ключевое слово TEMPORARY, таблица исчезнет, как только разорвется соединение, в ходе которого ее создали. Временная таблица, созданная под тем же именем, что и существующая, будет скрывать последнюю от текущей клиентской сессии до тех пор, пока временную таблицу не удалят или пока не завершится текущая сессия. Другие клиенты смогут видеть настоящую таблицу.

Запрос завершится неудачно, если таблица с таким именем уже существует, а ключевое слово TEMPORARY не указано. Однако этого можно избежать: команда IF NOT EXISTS позволит проигнорировать запрос. Ошибка также произойдет, если у вас нет необходимых прав для выполнения данного запроса.

В большинстве случаев после имени таблицы следует объявить ее столбцы (см. выше параметр *create_definition*). Описание каждого столбца состоит из имени, типа данных и любых приведенных ниже параметров.

- NOT NULL. При наличии этого параметра столбец нельзя оставить пустым (NULL). Параметр NULL, или «без значения», далеко не одно и то же, что

пустая строка (''). Например, столбцу типа VARCHAR, для которого указан параметр NOT NULL, можно присвоить '', но не NULL. Аналогично ненулевой столбец типа INT может содержать ноль, который также является значением, но не может равняться NULL, то есть не содержать ничего.

- DEFAULT default_value. Ключевое слово DEFAULT позволяет задать определенное значение столбцу, для которого в запросе INSERT не было ничего указано. Как правило, в такой ситуации столбцам, для которых не указан параметр NOT NULL, присваивается значение NULL. Если же указать DEFAULT, то даже ненулевые столбцы получат значение по умолчанию в зависимости от типа: пустая строка (''), ноль (0), '0000-00-00' или текущее время.
- AUTO_INCREMENT. Как упоминалось в главе 2, если столбцу с параметром AUTO_INCREMENT передать значение NULL, то он автоматически получит номер, который на единицу больше, чем самое большое число в этом столбце на текущий момент. Столбец с таким параметром также должен быть помечен как NOT NULL и быть либо первичным ключом (PRIMARY KEY), либо уникальным (UNIQUE).
- PRIMARY KEY. Параметр указывает на то, что заданный столбец является первичным ключом для таблицы, то есть каждая строка в нем определяется однозначно. Следовательно, значения в столбце являются уникальными и должны индексироваться, чтобы ускорить поиск соответствующих элементов.
- UNIQUE. Параметр похож на PRIMARY KEY: он требует, чтобы все значения в столбце были уникальными и индексировались для высокоскоростного поиска.

Помимо описания столбцов вы имеете возможность указать дополнительные индексы, которые хотите создать в таблице. Для этого в *create_definition* используются такие параметры, как PRIMARY KEY, INDEX, UNIQUE, FULLTEXT и FOREIGN KEY. Описание аналогичных параметров для команды ALTER TABLE находится в соответствующем разделе.

Фрагмент *table_options* используется в запросе CREATE TABLE для задания менее очевидных параметров таблицы, таких как DEFAULT CHARACTER SET utf8 и ENGINE=InnoDB. Более подробно с ними можно ознакомиться в справочном руководстве по MySQL (http://dev.mysql.com/doc/mysql/en/create-table.html).

Фрагмент *select_statement* позволяет создавать таблицу на основе результатов выполнения запроса SELECT (см. также раздел SELECT). При этом следует обязательно объявить отдельные столбцы, которые соотносятся с полученными результатами. Такой подход пригодится, если вы хотите сохранить команду SELECT во временную таблицу, а затем выполнить весь набор запросов к таблице.

Чтобы не описывать таблицу с нуля, сделайте так, чтобы при ее создании использовалась уже имеющаяся структура. Для этого опустите параметры *create_definitions* и *table_options* и укажите в конце запроса CREATE TABLE имя уже существующей таблицы с нужной структурой, написав после него ключевое слово LIKE.

DELETE

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    {   FROM tbl_name
          [WHERE where_clause]
          [ORDER BY order_by_expr]
          [LIMIT rows]

    | tbl_name[, tbl_name ...]
          FROM table_references
          [WHERE where_clause]

    | FROM tbl_name[, tbl_name ...]
          USING table_references
          [WHERE where_clause] }
```

Если для первой разновидности этого запроса не указать необязательные (но желательные) команды `WHERE` или `LIMIT`, он удалит в указанной таблице все строки. Оператор `WHERE` здесь работает так же, как и в запросе `SELECT` (см. раздел SELECT). Команда `LIMIT` позволяет указать максимальное количество удаляемых строк. С помощью оператора `ORDER BY` можно задать порядок, в котором удаляются записи, а в сочетании с командой `LIMIT` он позволяет выполнять такие действия, как, например, удаление десяти строк с самыми старыми записями в таблице.

Вторая и третья разновидности равнозначны. Они позволяют удалять строки из нескольких таблиц с помощью одной команды, подобно тому как запрос `SELECT` извлекает и объединяет данные из разных источников (см. раздел SELECT). Параметры в *`table_references`* работают точно так же, как и в случае с `SELECT`: вы можете создавать простые и внешние объединения. Оператор `WHERE` позволяет сузить диапазон удаляемых строк. Первый список в *`table_references`* определяет таблицы, в которых производится удаление. Таким образом, вы имеете возможность ограничить набор результатов, используя сложные объединения, а потом удалить строки только в одной из охваченных таблиц.

Параметр `LOW_PRIORITY` позволяет дождаться, чтобы перед выполнением запроса к указанной таблице перестали обращаться другие клиенты. Параметр `QUICK` пытается ускорить выполнение длительных операций удаления и повлиять на процесс обновления табличных индексов. Если указать ключевое слово `IGNORE`, MySQL перестанет сообщать об ошибках, возникающих в процессе удаления.

DESCRIBE/DESC

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

Команда предоставляет информацию о столбцах конкретной таблицы. Это может быть как набор столбцов, удовлетворяющих заданному шаблону с метасимволами % и _ (*`wild`*), так и один столбец с определенным именем (*`col_name`*). Полученный результат включает имя столбца, его тип, значение по умолчанию, дополнительные параметры, например `AUTO_INCREMENT`, и информацию о том, допускает ли столбец `NULL` в качестве значения и содержит ли индекс.

DROP DATABASE

```
DROP DATABASE [IF EXISTS] db_name
```

Команду следует использовать *очень осторожно*: она мгновенно удалит базу данных вместе со всеми таблицами. Запрос завершится ошибкой, если у вас не хватает полномочий или указанной базы данных не существует. В последнем случае избежать сообщения об ошибке можно, указав параметр `IF EXISTS`.

DROP INDEX

```
DROP INDEX index_name ON tbl_name
```

Работает аналогично команде `ALTER TABLE DROP INDEX`, описанной в разделе ALTER TABLE.

DROP TABLE

```
DROP TABLE [IF EXISTS] tbl_name[, tbl_name, ...]
```

Запрос полностью удаляет одну или несколько таблиц. Использовать его следует *осторожно*, поскольку после его выполнения данные уже нельзя восстановить. При нехватке полномочий команда завершится ошибкой. То же произойдет, если указанной таблицы не существует. Чтобы избежать появления сообщения об ошибке во втором случае, укажите параметр `IF EXISTS`.

EXPLAIN

Запрос имеет две разные формы. Первая является эквивалентом команд `DESCRIBE tbl_name` и `SHOW COLUMNS FROM tbl_name`.

```
EXPLAIN tbl_name
```

Вторая выглядит так:

```
EXPLAIN select_statement
```

В качестве `select_statement` выступает корректный запрос `SELECT`. Команда возвращает описание того, как MySQL определяет результаты выполнения оператора `SELECT`. Она пригодится при поиске участков выборки, на которых можно ускорить выполнение запроса с использованием индексов. Команда также позволяет определить, выполняются ли многотабличные запросы в оптимальном порядке. Чтобы узнать, как управлять этим порядком, переопределяя оптимизатор MySQL, обратитесь к разделу SELECT и ознакомьтесь с параметром выборки `STRAIGHT_JOIN`. Исчерпывающую информацию о том, как интерпретировать

результаты выполнения запроса EXPLAIN, вы найдете в справочном руководстве по MySQL (http://dev.mysql.com/doc/mysql/en/explain.html).

GRANT

```
GRANT priv_type [(column_list)], …
      ON {tbl_name | * | *.* | db_name.*}
      TO username [IDENTIFIED BY 'password'], …
      [WITH GRANT OPTION]
```

Команда предоставляет дополнительные права доступа для существующей учетной записи или создает новую, если в базе данных нет указанного имени *username*. Она также позволяет изменить имеющийся пароль учетной записи с помощью конструкции IDENTIFIED BY '*password*'.

Полное описание этого и других запросов, предназначенных для управления учетными записями пользователей, ищите в справочном руководстве по MySQL (http://dev.mysql.com/doc/mysql/en/account-management-sql.html).

INSERT

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE] [INTO] tbl_name
    {   [(col_name, …)] VALUES (expression, …), …

      | SET col_name=expression, col_name=expression, …

      | [(col_name, …)] SELECT … }

      [ON DUPLICATE KEY UPDATE col_name=expression[, …]]
```

Запрос используется для добавления новых записей в таблицу и поддерживает три главных параметра.

- LOW_PRIORITY. Выполнение запроса отложится до тех пор, пока таблицу не перестанут использовать другие клиенты.
- DELAYED. С точки зрения клиента запрос завершается мгновенно, однако операция добавления продолжает выполняться в фоновом режиме. Данный параметр следует использовать, если необходимо добавить большое количество строк, но нет желания ждать завершения процесса. Имейте в виду, что после окончания работы такого отложенного запроса клиент не будет знать идентификатор последней добавленной записи в столбце AUTO_INCREMENT. К примеру, в PHP метод lastInsertId из объекта PDO не сможет работать корректно.
- IGNORE. Как правило, когда операция добавления вызывает конфликт в столбцах, помеченных как PRIMARY KEY или UNIQUE, она завершается неудачно и возвращает ошибку. Данный параметр позволяет скрыть неудачу: новая строка не добавится, но и не появится сообщение об ошибке.

Использовать слово INTO необязательно: оно не влияет на выполнение запроса.

Исходя из вышесказанного, существует три вида запросов INSERT. Первый позволяет добавлять одну или несколько строк, указывая в круглых скобках значения для полей. Если опустить необязательный список с именами столбцов, значения для каждого столбца необходимо перечислить в том порядке, в котором они размещены в таблице.

Вторая разновидность команды INSERT используется для добавления одной строки. Значения для полей строки указываются в виде *col_name=value*.

Третья и последняя форма команды INSERT подразумевает, что в качестве добавляемых строк выступают результаты выполнения запроса SELECT. Если вы хотите опустить список с названиями строк, то результирующий набор выборки должен содержать значения для каждого столбца в нужном порядке. Запрос SELECT, входящий в состав выражения INSERT, позволяет обойтись без оператора ORDER BY, но при этом в операторе FROM нельзя использовать таблицу, в которую добавляются записи.

Столбцы, которым ничего не присваивается (например, они не указаны в списке), принимают значения по умолчанию. Как правило, то же происходит со столбцами, помеченными как NOT NULL, если им задается значение NULL. Однако, если в MySQL включен параметр DONT_USE_DEFAULT_FIELDS, данная операция вызовет ошибку, поэтому таких ситуаций лучше избегать.

Необязательный оператор ON DUPLICATE KEY UPDATE срабатывает, когда запрос INSERT пытается добавить в таблицу новую запись, дублируя уже имеющееся значение. Это происходит только при наличии параметров UNIQUE или PRIMARY KEY. Данный оператор предотвращает ошибку и описывает, каким образом должна обновиться существующая запись. Внешне он очень похож на команду UPDATE: в нем указывается один или несколько столбцов и новые значения, которые им следует присвоить. Более подробно об этом рассказано в разделе UPDATE.

LOAD DATA INFILE

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE
      'file_name.txt' [REPLACE | IGNORE] INTO TABLE tbl_name
      [FIELDS
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char'] ]
      [LINES [STARTING BY ''] [TERMINATED BY 'string']]
      [IGNORE number LINES]
      [(col_name, ...)]
```

Запрос используется для импорта данных из текстового файла, который хранится на MySQL-сервере или в клиентской системе (LOCAL). Это может быть файл, созданный с помощью команды SELECT INTO OUTFILE. Чтобы получить исчерпывающую информацию о запросе и о проблемах, которые возникают в результате его использования, обратитесь к справочному руководству по MySQL (http://dev.mysql.com/doc/mysql/en/load-data.html).

OPTIMIZE TABLE

```
OPTIMIZE TABLE tbl_name[, tbl_name …]
```

Удаление или изменение размеров файлов со временем приводит к фрагментированию разделов жесткого диска. То же происходит и с таблицами в MySQL, в которых удаляются строки или изменяются столбцы переменной длины, такие как VARCHAR или BLOB. Этот запрос является для таблиц базы данных тем же, что и команда `defrag` для файловой системы: он реорганизует хранящиеся данные и устраняет неиспользуемое пространство.

В процессе оптимизации таблица *блокируется*, поэтому приложение, которому необходим постоянный доступ к ней, не сможет продолжать свою работу, пока не завершится оптимизация. В таких случаях лучше создать копию таблицы, оптимизировать ее, а затем заменить ею прежнюю версию с помощью запроса RENAME. Изменения, внесенные в оригинальную таблицу в этот промежуток времени, будут утеряны, поэтому данный прием подходит лишь для некоторых приложений.

RENAME TABLE

```
RENAME TABLE tbl_name TO new_table_name[, tbl_name2 TO …, …]
```

Запрос позволяет быстро и удобно переименовать одну или несколько таблиц. От `ALTER TABLE tbl_name RENAME` он отличается тем, что на время выполнения запроса все затронутые таблицы блокируются, поэтому ни один клиент не получит к ним доступ. Согласно справочному руководству по MySQL (http://dev.mysql.com/doc/mysql/en/rename-table.html) атомарность, которая обеспечивается этой командой, позволяет заменить существующую таблицу ее пустым эквивалентом. К примеру, следующий код позволяет создать новую таблицу при достижении какого-то определенного количества записей.

```
CREATE TABLE new_table (…);
RENAME TABLE old_table TO backup_table, new_table TO old_table;
```

Задав имя таблицы как `db_name.tbl_name`, вы можете перемещать ее между базами данных. Для этого обе копии должны находиться на одном физическом диске, чаще всего так оно и есть.

Чтобы выполнить данный запрос, нужно иметь право на использование команд ALTER и DROP по отношению к оригинальной таблице, а также INSERT по отношению к новой. Запрос RENAME TABLE, завершившийся неудачей на полпути, автоматически отменяется, при этом изначальное состояние таблицы восстанавливается.

REPLACE

```
REPLACE [LOW_PRIORITY | DELAYED] [INTO] tbl_name
    {   [(col_name, …)] VALUES (expression, …), …
```

```
    | [(col_name, …)] SELECT …

    | SET col_name=expression, col_name=expression, … }
```

Команда похожа на INSERT. Если из-за столбца, отмеченного как PRIMARY KEY или UNIQUE, новая строка станет конфликтовать с уже имеющимся значением, команда REPLACE заменит старую запись.

REVOKE

```
REVOKE priv_type [(column_list)], …
    ON {tbl_name | * | *.* | db_name.*}
    FROM user, …
```

Запрос аннулирует права доступа для учетной записи. Если пользователя лишить всех прав, он по-прежнему сможет входить в систему, но доступ к любой информации ему будет закрыт. Исчерпывающее описание данного запроса вы найдете в главе 10 в разделе «Советы по контролю доступа в MySQL».

ROLLBACK

```
ROLLBACK
```

После того как вы инициируете транзакцию с помощью команды START TRANSACTION, MySQL начнет накапливать изменения, вносимые в базу данных, чтобы затем применить их единовременно. Указанный запрос сбрасывает все изменения и отменяет транзакцию.

SELECT

```
SELECT [select_options]
    select_expression, …
    [INTO {OUTFILE | DUMPFILE} 'file_name' export_options]
    [FROM table_references
      [WHERE where_definition]
      [GROUP BY {col_name | col_pos } [ASC | DESC], …]
      [HAVING where_definition]
      [ORDER BY {col_name | col_pos } [ASC | DESC], …]
      [LIMIT [offset,] rows]]
```

Самая сложная команда в языке SQL, которая используется во всех операциях по извлечению данных. Ниже представлены параметры *select_options*, которые она поддерживает. Вы можете перечислять их, разделяя пробелами, и составлять с их помощью любые разумные комбинации.

- ALL, DISTINCT и DISTINCTROW. Любой из этих параметров позволяет описывать поведение запроса при дублировании строк в результирующем наборе. ALL применяется по умолчанию и приводит к тому, что в ходе

выполнения запроса выводятся все дублирующиеся строки. Параметры DISTINCT и DISTINCTROW работают одинаково и убирают дубликаты.

- HIGH_PRIORITY. Назначает запросу SELECT высокий приоритет. Как правило, при обновлении таблицы все запросы, в ходе которых происходит считывание данных (к ним относится и SELECT), переходят в режим ожидания. В этом случае, как только появится возможность выполнить запрос, команда SELECT HIGH_PRIORITY будет выполнена в первую очередь.
- STRAIGHT_JOIN. Заставляет MySQL объединять таблицы, указанные в *table_references* в порядке их перечисления. Если вы считаете, что MySQL плохо оптимизирует данный запрос и выполняет его медленно, вы можете вмешаться в процесс с помощью данного аргумента. Более подробно об этом рассказано в подразделе «Объединения».
- SQL_BUFFER_RESULT. Требует от MySQL сохранить результирующий набор во временной таблице. Пока результаты передаются клиенту, другие процессы имеют возможность получить доступ к таблицам, задействованным в запросе.
- SQL_CACHE. Сообщает MySQL о том, что результат выполнения запроса необходимо сохранять в **кеше** — области памяти, которая выделяется сервером для хранения результатов работы часто запускаемых запросов. Таким образом, если содержимое соответствующих таблиц осталось прежним, выполнять запрос заново не нужно. При этом MySQL можно настроить так, чтобы кешировались только запросы с параметром SQL_CACHE. Если кеш выключен, данный параметр ни на что не повлияет.
- SQL_NO_CACHE. При его указании MySQL не станет кешировать результат текущего запроса. MySQL-сервер можно настроить таким образом, чтобы он помещал в кеш результаты выполнения всех запросов, для которых данный параметр не указан. Если кеш выключен, параметр SQL_NO_CACHE ни на что не повлияет.
- SQL_CALC_FOUND_ROWS. Применяется в сочетании с оператором LIMIT. Вычисляет и сохраняет общее количество строк, которые были бы возвращены, если бы оператор LIMIT не был указан. Данное число можно извлечь с помощью запроса SELECT FOUND_ROWS() (см. приложение В).

Параметр *select_expression* определяет поле результирующего набора, который возвращается в результате запроса. Обычно это столбец таблицы, записанный в виде *col_name*, *tbl_name.col_name* или *db_name.tbl_name.col_name*. Форма записи зависит от того, какая точность требуется для верной идентификации сервером MySQL. Данный параметр способен ссылаться не только на столбец базы данных, но и на другие выражения. Вы можете использовать простые математические формулы с именами полей в качестве переменных, а также сложные выражения, вычисляемые с помощью функций MySQL. Например, вам необходимо вернуть дату в формате «Январь 1, 2010», которая наступит через один месяц с настоящего момента.

```
SELECT DATE_FORMAT(DATE_ADD(CURDATE(), INTERVAL 1 MONTH), '%M %D, %Y')
```

Параметр *select_expressions* также может содержать псевдоним (или назначенное имя) для итогового столбца. Псевдоним указывается с помощью ключевого слова [AS], которое является необязательным. Подобное выражение используется, если на указанный столбец необходимо сослаться в каком-то другом участке запроса, например в операторах WHERE и ORDER BY.

```
SELECT jokedate AS jd FROM joke ORDER BY jd ASC
```

MySQL позволяет использовать оператор INTO, чтобы выводить результаты запроса в файл, а не возвращать их клиенту. Чаще всего этот оператор применяется для экспорта содержимого таблицы в файл формата CSV, внутри которого значения разделяются запятыми.

```
SELECT * INTO OUTFILE '/home/user/myTable.txt'
    FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
    LINES TERMINATED BY '\n'
    FROM myTable
```

Файл, куда записываются результаты, не должен существовать до выполнения запроса. Это ограничение необходимо для того, чтобы с помощью SQL-запросов нельзя было перезаписывать важные системные файлы. В операционных системах, которые поддерживают контроль доступа к файлам, извлеченные данные окажутся доступными для чтения любому пользователю системы, поэтому хорошо подумайте, прежде чем экспортировать важную информацию.

Чтобы записать все данные в одну строку без переносов и разделителей между столбцами, вместо OUTFILE используйте параметр DUMPFILE. С его помощью можно сбросить в файл хранящийся в таблице бинарный BLOB (SELECT *blobCol* INTO DUMPFILE ...). Исчерпывающие сведения об операторе INTO вы найдете в справочном руководстве по MySQL (http://dev.mysql.com/doc/mysql/en/select.html). Информация о том, как считать данные из созданного файла, находится в разделе LOAD DATA INFILE.

В операторе FROM содержится список таблиц, на основе которых формируется результирующий набор, а также инструкции о том, каким образом их следует объединить. Параметр *table_references*, по сути, имя одной таблицы, которая может быть назначена в качестве псевдонима с помощью (или без) ключевого слова AS, как было описано выше для *select_expression*. Если указать несколько таблиц, они **объединятся**. Более подробно вы узнаете об этом в подразделе «Объединения».

В участке *where_definition* оператора WHERE задается условие для строки, которая включается в результирующую таблицу, возвращаемую в ответ на запрос SELECT. Это может быть как простое условие (например, id=5), так и сложное с использованием функций MySQL и комбинаций из нескольких условий, совмещенных с помощью булевых операторов AND, OR или NOT.

Оператор GROUP BY позволяет указать один или несколько столбцов по имени, псевдониму или позиции (единица обозначает первый столбец в результирующем наборе). Строки, значения которых совпадают по полям, представлены в результирующем наборе одной записью. Как правило, данный оператор используется в сочетании с функциями группирования, такими как COUNT, MAX и AVG (см. приложение В), чтобы итоговые столбцы содержали сводку о созданных

группах. По умолчанию результаты сортируются в порядке возрастания значений, содержащихся в сгруппированных полях. Возрастающий или убывающий порядок сортировки задается явно с помощью соответствующих аргументов ASC или DESC, указанных после каждого столбца. Связанные наборы строк сортируются вначале по первому столбцу в списке, затем по второму и т. д.

Оператор WHERE выполняется до GROUP BY, поэтому условие не вправе ссылаться на столбцы, которые зависят от операций группирования. Чтобы описать условие для уже сгруппированного результирующего набора, воспользуйтесь оператором HAVING. Его синтаксис аналогичен синтаксису WHERE, но заданные условия обрабатываются непосредственно перед возвращением результатов и не подлежат оптимизации, поэтому применяйте оператор WHERE везде, где это возможно. Больше информации о командах GROUP BY и HAVING вы найдете в главе 11.

Оператор ORDER BY позволяет сортировать результаты в соответствии со значениями в одной или нескольких строках перед тем, как они будут возвращены. Каждый столбец указывается с помощью имени, псевдонима или позиции (единица обозначает первый столбец в результирующем наборе) и содержит аргумент ASC или DESC для задания порядка сортировки — по возрастанию (применяется по умолчанию) или убыванию соответственно. Связанные наборы строк сортируются сначала по первому столбцу, затем по второму и т. д.

Оператор LIMIT позволяет вернуть только часть тех результатов, которые запрос сгенерировал бы в обычных условиях. В простейшем случае LIMIT n возвращает только первые *n* строк из окончательного результирующего набора. Вы также можете указать сдвиг, используя запись вида LIMIT *k*, *n*. Таким образом, из всего результирующего набора вернется не более *n* строк, начиная с *k*-й. Для первой строки $k = 0$, для второй $k = 1$ и т. д.

Объединения

Оператор FROM в запросе WHERE позволяет указать таблицы, которые участвуют в формировании результирующего набора. Процесс, в ходе которого несколько таблиц совмещаются подобным образом, называется **объединением**. MySQL поддерживает несколько видов объединений. Ниже представлены варианты их синтаксиса для компонента *table_references*, принадлежащего оператору FROM.

```
table_ref

table_references, table_ref

table_references [CROSS] JOIN table_ref

table_references INNER JOIN table_ref join_condition

table_references STRAIGHT_JOIN table_ref

table_references LEFT [OUTER] JOIN table_ref join_condition
    { OJ table_ref LEFT OUTER JOIN table_ref ON cond_expr }

table_references NATURAL [LEFT [OUTER]] JOIN table_ref

table_references RIGHT [OUTER] JOIN table_ref join_condition

table_references NATURAL [RIGHT [OUTER]] JOIN table_ref
```

Параметр *table_ref* определяется следующим образом.

```
table_name [[AS] alias] [USE INDEX (key_list)]
      [IGNORE INDEX (key_list)]
```

В роли *join_condition* может выступать одно из выражений.

```
ON cond_expr

USING (column_list)
```

Пусть вас не пугает разнообразие типов объединений. Рассмотрим принцип работы каждого из них.

В результате самого простого внутреннего объединения возвращаются все возможные сочетания строк из первой и второй таблицы. В MySQL при выполнении внутреннего объединения вы можете разделять имена таблиц как посредством запятых, так и с помощью слов JOIN, CROSS JOIN или INNER JOIN (все они равнозначны).

Как правило, для создания внутренних объединений используют запятые, особенно в старом PHP-коде. Затем, применяя оператор WHERE внутри запроса SELECT, объявляется условие, сужающее набор возвращаемых комбинированных строк. Например, это может быть сопоставление первичного ключа первой таблицы со столбцом второй. Однако сегодня такой неаккуратный подход считается дурным тоном.

Вместо этого INNER JOIN следует размещать после выражения *join_condition*. При записи этого же выражения с ключевым словом ON условие (одно или несколько), необходимое для объединения двух таблиц, размещается сразу после их имен, а в операторе WHERE указываются другие условия, не связанные с операцией объединения.

Еще одна разновидность выражения *join_condition*, в котором используется ключевое слово USING (*column_list*), позволяет указать столбцы двух таблиц, которые должны соответствовать друг другу.

```
SELECT * FROM t1 INNER JOIN t2 USING (tid)
```

Это выражение эквивалентно следующему.

```
SELECT * FROM t1 INNER JOIN t2 ON t1.tid = t2.tid
```

Оператор STRAIGHT_JOIN работает так же, как внутреннее объединение, за исключением того, что таблицы в нем обрабатываются в порядке перечисления (сначала левая, потом правая). Как правило, MySQL выбирает тот порядок, который обеспечивает наиболее быструю обработку, но если вы уверены, что ваш способ лучше, воспользуйтесь STRAIGHT_JOIN.

Второй вид объединений называется **внешним** и реализуется в MySQL операторами LEFT/RIGHT [OUTER] JOIN. Слово OUTER не играет особой роли и не является обязательным. При левом внешнем объединении (LEFT) любая запись из таблицы слева, которая не имеет соответствий в таблице справа, определяемых условием *join_condition*, будет значиться в результирующем наборе как одна строка. Всем столбцам, полученным из правой таблицы, присвоится значение NULL.

Синтаксис { OJ ... } эквивалентен обычному левому внешнему объединению. Он добавлен для совместимости с другими базами данных, поддерживающими стандарт ODBC (Open Database Connectivity).

Правое внешнее объединение работает так же, как и левое, при этом в результирующий набор попадают записи из правой таблицы, даже если у них нет соответствий в левой. Правое внешнее объединение является нестандартным, поэтому для совместимости с другими базами данных лучше использовать LEFT JOIN.

Некоторые примеры использования внешних объединений рассмотрены в главе 11.

Естественное объединение является автоматическим в том смысле, что оно само сопоставляет строки из двух разных таблиц по столбцам с одинаковыми именами. Таким образом, если таблица joke имеет столбец authorid, который ссылается на записи в таблице author, чей первичный ключ — столбец с аналогичным именем authorid, то вы можете выполнить объединение двух таблиц по общему столбцу (если исходить из того, что больше столбцов с одинаковыми именами нет).

```
SELECT * FROM joke NATURAL JOIN author
```

Оператор UNION

Оператор UNION совмещает результаты выполнения нескольких запросов SELECT, возвращая единый результирующий набор. Каждый из запросов выдает одно и то же количество столбцов, типы которых должны совпадать. В результирующем наборе используются имена столбцов, взятые из первого запроса.

```
SELECT …
    UNION [ALL | DISTINCT]
    SELECT …
      [UNION [ALL | DISTINCT]
        SELECT …] …
```

По умолчанию оператор UNION устраняет дублирующиеся строки, чтобы все записи в результирующем наборе являлись уникальными. Такое поведение можно задать явно с помощью параметра DISTINCT, но в сущности это ничего не изменит. Чтобы разрешить дублирование результатов, воспользуйтесь параметром ALL.

SET

```
SET option = value, …
```

Запрос позволяет задать набор параметров как на клиентской, так и на серверной стороне.

Например, чтобы отключить режим автоматического выполнения для текущей сессии, используется команда SET autocommit = 0. В сущности, результат будет тем же, если выполнить запрос START TRANSACTION, а затем запустить его снова после выполнения команды COMMIT или ROLLBACK. Если параметр autocommit

отключен, вы всегда будете находиться в режиме транзакции и такие запросы, как INSERT, UPDATE и DELETE не вступят в силу до тех пор, пока вы не примените их с помощью команды COMMIT.

С полным списком параметров, доступных для использования в запросе SET, можно ознакомиться в справочном руководстве по MySQL (http://dev.mysql.com/doc/mysql/en/set-option.html).

SHOW

Запрос имеет множество разных форм и позволяет получить информацию о сервере MySQL, а также о хранящихся в нем базах данных и таблицах. Во многих его формах предусмотрен необязательный компонент LIKE *wild*, где *wild* — строка для фильтрации списка с результатами, которая может содержать метасимволы (% для множественных символов и _ для одиночных). Ниже приведены разновидности запроса.

- SHOW DATABASES [LIKE *wild*]. Перечисляет базы данных, которые доступны на MySQL-сервере.
- SHOW [OPEN] TABLES [FROM *db_name*] [LIKE *wild*]. Перечисляет все таблицы или открытые в настоящий момент (если указан параметр OPEN) в стандартной или явно заданной базе данных.
- SHOW [FULL] COLUMNS FROM tbl_name [FROM *db_name*] [LIKE *wild*]. Если параметр FULL не используется, запрос возвращает ту же информацию, что и команда DESCRIBE (см. раздел DESCRIBE/DESC). Параметр FULL добавляет в список результатов перечень ваших прав для доступа к каждому столбцу. Команды SHOW FIELDS и SHOW COLUMNS эквивалентны.
- SHOW INDEX FROM *tbl_name* [FROM *db_name*]. Предоставляет подробную информацию об индексах, которые содержит заданная таблица. Описание результатов, возвращаемых данным запросом, ищите в справочном руководстве по MySQL (http://dev.mysql.com/doc/en/show-index.html). Команды SHOW KEYS и SHOW INDEX эквивалентны.
- SHOW TABLE STATUS [FROM *db_name*] [LIKE *wild*]. Выводит подробную информацию о таблицах в стандартной или явно заданной базе данных.
- SHOW STATUS [LIKE *wild*]. Выводит подробную статистику о сервере. Значение каждого параметра описано в справочном руководстве по MySQL (http://dev.mysql.com/doc/en/show-status.html).
- SHOW VARIABLES [LIKE *wild*]. Перечисляет переменные и их значения в конфигурации MySQL. Исчерпывающее описание значений содержится в справочном руководстве по MySQL (http://dev.mysql.com/doc/en/show-variables.html).
- SHOW [FULL] PROCESSLIST. Отображает все потоки, запущенные на MySQL-сервере, а также запросы, которые выполняются в каждом из них. Если у вас нет права доступа process, вы сможете просматривать только потоки с собственными запросами. При указании параметра FULL текст запросов выводится полностью (по умолчанию отображаются только первые 100 символов).

- SHOW GRANTS FOR *user*. Перечисляет команды GRANT, с помощью которых возможно воссоздать права доступа заданного пользователя.
- SHOW CREATE TABLE *table_name*. Отображает команду CREATE TABLE, которая позволяет воссоздать указанную таблицу.

START TRANSACTION

```
START TRANSACTION
```

После того как транзакция инициирована (то есть отключен режим автоматического выполнения), с помощью данной команды MySQL начнет накапливать изменения, вносимые в базу данных, чтобы выполнить (COMMIT) или отменить (ROLLBACK) их все сразу.

TRUNCATE

```
TRUNCATE [TABLE] tbl_name
```

Результат выполнения команды таков, как и при выполнении запроса DELETE без оператора WHERE, — удаление всех строк таблицы. Однако в TRUNCATE есть набор оптимизаций, которые позволяют ускорить процесс, особенно если речь идет о больших таблицах. Фактически команда сначала удаляет таблицу с помощью запроса DROP TABLE, а затем создает ее заново с помощью CREATE TABLE.

UPDATE

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
      SET col_name = expr[, …]
      [WHERE where_definition]
      [ORDER BY …]
      [LIMIT #]
```

Запрос обновляет имеющиеся в таблице записи, присваивая заданным столбцам новые значения. Столбцы, не указанные в списке, остаются без изменений, за исключением столбцов типа TIMESTAMP (см. приложение Г). Оператор WHERE позволяет задать условие *where_definition*, которое должны удовлетворять обновляемые записи. С помощью оператора LIMIT можно указать максимальное количество строк, подлежащих обновлению.

НЕ ИГНОРИРУЙТЕ ОПЕРАТОРЫ WHERE И LIMIT

Если операторы WHERE и LIMIT пропущены, обновление коснется всех строк таблицы.

Оператор ORDER BY позволяет задать порядок обновления записей. Его хорошо использовать в сочетании с оператором LIMIT: вместе они позволяют формировать запросы вроде «обновить десять самых свежих записей».

Операция обновления завершится ошибкой, если новое значение, которое присваивается строке, вступит в конфликт с содержимым столбца, помеченного как PRIMARY KEY или UNIQUE. Чтобы этого избежать, укажите параметр IGNORE — запрос пропустит подобную строку.

Параметр LOW_PRIORITY обязует MySQL отложить обновление до тех пор, пока с таблицей не перестанут работать другие клиенты.

Как и у DELETE (см. раздел DELETE), у запроса UPDATE есть альтернативный синтаксис, который позволяет в ходе одной операции затронуть несколько таблиц.

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name[, tbl_name ...]
      SET col_name = expr[, ...]
      [WHERE where_definition]
```

USE

```
USE db_name
```

Команда задает базу данных, которая используется по умолчанию во всех запросах к MySQL в текущей сессии. Доступ к таблицам из других баз данных можно получить с помощью записи вида *db_name.tbl_name*.

Приложение В
ФУНКЦИИ MySQL

MySQL предоставляет обширную библиотеку функций, которые используются для форматирования и объединения данных в рамках SQL-запросов, гарантирующих получение желаемого результата в нужном формате. Приложение содержит перечень наиболее полезных функций, реализованных в MySQL версии 5.5.22 — последней на момент написания книги.

Самую полную и наиболее актуальную информацию об SQL-функциях вы найдете в справочном руководстве по MySQL (http://dev.mysql.com/doc/mysql/en/functions.html).

Функции для управления потоком данных

IFNULL(*expr1*, *expr2*)

Возвращает выражение *expr1*, если оно не равно NULL. В противном случае возвращается выражение *expr2*.

NULLIF(*expr1*, *expr2*)

Возвращает выражение *expr1*, если оно не равно *expr2*. В противном случае возвращается NULL.

IF(*expr1*, *expr2*, *expr3*)

Если выражение *expr1* истинно, то есть не равно NULL или 0, функция возвращает выражение *expr2*, в противном случае — *expr3*.

CASE *value* WHEN [*compare-value1*] THEN *result1* [WHEN ...] [ELSE *else-result*] END

Возвращает *result1*, если *value=compare-value1* (для сравнения значений можно определять несколько пар). В противном случае вернется результат *else-result* или NULL, если таковой не определен.

CASE WHEN [*condition1*] THEN *result1* [WHEN ...] [ELSE *else-result*] END

Если условие *condition1* истинно, функция возвращает *result1* (можно определить несколько пар «условие — результат»). В противном случае вернется результат *else-result* или NULL, если таковой не определен.

Математические функции

ABS(*expr*)

Возвращает абсолютное (положительное) значение выражения *expr*.

SIGN(*expr*)

Возвращает –1, 0 или 1 в зависимости от того, каково значение выражения *expr* (отрицательное, равное нулю или положительное).

MOD(*expr1, expr2*)

expr1* % *expr2

Возвращает остаток от деления *expr1* на *expr2*.

FLOOR(*expr*)

Округляет выражение *expr* в меньшую сторону, то есть возвращает наибольшее целое число, не превышающее *expr*.

CEILING(*expr*)

CEIL(*expr*)

Округляет выражение *expr* в большую сторону, то есть возвращает минимальное целое число, не меньше *expr*.

ROUND(*expr*)

Возвращает выражение *expr*, округленное до ближайшего целого числа. Если значение равно целому числу с половиной, поведение функции зависит от системы. Таким образом, при переходе на новую платформу полагаться на результаты выполнения ROUND не следует, какими бы они ни были.

ROUND(*expr, num*)

Округляет выражение *expr* до числа, где количество знаков после запятой равно *num*, включая нули в конце. Например, чтобы представить число в формате доллары/центы, используйте *num* со значением 2. С округлением половинных значений существует та же неопределенность, что и в предыдущем пункте.

EXP(*expr*)

Возвращает e^{expr} — основание натурального логарифма, возведенное в степень expr.

LOG(*expr*)

Возвращает ln(*expr*) или log*e*(*expr*) — натуральный логарифм выражения expr.

LOG(*B*, *expr*)

Возвращает логарифм *expr* с произвольным основанием B.

ИЗ ШКОЛЬНОГО КУРСА МАТЕМАТИКИ

LOG(*B*, *expr*) = LOG(*expr*) / LOG(*B*)

LOG10(*expr*)

Возвращает десятичный логарифм выражения *expr*.

POW(*expr1*, *expr2*)

POWER(*expr1*, *expr2*)

Возвращает выражение *expr1*, возведенное в степень *expr2*.

SQRT(*expr*)

Возвращает квадратный корень выражения *expr*.

PI()

Возвращает значение ϖ.

COS(*expr*)

Возвращает косинус выражения *expr* в радианах, например COS(PI())=-1.

SIN(*expr*)

Возвращает синус выражения *expr* в радианах, например SIN(PI())=0.

TAN(*expr*)

Возвращает тангенс выражения *expr* в радианах, например TAN(PI())=0.

ACOS(*expr*)

Возвращает арккосинус ($\cos^{-1}$, или обратный косинус) выражения *expr* в радианах, например ACOS(-1)=PI().

ASIN(*expr*)

Возвращает арксинус ($\sin^{-1}$, или обратный синус) выражения *expr* в радианах, например ASIN(0)=PI().

ATAN(*expr*)

Возвращает арктангенс ($\tan^{-1}$, или обратный тангенс) выражения `expr` в радианах, например `ATAN(0)=PI()`.

ATAN(*y*, *x*)

ATAN2(*y*, *x*)

Возвращает угол в радианах, образовавшийся в начале координат между участком оси x с положительными значениями и точкой (x, y), например `ATAN(1,0)=PI()/2`.

COT(*expr*)

Возвращает котангенс выражения *expr*, например `COT(PI()/2)=0`.

RAND()

RAND(*expr*)

Возвращает случайное число с плавающей точкой в диапазоне от 0 до 1,0. Если указать выражение *expr*, оно будет использоваться как основа для случайной величины, а сама величина изменится.

LEAST(*expr1*, *expr2*, ...)

Возвращает наименьшее из указанных значений.

GREATEST(*expr1*, *expr2*, ...)

Возвращает наибольшее из указанных значений.

DEGREES(*expr*)

Переводит радианы в градусы.

RADIANS(*expr*)

Переводит градусы в радианы.

TRUNCATE(*expr*, *num*)

Возвращает *expr* в виде числа с плавающей точкой с количеством десятичных знаков, равным *num*, то есть округленное до наименьшего.

BIN(*expr*)

Преобразует значение *expr* из десятичного в двоичное. Аналогично функции `CONV(expr, 10, 2)`.

OCT(*expr*)

Преобразует значение *expr* из десятичного в восьмеричное. Аналогично функции `CONV(expr, 10, 8)`.

HEX(*expr*)

Преобразует значение *expr* из десятичного в шестнадцатеричное. Аналогично функции CONV(*expr*, 10, 16).

CONV(*expr*, *from_base*, *to_base*)

Преобразует значение *expr* из системы счисления *from_base* в *to_base*. Возвращает NULL, если любой из аргументов равен NULL.

Строковые функции

ASCII(*str*)

Возвращает ASCII-код крайнего слева символа строки *str*; 0, если строка является пустой, и NULL, если она равна NULL.

ORD(*str*)

Возвращает ASCII-код крайнего слева символа строки *str*, учитывая, что символ может быть многобайтным.

CHAR(*expr*, ...)

Создает строку, состоящую из ASCII-кодов, переданных в качестве аргументов.

CONCAT(*str1*, *str2*, ...)

Возвращает строку, являющуюся результатом последовательного соединения подстрок, переданных в качестве аргументов. Если какой-либо из аргументов равен NULL, в качестве значения вернется NULL.

CONCAT_WS(*separator*, *str1*, *str2*, ...)

Аналогична функции CONCAT и отличается лишь разделителем (WS – сокращенное от with separator). При объединении первый аргумент помещается между всеми остальными.

LENGTH(*str*)

OCTET_LENGTH(*str*)

CHAR_LENGTH(*str*)

CHARACTER_LENGTH(*str*)

Возвращают длину строки *str* в символах. При этом CHAR_LENGTH и CHARACTER_LENGTH при подсчете учитывают мультибайтные символы.

BIT_LENGTH(*str*)

Возвращает длину строки *str* в битах, то есть BIT_LENGTH(*str*)= =8*LENGTH(*str*).

LOCATE(*substr, str*)

POSITION(*substr* IN *str*)

Возвращает позицию вхождения подстроки *substr* в строку *str* (1 — если она находится в начале, 2 — если сдвинута на один символ и т. д.). Если *substr* не входит в *str*, вернется 0.

LOCATE(*substr, str, pos*)

Аналогична функции LOCATE(*substr, str*), но начинает поиск с символа, номер которого определяется аргументом *pos*.

INSTR(*str, substr*)

Отличается от функции LOCATE(*substr, str*) только порядком следования аргументов.

LPAD(*str, len, padstr*)

Укорачивает или удлиняет строку *str* до размеров *len*. Удлинение достигается путем добавления выражения *padstr* слева от *str*. Например, LPAD('!', '5', '.')='....!'.

RPAD(*str, len, padstr*)

Укорачивает или удлиняет строку *str* до размеров *len*. Удлинение достигается путем добавления выражения *padstr* справа от *str*. Например, RPAD('!','5','.') = '!....'.

LEFT(*str, len*)

Возвращает крайние слева символы в строке *str* в количестве, равном *len*. Если символов меньше, чем указано в *len*, значение *str* вернется без дополнительного сдвига.

RIGHT(*str, len*)

Возвращает крайние справа символы в строке str в количестве, равном len. Если символов меньше, чем указано в *len*, значение *str* вернется без дополнительного сдвига.

SUBSTRING(*str, pos, len*)

SUBSTRING(*str* FROM *pos* FOR *len*)

MID(*str, pos, len*)

Возвращает часть строки *str*, состоящую из символов, количество которых равно *len*, начиная с позиции *pos*, где 1 обозначает первый символ. Форма SUBSTRING входит в стандарт ANSI.

SUBSTRING(*str, pos*)

SUBSTRING(*str* FROM *pos*)

Возвращает часть строки *str*, начиная с позиции *pos*, где 1 обозначает первый символ.

SUBSTRING_INDEX(*str*, *delim*, *count*)

MySQL отсчитывает число вхождений *count* разделителя *delim* в строку *str* и возвращает подстроку с высчитанной позиции. Если *count* больше нуля, подсчет ведется вправо от начала строки и охватывает все символы до разделителя, за исключением самого разделителя. Если *count* меньше нуля, подсчет ведется влево от конца строки, при этом берется подстрока справа от разделителя и до конца *str*.

LTRIM(*str*)

Возвращает строку *str*, удаляя пробелы в начале.

RTRIM(*str*)

Возвращает строку *str*, удаляя пробелы в конце.

TRIM([[BOTH | LEADING | TRAILING] [*remstr*] FROM] *str*)

Удаляет в строке *str* пробелы (по умолчанию) или вхождения подстроки *remstr*. Удаление выполняется в начале или в конце строки либо с обоих концов (по умолчанию) — LEADING, TRAILING и BOTH соответственно.

SOUNDEX(*str*)

Возвращает строку, которая описывает звучание текста, содержащегося в *str*. При этом описание слов, схожих по звучанию, выглядит одинаково.

Например:

```
SOUNDEX("tire") = "T600"
SOUNDEX("tyre") = "T600"
SOUNDEX("terror") = "T600"
SOUNDEX("tyrannosaur") = "T6526"
```

SPACE(*num*)

Возвращает строку, состоящую из символов пробела. Число пробелов равняется *num*.

REPLACE(*str*, *from_str*, *to_str*)

Возвращает строку *str*, в которой все вхождения подстроки *from_str* заменены на *to_str*.

REPEAT(*str*, *count*)

Возвращает строку *str*, повторенную столько раз, сколько задано в выражении *count*. Если *count* меньше либо равно 0, вернется пустая строка. Если любой из аргументов равен NULL, вернется значение NULL.

REVERSE(*str*)

Меняет порядок символов в строке *str* на обратный.

INSERT(*str, pos, len, newstr*)

Удаляет из строки *str* подстроку длиною *len*, начиная с позиции *pos*, где 1 обозначает первый символ в строке. На место удаленной подстроки помещается выражение *newstr*. Если *len* равно 0, функция вставит *newstr* в указанную позицию.

ELT(*N, str1, str2, str3, ...*)

Возвращает N-й строковый аргумент (*str1*, если N = 1; *str2*, если N = 2, и т. д.) или NULL, если соответствующего аргумента нет.

FIELD(*str, str1, str2, str3, ...*)

Возвращает позицию строки *str* в списке заданных аргументов (1, если *str* равно *str1*; 2, если *str* равно *str2*, и т. д.).

FIND_IN_SET(*str, strlist*)

Если *strlist* представляет собой список вида 'string1,string2,string3,...', функция возвращает индекс строки *str*, входящей в этот список, или 0, если такой строки нет. Подходит и имеет соответствующие оптимизации для определения того, выбрана ли строка *str* в столбце типа SET (см. приложение Г).

MAKE_SET(*bits, str1, str2, ...*)

Возвращает список строк в формате 'string1,string2,string3,...', используя строковые параметры (*str1*, *str2* и т. д.), соответствующие числу бит, заданному с помощью *bits*. Например, если *bits* равно 10 (1010 в двоичной системе), что соответствует битам с номерами 2 и 4, то MAKE_SET вернет 'str2,str4'.

EXPORT_SET(*bits, on_str, off_str*[, *separator*[, *number_of_bits*]])

Возвращает строковое представление бит, которые были (или не были) установлены с помощью значения *bits*. Установленные биты представлены строкой *on_str*, неустановленные — *off_str*. Значения разделяются посредством запятых, однако аргумент *separator* позволяет задать собственный разделитель. По умолчанию считываются порядка 64 бит, их количество можно уменьшить, используя параметр *number_of_bits*.

Например:

```
EXPORT_SET(10, 'Y', 'N', ',', 6) = 'N,Y,N,Y,N,N'
```

LCASE(*str*)
LOWER(*str*)

Возвращает строку *str*, в которой все символы переведены в нижний регистр.

UCASE(*str*)

UPPER(*str*)

Возвращает строку *str*, в которой все символы переведены в верхний регистр.

LOAD_FILE(*filename*)

Возвращает содержимое файла, указанного в аргументе *filename*, где прописывается полный путь к документу, который может быть прочитан MySQL-сервером. При этом ваша учетная запись должна предоставлять право для работы с файлами.

QUOTE(*str*)

Заключает строку *str* в одинарные кавычки и экранирует все содержащиеся в ней специальные символы с помощью обратной косой черты. Если *str* равно NULL, функция возвращает NULL (без кавычек).

Функции даты и времени

DAYOFWEEK(*date*)

Возвращает день недели для заданной даты *date* в виде целого числа и с учетом стандарта ODBC: 1 — воскресенье, 2 — понедельник, 3 — вторник... 7 — суббота.

WEEKDAY(*date*)

Возвращает день недели для заданной даты *date* в виде целого числа: 0 — понедельник, 1 — вторник, 2 — среда... 6 — воскресенье.

DAYOFMONTH(*date*)

Возвращает день месяца для заданной даты *date* в диапазоне от 1 до 31.

DAYOFYEAR(*date*)

Возвращает порядковый номер дня в году для даты date в диапазоне от 1 до 366 (помните о високосных годах).

MONTH(*date*)

Возвращает порядковый номер месяца для даты *date* в диапазоне от 1 (январь) до 12 (декабрь).

DAYNAME(*date*)

Возвращает название дня недели для даты *date*. Например, 'Вторник'.

MONTHNAME(*date*)

Возвращает название месяца для даты *date*. Например, 'Апрель'.

QUARTER(*date*)

Возвращает номер квартала года для даты *date*. Например, QUARTER ('2005-04-12')=2.

WEEK(*date*[, *mode*])

Возвращает порядковый номер недели в году для даты *date* в диапазоне от 0 до 53, где 1 — это первая неделя в году. При этом первым днем недели считается воскресенье.

Чтобы изменить способ вычисления данного значения, воспользуйтесь сведениями из табл. В.1.

Таблица В.1. Режимы вычисления порядкового номера недели

Режим	Первый день недели	Диапазон возвращаемых значений	Первая неделя
0	Воскресенье	от 0 до 53	Начинается в этом году
1	Понедельник	от 0 до 53	Имеет больше трех дней в этом году
2	Воскресенье	от 1 до 53	Начинается в этом году
3	Понедельник	от 1 до 53	Имеет больше трех дней в этом году
4	Воскресенье	от 0 до 53	Имеет больше трех дней в этом году
5	Понедельник	от 0 до 53	Начинается в этом году
6	Воскресенье	от 1 до 53	Имеет больше трех дней в этом году
7	Понедельник	от 1 до 53	Начинается в этом году

YEAR(*date*)

Возвращает год для даты *date* в диапазоне от 1000 до 9999.

YEARWEEK(*date*)

YEARWEEK(*date, first*)

Возвращает год и порядковый номер недели для даты *date* в виде 'YYYYWW'. Обратите внимание, что первые или последние несколько дней могут принадлежать неделе, которая относится к предыдущему или следующему году соответственно.

Например:

```
YEARWEEK("2006-12-31")=200701
```

HOUR(*time*)

Возвращает час для аргумента *time* в диапазоне от 0 до 23.

MINUTE(*time*)

Возвращает минуту для аргумента *time* в диапазоне от 0 до 59.

SECOND(*time*)

Возвращает секунду для аргумента *time* в диапазоне от 0 до 59.

PERIOD_ADD(*period, num_months*)

Добавляет к периоду *period* количество месяцев *num_months*, заданных в формате 'YYMM' или 'YYYYMM', и возвращает значение в виде 'YYYYMM'.

PERIOD_DIFF(*period1, period2*)

Возвращает количество месяцев между периодами *period1* и *period2*, каждый из которых задается в формате 'YYMM' или 'YYYYMM'.

DATE_ADD(*date*, INTERVAL *expr type*)
DATE_SUB(*date*, INTERVAL *expr type*)
ADDDATE(*date*, INTERVAL *expr type*)
SUBDATE(*date*, INTERVAL *expr type*)

Возвращает результат добавления к дате *date* или вычитания из нее указанного интервала времени *time* (значения DATE или DATETIME). Функции DATE_ADD и ADDDATE, а также DATE_SUB и SUBDATE идентичны. Выражение *expr* описывает интервал, который добавляется или вычитается. При желании вы можете задать ему отрицательное значение. Формат выражения *expr* определяется параметром *type* (табл. В.2).

Если аргументы *date* и *expr* содержат только дату, результат будет иметь тип DATE. В противном случае функция вернет значение DATETIME.

Рассмотрим работу функций этого семейства на примерах.

Представленные ниже выражения возвращают дату, которая наступит через шесть месяцев, начиная с текущего момента:

```
ADDDATE(CURDATE(), INTERVAL 6 MONTH)

DATE_ADD(CURDATE(), INTERVAL '0-6' YEAR_MONTH)
```

Следующие выражения возвращают текущее время, но на день позже.

```
ADDDATE(NOW(), INTERVAL 1 DAY)

SUBDATE(NOW(), INTERVAL -1 DAY)

DATE_ADD(NOW(), INTERVAL '24:0:0' HOUR_SECOND)

DATE_ADD(NOW(), INTERVAL '1 0:0' DAY_MINUTE)
```

Таблица В.2. Типы интервалов в функциях для арифметических операций с датами

Тип	Формат для expr
SECOND	количество секунд
MINUTE	количество минут
HOUR	количество часов

Тип	Формат для expr
DAY	количество дней
MONTH	количество месяцев
YEAR	количество лет
MINUTE_SECOND	'минуты:секунды'
HOUR_MINUTE	'часы:минуты'
DAY_HOUR	'дни часы'
YEAR_MONTH	'года-месяцы'
HOUR_SECOND	'часы:минуты:секунды'
DAY_MINUTE	'дни часы:минуты'
DAY_SECOND	'дни часы:минуты:секунды'

TO_DAYS(*date*)

Преобразует дату в количество дней, прошедших с нулевого года. Данная функция позволяет определять разницу между датами, то есть O_DAYS(*date1*) – TO_DAYS(*date2*) = количество дней между *date1* и *date2*.

FROM_DAYS(*days*)

Превращает в дату количество дней, прошедших с нулевого года. Результат аналогичен тому, что возвращает функция TO_DAYS.

DATE_FORMAT(*date, format*)

Принимает значение времени или даты *date* и форматирует его согласно строке *format*. Символы-заполнители, которые может содержать данная функция, представлены в табл. В.3.

Таблица В.3. Символы-заполнители для функции DATE_FORMAT (2004-01-01 01:00:00)

Символ	Значение	Пример
%M	Название месяца	Январь
%W	Название дня недели	Четверг
%D	День месяца с английским суффиксом	1st
%Y	Год (число, 4 разряда)	2004
%y	Год (число, 2 разряда)	03
%a	Сокращенное название дня недели	Чт
%d	День месяца	01
%e	День месяца	1
%m	Порядковый номер месяца в году	01
%c	Порядковый номер месяца в году	1
%b	Сокращенное название месяца	Янв
%j	День года	001
%H	Час (24-часовой формат, 00-23)	01

Продолжение таблицы В.3

Символ	Значение	Пример
%k	Час (24-часовой формат, 00-23)	1
%h	Час (12-часовой формат, 01-12)	01
%I	Час (12-часовой формат, 01-12)	01
%l	Час (12-часовой формат, 1-12)	1
%i	Минуты	00
%r	Время (12-часовой формат, hh:mm:ss AM/PM)	01:00:00 AM
%T	Время (24-часовой формат, hh:mm:ss)	01:00:00
%S	Секунды	00
%s	Секунды	00
%p	AM или PM	AM
%w	День недели, число (где 0 — воскресенье)	4
%U	Неделя (00–53, где первый день недели — воскресенье)	00
%u	Неделя (00–53, где первый день недели — понедельник)	01
%X	Год для недели, где первым днем недели считается воскресенье (число, 4 разряда, используется с %V)	2003
%V	Неделя (01-53, где первый день недели — воскресенье (%X))	53
%x	Аналогично %X, но первый день недели — понедельник (используется с %v)	2004
%v	Неделя (01-53, где первый день недели — понедельник (%x))	01
%%	Символ процента	%

TIME_FORMAT(*time*, *format*)

Аналогична DATE_FORMAT. Разница заключается лишь в том, что аргумент *format* содержит только те символы, которые приняты для обозначения часов, минут и секунд.

CURDATE()

CURRENT_DATE

Возвращает текущую системную дату в формате SQL — 'YYYY-MM-DD' (если используется как дата) или YYYYMMDD (если используется как число).

CURTIME()

CURRENT_TIME

CURRENT_TIME()

Возвращает текущее системное время в формате SQL— 'HH:MM:SS' (если используется как время) или HHMMSS (если используется как число).

NOW()

SYSDATE()

CURRENT_TIMESTAMP

CURRENT_TIMESTAMP()

LOCALTIME

LOCALTIME()

LOCALTIMESTAMP

LOCALTIMESTAMP()

Возвращает текущие системные дату и время в формате SQL — `'YYYY-MM-DD HH:MM:SS'` (если используется как дата/время) или `YYYYMMDDHHMMSS` (если используется как число).

UNIX_TIMESTAMP()

UNIX_TIMESTAMP(*date*)

Возвращает текущие системные дату/время либо указанные дату/время в виде количества секунд, прошедших с 1970-01-01 00:00:00 GMT.

FROM_UNIXTIME(*unix_timestamp*)

Аналогично `UNIX_TIMESTAMP`, только преобразует количество секунд, прошедших с 1970-01-01 00:00:00 GMT, в формат `'YYYY-MM-DD HH:MM:SS'` (если используется как дата/время) или `YYYYMMDDHHMMSS` (если используется как число). Время локальное.

FROM_UNIXTIME(*unix_timestamp, format*)

Форматирует UNIX-время в соответствии со строкой *format*, которая может содержать любой из символов-заполнителей, перечисленных в табл. В.3.

SEC_TO_TIME(*seconds*)

Преобразует указанное количество секунд в формат `'HH:MM:SS'` (если используется как время) или `HHMMSS` (если используется как число).

TIME_TO_SEC(*time*)

Преобразует время в формате `'HH:MM:SS'` в количество секунд.

Другие функции

DATABASE()

Возвращает имя текущей базы данных. Если в данный момент ни одна из них не выбрана, результатом станет пустая строка.

USER()

SYSTEM_USER()

SESSION_USER()

Возвращает имя текущего пользователя MySQL и имя клиентского компьютера, например `'kevin@localhost'`. Функция SUBSTRING_INDEX позволяет получить только имя пользователя.

```
SUBSTRING_INDEX(USER(), "@", 1) = 'kevin'
```

CURRENT_USER()

Возвращает запись о пользователе из системы контроля доступа MySQL в формате `'user@host'`, которая описывает права пользователя и с помощью которой произошла аутентификация текущего соединения. Как правило, возвращаемое значение совпадает с результатом выполнения функции USER. Однако, если запись в системе контроля доступа содержит метасимволы, функция CURRENT_USER может выдать меньше подробностей, например `'@%.mycompany.com'`.

PASSWORD(*str*)

Однонаправленная функция для шифрования паролей. Преобразует любой текст (обычно текстовый пароль) в зашифрованную строку длиной 16 символов. При этом каждая отдельная строка – постоянный набор символов. Полученное таким образом значение можно использовать для проверки правильности введенного пароля без необходимости хранить сам пароль в базе данных.

Механизм шифрования PASSWORD отличается от того, который используется в системе UNIX. Последний воспроизводится с помощью функции ENCRYPT.

ENCRYPT(*str*[, *salt*])

Шифрует строку *str* с помощью стандартного для UNIX механизма шифрования с использованием системного вызова `crypt()`. Аргумент *salt* является необязательным, но позволяет изменять начальное значение, на основе которого генерируется пароль. Если вы хотите, чтобы полученное значение совпадало с тем, которое хранится в системном файле с паролями, аргумент *salt* должен состоять из первых двух символов шифруемой строки. В зависимости от реализации функции `crypt()` в системе зашифрованное значение может основываться на первых восьми символах оригинальной строки.

В системах, где вызов `crypt()` недоступен, данная функция возвращает NULL.

ENCODE(*str*, *pass_str*)

Шифрует строку *str*, используя двунаправленный алгоритм и пароль *pass_str*. Для расшифровки полученного значения применяется функция DECODE.

DECODE(*crypt_str*, *pass_str*)

Расшифровывает строку crypt_str, используя двунаправленный алгоритм и пароль *pass_str*. Оригинальная строка будет восстановлена в том случае, если предоставлен тот же пароль, который применялся в функции ENCODE.

MD5(*string*)

Вычисляет контрольную сумму для строки *string* с помощью алгоритма MD5. Полученное значение представляет собой 32-разрядное шестнадцатеричное число. При этом контрольная сумма отдельно взятой строки является величиной постоянной. `MD5(NOW())` применяется, например, для получения псевдослучайных строк, используемых в качестве паролей по умолчанию.

LAST_INSERT_ID()

Возвращает последнее число, сгенерированное автоматически в рамках текущей сессии для столбца `AUTO_INSERT`.

FOUND_ROWS()

Иногда при использовании оператора `LIMIT` в запросе `SELECT` возникает необходимость узнать, сколько строк вернул бы запрос, если бы не было ограничений. Для этого следует вызвать две команды `SELECT`: первую с параметром `SQL_CALC_FOUND_ROWS` (см. приложение А), а вторую с вызовом функции `FOUND_ROWS`. При этом последняя срабатывает намного быстрее, чем тот же запрос без оператора `LIMIT`, поскольку отпадает необходимость возвращения всего результирующего набора на клиентскую сторону.

FORMAT(*expr*, *num*)

Меняет формат числа *expr*, оставляя столько десятичных знаков, сколько указано в аргументе *num*, и используя запятые в качестве разделителей групп разрядов. Округление выполняется до ближайшего значения, при необходимости в конце добавляются нули.

VERSION()

Возвращает версию MySQL-сервера, например `'5.1.34'`.

CONNECTION_ID()

Возвращает идентификатор потока для текущего соединения.

GET_LOCK(*str*, *timeout*)

Если нескольким клиентам необходимо синхронизировать выполнение задач, предусматривающих блокировку таблицы, они могут воспользоваться именными блокировками. Данная функция пытается получить блокировку по заданному имени *str*. Если блокировка уже используется другим клиентом, запрос подождет ее освобождения в течение *timeout* секунд, прежде чем прекратит попытку.

При этом клиент может освободить полученную блокировку с помощью функции `RELEASE_LOCK` или запросить новую, вызвав `GET_LOCK`. Если блокировка получена, `GET_LOCK` возвращает 1, если время ожидания *timeout* истекло — 0. В результате возникновения какой-либо ошибки функция примет значение `NULL`.

Функция GET_LOCK не является самостоятельной командой MySQL и используется как часть другого запроса.

Например:

```
SELECT GET_LOCK("mylock", 10)
```

RELEASE_LOCK(str)

Освобождает именную блокировку, полученную с помощью GET_LOCK. Возвращает 1, если блокировка освобождена, 0, если блокировка не принадлежит текущему потоку, и NULL, если она не существует.

IS_FREE_LOCK(*str*)

Проверяет, свободна ли именная блокировка и можно ли ее получить. Возвращает 1, если блокировка свободна, 0, если она уже кем-то используется, и NULL, если произошла ошибка.

BENCHMARK(*count, expr*)

Оценивает скорость выполнения выражения expr определенное количество раз, заданное с помощью аргумента *count*. Консольный клиент MySQL позволяет выводить время выполнения данной операции.

INET_NTOA(*expr*)

Возвращает IP-адрес, заданный в виде целого числа *expr*. Генерация числа осуществляется с помощью функции INET_ATON.

INET_ATON(*expr*)

Преобразует IP-адрес *expr* в числовой формат.

Например:

```
INET_ATON('64.39.28.1') = 64 * 2553 + 39 * 2552 + 28 * 255 + 1
                        = 1063751116
```

Функции, используемые в операторах GROUP BY

Функции, предназначенные для использования в рамках операторов GROUP BY, называют **групповыми**. Они генерируют значения на основе набора записей, каждая из которых включается в результирующий набор.

Используемые за пределами оператора GROUP BY, данные функции выводят результат в виде единой строки, значение которой формируется с учетом всех записей полноценного результирующего набора. Если такие функции смешать со столбцами, для которых не указаны групповые функции, произойдет ошибка, поскольку невозможно будет объединить их в единую строку, чтобы получить корректное значение.

COUNT(*expr*)

Возвращает число, равное количеству величин *expr* со значением не NULL в несгруппированном результирующем наборе. Запись COUNT(*) вернет количество строк в группе, независимо от того, содержат они значения NULL или нет.

COUNT(DISTINCT *expr*[, *expr* ...])

Возвращает количество различающихся величин *expr* со значением, не равным NULL (или набором значений, если задано несколько выражений).

AVG(*expr*)

Вычисляет среднее арифметическое значение выражения *expr*, которое встречается в строках данной группы.

MIN(*expr*)

MAX(*expr*)

Возвращают соответственно наименьшее и наибольшее значения выражения *expr* в строках группы.

SUM(*expr*)

Возвращает сумму значений для выражения *expr* в строках группы.

STD(*expr*)

STDDEV(*expr*)

Возвращают среднеквадратичное отклонение значения для выражения *expr* в строках группы.

BIT_OR(*expr*)

BIT_AND(*expr*)

Возвращают побитовые ИЛИ/И для значений выражения *expr* в строках группы.

Приложение Г
ТИПЫ СТОЛБЦОВ В MySQL

При создании таблицы в MySQL необходимо указать тип данных для каждого ее столбца. Данное приложение содержит все типы столбцов, которые поддерживаются MySQL версии 5.5.22 — последней на момент написания книги.

Многие из представленных типов столбцов могут иметь **необязательные параметры**, которые позволяют более подробно описать способ хранения и отображения данных. Прежде всего речь идет о параметрах *M* и *D*, которые указываются сразу после названия типа столбца. Квадратные скобки обозначают, что параметр является необязательным.

Параметр *M* определяет максимальное количество символов, используемых для отображения содержимого столбца, и в большинстве случаев сужает диапазон хранящихся в нем значений. Он может быть любым целым числом от 1 до 255. При этом, если столбец имеет тип `INT`, параметр *M* не влияет на значения, которые столбец содержит: чтобы при отображении на экране данные имели нужную длину, к ним добавляются пробелы или нули, если речь идет о столбце `ZEROFILL` (более подробно об этом рассказано далее). Тем не менее по возможности не сохраняйте значения, длина которых превышает число, указанное в параметре *M*, поскольку это может вызвать проблемы во время сложных объединений.

Параметр *D* задает количество разрядов, которые отводятся для хранения дробной части чисел с плавающей точкой. Максимальное значение данного параметра равно 30 и не должно превышать значение, указанное в параметре *M*. Таким образом, *D* всегда меньше или равен *M*-2. Это нужно для того, чтобы вместить ноль и десятичный разделитель.

Еще один вид параметров — необязательные **атрибуты столбцов**. Ниже перечислены атрибуты, которые поддерживаются разными типами столбцов, они указываются после типа и разделяются пробелами.

- `ZEROFILL`. Длина выводимых значений в столбце является максимальной. При этом недостающие разряды заполняются нолями. К этому параметру автоматически добавляется параметр `UNSIGNED`.
- `UNSIGNED`. Столбец принимает только положительные числовые значения или ноль. Такое ограничение освобождает место для чисел больше ноля, удваивая диапазон допустимых значений. Его устанавливают в том случае, когда есть уверенность в том, что в столбце не будет отрицательных чисел.
- `BINARY`. По умолчанию сравнение символьных значений в MySQL, включая сортировку, выполняется с учетом регистра, однако в столбцах с параметром `BINARY` регистр не учитывается.

Полную и наиболее актуальную информацию о поддерживаемых типах столбцов вы найдете в справочном руководстве по MySQL (http://dev.mysql.com/doc/mysql/en/data-types.html).

Числовые типы

`TINYINT[(M)]`

Описание: очень малое целое число.

Допустимые атрибуты: `UNSIGNED`, `ZEROFILL`.

Диапазон значений: от –128 до 127 (от 0 до 255, если `UNSIGNED`).

Объем занимаемой памяти: 1 байт (8 бит).

`SMALLINT[(M)]`

Описание: малое целое число.

Допустимые атрибуты: `UNSIGNED`, `ZEROFILL`.

Диапазон значений: от –32 768 до 32 767 (от 0 до 65 535, если `UNSIGNED`).

Объем занимаемой памяти: 2 байта (16 бит).

`MEDIUMINT[(M)]`

Описание: целое число среднего размера.

Допустимые атрибуты: `UNSIGNED`, `ZEROFILL`.

Диапазон значений: от –8 588 608 до 8 388 607 (от 0 до 16 777 215, если `UNSIGNED`).

Объем занимаемой памяти: 3 байта (24 бита).

`INT[(M)]`

Описание: обычное целое число.

Допустимые атрибуты: `UNSIGNED`, `ZEROFILL`.

Диапазон значений: от –2 147 483 648 до 2 147 483 647 (от 0 до 4 294 967 295, если `UNSIGNED`).

Объем занимаемой памяти: 4 байта (32 бита).

Альтернативный синтаксис: `INTEGER[(M)]`.

`BIGINT[(M)]`

Описание: большое целое число.

Допустимые атрибуты: `UNSIGNED`, `ZEROFILL`.

Диапазон значений: от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 (от 0 до 18 446 744 073 709 551 615, если `UNSIGNED`).

Объем занимаемой памяти: 8 байт (64 бита).

Примечание. MySQL выполняет все арифметические функции с целыми числами в формате BIGINT. Таким образом, значения типа BIGINT UNSIGNED, которые превышают 9 223 372 036 854 775 807 (63 бита), работают корректно только с булевыми функциями, например с побитовыми AND, OR и NOT. Использование подобных значений в арифметических операциях с целыми числами может привести к неточностям в результатах, возникающих из-за ошибок, связанных с округлением.

FLOAT[(*M*, *D*)]

FLOAT(*precision*)

Описание: число с плавающей точкой.

Допустимые атрибуты: ZEROFILL.

Диапазон значений: 0 и от ±1,175494351E–38 до ±3,402823466E+38.

Объем занимаемой памяти: 4 байта (32 бита).

Примечание. Аргумент *precision* является необязательным и не должен превышать 24 бит, иначе будет создан столбец типа DOUBLE.

DOUBLE[(*M*, *D*)], DOUBLE(*precision*)

Описание: число с плавающей точкой повышенной точности.

Допустимые атрибуты: ZEROFILL.

Диапазон значений: 0 и от ±2,2250738585072014–308 до ±1,7976931348623157E+308.

Объем занимаемой памяти: 8 байт (64 бита).

Примечание. Аргумент *precision* является необязательным и не должен быть меньше 25 бит, иначе будет создан столбец типа FLOAT (см. выше). Максимальное значение *precision* равно 53.

Альтернативный синтаксис: DOUBLE PRECISION[(*M*,*D*)] или REAL[(*M*,*D*)].

DECIMAL[(*M*[, *D*])]

Описание: число с плавающей точкой для хранения символьных строк.

Допустимые атрибуты: ZEROFILL.

Диапазон значений: аналогичен DOUBLE, но ограничен значениями *M* и *D* (см. примечание).

Объем занимаемой памяти. Зависит от сохраняемого значения. Если оно имеет *X* разрядов перед запятой и *Y* разрядов после, то используется приблизительно $(X + Y) \times 4 \div 10$ байт.

Примечание. Если параметр *D* не указан, подразумевается, что его значение равно нолю, а числа в данном столбце не имеют десятичного разделителя или дробной части. По умолчанию параметр *M* принимает значение, равное 10.

Альтернативный синтаксис: NUMERIC([*M*[,*D*]]).

BIT (*M*)

Описание. Бинарное значение размером *M* бит, где *M* находится в диапазоне от 1 до 64. Другими словами, это последовательность цифр длиною *M*, где каждая из цифр равна 0 или 1.

Диапазон значений: ограничен величиной *M*.

Объем занимаемой памяти: *M* + 2 байта (8 × *M* + 16 бит).

Примечание. Предназначен для хранения последовательности булевых флагов (`true` или `false`). Значения типа `BIT` записываются следующим образом: `b'ddd...'`, где `d` — это 1 `(true)` или 0 `(false)`. Например, у восьмибитного бинарного значения все флаги равны true: `b'11111111'`.

Строковые типы

CHAR (*M*)

Описание: строка фиксированной длины.

Допустимые атрибуты: `BINARY`.

Занимаемое место: *M* байт (8 × *M* бит).

Примечание. Содержимое столбца типа `CHAR` хранится в виде строки длиной *M*, при этом присваиваемые значения могут быть короче. Конец строки, которая меньше указанной для поля длины, дополняется пробелами, чтобы длина строки соответствовала размеру, заданному в *M*. При извлечении значения пробелы удаляются.

В столбцах типа `CHAR` поиск осуществляется гораздо быстрее, чем в столбцах со значениями переменной длины, таких как `VARCHAR`. Заранее определенный размер значений делает файлы базы данных более однородными.

M принимает любое целочисленное значение от 0 до 255. Столбец типа `CHAR(0)` занимает 1 бит и может хранить `NULL` либо пустую строку (`''`).

Альтернативный синтаксис: `CHARACTER(M)`.

VARCHAR (*M*)

Описание: строка переменной длины.

Допустимые атрибуты: `BINARY`.

Максимальная длина: *M* символов.

Объем занимаемой памяти: размер хранимого значения плюс 1 байт для записи его длины (2 байта, если *M* > 255)

Примечание. Значения типа `VARCHAR` занимают столько места, сколько им необходимо, поэтому обычно нет смысла задавать максимальную длину поля *M* меньше 255 — максимального размера для версий MySQL, предшествовавших 5.0.3. Начиная с данной версии, MySQL позволяет задавать максимальную длину вплоть до 65 535 символов. Однако следует учесть, что это значение является максимальным количеством байтов, которое может содержаться в одной строке таблицы,

поэтому на практике данные границы лучше уменьшать. Как вариант, можно указать более подходящий тип столбца, например TEXT.

Строки, которые выходят за рамки указанного значения, при добавлении сокращаются до максимально возможного. В версиях MySQL, предшествующих 5.0.3, перед сохранением в конце строки удалялись все пробелы, однако предсказать результаты подобной операции было невозможно. Начиная с версии 5.0.3, значения сохраняются в том виде, в котором они передаются.

Альтернативный синтаксис: CHARACTER VARYING(*M*).

BINARY(*M*)

Описание: бинарная строка фиксированной длины.

Максимальная длина: *M* символов.

Объем занимаемой памяти: *M* байт (8 × *M* бит).

Примечание. Отличается от CHAR только тем, что MySQL обращается со значениями, хранящимися в таком столбце, как с нетекстовой байтовой строкой.

VARBINARY(*M*)

Описание: бинарная строка переменной длины.

Максимальная длина: *M* символов.

Объем занимаемой памяти: размер хранимого значения плюс 1 байт для записи его длины.

Примечание. Отличается от VARCHAR только тем, что MySQL обращается со значениями, хранящимися в таком столбце, как с нетекстовой байтовой строкой.

TINYBLOB

TINYTEXT

Описание: короткая строка переменной длины.

Максимальная длина: 255 символов.

Объем занимаемой памяти: размер хранимого значения плюс 1 байт для записи его длины

Примечание. Эквивалентны VARCHAR(255) BINARY и VARCHAR(255) соответственно. Однако при добавлении значений данные типы не удаляют пробелы, идущие в конце. Единственная разница между типами TINYBLOB и TINYTEXT заключается в том, что при сравнении значений первый тип учитывает регистр, а второй нет.

BLOB

TEXT

Описание: строка переменной длины.

Максимальная длина: 65 535 символов (65 Кбайт).

Объем занимаемой памяти: размер хранимого значения плюс 2 байта для записи его длины.

Примечание. Единственная разница между типами BLOB и TEXT заключается в том, что при сравнении значений первый тип учитывает регистр, а второй нет.

MEDIUMBLOB

MEDIUMTEXT

Описание: строка переменной длины средних размеров.

Максимальная длина: 16 777,215 символов (16,8 Мбайта).

Объем занимаемой памяти: размер хранимого значения плюс 3 байта для записи его длины.

Примечание. Единственная разница между типами MEDIUMBLOB и MEDIUMTEXT заключается в том, что при сравнении значений первый тип учитывает регистр, а второй нет.

LONGBLOB

LONGTEXT

Описание: строка переменной длины большого размера.

Максимальная длина: 4 294 967 295 символов (4,3 Гбайта).

Объем занимаемой памяти: размер хранимого значения плюс 4 байта для записи его длины.

Примечание. Единственная разница между типами LONGBLOB и LONGTEXT заключается в том, что при сравнении значений первый тип учитывает регистр, а второй нет.

ENUM(*value1, value2, ...*)

Описание: набор значений, где для каждой отдельной строки можно выбрать только одно из указанных.

Максимальная длина: не более 65 535 значений.

Объем занимаемой памяти: от 1 до 255 значений — 1 байт (8 бит); от 256 до 65 535 значений — 2 байта (16 бит).

Примечания. В полях данного типа значения хранятся в виде целых чисел, которые соответствуют выбранному элементу: 1 — первый элемент, 2 — второй и т. д. Если присваиваемое значение при объявлении столбца не указано, для обозначения пустой строки (`''`) используется ноль.

Столбцы данного типа, помеченные как NOT NULL, по умолчанию содержат первый объявленный элемент, если для этого предусмотрено иное значение.

SET(*value1, value2, ...*)

Описание: набор значений, каждое из которых может быть выбрано (допускается вариант, когда не выбрано ни одно значение).

Максимальная длина: не более 64 значений.

Объем занимаемой памяти: от 1 до 8 значений — 1 байт (8 бит); от 9 до 16 значений — 2 байта (16 бит); от 17 до 24 значений — 3 байта (24 бита); от 25 до 32 значений — 4 байта (32 бита); от 33 до 64 значений — 8 байт (64 бита).

Примечание. В полях данного типа значения хранятся в виде целых чисел, которые представляют собой битовую маску для установленных и неустановленных элементов. Предположим, что набор содержит восемь значений и в конкретной строке заданы нечетные числа. Тогда двоичная величина 01010101 будет представлена в виде десятичного значения 85. Значения присваиваются как в виде целых чисел, так и в виде строки, содержащей набор элементов, разделенных запятыми. Например, `'значение1,значение3,значение5,значение7'=85`. Поиск по содержимому столбца выполняется с помощью оператора `LIKE` или с использованием функции `FIND_IN_SET`.

Типы данных даты и времени

`DATE`

Описание: дата.

Диапазон значений: от `'1000-01-01'` до `'9999-12-31'`, а также `'0000-00-00'`.

Объем занимаемой памяти: 3 байта (24 бита).

`TIME`

Описание: время.

Диапазон значений: от `'-838:59:59'` до `'838:59:59'`.

Объем занимаемой памяти: 3 байта (24 бита).

`DATETIME`

Описание: дата и время.

Диапазон значений: от `'1000-01-01 00:00:00'` до `'9999-12-31 23:59:59'`.

Объем занимаемой памяти: 8 байт (64 бита).

`YEAR`

Описание: год.

Диапазон значений: от 1901 до 2155, а также 0000.

Объем занимаемой памяти: 1 байт (8 бит).

Примечание. Год указывается в виде четырехзначного числа (от 1901 до 2155 или 0000), четырехсимвольной строки (от `'1901'` до `'2155'` или `'0000'`), двухзначного числа (от 70 до 99 для периода с 1970 по 1999, от 1 до 69 для периода с 2001 по 2069 и 0 для нулевого значения) или двухсимвольной строки (от `'70'` до `'99'` для периода с 1970 по 1999 и от `'00'` до `'69'` для периода с 2000 по 2069).

При этом 2000-й год нельзя указать в виде двухзначного числа, а нулевой — в виде двухсимвольной строки. Все некорректные значения преобразуются в 0.

`TIMESTAMP [ (M) ]`

Описание: временная отметка (дата/время) в формате `YYYYMMDDHHMMSS`.

Диапазон значений: от 19700101000000 до определенной даты в 2037 году (в современных системах).

Объем занимаемой памяти: 4 байта (32 бита).

Примечание. При добавлении или обновлении строки (операции `INSERT` и `UPDATE` соответственно), которая содержит один или несколько столбцов типа `TIMESTAMP`, столбцу, указанному при объявлении первым, автоматически присваиваются текущие дата/время. В таком столбце удобно хранить информацию о дате/времени последнего изменения строки. Аналогичного результата можно добиться, если присвоить столбцу значение `NULL` — происходит своеобразная фиксация времени доступа. Однако, как и в случае с любыми другими столбцами, вы можете сами выбирать, какие значения в данном столбце следует хранить.

Параметр *M* принимает значения 14, 12, 10, 8, 6, 4 и 2, что соответствует форматам вывода `YYYYMMDDHHMMSS`, `YYMMDDHHMMSS`, `YYMMDDHHMM`, `YYYYMMDD`, `YYMMDD`, `YYMM` и `YY`. Нечетные числа от 1 до 13 автоматически приводятся к следующему четному значению. Ноль и все числа больше 14 по умолчанию приводятся к 14.

Алфавитный указатель

B

C

D

Научно-популярное издание

МИРОВОЙ КОМПЬЮТЕРНЫЙ БЕСТСЕЛЛЕР

Кевин Янк

PHP и MySQL
От новичка к профессионалу
(орыс тілінде)

Директор редакции *Е. Капьёв*
Ответственный редактор *В. Обручев*
Художественный редактор *Е. Мишина*

ООО «Издательство «Эксмо»
123308, Москва, ул. Зорге, д. 1. Тел. 8 (495) 411-68-86, 8 (495) 956-39-21.
Home page: **www.eksmo.ru** E-mail: **info@eksmo.ru**

Өндіруші: «ЭКСМО» АҚБ Баспасы, 123308, Мәскеу, Ресей, Зорге көшесі, 1 үй.
Тел. 8 (495) 411-68-86, 8 (495) 956-39-21
Home page: www.eksmo.ru E-mail: info@eksmo.ru.
Тауар белгісі: «Эксмо»
Қазақстан Республикасында дистрибьютор және өнім бойынша
арыз-талаптарды қабылдаушының
өкілі «РДЦ-Алматы» ЖШС, Алматы қ., Домбровский көш., 3«а», литер Б, офис 1.
Тел.: 8 (727) 2 51 59 89,90,91,92, факс: 8 (727) 251 58 12 вн. 107; E-mail: RDC-Almaty@eksmo.kz
Өнімнің жарамдылық мерзімі шектелмеген.
Сертификация туралы ақпарат сайтта: www.eksmo.ru/certification

Сведения о подтверждении соответствия издания согласно законодательству РФ
о техническом регулировании можно получить по адресу: http://eksmo.ru/certification/

Өндірген мемлекет: Ресей
Сертификация қарастырылмаған

Подписано в печать 10.10.2013. Формат 70x100$^{1}/_{16}$.
Печать офсетная. Усл. печ. л. 31,11.
Тираж 2000 экз. Заказ 8413.

Отпечатано с готовых файлов заказчика
в ОАО «Первая Образцовая типография»,
филиал «УЛЬЯНОВСКИЙ ДОМ ПЕЧАТИ»
432980, г. Ульяновск, ул. Гончарова, 14

ISBN 978-5-699-67363-6