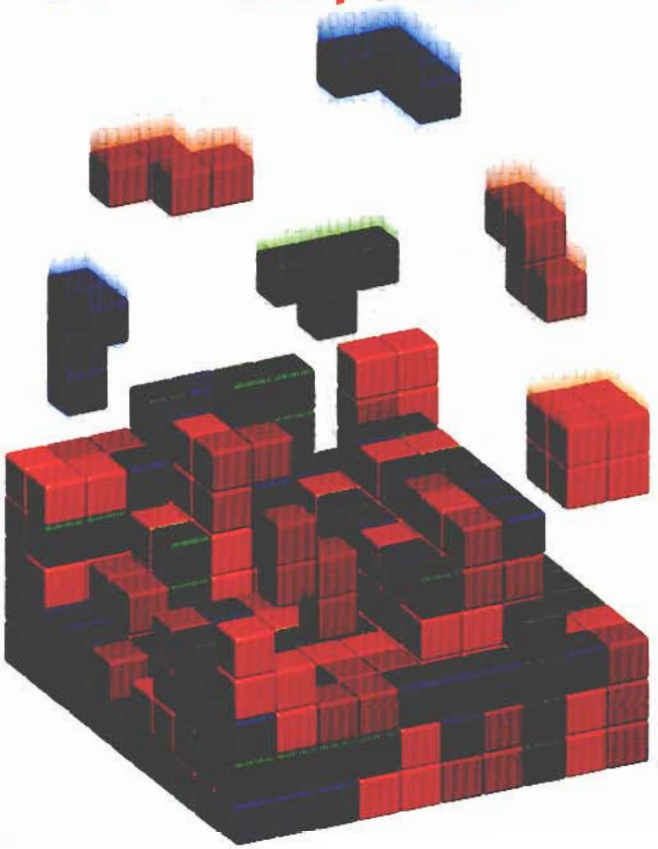
phpclub
http://phpclub.ru
рекомендует!

ЭНДИ ХАРРИС

PHP/MySQL для начинающих

ОПЫТА НЕ ТРЕБУЕТСЯ



КУДИЦ-ОБРАЗ

Посвящается Хизер, Элизабет, Мэтью и Джейкобу

Andy Harris

PHP/MySQL Programming

for the absolute beginner

Энди Харрис

PHP/MySQL

для начинающих

Перевод с английского

КУДИЦ-ОБРАЗ
Москва • 2005

ББК 32.973.26–018.2

Харрис Э.

PHP/MySQL для начинающих. / Пер. с англ. – М.: КУДИЦ-ОБРАЗ, 2005. – 384 с.

Вы держите в руках книгу, которая поможет вам научиться программировать на PHP и создвать базы данных на MySQL. Если вашей целью является написание программ для веб-серверов, то эта книга для вас. Вы узнаете все основные концепции языков программирования, в частности, изучите команды и синтаксис языка PHP. Вы также узнаете, как в современных средах используются данные, кроме всего этого, также обучитесь самому процессу программирования.

PHP – это мощный язык программирования, который позволяет создавать динамические веб-сайты. Он хорошо работает на разнообразных платформах и весьма доступен для понимания. MySQL является впечатляющей системой управления реляционными данными, используемой для создания высококачественных коммерческих баз данных. PHP и MySQL являются настолько мощными и простыми в использовании платформами, что позволяет заниматься веб-программированием даже новичкам.

ISBN 5-9579-0046-X
ISBN 1-931841-32-2

Энди Харрис
PHP/MySQL для начинающих

Учебно-справочное издание

Перевод с англ. В. Ю. Щербаков
Научный редактор Е. В. Петрова
ООО "ИД КУДИЦ-ОБРАЗ".
119034, Москва, Ленинский. Тел.: 333-82-11, ok@kudits.ru
Подписано в печать 23.09.2004.
Формат 70x90/16. Бум. газ. Печать офс
Усл. печ. л. 28,6. Тираж 2000. Заказ 2050

Отпечатано в ОАО "Щербинская типография"
117623, Москва, ул. Типографская, д. 10

ISBN 1-931841-32-2
ISBN 5-9579-0046-X

Краткое содержание

Благодарности

Прежде всего, Ему, благодаря кому мы все существуем.

Хизер, ты всегда работаешь над этими книгами больше, чем я. Спасибо тебе за твою любовь и поддержку. Спасибо вам, Элизабет, Мэтью и Джейкоб, за понимание, почему папа все время что-то печатал.

Спасибо сообществу Open Source за создание такого замечательного бесплатного программного обеспечения, как PHP и MySQL.

Спасибо Стейси Хикет (Stacy Hiquet) за постоянную поддержку и ободрение в этом и других проектах.

Спасибо Тоду Йенсену (Todd Jensen) за то, что соединил эту книгу.

Отдельное спасибо Сэнди Доэл (Sandy Doell) за превращение моих рукописей во что-то читаемое.

Спасибо J Wynia (www.phpgeek.com) за техническое редактирование. Также спасибо Джейсону (Jason).

Спасибо команде разработчиков webyog (http://www.webyog.com/sqlyog/) за использование инструмента SQLyog.

Большое спасибо Кейт Девенпорт (Keith Davenport).

Спасибо множеству членов команды Premier/Course, кто работал над этой книгой.

Огромное спасибо моей группе CSCI N399 Server Side Web Development, обучавшейся весной 2003 года. Спасибо вам за терпеливую работу с моей рукописью, за помощь в выявлении множества ошибок и за предоставление бесценных советов. Я научился у вас столькому же, сколькому и вы у меня.

Об авторе

Энди Харрис начал преподавательскую карьеру в качестве преподавателя в средней школе. В то время он в достаточной мере изучил вычислительную технику, чтобы заниматься компьютерным консультированием и работой с базами данных. Он начал преподавание программирования на университетском уровне в конце 80-х в качестве дополнительной работы. С 1995 он работает лектором на факультете Вычислительной техники (Computer Science Department) университетов Indiana и Purdue в штате Indianapolis, где он заведует лабораторией Потокового мультимедиа (Streaming Media Lab) и читает лекции по нескольким языкам программирования. Его основными увлечениями являются Java, языки Microsoft, Perl, JavaScript, PHP, Web Data, виртуальная реальность, переносные устройства и потоковое мультимедиа.

Предисловие

Если вы следили некоторое время за Интернетом, вы, вероятно, заметили, что он изменяется. Когда Интернет впервые вошел в общественное сознание, он был способом размещения документов. Эти документы было достаточно легко создавать. Любой за выходные мог с помощью текстового редактора создать веб-страницу. Вначале создание веб-сайтов заключалось в создании таких документов.

В наши дни Интернет является намного более сложной структурой, чем тогда. Интересные сайты не являются просто *документами*; они являются *приложениями*. Они имеют намного больше гибкости и мощности. Вам может показаться, что теперь Интернет не место для отдельного человека или начинающего программиста. Многие инструменты для разработки приложений дороги и сложны.

Для меня наиболее замечательной стороной Интернета является его социальная часть. Существует огромное сообщество, которое верит в возможность существования мощного, легко используемого бесплатного программного обеспечения. Это сообщество создало большое количество отличных программ, таких, как PHP и MySQL.

PHP – это мощный язык программирования, который позволяет создавать динамические веб-сайты. Он хорошо работает на разнообразных платформах и его легко понять. MySQL является впечатляющей системой управления реляционными данными, используемой для создания высококачественных коммерческих баз данных. PHP и MySQL являются настолько мощными и простыми в использовании платформами, что они позволяют заниматься веб-программированием даже новичкам.

В этой книге я научу вас программировать. В частности, вы научитесь создавать программы для веб-серверов. Вы узнаете все основные концепции языков программирования. Вы также узнаете, как в современных средах используются данные. Вы изучите команды и синтаксис, однако, кроме этого, вы также обучитесь процессу программирования.

Если вы до этого не создавали компьютерных программ, эта книга будет хорошим введением. Если вы опытный программист и хотите изучить PHP и MySQL, вы обнаружите, что эта книга является хорошим введением.

Программирование является тяжелой работой, однако оно также приносит много веселья. Я замечательно провел время при написании этой книги, и я надеюсь, что вам понравится ее чтение. Мне не терпится услышать о том, что вы сможете сделать после прочтения этой книги.

– Энди

От издательства: О программных примерах

Коды всех приведенных в книге примеров выложены на сайте издательства, (http://books.kudits.ru) в разделе "Дополнительные материалы", в том виде как они даны в оригинале.

Специалисты PHPClub (http://phpclub.ru/) испытывают определенные сомнения по поводу работы всех оригинальных кодов (в частности из-за опции register_globals = off), и поэтому приглашают читателей данной книги на свой сайт, где будет открыт форум для обсуждения этой и других книг по PHP.

Глава 1

Знакомимся с PHP

Веб-страницы имеют широкие функциональные возможности и часто бывают интересны по содержанию, но сами по себе они являются простыми документами. Однако их возможности можно расширить, и поможет вам в этом язык PHP. С помощью языка сценариев, такого, как PHP, любой веб-сайт можно превратить из набора статических документов в настоящее интерактивное приложение. В этой главе вы научитесь использовать при создании своих веб-страниц простейшие функции PHP. Вы:

- вспомните команды языка HTML;
- научитесь использовать каскадные таблицы стилей (CSS – Cascading Style Sheets);
- создадите несколько HTML-форм;
- удостоверитесь в том, что язык PHP установлен на вашем компьютере;
- проведете короткое исследование конфигурации установленного у вас PHP;
- и наконец, добавите к своей веб-странице код, написанный на PHP.

Знакомьтесь: программа «Совет дня»

Ваша первая программа вряд ли выиграет какую-нибудь Интернет-награду, но с ее помощью вы сразу же шагнете за пределы функциональности, доступной для обычных HTML-страниц. На рис. 1.1 показана страница «Совет дня», предлагающая дружескую и временами такую необходимую подсказку.

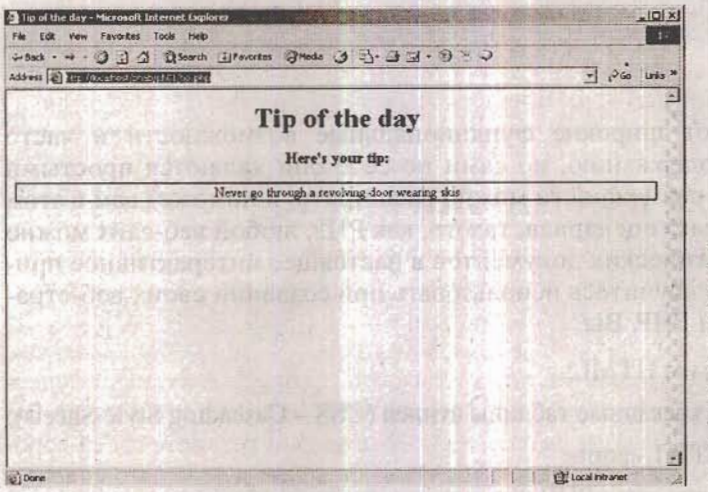


Рис. 1.1. На первый взгляд кажется, что это очень простая страница, но на самом деле она является настоящим чудом современных веб-технологий: страница создана с помощью HTML, CSS и PHP

Конечно, подобную страницу можно создать и не используя язык PHP, но наша программа изнутри немного сложнее, чем может показаться на первый взгляд. Текст подсказок на самом деле хранится не в теле веб-страницы, а в отдельном файле. Программа вставляет текст из этого файла прямо в HTML-страницу. Поэтому для изменения текста подсказок владельцу страницы нужно будет всего лишь отредактировать обычный текстовый файл, в котором они хранятся.

Начнем мы с того, что освежим в памяти навыки создания HTML-страниц. Очень скоро вам придется писать программы, которые будут создавать веб-страницы, поэтому вы должны уверенно владеть языком HTML. Если вы обычно пишете HTML-код в стандартном текстовом редакторе, то проблем у вас не возникнет. Если же вы предпочитаете специализированные редакторы, вроде Microsoft FrontPage или Macromedia Dreamweaver, будет лучше на время отказаться от их использования и немного потренироваться, создавая HTML-страницы вручную.

ВОЗВРАЩАЯСЬ К РЕАЛЬНОСТИ

Наша страница «Совет дня» иллюстрирует одну из важнейших концепций в современном веб-программировании – концепцию системы управления содержимым (CMS – content management system). Согласно этой концепции, содержимое веб-страниц отделяется от их дизайна, так что программисты разрабатывают только внешний вид страниц, а их владельцы могут просто менять текст в обычном текстовом файле, не подвергая риску программный код сайта. При этом владельцы могут и не знать, как вносить изменения непосредственно на страницу. Эта книга поможет вам научиться создавать мощные системы управления содержимым и многое другое

Программируем на стороне веб-сервера

Интернет построен на том, что множество компьютеров обмениваются данными друг с другом. И чаще всего их общение организовано по модели «клиент-сервер». Для того чтобы лучше в ней разобраться, давайте представим себе обычное кафе, где можно купить еду, не выходя из машины. Сначала вы подъезжаете к небольшому динамику, и едва различимый голос спрашивает ваш заказ. Вы хотите купить «Супергамбургер», и где-то внутри кафе скучающий подросток укладывает его в пакет. Вы проезжаете вперед, отдаете на выезде деньги, получаете заказ и уезжаете своей дорогой. А тем временем подросток, который вас обслуживал, ждет следующего клиента. Все это очень похоже на то, как устроен Интернет. Веб-страницы и другие полезные данные размещаются на мощных, постоянно работающих компьютерах, которые называются веб-серверами. Эти серверы очень напоминают наше с вами кафе. Посетители «приезжают» на веб-сервер с помощью браузера. Происходит обмен данными, и пользователь видит в окне браузера полученную информацию.

Но особенно интересен в этой модели тот факт, что взаимодействие между компьютерами на этом вовсе не заканчивается. Команды ведь можно давать и клиентскому компьютеру. Обычно для их включения в веб-страницу используют язык JavaScript. Для сервера такие команды (как и собственно HTML-код) не имеют никакого значения. Но как только страница попадает на машину-клиент, браузер интерпретирует HTML-код, а вместе с ним и команды языка JavaScript. И хотя основную работу в этом случае выполняет клиент, при таком подходе существуют свои недостатки. Программы, рассчитанные на работу в окне веб-браузера, обычно сильно ограничены в своих возможностях. Они, как правило, не могут получить доступ к принтеру или дискам на компьютере пользователя. Уже одно это ограничение не позволяет использовать эти программы для выполнения самых полезных функций Интернета, таких, как подключение к удаленным базам данных или отслеживание пользователей.

Сервер – это тоже компьютер, а значит, можно написать программу, которая будет исполняться не на клиенте, а на сервере. В таком подходе есть несколько плюсов.

- Эти программы будут выполняться на мощных компьютерах веб-серверов.
- Сервер может свободно работать с файлами и базами данных.
- В результате работы такой программы пользователь получает простую HTML-страницу, отобразить которую способен любой браузер.

Создаем простые HTML-страницы

Основная единица веб-разработки – это HTML-страница. Она представляет собой простой текстовый документ, в который включены специальные элементы – теги, описывающие данные, которые содержатся на странице. Даже если вы уже знакомы с HTML, имеет смысл освежить ваши навыки, поскольку программирование на PHP неразрывно связано с HTML.

Я настоятельно рекомендую с самого начала пользоваться обычным текстовым редактором. Например, это может быть «Блокнот» (Notepad). Текстовые процессоры же, как правило, не сохраняют свои файлы в формате «только текст» (что необходимо для работы с PHP и HTML), а специальные веб-редакторы (вроде FrontPage или Dreamweaver) создают весьма неуклюжий HTML-код, который станет серьезной помехой, как только вы захотите добавить к своей странице программный код.

Создаем на HTML страничку приветствия

HTML – это прежде всего текст. Автор страницы добавляет к текстовому документу специальную разметку, которая определяет значение различных элементов. Когда пользователь запрашивает веб-страницу, текстовый документ поступает с сервера в браузер, и тот интерпретирует теги, определяя, как страница должна быть показана на экране. На рис. 1.2 показана очень простая веб-страница.

Посмотрев на код этой страницы, легко убедиться, что понять его очень просто даже людям, не слишком знакомым с HTML.

```
<html>
<head>
<title>Hello, World</title>
</head>
<body>
<center>
<h1>Hello, World!</h1>
</center>
</body>
</html>
```

Рис. 1.2. Самая простая веб-страница

Как вы, наверное, заметили, многие слова в тексте заключены в угловые скобки (< >). Эти слова называются тегами, и при обработке страницы браузер интерпретирует их как специальные команды. Большинство тегов имеют пару. К примеру, весь документ начинается с тега `<html>` и оканчивается тегом `</html>`. Косая черта (/) указывает на то, что это – закрывающий тег.

Каждый HTML-документ имеет заголовок, который ограничивается парой тегов `<head></head>`. В заголовке содержится информация о документе в целом. В него почти всегда включается название документа, которое часто отображается в строке заголовка окна браузера. Однако никаких обязательных требований на этот счет не существует. HTML-теги описывают значение элемента, и совсем не обязательно – его внешний вид. Каждый браузер самостоятельно устанавливает, как должен выглядеть тот или иной элемент.

Основная часть HTML-документа – это его тело, заключенное между тегами `<body></body>`.

В теле документа при помощи тегов задаются самые разнообразные характеристики страницы. Как правило, по названию тега можно догадаться о его назначении. Например, текст, заключенный между парой тегов `<center></center>`, будет размещен по центру (если браузер поддерживает такую возможность).

Важно осознавать, что теги HTML являются не столько командами, сколько указаниями для браузера. Существует огромное количество компьютеров и браузеров. А потому всегда есть вероятность, что кто-то станет просматривать вашу веб-страницу с карманного компьютера или мобильного телефона. Эти устройства не могут отображать данные в том же виде, в котором они показаны на обычных компьютерах. Браузер постарается выполнить все ваши инструкции, но в конечном итоге вы не можете контролировать то, как будет выглядеть страница у каждого пользователя

Теги `<h1></h1>` определяют текст, помещенный между ними, как заголовок первого (самого высокого) уровня. В HTML предусмотрено шесть уровней заголовков: от `<h1>` до `<h6>`. Невозможно точно сказать, как именно эти заголовки будут выглядеть в окне браузера, однако любой текст, расположенный внутри пары тегов `<h1>`, будет выделен очень сильно, а каждый следующий уровень заголовков будет выделяться немного слабее, чем предыдущий.

Основные теги

В HTML существует довольно много тегов. Большинство из них используются для того, чтобы определить значение какого-либо отдельного куска текста. В таблице 1.1 приводятся примеры некоторых таких тегов.

Конечно же, существует и множество других HTML-тегов, но в таблице. 1.1 представлены наиболее часто используемые. На рис. 1.3 показано применение нескольких тегов из перечисленых в таблице 1.1.

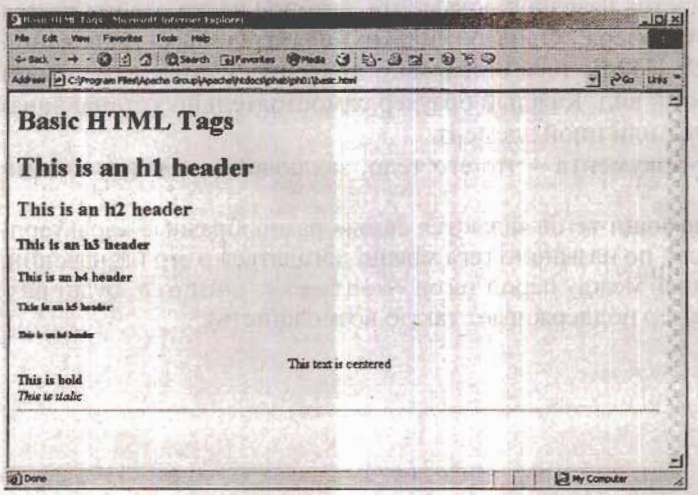


Рис. 1.3. Страница с примерами наиболее часто применяемых HTML-тегов

Табл. 1.1. Основные теги языка HTML

Тег	Значение	Комментарий
`<b></b>`	Жирный шрифт	Может работать не во всех браузерах
`<i></i>`	Курсив	Может работать не во всех браузерах
`<h1></h1>`	Заголовок 1-го уровня	Самые крупные заголовки
`<h6></h6>`	Заголовок 6-го уровня	Самые мелкие заголовки (также существуют заголовки промежуточных уровней)

`<ul>`
`<li></li>`
`</ul>`
	Ненумерованный список	Должен содержать элементы списка (`<li></li>`). Может содержать любое число элементов

`<ol>`
`<li></li>`
`</ol>`
	Нумерованный список	Должен содержать элементы списка (`<li></li>`). Может содержать любое число элементов

`<a href ="another Page.html"> переход на другую страницу</a>`	Гиперссылка	Ссылка на другую страницу. Текст между тегами <a> и на странице будет виден как гиперссылка. Когда пользователь щелкает мышкой по ссылке, браузер переходит на указанную страницу
	Рисунок	Размещает на странице рисунок. Поддерживаются графические форматы GIF, JPG и PNG
`<font color = "red" эта надпись - красного цвета </font>`	Изменить шрифт	Поддерживается не во всех браузерах. Можно изменять цвет, размер и вид (гарнитуру) шрифта, однако выбранный вид не всегда можно передать на клиентскую машину
` `	Перенос строки	Перенос строки в тексте. Не имеет закрывающего тега
`<hr>`	Горизонтальная линия	Помещает на страницу горизонтальную линию. Не имеет закрывающего тега

А вот исходный код страницы `basic.html`, иллюстрирующей основные теги.

```
<html>
<head>
<title>Basic HTML Tags</title>
</head>
<body>
<h1>Basic HTML Tags</h1>
<h1>This is an h1 header</h1>
```

```
<h2>This is an h2 header</h2>
<h3>This is an h3 header</h3>
<h4>This is an h4 header</h4>
<h5>This is an h5 header</h5>
<h6>This is an h6 header</h6>

<center>
This text is centered
</center>

<b>This is bold</b>
<br>
<i>This is italic</i>
<hr>

</body>
</html>
```

При помощи тегов H1-H6 создаются заголовки разного размера. Тег <b> делает текст жирным, а <i> превращает его в курсив. И, наконец, благодаря <hr> на странице появилась горизонтальная линия.

Другие HTML-теги

Применение остальных тегов из таблицы 1.1. показано на рис. 1.4.

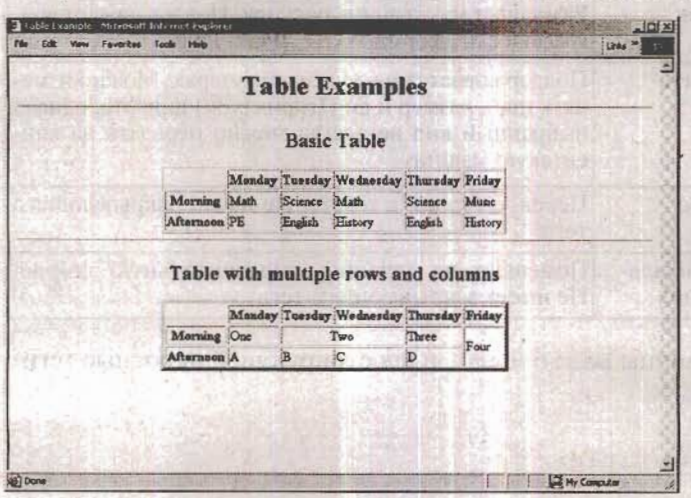


Рис. 1.4. Примеры еще нескольких основных HTML-тегов

Теги из файла `more.html` используются для помещения на страницу списков, ссылок и изображений. А вот исходный код страницы.

```
<html>
<head>
<title>More HTML Tags</title>
</head>
<body>
<h1>More HTML Tags</h1>
<h3>Ordered List</h3>
<ol>
   <li>alpha</li>
   <li>beta</li>
   <li>charlie</li>
</ol>
<h3>Unordered List</h3>
<ul>
   <li>alpha</li>
   <li>beta</li>
   <li>charlie</li>
</ul>
<h3>Hyperlink</h3>
<a href="http://www.cs.iupui.edu/~aharris">Andy's Home page</a>
<h3>Image</h3>
<img src="silly.gif"
     height = 100
     width = 100>
</body>
</html>
```

В HTML используется два вида списков. При помощи тега `<ol></ol>` создают нумерованные списки. Каждый элемент, обозначенный парой `<li></li>`, в таком списке автоматически получает номер. Теги `<ul></ul>` используют для создания ненумерованных списков. В них перед каждым элементом `<li></li>` ставится маркер.

При помощи гиперссылок пользователи переходят с одной веб-страницы на другую. Для размещения гиперссылки применяется пара тегов `<a></a>`. Тег `<a>` почти всегда включает атрибут `href`, после которого идет адрес. Когда пользователь щелкнет по ссылке, в его браузере откроется страница с этим адресом. Текст (или HTML-код), размещенный между тегами `<a>` и `</a>`, отображается браузером как гиперссылка. Он будет оформлен соответствующим образом (обычно ссылки выделены синим цветом и подчеркнуты). Пример `more.html` включает

в себя ссылку на одну из моих домашних страниц (`http://www.cs.iupui.edu`). Когда пользователь в окне своего браузера нажмет на ссылку «Andy's Home Page», он перейдет на эту страницу.

Еще один тег, применение которого показано в файле `more.html` – это `<img>`. С помощью этого тега на веб-страницах размещают изображения. Большинство браузеров поддерживают изображения в форматах `.gif` и `.jpg`, а многие «понимают» также и формат `.png`.

Если у вас есть картинка в другом формате или вы хотите сперва что-либо в ней изменить, а потом уже поместить на свою страницу, вы можете воспользоваться, например, бесплатными программами, такими, как irfanView или Gimp.

Таблицы

Наверняка вам нередко приходилось работать с большими объемами данных, которые было бы удобно представить в виде таблицы. В HTML существует набор тегов, при помощи которых таблицу можно разместить на веб-странице. Принцип работы этих тегов иллюстрирует рис. 1.5.

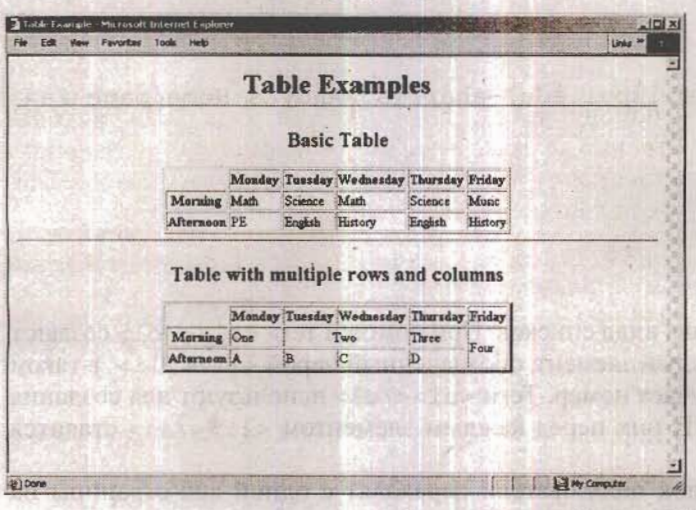


Рис. 1.5. Таблицы могут быть простыми, а могут содержать ячейки, занимающие несколько строк или столбцов

Код для создания простой таблицы (первой из изображенных на рисунке) выглядит следующим образом.

```
<table border = "1">
<tr>
  <th></th>
```

```
  <th>Monday</th>
  <th>Tuesday</th>
  <th>Wednesday</th>
  <th>Thursday</th>
  <th>Friday</th>
</tr>
<tr>
  <th>Morning</th>
  <td>Math</td>
  <td>Science</td>
  <td>Math</td>
  <td>Science</td>
  <td>Music</td>
</tr>
<tr>
  <th>Afternoon</th>
  <td>PE</td>
  <td>English</td>
  <td>History</td>
  <td>English</td>
  <td>History</td>
</tr>
</table>
```

Таблицы создаются при помощи тегов `<table></table>`. Между этими тегами можно создавать строки таблицы, используя элементы `<tr></tr>`. Каждая строка может включать в себя элементы, определяющие заголовок таблицы (`<th></th>`) или ячейку с данными (`<td></td>`).

Веб-браузер не отображает лишние пробелы и отступы, но при написании HTML-кода все-таки лучше их использовать, поскольку форматированный код легче читать. Обратите внимание на то, как в моем коде сдвинуты элементы внутри каждой строки таблицы. Благодаря этому намного проще заметить, что все данные, заключенные в <tr></tr>, являются частью одной группы.

У тега `<table>` можно установить атрибут `border`, с помощью которого задается толщина рамки вокруг таблицы.

Следует помнить, что не все браузеры имеют одинаковые настройки по умолчанию. Если вы не установите ширину рамки, некоторые браузеры покажут стандартную рамку, а некоторые не покажут вовсе. Поэтому лучшим решением будет всегда указывать ширину рамки. Если рамка вам не нужна, установите для нее ширину 0.

В какой-то момент вам может потребоваться, чтобы ячейки в таблице занимали больше одной строки или одного столбца. Ниже приведен код, с помощью которого была создана вторая таблица в файле `table.html`.

```
<table border = "4">
<tr>
  <th></th>
  <th>Monday</th>
  <th>Tuesday</th>
  <th>Wednesday</th>
  <th>Thursday</th>
  <th>Friday</th>
</tr>
<tr>
<  th>Morning</th>
  <td>One</td>
  <td colspan = "2"><center>Two</center></td>
  <td>Three</td>
  <td rowspan = "2">Four</td>
</tr>
<tr>
  <th>Afternoon</th>
  <td>A</td>
  <td>B</td>
  <td>C</td>
  <td>D</td>
</tr>
</table>
```

Обратите внимание, что ячейка, в которую записано значение «Two», имеет атрибут `colspan`, которому присвоено значение 2. Таким образом мы указываем, что эта ячейка должна занять два столбца по ширине. Поскольку она требует в два раза больше места, чем обычная, в этом ряду нам необходимо задать только пять элементов `<td>` или `<th>`, а не шесть, как это было в каждом ряду простой таблицы.

Взгляните также на ячейку, в которую записано значение «Four». Для того чтобы она заняла сразу две строки, я установил атрибут `rowspan` раным 2. Обратите внимание, что в следующем ряду мне пришлось определить на один элемент меньше, поскольку один из столбцов уже был занят нашим удлиненным элементом.

Учимся использовать каскадные таблицы стилей

Нетрудно писать обычный HTML-код, но страницы в результате получаются скучными. Современные браузеры поддерживают каскадные таблицы стилей (CSS – Cascading Style Sheets), с помощью которых вы можете явно указать, как должен отображаться тот или иной тег. О CSS написана не одна книга, но основные идеи весьма просты. Вы можете задать стиль, представляющий собой набор правил форматирования, и назначить его различным элементам на ваших страницах. Небольшой пример поможет вам лучше понять, о чем идет речь.

Создаем стили отдельных элементов

На рис. 1.6 показана веб-страница, некоторые элементы которой невозможно реализовать при помощи обычного HTML.

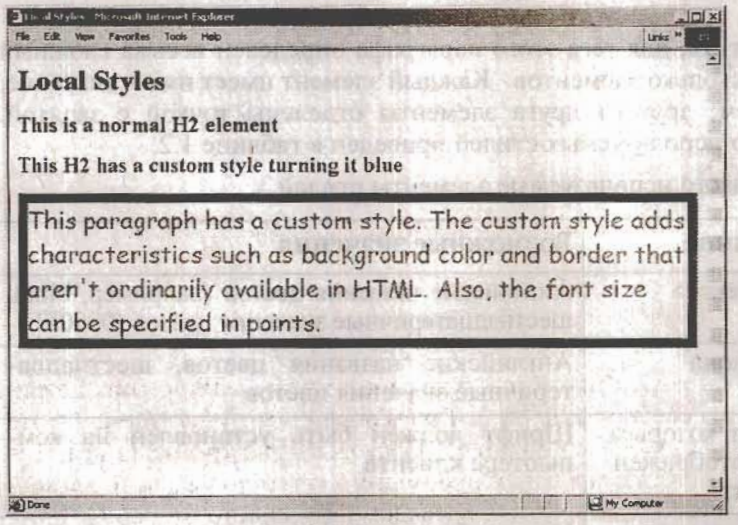


Рис. 1.6. Для оформления этой страницы я использовал CSS

Тег H2 обычно не делает текст голубым, но я добился этого, настроив стиль для данного тега при помощи следующего кода.

```
<h2 style = "color:blue">
This H2 has a custom style turning it blue
</h2>
```

Я добавил к тегу <h2> атрибут style. Этот атрибут имеет несколько параметров. С помощью color устанавливает цвет стиля. Объект, использующий этот стиль, будет отображен соответствующим цветом.

Кроме того, существует множество других параметров, применяемых для стилей. Для создания большого параграфа на рис. 1.6 был использован стиль с несколькими параметрами. Исходный код, создающий наш параграф, показан ниже.

```
<p style = "color:red;
            background-color: yellow;
            font-family: 'comic sans ms';
            font-size: 20pt;
            border-width: 10px;
            border-style: groove;
            border-color: green">
This paragraph has a custom style. The custom style adds characteristics such as background color and border that aren't ordinarily available in HTML. Also, the font size can be specified in points by specifying pt in the font size.
</p>
```

Как вы могли заметить, для тега этого параграфа определен весьма сложный стиль, содержащий несколько элементов. Каждый элемент имеет имя и значение, отделяемое двоеточием; друг от друга элементы отделены точкой с запятой. Список наиболее часто используемых стилей приведен в таблице 1.2.

Табл. 1. 2. Наиболее часто используемые элементы стилей

Элемент	Описание	Возможные значения
Color	Цвет текста	Английские названия цветов (например, blue), шестнадцатеричные значения цветов (0000FF)
Background-color	Цвет фона	Английские названия цветов, шестнадцатеричные значения цветов
Font-family	Шрифт, которым будет отображен элемент	Шрифт должен быть установлен на компьютере клиента
Font size	Размер шрифта	Может быть установлен в пикселах (px), точках (pt), сантиметрах (cm) или дюймах (in)
Border-width	Ширина рамки	Обычно измеряется в пикселах (px), сантиметрах (cm) или дюймах (in)
Border-style	Тип рамки	Некоторые значения: groove, double, ridge, solid, inset, outset
Border-color	Цвет рамки	Английские названия цветов (blue), шестнадцатеричные значения цветов (0000FF)

Внутренние стили

Несмотря на то что иногда бывает удобно задать стиль непосредственно для HTML-элемента, чаще случается так, что нужно изменить сразу несколько элементов на одной странице. Это можно сделать, создав для документа внутреннюю таблицу стилей. На рис. 1.7 показана страница, для которой определены внутренние стили.

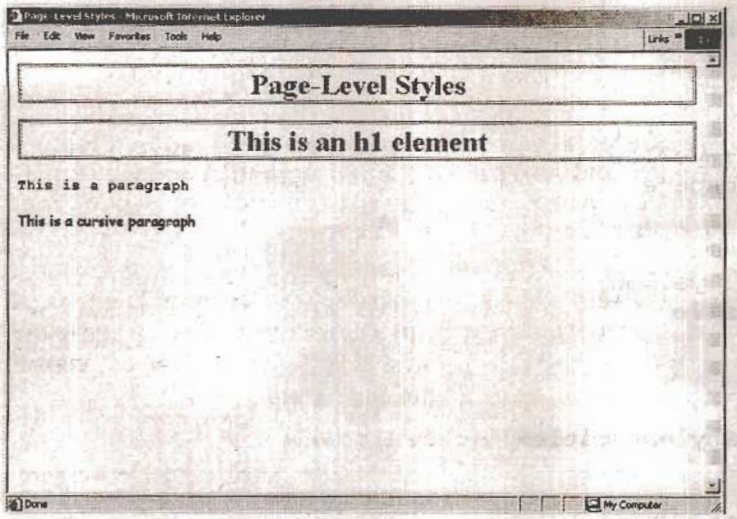


Рис. 1.7. Стиль тега H1 и два различных стиля параграфов определены во внутренней таблице стилей

Применение элементов DIV и SPAN в CSS

Стили CSS можно назначить для любого HTML-тега. Но на практике многие авторы предпочитают для назначения своих стилей использовать элементы `span` и `div`. Тег `span`, по существу, не имеет собственных свойств. Это делает его поведение очень предсказуемым, поскольку стиль CSS будет практически полностью определять внешний вид текста внутри этого элемента. Элемент `div` очень похож на `span`. Иногда его используют вместо `span` в качестве «группового» элемента, для которого может быть установлен стиль. В большинстве случаев элемент div создает отдельный параграф, а элемент `span` может использоваться внутри параграфа

Для установки стилей на уровне страницы в заголовке документа создается раздел `<style></style>`, внутри которого задается, как будет отображен тот или иной тег. Исходный код страницы `pageStyle.html` демонстрирует создание таблицы стилей на уровне всей страницы.

```
<html>
<head>
<style type = "text/css">
h1 {
  text-align:center;
  color:green;
  border-color:red;
  border-style:double;
  border-size: 3px
}
p {
  background-color: yellow;
  font-family: monospace
}
p.cursive {
  background-color: yellow;
  font-family: cursive
}
</style>

<title>Page-Level Styles</title>
</head>
<body>
<h1>Page-Level Styles</h1>
<h1>This is an h1 element</h1>
<p>This is a paragraph</p>
<p class = cursive>
This is a cursive paragraph
</p>
</body>
</html>
```

Если мы посмотрим на тело страницы, заключенное в теги `<body></body>`, то увидим, что выглядит оно как обычный HTML-код (которым и является на самом деле). Нас же сейчас интересует код, расположенный между тегами `<style>` и `</style>`. Он определяет, как те или иные теги будут выглядеть на экране. Открывающий тег должен выглядеть так: `<style type = "text/css">`, – этим мы даем понять браузеру, что он имеет дело с обыкновенной таблицей стилей. Внутри элемента `style` мы перечисляем все теги, внешний вид которых хотим задать. После каждого названия тега следуют различные элементы стиля, обрамленные парой фигурных скобок (`{}`). Элементы перечис-

ляются точно так же, как и для атрибута `style`. Каждый элемент представляет собой пару «название/значение». Двоеточие отделяет название от значения, а точкой с запятой одна пара отделяется от другой.

Как и в большинстве случаев при программировании на HTML, для элемента style не имеет значения, где вы поставите пробел или перейдете на новую строку. Однако при разумной расстановке пробелов, символов табуляции и перевода строки ваш код будет намного легче читать и изменять. Посмотрите, как у меня выровнен каждый элемент и насколько это упрощает чтение кода. Обратите внимание на то, что закрывающую скобку (}) я расположил прямо под именем HTML-элемента – так я с первого взгляда вижу, как соотносятся между собой различные части кода. Подобное внимание к форматированию кода вы будете встречать на протяжении всего времени, в течение которого станете заниматься программированием.

Обратите внимание на второй элемент параграфа, который выглядит следующим образом.

```
p.cursive {
  background-color: yellow;
  font-family: cursive
}
```

Добавив к названию HTML-элемента точку и приписав еще одно имя (в нашем случае `.cursive`), я создал еще один тип тега `<p>`, определяющего параграф. Вариаций одного и того же тега вы можете создать столько, сколько пожелаете. Это бывает особенно удобно тогда, когда вы хотите оформить текст несколькими разными стилями. Например, установить один стиль для цитат, а другой – для простого текста. Для указания той или иной формы тега используется атрибут `class`, так, как я это сделал в следующем примере.

```
<p class = cursive>
This is a cursive paragraph
</p>
```

Внешние таблицы стилей

Большинство веб-браузеров поддерживают и третий вид таблиц стилей – внешние. На рис. 1.8 показана страница, использующая внешнюю таблицу стилей.

Не глядя на исходный код страницы, пользователь никак не определит, какой вид таблиц стилей использовался. И хотя пример страницы, иллюстрирующей использование внешних таблиц стилей, почти не отличается от страницы, для создания которой использовались внутренние таблицы стилей, их исходный код раз-

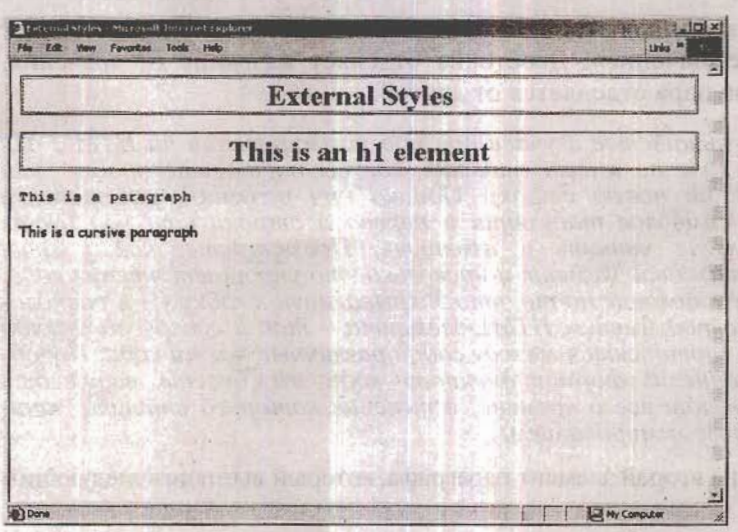


Рис. 1.8. Внешние таблицы стилей ничем не отличаются от всех прочих с точки зрения пользователя, но имеют свои преимущества для программиста

личен. Ниже приведен код, с помощью которого создана страница `externStyle.html`.

```
<html>
<head>
<link rel="stylesheet"
  type="text/css"
  href = "externStyle.css">
<title>External Styles</title>
</head>
<body>
<h1>External Styles</h1>
<h1>This is an h1 element</h1>
<p>This is a paragraph</p>
<p class = cursive>
This is a cursive paragraph
</p>
</body>
</html>
```

Основной код идентичен исходному коду примера с внутренними стилями, но вы, наверное, заметили, что на этот раз таблица стилей не включена в документ непосредственно. Напротив, стили хранятся в отдельном файле с именем `externStyle.css`. Содержимое этого файла в точности повторяет стили, созданные для предыдущей страницы-примера.

```
h1 {
  text-align:center;
  color:green;
  border-color:red;
  border-style:double;
  border-size: 3px
}
p {
  background-color: yellow;
  font-family: monospace
}
p.cursive {
  background-color: yellow;
  font-family: cursive
}
```

Когда таблица стилей хранится в отдельном файле, правила из нее можно импортировать при помощи тега `link`. Преимущество такого подхода состоит в том, что один и тот же набор правил может быть использован для множества страниц.

ВОЗВРАЩАЯСЬ К РЕАЛЬНОСТИ

Внешние таблицы стилей очень полезны, когда вы работаете над проектом, состоящим из множества страниц, которые должны выглядеть одинаково. Дизайн большинства сайтов за время создания может измениться несколько раз, и если каждый раз, когда клиенту захочется поменять цвет шрифта, вам придется переделывать каждую из 20 страниц сайта, это превратится в настоящую пытку. Если же все ваши стили хранятся в одном CSS-документе, а все страницы обращаются к нему, тогда все, что вам понадобится сделать, – это поменять правила стилей один раз, и внешний вид каждой страницы, использующей этот набор правил, изменится автоматически

Используем элементы форм

Для работы с данными, вводимыми пользователями, на HTML-страницах часто используются элементы форм. Они включают в себя основные инструменты, с помощью которых пользователь может осуществить ввод данных. Но если создавать страницы на «чистом» HTML, размещать элементы форм на них не имеет смысла. И хотя совсем не сложно разместить такие элементы на странице, без обрабатывающей программы пользы от них будет немного. Ваша работа как программиста, пишущего на PHP, во многом будет состоять в том, чтобы получить информацию из веб-форм,

а значит, очень важно знать основные элементы, применяемые при создании форм. Писать программы, извлекающие данные из веб-форм, вы начнете уже в следующей главе, так что сейчас самое время разобраться в том, как они работают.

Текстовые элементы

Большинство элементов форм служат для передачи текстовой информации от пользователя в программу. И первый тип – это элементы, которые просто позволяют пользователю ввести некий текст. На рис. 1.9 показаны четыре таких элемента.

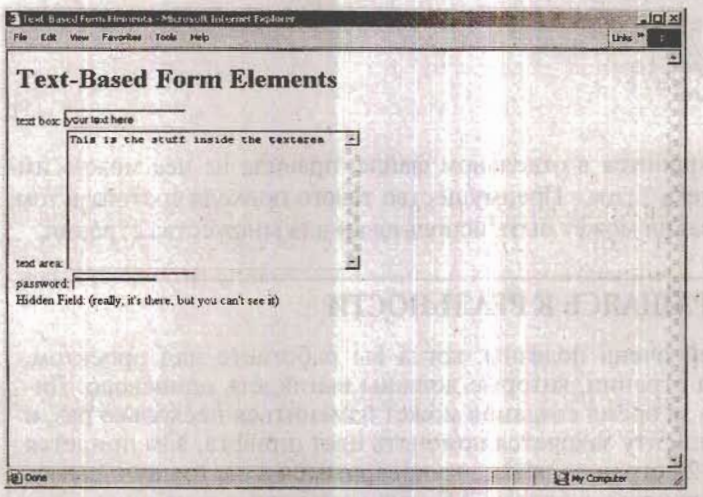


Рис. 1.9. В свои веб-страницы вы можете включить текстовые поля, текстовые области, поля для ввода пароля и скрытые поля (последние пользователю не видны)

Ниже приведен код, создающий страницу `textForm.html`.

```
<html>
<head>
<title>Text-Based Form Elements</title>
</head>
<body>
<h1>Text-Based Form Elements</h1>
<form>
text box:
<input type = "text"
  name = "txtInput"
  value = "your text here">
<br>
text area:
<textarea name = "txtBigInput"
```

```
  rows = 10
  cols = 40>
  This is the stuff inside the textarea
</textarea>
<br>
password:
<input type = "password"
  name = "secret"
  value = "you can't read this">
<br>
Hidden Field: (really, it's there, but you can't see it)
<input type = "hidden"
  name = "mystery"
  value = "secret formula">
</form>
</body>
</html>
```

Любой элемент, предполагающий взаимодействие с пользователем, должен быть размещен между тегами `<form></form>`. Чаще всего используется элемент `<input>`, который может принимать самый разный вид в зависимости от значения атрибута type.

Создаем текстовое поле

Самый используемый элемент ввода – это скромное текстовое поле. Для создания самого простого текстового поля я использовал следующий код.

```
<input type = "text"
       name = "txtInput"
       value = "your text here">
```

Здесь использован простой элемент `input`. Установив для него тип `"text"`, я определил, каким образом элемент будет отображен на экране: в данном случае это будет поле, в которое пользователь сможет ввести текст. Элемент `input`, у которого атрибуту `type` присвоено значение `text`, обычно называют текстовым полем. В текстовое поле нельзя ввести несколько строк, а длина его может быть ограничена при помощи атрибута `size`. (Если вы присвоите атрибуту `size` значение 20, то пользователь сможет ввести в поле примерно 20 символов.) Для текстовых полей (как, собственно, и для всех элементов форм) очень важно назначить атрибут `name` (имя), поскольку чуть позже вы станете писать программы, которые будет извлекать данные из форм. Для того чтобы ваши программы могли обращаться к данным, введенным пользователем, вы будете использовать имена элементов форм.

Назначение имен элементам пользовательского ввода сродни искусству. Название должно быть в достаточной степени описательным (`r` *или* `albert` *вряд ли можно назвать хорошими именами для подобных объектов, поскольку они никак не поясняют, какого типа данные предполагается получать посредством этого объекта). Имена объектов не могут содержать пробелы, поскольку в дальнейшем это может привести к путанице. Об этом вы больше узнаете в следующей главе, когда начнете работать с переменными, которые в PHP имеют очень тесную связь с элементами форм.*

С помощью атрибута `value` устанавливается значение по умолчанию для текстовой области. Это значение пользователь увидит при первой загрузке вашей формы. Установка значений по умолчанию – это хорошая практика, поскольку в таком случае пользователь сразу видит, данные какого типа он должен ввести.

Создаем текстовую область

Текстовые поля очень удобны, но в какой-то момент вы обязательно захотите, чтобы пользователь ввел не одну, а несколько строк. К примеру, вам может понадобиться страница обратной связи, где пользователь смог бы написать свои комментарии, которые будут отправлены вам по электронной почте. В таком случае используется объект, называемый «текстовая область». А вот с помощью какого кода он создается:

```
<textarea name = "txtBigInput"
          rows = 10
          cols = 40>
This is the stuff inside the textarea
</textarea>
```

Для создания текстовой области используется пара тегов `<textarea></textarea>`. Текстовая область имеет атрибут `name`, а также атрибуты `rows` и `cols`, определяющие ее размер в строках и столбцах. Атрибута `value` текстовая область не имеет. Вместо этого любой текст, расположенный между тегами `<textarea>` и `</textarea>`, воспринимается браузером в качестве содержимого области.

Не забывайте закрывать текстовую область тегом `</textarea>`*. Если вы этого не сделаете, то даже если браузер и сумеет отобразить такую страницу, все ее содержимое, расположенное после тега* `<textarea>`*, окажется внутри этой области.*

Создаем поле для ввода пароля

Поля для ввода пароля практически идентичны текстовым полям. Код, при помощи которого создается поле для ввода пароля, очень похож на код текстовой области:

```
<input type = "password"
       name = "secret"
       value = "you can't read this">
```

Единственная существенная разница между этими полями состоит в том, что текст, вводимый в поле для ввода пароля, на экране отображается звездочками. Считается, что таким образом любопытные не смогут подсмотреть вводимый пароль, заглянув через плечо пользователя.

Очень важно понимать, что поле для ввода пароля на самом деле практически не обеспечивает безопасности. В следующей главе вы узнаете, что информация, вводимая в такое поле, передается на сервер в открытом виде, так что секретным это поле можно назвать лишь условно.

Создаем скрытое поле

У текстового поля есть еще более скрытный родственник, чем поле для ввода пароля. Скрытое поле в коде очень похоже на текстовое поле, однако на странице оно не отображается. Код, создающий его, выглядит следующим образом:

```
<input type = "hidden"
       name = "mystery"
       value = "secret formula">
```

Сейчас вам, может быть, не совсем ясно, как можно использовать поля, скрытые от глаз пользователя, но их удобно применять, когда вашей странице нужно, например, установить связь или обменяться информацией с сервером, а пользователю не обязательно знать все эти детали. (Обещаю, что вскоре покажу вам такую ситуацию на примере.)

Создаем элементы выбора

Совсем не трудно разместить на вашей веб-странице текстовые элементы, но необходимость ввода текста может сильно усложнить работу программы. Опытные программисты предпочитают при любой возможности исключать ручной ввод текста пользователем, взамен предоставляя ему варианты для выбора. На HTML-формы могут быть помещены несколько простых элементов, позволяющих пользователю при помощи мыши выбрать один или несколько из предлагаемых вариантов.

Удобство использования – одна из причин, по которым применяются элементы выбора, но не единственная. Невозможно предсказать, что именно введет пользователь в текстовое поле. Очень трудно написать код, в котором была бы предусмотрена реакция на любые неправильно вводимые данные, в то время как при использовании элементов выбора, описанных ниже, вы заранее определяете все допустимые значения, с которыми придется работать вашей программе (по крайней мере для большинства ситуаций).

На рис. 1.10 показано несколько элементов выбора, размещенных на веб-странице.

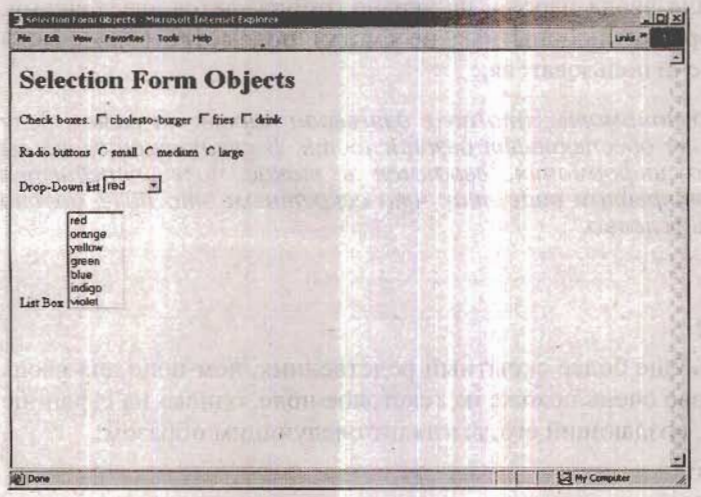


Рис. 1.10. HTML-элементы, позволяющие пользователю вводить информацию без необходимости набирать текст вручную

Создаем флажки

Первым из подобных элементов мы рассмотрим флажок. Обычно флажок выглядит как галочка, которую можно поставить в специально отведенном для этого месте. Рядом с ним, как правило, помещается какой-либо текст.

Флажок может быть установлен либо не установлен. На странице `selectForm.html` флажки создаются при помощи следующего кода.

```
<input type = "checkbox"
       name = "chkBurger">cholesto-burger
<input type = "checkbox"
       name = "chkFries">fries
<input type = "checkbox"
       name = "chkDrink">drink
```

Флажок – это элемент ввода типа `checkbox`. Для него можно установить атрибут `value`, однако в отличие от других элементов ввода делать это, как правило,

не требуется. Заметим, что текст, расположенный рядом с флажком, представляет собой обычный HTML. Каждый флажок является полностью независимым объектом. Несмотря на то что в HTML-документе несколько флажков могут находиться рядом, значение одного из них никак не влияет на значения остальных.

Флажки применяют тогда, когда возможен ввод любой комбинации различных элементов. Например, пользователь может захотеть гамбургер, картофель-фри и напиток. Он может выбрать любое их сочетание, а может и вовсе ничего не заказать. В тех же случаях когда каждый вариант выбора исключает все остальные, применение флажков не столь оправдано. К примеру, если нужно уточнить объем выбранного напитка, то необходимо разрешать пользователю выбор только одного варианта. В такой ситуации лучше использовать другой элемент, который назвается «переключатель».

Выбираем с помощью переключателей

Переключатели (их еще называют «кнопками выбора вариантов» и «радио-кнопками») можно использовать когда требуется, чтобы пользователь выбрал один из нескольких предлагаемых вариантов. Название «радио-кнопка» (`radio button`) произошло от автомобильных радиоприемников (по крайней мере, какими они были в моем детстве), имевших несколько выступающих кнопок. Для выбора диапазона (ДВ,СВ, КВ, УКВ и т. д.) нужно было нажать одну из них, при этом выбранная кнопка фиксировалась в утопленном положении, а все остальные автоматически выключались. В HTML переключатели работают похожим образом. Они сгруппированы так, что когда вы выбираете один из перелючателей, все остальные в группе автоматически выключаются.

Посмотрите на код, создающий переключатели, и попробуйте определить, каким образом они сгруппированы.

```
<input type = "radio"
       name = "size"
       value = "small">small
<input type = "radio"
       name = "size"
       value = "medium">medium
<input type = "radio"
       name = "size"
       value = "large">large
```

Обратите внимание на то, как перелючателям назначаются имена. В моем коде определено три кнопки-переключателя, но имя у них одно. С помощью такого приема переключатели объединяются в группу и реагируют на действия пользователя как единое целое: когда пользователь выбирает один из элементов в группе переключателей, все остальные элементы с тем же именем автоматически вы-

ключаются. Для каждого перелючателя назначен отдельный атрибут value. Обратившись к этому атрибуту, ваша программа определит, какой из переключателей в группе был выбран.

Создаем выпадающие списки

Еще один часто используемый элемент пользовательского интерфейса – это выпадающий список. Этот элемент позволяет пользователю выбирать значение из заданного набора вариантов, увидеть которые он сможет только непосредственно при выборе. Это бывает особенно полезно, когда на экране не хватает свободного места или когда вы не хотите перегружать интерфейс. Выпадающие списки создаются при помощи двух разных элементов. Основным является объект select. Он содержит несколько объектов option. (Похожим образом внутри объекта списка li содержатся элементы ul или ol.) Пример поможет вам лучше разобраться с выпадающими списками.

```
<select name = "selColor">
   <option value = "red">red</option>
   <option value = "orange">orange</option>
   <option value = "yellow">yellow</option>
   <option value = "green">green</option>
   <option value = "blue">blue</option>
   <option value = "indigo">indigo</option>
   <option value = "violet">violet</option>
</select>
```

У объекта select есть атрибут name. Для каждого элемента назначается атрибут value. В вашу программу поступит значение атрибута value для выбранного элемента, однако само это значение не отображается на странице. Текст, который должен быть показан для каждого из элементов списка, помещается в коде между тегами <option> и </option>.

Создаем список с множественным выбором

В некоторых ситуациях может оказаться полезным использовать еще один элемент выбора. В общем-то, это не совсем новый для нас объект, а вариант выпадающего списка.

Код этого элемента, последнего в файле selectForm.html, показан ниже.

```
<select name = "lstColor"
        size = 7
        multiple>
   <option value = "red">red</option>
   <option value = "orange">orange</option>
   <option value = "yellow">yellow</option>
```

```
    <option value = "green">green</option>
    <option value = "blue">blue</option>
    <option value = "indigo">indigo</option>
    <option value = "violet">violet</option>
</select>
```

Код выглядит так же, как для предыдущего (выпадающего) списка, и только в описании самого тега `select` есть небольшие отличия. Присвоив атрибуту `size` значение 7, я указал, что на экране элемент списка должен всегда занимать семь строк. Это полезно в тех случаях, когда вы хотите, чтобы пользователь сразу видел все (или хотя бы многие) варианты выбора. Еще одна интересная особенность этого списка состоит в том, что он позволяет пользователю выбрать несколько элементов, если в описание тега включен атрибут `multiple`. В списке с множественным выбором пользователь может выбрать несколько элементов при помощи стандартных действий множественного выбора (например, Shift+щелчок мышью выделяет диапазон последовательно идущих элементов, а Ctrl+щелчок добавляет элемент к уже выделенным или снимает выделение с конкретного элемента).

Добавляем кнопки

Последний важный элемент форм – это кнопка. Кнопки играют большую роль, потому что пользователь привык нажимать их, когда хочет чтобы что-то произошло. Если ваши программы будут следовать этому правилу, они от этого только выиграют. На рис. 1.11 показана страница, содержащая три различных кнопки.

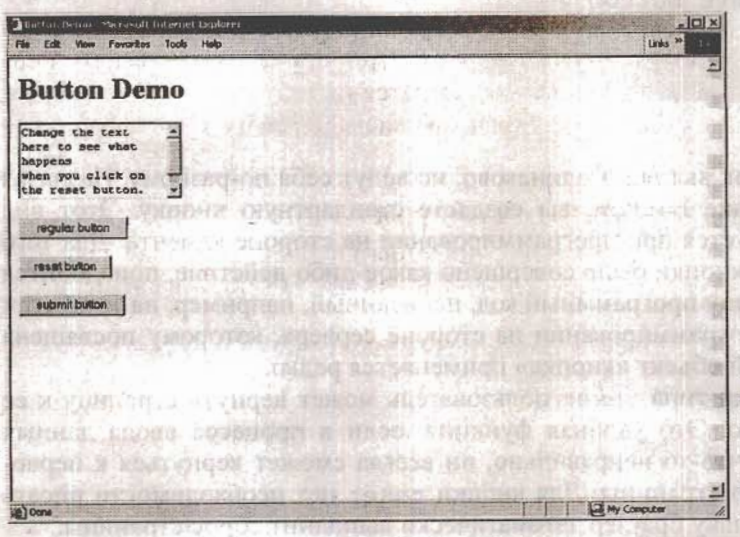


Рис. 1.11. Несмотря на то что для пользователя эти кнопки похожи одна на другую, они все же отличаются и имеют разное поведение

Каждая из кнопок создана как вариант основного тега input, с которым вы так часто сталкивались в этой главе. Код, положенный в основу страницы buttonForm.html, наглядно это иллюстрирует.

```
<html>
<head>
<title>Button Demo</title>.
</head>
<body>
<h1>Button Demo</h1>

<form>
<textarea rows = 5>
Change the text here to see what happens
when you click on the reset button.
</textarea>
<br><br>

<input type = "button"
     value = "regular button">
<br><br>

<input type = "reset"
     value = "reset button">
<br><br>

<input type = "submit"
     value = "submit button">

<br><br>
</form>

</body>
</html>
```

Эти три вида кнопок выглядят одинаково, но ведут себя по-разному. Назначая атрибуту type значение button, вы создаете стандартную кнопку. Этот вид кнопок часто используется при программировании на стороне клиента. Для того чтобы после нажатия кнопки было совершено какое-либо действие, понадобится включить в веб-страницу программный код, написанный, например, на JavaScript или VBScript. При программировании на стороне сервера, которому посвящена эта книга, стандартный объект «кнопка» применяется редко.

При помощи кнопки типа reset пользователь может вернуть страницу к ее начальному состоянию. Это удобная функция: если в процессе ввода данных пользователь сделает что-то неправильно, он всегда сможет вернуться к первоначальному состоянию страницы. Для кнопки reset нет необходимости писать какой-либо код, поскольку браузер автоматически выполнит сброс страницы.

Кнопка типа `submit`, вне всякого сомнения, является важнейшей для программирования на стороне сервера, которым мы займемся в этой книге. Кнопка `submit` обеспечивает связь между веб-страницами и вашими программами. Взаимодействие с пользователем при программировании на стороне сервера, как правило, начинается с отправки ему HTML-страницы с формой. После того как пользователь выбирает и вводит все требуемые значения в элементы формы, он нажимает кнопку «Отправить» (имеющую тип `submit`), и тогда из данных в элементах формы создается пакет, который отправляется на обработку на сервер. В следующей главе вы узнаете, как это работает, а сейчас очень важно научиться размещать кнопку `submit` на ваших формах, поскольку этот элемент будет использован на многих страницах.

Добавляем к странице код, написанный на PHP

Язык HTML – это здорово, но подозреваю, что вы взяли эту книгу, чтобы изучить PHP, поэтому сейчас самое время добавить на страницу код на этом языке. Использование PHP расширит ваши возможности при создании страниц по сравнению с обычным HTML и CSS.

Проверяем, поддерживает ли ваш сервер PHP

Страница, написанная на PHP, может быть такой же, как и обычная HTML-страница. И та, и другая создается в обычном текстовом редакторе и хранится на веб-сервере. Программа на PHP может содержать элементы `<script>`, включенные в страницу. Когда пользователь запрашивает страницу с PHP, сервер вначале исследует ее код, выполняет все сценарии и только потом отправляет итоговый HTML-код пользователю. Но так сервер будет работать, только если предварительно он был настроен на использование языка PHP. Возможно, вам придется обратиться к администратору своего сервера, чтобы выяснить, включена ли на нем поддержка PHP. На домашнем комьютере вы можете установить пакет PHP Triad. Тем самым вы установите и настроите все необходимые компоненты.

Для запуска всех программ из этой книги потребуется, чтобы на вашем сервере были установлены три разных компонента. Во-первых, вам понадобится веб-сервер, например Microsoft IIS или Apache. Во-вторых, интерпретатор языка PHP – программа, которая обрабатывает файлы, написанные на PHP, преобразовывая их в HTML-страницы. И наконец, для хранения информации вам потребуется система управления базами данных (СУБД). В пакет PHP Triad входят все три системы. Он включает в себя Apache – бесплатный и очень мощный веб-сервер, интерпретатор PHP и СУБД mySQL. Это

стандартный пакет для большинства серверов, поддерживающих PHP. Если веб-сервер, на котором вы размещаете свои страницы, еще не поддерживает PHP, вы все равно можете установить этот пакет и тренироваться на собственном компьютере (хотя никто и не сможет увидеть ваши программы извне).

Добавляем к HTML-странице команды на PHP

Простейший способ выяснить, поддерживается ли PHP вашим сервером, – это написать короткую программу на PHP и посмотреть, будет ли она работать. Вот очень простая PHP-программа.

```
<html>
<head>
<title>Hello in PHP</title>
</head>
<body>
<h1>Hello in PHP</h1>
<?
print "Hello, world!";
phpInfo();
?>
</body>
</html>
```

Последовательность символов <? ?> – самый простой, но не всегда лучший способ обозначения PHP-кода. Его можно обозначить и так: <?php ?>. Этот вариант более приемлем в ситуациях, когда ваш код будет обрабатываться как XML. Вы также можете определить код и при помощи стандарных HTML-тегов, аналогично с тем как вы определяете JavaScript: <script language="php"> </script>. Некоторые PHP-серверы настроены на использование только одного варианта тега script, так что от вас может потребоваться определенная гибкость. Однако все эти варианты работают совершенно одинаковым образом.

Программа на PHP внешне очень похожа на обычную HTML-страницу. Единственное отличие – специальный тег <? ?>. Этот тег указывает на наличие PHP-кода. Любой код, написанный внутри этого тега, будет обработан интерпретатором языка PHP и затем преобразован в HTML-код. В моем примере на страницу добавлены две команды, но посмотрите на результат работы программы, показанный на рис. 1.12, и возможно, он вас удивит.

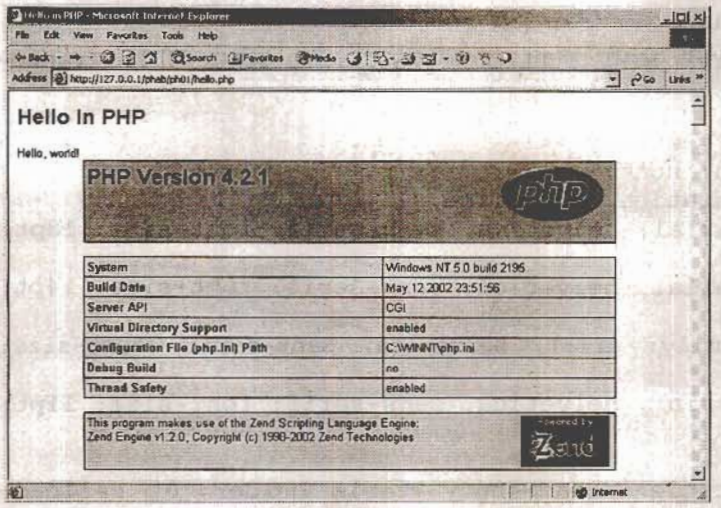


Рис. 1.12. Кроме HTML на этой странице есть еще кое-что...

Исследуем результаты

На этой странице представлены три различных вида текста. Во-первых, надпись «Hello in PHP» создана на обычном HTML. Я создал ее точно так же, как для обычной HTML-страницы, и отображается она тоже как обычный HTML-код. Надпись «Hello world» слегка отличается, поскольку создана при помощи PHP-программы, включенной в страницу. А оставшаяся часть страницы, наверное, является для вас загадкой. Она содержит множество данных об используемой PHP-машине. Эти данные занимают несколько страниц, хотя и сгенерированы всего одной командой `phpInfo()`. Данная команда отображает информацию об установленном на сервере языке PHP. Вам не обязательно понимать все, что показывает команда `phpInfo()`. Значительно важнее осознавать, что, когда пользователь запрашивает страницу `hello.html`, ее текст сперва обрабатывается интепретатором языка PHP. Эта программа сканирует страницу, находит на ней все PHP-команды, выполняет их и помещает на их место полученный HTML-код. К моменту, когда страница попадает к пользователю, PHP-кода на ней уже нет, поскольку на его основе сервер сгенерировал HTML-код. Чтобы убедиться в этом, откройте в вашем браузере страницу `hello.php` и запросите ее исходный код. Он будет выглядеть примерно следующим образом:

```
<html>
<head>
<title>Hello in PHP</title>
</head>
<body>
```

```
<h1>Hello in PHP</h1>
Hello, world!<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transi-
tional//EN">
<html>
<head><style type="text/css"><!--
a { text-decoration: none; }
a:hover { text-decoration: underline; }
h1 { font-family: arial, helvetica, sans-serif; font-size: 18pt;
font-weight: bold;}
h2 { font-family: arial, helvetica, sans-serif; font-size: 14pt;
font-weight: bold;}
body, td { font-family: arial, helvetica, sans-serif; font-size:
10pt; }
th { font-family: arial, helvetica, sans-serif; font-size: 11pt;
font-weight: bold; }
//--></style>
<title>phpinfo()</title></head><body><table border="0" cellpad-
ding="3" cellspacing="1"
width="600" bgcolor="#000000" align="center">.
<tr valign="middle" bgcolor="#9999cc"><td align="left">
<a href="http://www.php.net/"><img src="/phab/ph01/
hello.php?=PHPE9568F34-D428-11d2-A769-
00AA001ACF42" border=0 align="right" alt="PHP Logo"></a><h1>PHP
Version 4.2.1</h1>
```

Заметьте, что здесь я показал лишь небольшую часть кода, созданного командой `phpInfo()`, и что при выполнении программы на вашем компьютере отдельные участки этого кода могут отличаться. Главное в данном случае то, что PHP-код, создающий надпись «Hello World!» (`print "Hello World!"`), заменен на саму надпись «Hello World!». Еще важнее то, что простая команда `phpInfo()` оказалась заменена огромным количеством HTML-кода.

При помощи небольших кусков PHP-кода можно эффективно создавать большие и сложные HTML-документы. Это одно из существенных преимуществ PHP. Кроме того, к тому моменту, когда страница попадает в браузер, в ней остается только обыкновенный HTML-код, который будет понят любым браузером. Две этих особенности являются важными преимуществами программирования на стороне сервера вообще и программирования на PHP в частности. По мере прочтения этой книги вы узнаете множество команд для создания интересного HTML-кода, но основные положения останутся неизменными. Программа на языке PHP – это всегда не более чем HTML страница, содержащая специальную PHP-разметку. PHP-код обрабатывается отдельной программой на сервере, и результаты включаются в веб-страницу до отправки ее пользователю.

Создаем программу «Совет дня»

В самом начале этой главы я пообещал вам, что вы сможете написать программу «Совет дня», работа которой была продемонстрирована там же. Эта программа требует знания HTML, CSS и написания одной строки PHP-кода. Итоговый код покажет весьма простую страницу.

```
<html>
<head>
<title>Tip of the day</title>
</head>
<body>
<center>
<h1>Tip of the day</h1>
<div style = "border-color:green; border-style:groove; border-
width:2px">.
<?
readfile("tips.txt");
?>
</div>
</center>
</body>
</html>
```

Это простая HTML-страница. Она содержит один элемент `div`, для которого определен собственный стиль, рисующий рамку вокруг текста совета. Внутри элемента `div` я поместил PHP-код, обрамленный символами `<?` и `?>`. Этот код вызывает одну функцию языка PHP, которая называется `readFile()`. Команда `readFile()` принимает в качестве аргумента имя файла, запрашивает содержимое этого файла и отображает его на странице как простой HTML-код.

Как только эта строка кода заканчивает свою работу (т. е., когда весь текст из файла `tips.txt` будет отображен в окне браузера), символ `?>` указывает, что PHP-код закончен, и остаток страницы будет простым HTML.

Итоги

К этому моменту вы уже прошли немалый путь. Вы изучили или вспомнили все основные HTML-объекты. Вы исследовали каскадные таблицы стилей и то, как с их помощью можно изменить HTML-атрибут. Вы поэкспериментировали с основными элементами форм и научились помещать на свои страницы разнообразные текстовые поля и элементы выбора. Вы увидели, как в HTML-документ

можно включить PHP-код. И наконец, вы создали свою первую страницу, включающую все эти элементы. Сейчас вы уже можете гордиться своими усилиями. В следующей главе вы более подробно исследуете взаимосвязь между PHP и HTML и научитесь делать интересные вещи при помощи переменных и полей ввода.

ДОМАШНЕЕ ЗАДАНИЕ

1. Создайте веб-версию вашего резюме, включающую заголовки, списки и разные стили текста.
2. Измените одну из ваших страниц так, чтобы она включала каскадные таблицы стилей.
3. Установите тренировочную конфигурацию Apache, PHP и mySQL (или другие подобные продукты). Для облегчения задачи воспользуйтесь пакетом наподобие PHP Triad, если это возможно.
4. Создайте страницу, вызывающую команду `phpInfo()`, и запустите ее на своем веб-сервере. Проверьте, не является ли версия PHP, установленная на вашем сервере, слишком старой.

Глава 2

Используем переменные и элементы ввода

В главе 1, «Знакомимся с PHP», вы получили базовые знания, на которых построено все программирование на PHP. Теперь, освежив в памяти навыки программирования на HTML и использования CSS, вы готовы к тому, чтобы почувствовать всю мощь, заключенную в программировании на стороне веб-сервера, особенно с использованием PHP. Все компьютерные программы в конечном итоге занимаются обработкой данных. В этой главе вы узнаете о том, как данные хранятся в переменных и как программы манипулируют ими. Вы научитесь:

- создавать переменные языка PHP;
- распознавать основные типы переменных;
- правильно назначать имена переменным;
- выводить значения переменных в ваших сценариях;
- производить основные операции с переменными;
- считывать в переменные значения из HTML-форм.

Представляем программу, рассказывающую сказки

Прочитав эту главу, вы сможете написать программу, результат работы которой показан на рис. 2.1 и 2.2.

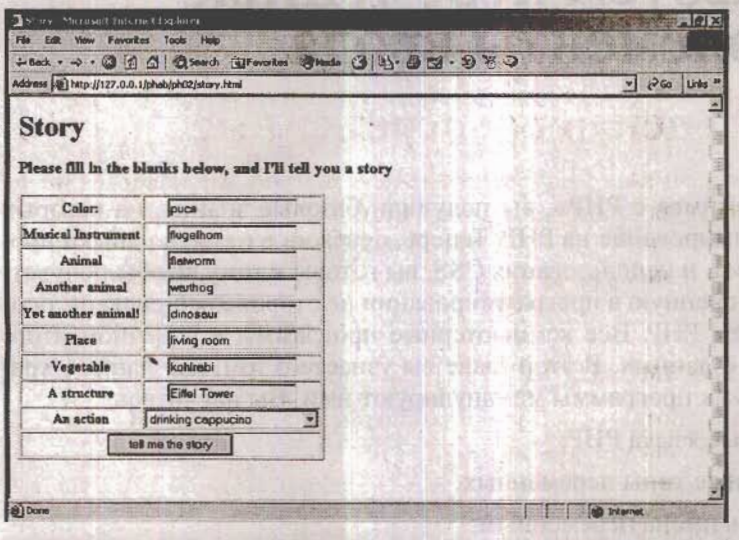


Рис. 2.1. Программа начинает работу с того, что запрашивает у пользователя данные

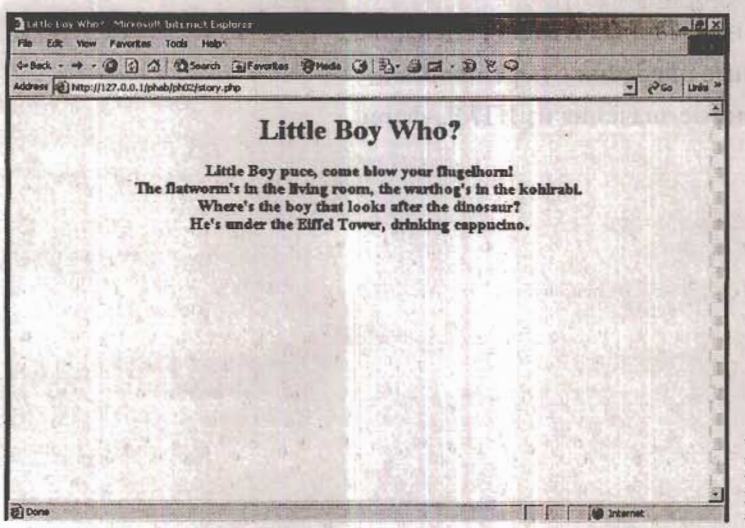


Рис. 2.2. Пример работы программы, рассказывающей сказки

Программа просит пользователя ввести некоторые значения в HTML-форму, а затем на их основе создает собственную версию известной детской сказки. Как и большинство программ, выполняемых на стороне сервера, она состоит из двух отдельных частей. Вначале пользователь вводит данные в обычную HTML-форму и нажимает кнопку отправки. Только после этого начинается работа второй, написанной на PHP, части программы. Она получает данные из формы, обрабатывает их неким образом и, как правило, возвращает пользователю результат в виде HTML-страницы.

Используем в сценариях переменные

Самая важная идея в этой главе – это понятие переменной. Переменная представляет собой контейнер для хранения информации в памяти компьютера. Для того чтобы программисту было легче ориентироваться, каждая переменная имеет имя. В переменную можно записать данные, а можно получить из нее ранее записанные данные.

Представляем программу «Hi Jacob»

Программа, результат работы которой показан на рис. 2.3, использует переменную, хотя с первого взгляда вы могли этого и не заметить.

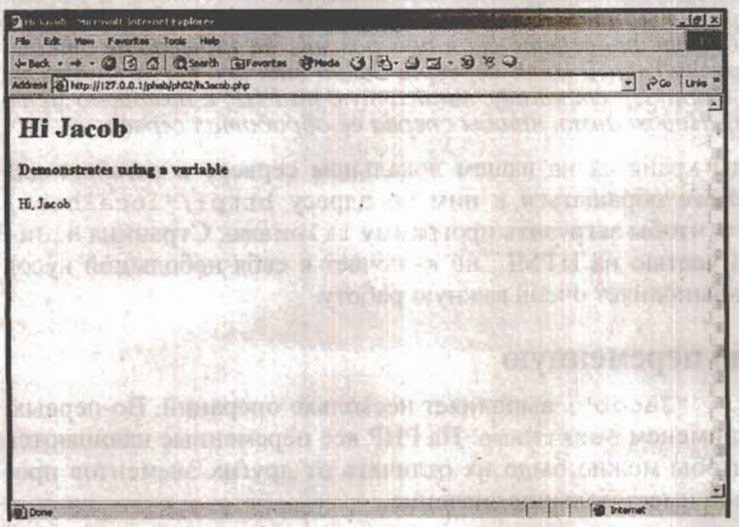


Рис. 2.3. Слово «Jacob» на этой странице хранится в переменной. На веб-странице, созданной программой, вы не увидите ничего необычного, даже если заглянете в ее HTML-код. Чтобы понять, что в ней нового, посмотрите на исходный код в файле `hiJacob.php`

```
<html>
<head>
<title>Hi Jacob</title>
</head>
<body>
<h1>Hi Jacob</h1>
<h3>Demonstrates using a variable</h3>
<?
$userName = "Jacob";
print "Hi, $userName";
?>
</body>
</html>
```

При программировании на HTML и JavaScript вы можете использовать функцию браузера «показать исходный код» (view source) для просмотра текстов ваших программ. Но при программировании на языках, выполняемых на стороне сервера, этого недостаточно. В коде, который вы увидите с помощью этой функции, не будет PHP-команд. Не забывайте, что оригинальный программный код не попадает в браузер. Программа выполняется на сервере, а результаты ее работы отправляются в браузер в виде обычного HTML. Когда вы будете проверять свои программы, не забывайте открывать их оригинальный код с сервера. Точно так же вы не сможете загрузить в браузер страницу, написанную на PHP, с помощью меню File (Файл). Необходимо, чтобы сперва ее обработал сервер.

Если ваши страницы хранятся на вашем локальном сервере в каталоге для HTML-файлов, вы можете обращаться к ним по адресу `http://localhost/filename.php` для того, чтобы загрузить программу `filename`. Страница `hiJacob` написана большей частью на HTML, но включает в себя небольшой кусок PHP кода. И этот кусок выполняет очень важную работу.

Создаем строковую переменную

Строка `$userName = "Jacob";` выполняет несколько операций. Во-первых, создается переменная с именем `$userName`. На PHP все переменные начинаются со знака доллара ($), чтобы можно было их отличать от других элементов программы. Имя переменной имеет большое значение.

Назначаем имена переменным

Как программисту вам часто придется придумывать имена. Опытные программисты разработали некоторые рекомендации относительно того, как именно следует присваивать имена переменным и другим объектам.

- Имя должно быть «говорящим». Намного легче понять, что может означать `$userName`, чем, например, `$myVariable` или `$r`. Всегда, когда только возможно, давайте своим переменным имена, которые описывают, какие данные в них содержатся.
- Подумайте о длине имени. Оно должно быть достаточно длинным, чтобы быть «говорящим», но не слишком длинным, иначе вам надоест каждый раз его набирать.
- Не используйте пробелы. Большинство языков программирования (включая и PHP) не поддерживают использование пробелов в именах переменных.
- Не используйте знаки препинания. Большинство специальных символов, таких, как #, * или /, в языках программирования уже зарезервированы и потому не могут быть использованы в именах переменных. Конечно, в PHP каждая переменная начинается со знака $, но в остальном вам следует избегать использования знаков препинания. Единственное исключение представляет символ подчеркивания (_), который поддерживается в большинстве языков, в том числе и в PHP.
- Будьте осторожны с регистром букв. В PHP различаются большие и маленькие буквы, иначе говоря, язык PHP воспримет имена `$userName`, `$USERNAME`, и `$UserName` как три разные переменные. По соглашению, принятому для PHP, в именах переменных используются маленькие буквы, за исключением букв, разделяющих слова (как заглавная «N» в `$userName`). Я рекомендую следовать этому соглашению и сам придерживаюсь его в этой книге.
- Следите за правописанием! Каждый раз, когда вы обращаетесь к переменной, PHP проверяет, существует ли она уже в вашей программе. Если да, то используется существующая переменная. Если же нет, то без какого-либо сообщения создается новая. Поэтому если вы неверно напишете имя переменной, PHP не посчитает это ошибкой. Он создаст новую переменную, и ваша программа, скорее всего, будет работать неправильно.

Переменная не обязательно должна быть создана в явном виде. Когда вы обращаетесь к переменной, PHP создает ее автоматически.

Присваиваем значение переменной

Знак равенства (=) играет в PHP особую роль. Он не означает «равно» (по крайней мере, в том смысле, в котором мы его рассматриваем сейчас). Этот знак используется для присваивания. Если прочитать его как «принимает значение», вы получите почти точное представление о том, как он воспринимается языком PHP. Например, строку

```
$userName = "Jacob"
```

нужно читать так: «Переменная `$userName` принимает значение «Jacob».

Как правило, при создании на PHP переменной вы будете также присваивать ей какое-либо значение. Операция присваивания выполняется справа налево.

Переменной `$userName` было присвоено значение «Jacob». Компьютеры весьма требовательны к тому, данные какого типа записываются в переменную, но PHP этот процесс автоматизирует. Тем не менее, важно понимать, что «Jacob» является текстовым значением, поскольку текст хранится и обрабатывается в памяти компьютера несколько иным образом, чем числовые данные.

Программисты почти никогда не называют текст словом `text` *(текст). Они предпочитают использовать менее понятный термин* `string` *(нитка)*[1]*. Происхождение этого термина можно в какой-то мере назвать поэтическим, поскольку механизм хранения текста в памяти компьютера напоминал первым программистам процесс нанизывания бусин на нить.*

Выводим значение переменной на экран

Следующая строка кода выводит на экран сообщение. Вы можете поместить на экран любой текст, какой пожелаете. Текст (его также называют строковыми данными) обычно заключается в кавычки. Если вы хотите вывести значение переменной, просто поместите имя переменной в текст, который хотите вывести. Строка

```
print "Hi, $userName";
```

в результате выведет на экран

```
Hi, Jacob,
```

поскольку как только сервер встретит переменную `$userName`, он подставит взамен ее значение, в данном случае «Jacob». Результат работы PHP-программы будет отправлен непосредственно в веб-браузер, поэтому, если вы захотите, вы можете включать в выводимый текст даже HTML-теги, поместив их между кавычек.

---

[1] В настоящее время термин «string» принято переводить как «строка». – *Примеч. пер.*

Возможность выводить значение переменной внутри некоторого текста называется «строковой интерполяцией». Это не обязательно запоминать, хотя возможно эта информация пригодится вам при участии в какой-нибудь телевикторине.

Завершаем строку точкой с запятой

Если вы снова посмотрите на полный текст программы `hiJacob`, то можете заметить, что она включает в себя две строки кода внутри блока PHP. Каждая из этих строк оканчивается точкой с запятой. PHP – более строгий язык, чем HTML, и, как большинство языков программирования, имеет несколько обязательных правил синтаксиса, которых необходимо придерживаться при создании страниц.

Каждая отдельная команда должна оканчиваться точкой с запятой. В PHP этим знаком вы будете заканчивать почти каждую строку. Если вы забудете об этом, то получите сообщение об ошибке, как на рис. 2.4.

Если вы увидите это сообщение, внимательно просмотрите свой код и убедитесь, что в конце каждой строки поставили точку с запятой.

Временами команда бывает длиннее, чем одна строка в редакторе. Точка с запятой ставится после окончания команды, которое часто (хотя и не всегда) совпадает с окончанием строки.

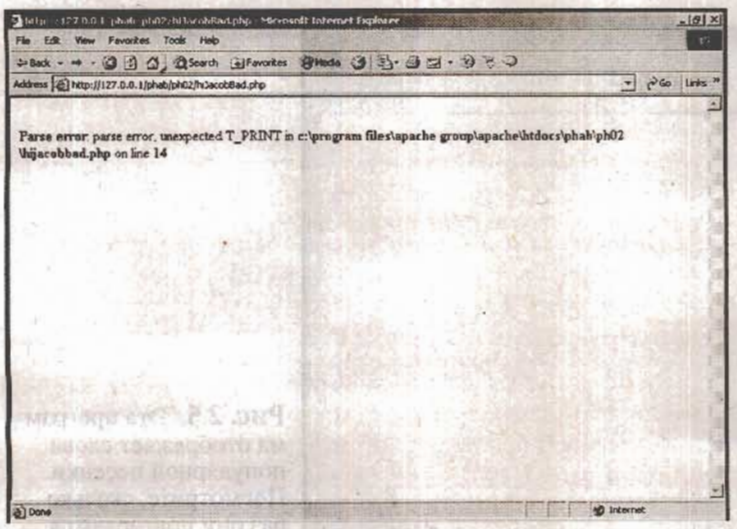


Рис. 2.4. Эта ошибка возникнет, если вы забудете поставить точку с запятой в конце каждой строки

Если программа вам выдаст пару сообщений об ошибке, не впадайте в панику. Они являются совершенной обычной частью программирования. Даже опытные программисты получают множество сообщений об ошибках при разработке и тестировании своих программ. Обычно код ошибки содержит важную информацию о том, что именно послужило ее причиной. Как следует проверьте строку кода, номер которой содержится в сообщении об ошибке. Хотя сама ошибка и не всегда находится именно в этой строке, ее внимательное изучение может подсказать вам причину сбоя. Во многих случаях (особенно в случае пропущенной точки с запятой) сообщение о синтаксической ошибке укажет на строку, следующую за той, в которой она допущена. Если вы получили сообщение о синтаксической ошибке в строке 14 и все дело в пропущенной точке с запятой, значит, вы забыли ее поставить в строке 13.

Используем переменные при создании более сложных страниц

Хотя программа `HiJacob` и продемонстрировала нам пример использования переменной, реальной выгоды от этого мы не получили. Настало время продемонстрировать вам, насколько полезным может оказаться применение переменных.

Создаем страницу с текстом песни

На рис. 2.5 показана страница с текстом песни «Row your boat» (Прокатись на своей лодке).

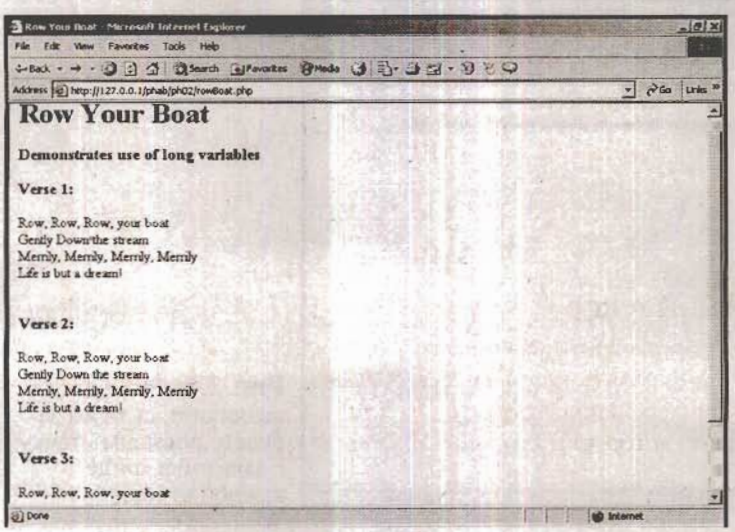


Рис. 2.5. Эта программа отображает слова популярной песенки. Посмотрите, сколько раз они повторяются

Я выбрал именно эту песню, потому что один и тот же куплет в ней повторяется три раза. Если вы посмотрите на исходный код программы `rowBoat.php`, то увидите, что с помощью небольшого трюка я избежал лишней работы по набору текста.

```
<html>
<head>
<title>Row Your Boat</title>
</head>
<body>
<h1>Row Your Boat</h1>
<h3>Demonstrates use of long variables</h3>
<?
$verse = <<<HERE
Row, Row, Row, your boat<br>
Gently down the stream<br>
Merrily, merrily, merrily, merrily<br>
Life is but a dream!<br>
<br><br>
HERE;
print "<h3>Verse 1:</h3>";
print $verse;
print "<h3>Verse 2:</h3>";
print $verse;
print "<h3>Verse 3:</h3>";
print $verse;
?>
</body>
</html>
```

Создаем многострочный текст

Довольно часто требуется вывести сразу несколько строк HTML-кода, и может оказаться утомительным обрамлять каждую строку кавычками (особенно потому, что в HTML они тоже нередко используются). В языке PHP существует специальный механизм цитирования, который идеально подходит в данной ситуации. Строка

```
$verse = <<<HERE
```

начинается с присваивания переменной `$verse` некоторого значения. Фрагмент `<<<HERE` показывает, что после него идет текст, занимающий несколько строк и заканчивающийся символом «`HERE`». Вы можете использовать вместо

HERE любую фразу на свое усмотрение, но я, как правило, пишу HERE. Другими словами, команду

```
$verse = <<<HERE
```

можно воспринимать так: «переменная verse получает в качестве значения весь текст вплоть до HERE».

Вы также можете считать <<<HERE особой открывающей кавычкой, закрывает которую только строка HERE.

Между <<<HERE и HERE вы можете поместить столько текста, сколько вам необходимо. Вы можете размещать в этом тексте переменные, и PHP запишет вместо них соответствующие значения, точно так же, как сделал бы это в обычной строке, заключенной в кавычки. Можете свободно использовать одинарные и двойные кавычки (" и '). Фраза-окончание (в данном случае HERE) должна занимать отдельную строку, и перед ней не должно быть пробелов.

У вас наверняка возник вопрос, почему строка `$verse = <<<HERE` *не оканчивается точкой с запятой. Хотя в редакторе она расположена на отдельной строке, именно с нее начинается сам многострочный текст. С технической точки зрения, все, что расположено между этой строкой и закрывающей конструкцией «*HERE;*», является частью одной логической строки, даже если этот код занимает в редакторе несколько строк. Внутри строки точка с запятой не выполняет никакой особой функции. Если такое объяснения непонятно, пусть вас это пока не беспокоит, так как возможность обдумать эту идею позже вам еще представится. Сейчас вы как минимум должны запомнить, что строка, начинающая многострочный текст, не требует после себя точки с запятой, а строка, которой такой текст завершается, требует.*

После того как многострочная конструкция создана, пользоваться ей очень легко. Фактически написание заголовков для куплетов потребует больше работы, чем написание самих куплетов. Оператор `print` просто размещает значение переменной `$verse` на подходящих местах в итоговом HTML.

Работаем с числовыми переменными

Компьютеры хранят данные в виде последовательностей единиц и нулей. Эти последовательности могут быть преобразованы в несколько более удобных типов представления информации. Для большинства ситуаций язык PHP скрывает от вас эти преобразования, но, тем не менее, необходимо знать, что строка (текст) обрабатывается в памяти не так, как числовые значения, а кроме того, существуют два основных вида числовых значений.

Создаем программу ThreePlusFive

В качестве примера того, как PHP обрабатывает числа, взгляните на программу `ThreePlusFive.php`, результат работы которой показан на рис. 2.6.

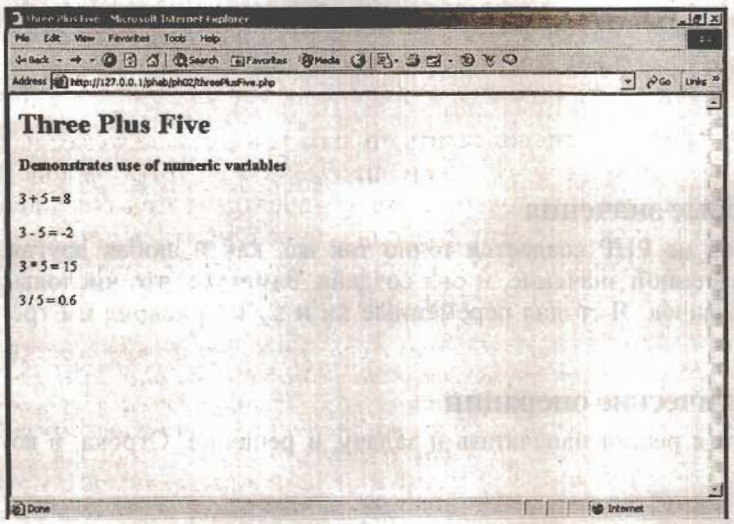


Рис. 2.6. Эта программа производит основные математические операции над переменными, содержащими значения 3 и 5

Вся работа в программе ведется с двумя переменными, которые называются `$x` и `$y`. (Я помню, что советовал вам назначать переменным более длинные и «говорящие» имена, но эти переменные часто используются при решении арифметических задач, так что в данном случае столь короткие имена вполне допустимы.) Код нашей программы выглядит так:

```
<html>
<head>
<title>Three Plus Five</title>
</head>
<body>
<h1>Three Plus Five</h1>
<h3>Demonstrates use of numeric variables</h3>
<?
$x = 3;
$y = 5;
print "$x + $y = ";
print $x + $y;
print "<br><br>";
print "$x - $y = ";
print $x - $y;
print "<br><br>";
```

```
print "$x * $y = ";
print $x * $y;
print "<br><br>";
print "$x / $y = ";
print $x / $y;
print "<br><br>";
?>
</body>
</html>
```

Присваиваем числовые значения

Числовая переменная на PHP создается точно так же, как и любая другая. Просто присвойте переменной значение, и она создана. Заметьте, что числовые значения не требуют кавычек. Я создал переменные `$x` и `$y` и присвоил им требуемые значения.

Используем математические операции

Для каждого расчета я решил напечатать и задачу, и решение. Строка, в которой содержится

```
print "$x + $y = ";
```

выводит значения переменных `$x` и `$y` и знак сложения между ними. В данном случае, так как переменной `$x` было присвоено значение 3, а переменной `$y` – 5, эта команда выведет следующее:

```
3 + 5 =
```

Знак сложения и знак равенства расположены внутри кавычек, поэтому воспринимаются они как обычные элементы текста и PHP не производит никаких расчетов (ни сложения, ни присваивания).

Следующая строка

```
print $x + $y;
```

не содержит кавычек. В ней рассчитывается значение `$x + $y` и результат этой операции (8) выводится на веб-страницу.

Большинство известных вам математических символов применимы для числовых переменных. Знак «плюс» (+) используется для сложения, знак «минус» (–) указывает на вычитание, звездочка (*) означает умножение, а косая черта (/) – деление. Оставшаяся часть программы иллюстрирует то, как PHP производит вычитание, умножение и деление.

Возвращаясь к реальности

Числа, не имеющие дробной части, называются целыми, а числа с дробной частью (например, 1.5, 0.333 и т. д.) – действительными. Компьютер по-разному хранит их в памяти, из-за чего иногда возникают трудности. Язык PHP старается оградить вас от необходимости явно определять тип числа. Например, если взять два целых числа, 3 и 5, то результат их сложения, вычитания или умножения тоже будет целым. Однако частное, полученное при делении двух целых чисел, часто может оказаться действительным. Во многих языках программирования задача деления либо вовсе не будет решена, либо результат будет получен не полностью. Например, в результате деления 3 на 5 может получиться 0 вместо 0.6. PHP старается преобразовывать типы во всех случаях, когда это можно сделать, и обычно ему это неплохо удается. Тем не менее, бывают случаи, когда вам необходимо проконтролировать эти преобразования. Это делается с помощью функции `setType()`, которая явно приводит переменную к определенному типу. Подробно о ее применении вы можете прочитать в системе интерактивной справки языка PHP. Примеры исходного кода вы найдете на нашем сайте.

Создаем форму, задающую пользователю вопрос

Программы, написанные на PHP, как правило, состоят из двух и более отдельных документов. Страница, написанная на обычном HTML, содержит форму, которую пользователь вначале заполняет, а затем нажимает кнопку «Отправить». После этого информация из формы передается в назначенную для нее программу, где производится ее обработка, после чего пользователю возвращается результат в виде обычной веб-страницы. В качестве примера посмотрите на страницу `whatsName.html`, показанную на рис. 2.7.

Страница `whatName.html` не содержит ни строчки кода, написанного на PHP. Это просто HTML-страница с формой. Когда пользователь щелкает мышкой по кнопке «Submit Query», страница передает данные из текстовой области в PHP-программу, которая называется `hiUser.php`. На рис. 2.8 показано, что получается в результате работы программы `hiUser.php`.:

Важно осознавать, что в ходе этой операции используется две разных страницы. В этом разделе вы научитесь связывать HTML-страницу с конкретным сценарием, а также создавать сценарии, которые запрашивают информацию из определенной формы.

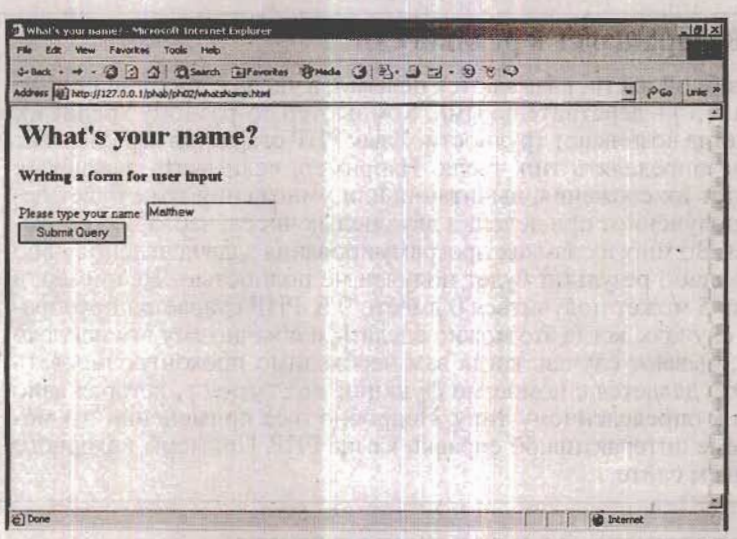


Рис. 2.7. Обычная HTML-страница с формой

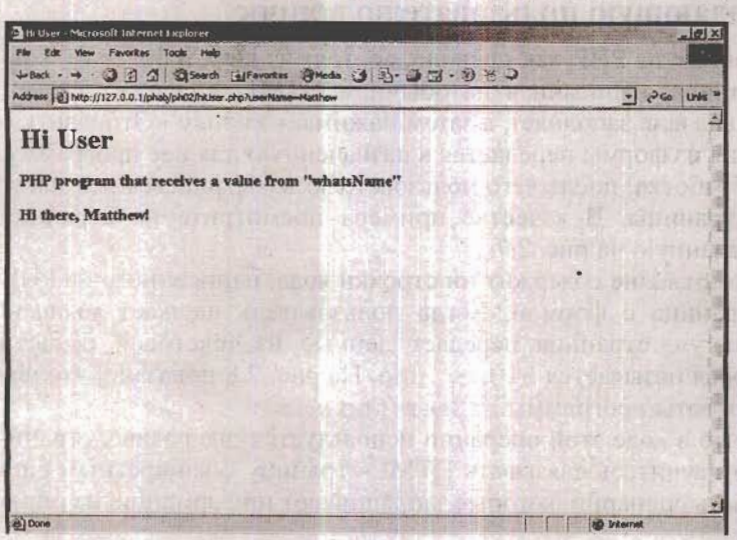


Рис. 2.8. При создании этой страницы использованы данные, которые были внесены в первую HTML-форму

Создаем HTML-страницу с формой

Формы бывают очень полезны, когда необходимо получить данные от пользователя. Чтобы увидеть, как это делается, посмотрите на код из файла `whatsName.html`.

```
<html>
<head>
<title>What's your name?</title>
</head>
<body>
<h1>What's your name?</h1>
<h3>Writing a form for user input</h3>
<form method = "post"
      action = "hiUser.php">
Please type your name:
<input type = "text"
       name = "userName"
       value = "">
<br>
<input type = "submit">
</form>
</body>
</html>
```

Только один элемент на этой странице может быть вам незнаком. Посмотрите внимательно на тег `form`. Он содержит два новых атрибута. Атрибут `method` устанавливает, каким образом данные будут отправлены на сервер. Существует два основных метода: `get` и `post`. Метод `post` является более мощным и гибким, поэтому чаще всего в этой книге я буду использовать именно его. Однако чуть дальше по ходу этой главы, в разделе «Отправляем данные без использования формы», вы узнаете несколько интересных способов использования метода `get`.

Назначаем для файла сценария атрибут Action

Другой атрибут тега `form` – это атрибут `action`. С его помощью устанавливают URL (Uniform resource locator – унифицированный указатель информационного ресурса) программы, которая будет обрабатывать форму. Этот атрибут связывает веб-страницу с программой, которая считывает с нее данные и выдает результат в виде другой страницы. В URL может быть записана абсолютная ссылка (начинается с `http://` и содержит полное доменное имя обрабатывающей про-

граммы) или относительная ссылка (это означает, что программа находится в том же каталоге, что и исходная веб-страница[1]).

Страница `whatsName.html` содержит форму, атрибуту `action` которой присвоено значение `hiUser.php`. Как только пользователь нажимает на кнопку отправки формы, значения всех ее полей (в данном случае единственного поля) собираются в один пакет и передаются в программу с названием `hiUser.php`, которая по умолчанию должна быть расположена в том же каталоге, что и исходная страница `whatsName.html`.

Создаем сценарий для извлечения данных

Код программы `hiUser.php` написан особым образом. Форма, вызывающая эту программу, должна содержать элемент с именем `userName`. Посмотрите на код в файле `hiUser.php`, и вы поймете, что я имею в виду.

Возвращаясь к реальности

На некоторых PHP-серверах функция автоматического создания переменной из формы бывает выключена. Возможно, вам удастся уговорить администратора сервера установить для параметра `register_globals` в файле PHP.INI значение `on`. Если нет, существует обходной путь: если на вашей форме присутствует поле с именем userName, добавьте в начало программы, получающей значение из этого поля, следующий код

```
$userName = $_REQUEST["userName"];
```

Повторите это выражение для каждой переменной, значение которой вам необходимо извлечь из формы.
Исчерпывающее объяснение того, как работает эта строка кода, вы получите в главе 5. В ней вы также научитесь извлекать значения всех полей из формы, даже не зная их имен

```
<html>
<head>
<title>Hi User</title>
</head>
<body>
<h1>Hi User</h1>
<h3>PHP program that receives a value from "whatsName"</h3>
<?
```

[1] В общем случае, относительные ссылки применяются тогда, когда необходимо указать на файл, расположенный на том же компьютере, на котором находится ссылающийся документ. – *Примеч. пер.*

```
print "<h3>Hi there, $userName!</h3>";
?>
</body>
</html>
```

Как и многие PHP-страницы, `hiUser.html` большей частью написана на HTML. Единственный оператор языка PHP – это `print`. Он включает в себя переменную `$userName`. Главная загадка состоит в том, что данная переменная ни до, ни после этого не упоминается в коде.

Когда пользователь передает форму в программу, интерпретатор PHP для каждого элемента формы исходной HTML-страницы автоматически создает переменную с таким же именем. Поскольку на странице `whatsName.html` присутствует элемент с именем `userName`, любая PHP-программа, к которой обратится страница `whatsName.html`, автоматически получит доступ к переменной с именем `$userName`. Эта переменная будет содержать значение, которое пользователь введет в поле перед тем, как нажать кнопку отправки формы.

Отправляем данные без использования форм

Иногда бывает удобно отправить данные в программу, хранящуюся на сервере, без использования формы. Немногие знают о такой тонкости, а между тем с ее помощью можно существенно расширить возможности ваших страниц без единой строчки PHP-кода. Страница linkDemo.html, показанная на рис. 2.9 и 2.10, служит иллюстрацией этого необычного явления.

Как работает метод get

Все ссылки на странице `linkDemo.html` работают похожим образом. Как мы уже говорили в этой главе, данные из формы могут быть переданы в программу двумя различными методами. Как правило, вы будете пользоваться методом `post` при отправке данных из форм, однако все это время вы использовали метод get, так как стандартные HTML-запросы передаются именно по методу `get`. Наиболее интересным моментом является то, что, включив запрос по методу `get` в URL, можно передать данные из формы в любую программу, умеющую обрабатывать такие запросы. В качестве эксперимента попробуйте изменить атрибут `method` в файле `whatsName.html` следующим образом.

```
<form method = "get"
      action = "hiUser.php">
```

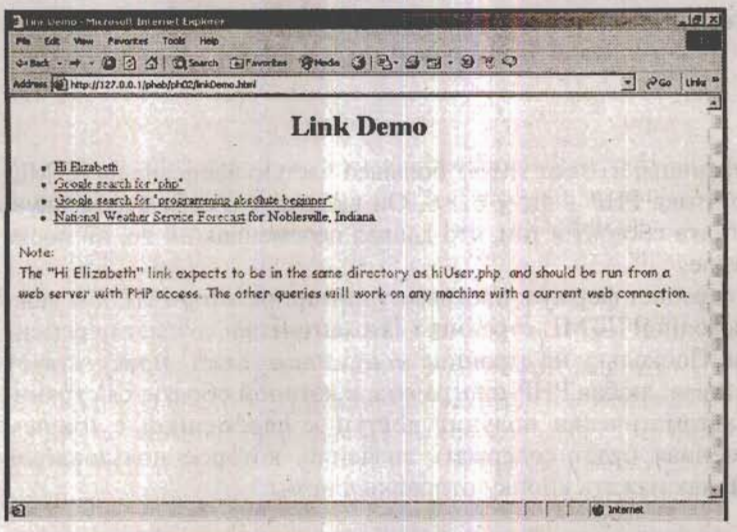


Рис. 2.9. Ссылки на этой странице выглядят как обычно, но обладают удивительными возможностями

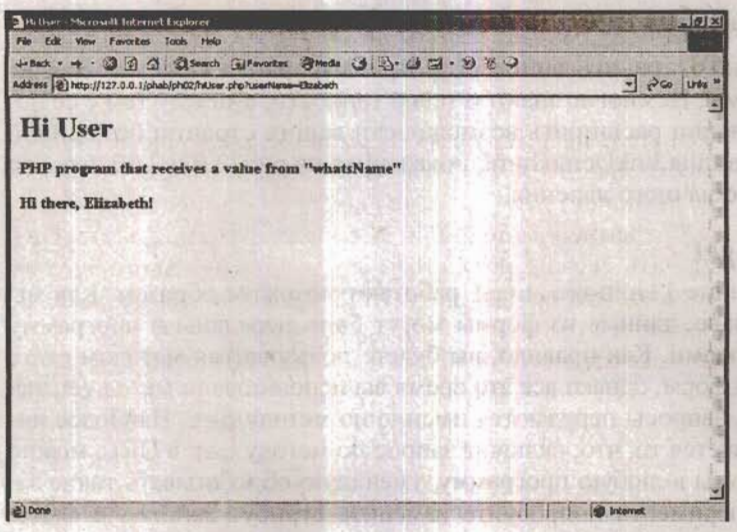


Рис. 2.10. Щелкнув по ссылке «`Hi Elizabeth`», я попал в программу HiUser, в которую автоматически было передано значение «`Elizabeth`»

Откройте эту страницу снова. Она будет работать как и раньше, но URL итоговой страницы станет примерно таким (если считать, что вы выбрали имя пользователя «Andy»):

```
http://127.0.0.1/phab/ph02/hiUser.php?userName=Andy
```

Метод `get` присоединяет все данные из формы к URL в особом формате. Если вы вернетесь к странице `whatsName` и введете «Andy Harris», то результат получится несколько иным:

```
http://127.0.0.1/phab/ph02/hiUser.php?userName=Andy+Harris
```

Пробел между «Andy» и «Harris» был преобразован в плюс, поскольку пробелы вызывают много путаницы. Данные при передаче из формы, как правило, претерпевают несколько подобных изменений. При программировании на PHP все преобразования производятся автоматически, поэтому вам не нужно об этом беспокоиться.

Используем URL для передачи данных из форм

Если вы научитесь пользоваться данным методом, то сможете управлять любой Интернет-программой, выполняемой на стороне сервера. (В случае если она умеет принимать данные по методу `get`.) Исследовав несколько URL запросов к поисковой машине Google, я обнаружил, что данные, вводимые мной при поиске, содержатся в поле с именем «q» (вероятно, от английского «query» – запрос). Я предположил, что остальные поля имеют значения по умолчанию, и написал гиперссылку, включающую в себя такой запрос. Она выглядела следующим образом

```
<li><a href = "http://www.google.com/search?q=php">
Google search for "php"</a></li>
```

Всякий раз, когда пользователь щелкает по этой ссылке, она отправляет запрос по методу `get` к поисковой машине Google. В итоге пользователь попадает на страницу с результатами поиска. Вы можете захотеть написать страницу с заранее подготовленными версиями ваших наиболее частых запросов к поисковым машинаи, чтобы получать свежие результаты посредством одного щелчка мышью. На рис. 2.11 показано, что происходит, когда пользователь щелкает по ссылке «google php» на странице `linkDemo`.

Рис. 2.12 показывает результаты этого, чуть более сложного поиска.

```
<li><a href =
    "http://www.google.com/search?q=programming for the absolute
beginner">
     Google search for "programming absolute beginner"</a></li>
```

У такого подхода есть и свои недостатки. Владелец программы может изменить ее, не сообщив об этом вам, и ваша ссылка начнет работать неправильно. Большинство веб-программистов исходят из того, что их программы будут вызываться только из тех форм, которые они же и написали. Небходимо также помнить, что подобные трюки могут быть проделаны и с вашими программами. Вы рассчитываете, что вашу программу будет вызывать только ваша форма, однако это может оказаться не так. В этом вся сущность свободного Интернета.

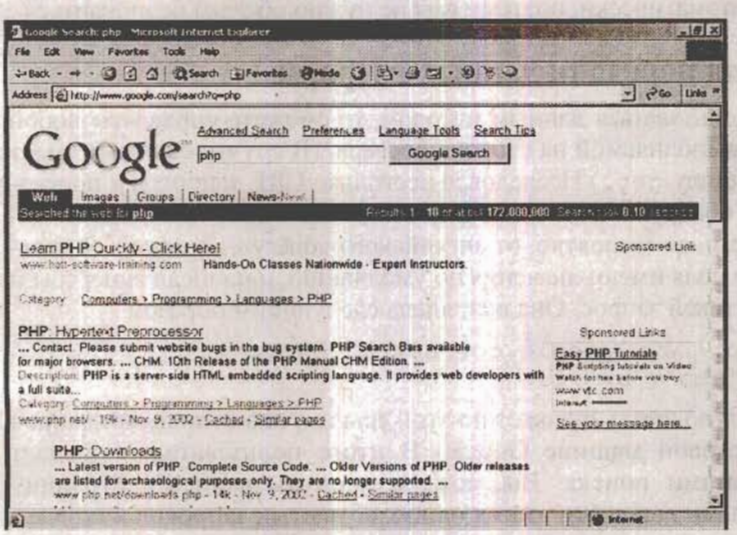


Рис. 2.11. Ссылка `Google PHP` запускает на сервере www.google.com поиск по ключевому слову PHP

Как работают запросы с несколькими полями

В качестве более применимого в реальной жизни примера рассмотрим ссылку на Национальную метеорологическую службу.

```
<li><a href = "http://www.crh.noaa.gov/data/forecasts/
INZ039.php?warncounty=INC057&city=Noblesville">
National Weather Service Forecast</a>
for Noblesville, Indiana.
```

Возможно, она покажется вам более сложной, но никаких специальных знаний для ее написания не потребовалось. Я просто нашел на сайте Национальной метеорологической службы автоматически создаваемую страницу с прогнозом погоды для моего города. Посмотрев на итоговый URL, я с удовлетворением (но без удивления) отметил, что он получен при помощи PHP-сценария. (Обратите внимание на расширение `.php` в имени файла). Я скопировал ссылку в своем браузере и поместил ее в код страницы `linkDemo.html`. Страница с прогнозом погоды создается PHP-сценарием

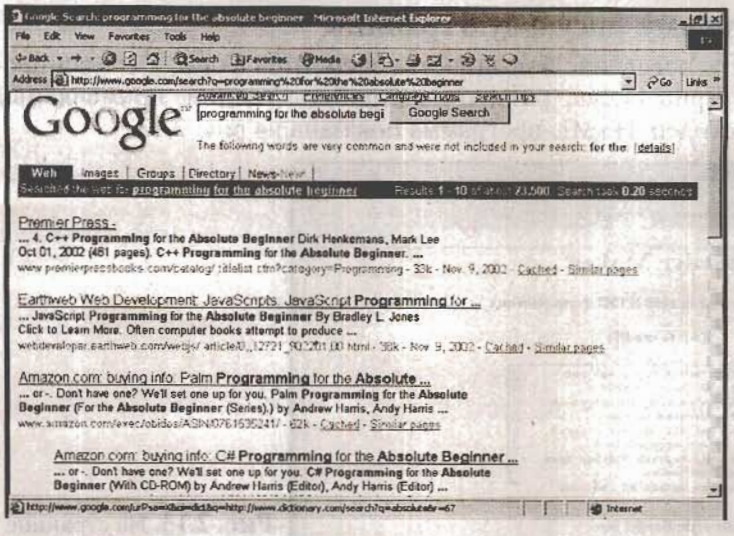


Рис. 2.12. В результатах поиска по фразе «Absolute Beginners Programming» содержится немало предложений о покупке интересных книг!

на основе двух входных параметров (название страны и города). Когда бы я ни захотел узнать местный прогноз погоды, я могу вызвать все тот же запрос, несмотря на то что он будет исходить уже не со страницы Национальной метеорологической службы. Таким образом можно очень легко переделать вашу веб-страницу.

В предыдущем параграфе я упомянул, что данная программа на PHP требует ввода двух параметров. На самом деле, я никогда не видел кода этой программы, но мне стало это известно после того, как я внимательно изучил URL. Подстрока `warncounty=INCO57` содержит название страны и штата (и это предположение очень похоже на истину), а подстрока `city=Noblesville` указывает на конкретный город в этой стране. Когда на форме расположено два или более элемента, они соединяются между собой при помощи символа &, как вы могли заметить в примере с Национальной метеорологической службой.

Извлекаем данные из других элементов форм

Программа на PHP способна считать данные из любого элемента HTML-формы. Во всех случаях атрибут `name` объекта HTML-формы становится на PHP именем переменной. В общем случае этой переменной присваивается значение атрибута `value` объекта формы.

Представляем программу, рисующую границы

Чтобы лучше изучить поведение большинства элементов форм, я создал простую страницу, демонстрирующую различные атрибуты границ, применяемые в каскадных таблицах стилей. HTML-программа показана на рис. 2.13.

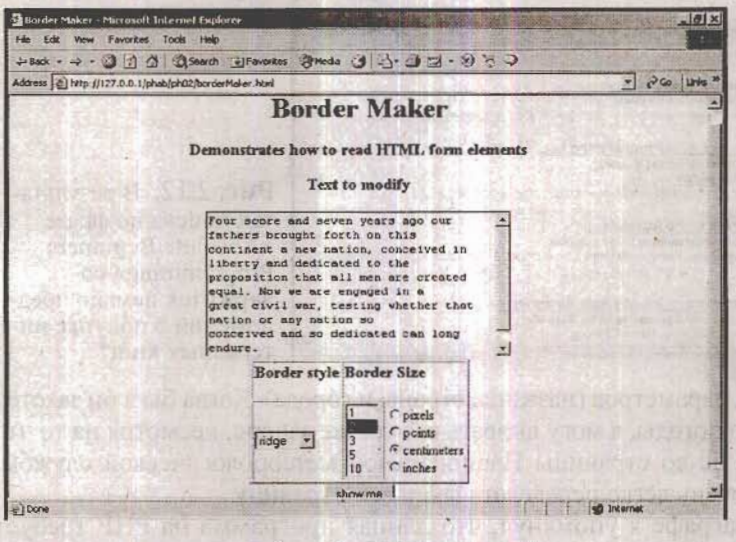


Рис. 2.13. На странице программы, рисующей границы, размещена текстовая область, два списка и группа элементов выбора

Создаем страницу borderMaker.html

Страница `borderMaker.html` содержит самую обыновенную форму, вклю чающую в себя большинство основных элементов ввода. Код этой страницы приведен ниже.

```
<html>
<head>
<title>Font Choices</title>
</head>
<body>
<center>
<h1>Font Choices</h1>
<h3>Demonstrates how to read HTML form elements</h3>
<form method = "post"
      action = "borderMaker.php">
<h3>Text to modify</h3>
<textarea name = "basicText"
          rows = "10"
          cols = "40">
```

```
Four score and seven years ago our fathers brought forth on this
continent a new nation, conceived in liberty and dedicated to the
proposition that all men are created equal. Now we are engaged in a
great civil war, testing whether that nation or any nation so
conceived and so dedicated can long endure.
</textarea>
<table border = 2>
<tr>
  <td><h3>Border style</h3></td>
  <td colspan = 2><h3>Border Size</h3></td>
</tr>
<tr>
<td>
<select name = borderStyle>
  <option value = "ridge">ridge</option>
  <option value = "groove">groove</option>
  <option value = "double">double</option>
  <option value = "inset">inset</option>
  <option value = "outset">outset</option>
</select>
</td>
<td>
<select size = 5
        name = borderSize>
  <option value = "1">1</option>
  <option value = "2">2</option>
  <option value = "3">3</option>
  <option value = "5">5</option>
  <option value = "10">10</option>
</select>
</td>
<td>
<input type = "radio"
       name = "sizeType"
       value = "px">pixels<br>
<input type = "radio"
       name = "sizeType"
       value = "pt">points<br>
<input type = "radio"
       name = "sizeType"
       value = "cm">centimeters<br>
<input type = "radio"
```

```
        name = "sizeType"
        value = "in">inches<br>
</td>
</tr>
</table>
<input type = "submit"
        value = "show me">
</form>
</center>
</body>
</html>
```

Страница `borderMaker.html` разработана так, чтобы она могла взаимодействовать с программой на PHP. Эта программа называется `borderMaker.php`, как вы могли заметить, посмотрев на значение атрибута `action`. Обратите внимание на то, что для каждого элемента `option` я назначил атрибут `value`, а кнопки-переключатели имеют одно и то же имя, но разные значения. Атрибут `value` становится необычайно важным, когда данные из вашей программы должны быть переданы в другую программу, и вскоре вы в этом убедитесь. И наконец, нельзя забывать о кнопке типа `submit`, поскольку главное действие начнется лишь тогда, когда пользователь отправит форму.

Возможно, вы заметили, что в этом примере нет флажков. Они работают очень похоже на все прочие элементы, но на практике пользы от них будет больше, когда вы изучите операторы условия, которые будут главной темой следующей главы. Когда мы до нее дойдем, вам представится немало возможностей поэкспериментировать с данными элементами.

Получаем данные из элементов форм

Программа `borderMaker.php` ожидает данные со страницы `borderMaker.html`. Когда пользователь отправляет HTML-форму, PHP-программа выдает результат, подобный представленному на рис. 2.14.

В общем случае тип элемента, помещенного на HTML-форму, не играет никакой роли. Интерпретатор PHP просто обрабатывает имя каждого элемента и его значение. К тому моменту, когда эти данные попадают на сервер, уже не имеет значения, элементы какого типа были использованы. PHP автоматически создает переменную, соответствующую каждому элементу формы. Значением этой переменной становится значение элемента. Код программы `borderMaker.php` иллюстрирует сказанное.

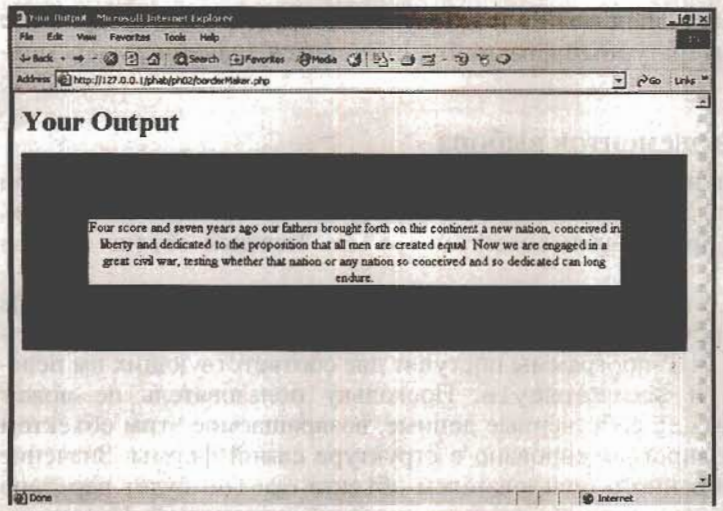


Рис. 2.14. Код программы `borderMaker.php` реагирует на значения всех элементов форм

```
<html>
<head>
<title>Your Output</title>
</head>
<body>
<h1>Your Output</h1>
<center>
<?
$theStyle = <<<HERE
"border-width:$borderSize$sizeType;
border-style:$borderStyle;
border-color:green"
HERE;
print "<div style = $theStyle>";
print $basicText;
print "</span>";
?>
</center>
</body>
</html>
```

В случае с текстовыми полями и текстовыми областями пользователь вводит данные вручную. На странице `borderMaker.html` существует текстовая область с именем `basicText`. Интерпретатор PHP создает переменную с именем

$basicText. Любой текст, вводимый в текстовую область (по умолчанию в нее помещен небольшой литературный фрагмент), становится значением переменной $basicText.

Получаем данные из элементов выбора

Если вы помните, и выпадающие, и обыкновенные списки создаются при помощи объекта select. У этого объекта есть атрибут name. Каждый из возможных выборов в списке представлен объектом option. Каждый объект option имеет атрибут value.

Имя объекта select становится именем переменной. Например, на странице borderMaker.html присутствует два объекта выбора: borderSize и borderStyle. В распоряжение PHP-программы поступят две соответствующих им переменные: $borderSize и $borderStyle. Поскольку пользователь не может ввести в объект типа select собственные данные, возвращаемое этим объектом значение должно быть запрограммировано в структуре самой формы. Значение атрибута value для выбранного пользователем объекта option будет передано в PHP-программу в качестве значения соответствующей переменной. Например, если пользователь выберет в качестве стиля границы groove, переменной $borderStyle будет присвоено это же значение.

Возвращаясь к реальности

Заметьте, что атрибут value элемента option не обязательно должен содержать то же значение, которое пользователь видит на экране. Это бывает удобно в тех случаях, когда пользователю необходимо показать одно значение, а на сервер отправить другое. Например, вы можете предложить пользователю выбрать цвет из списка возможных. Для этого можно создать список, в котором будут показаны их названия, и в то же время атрибут value для каждого элемента option может содержать шестнадцатеричное значение, с помощью которого этот цвет отображается. Подобные трюки используются в Интернет-магазинах, когда пользователь выбирает покупки по их наименованиям, а значение, связанное с каждым именем, может быть номером по каталогу, на основании которого требуемый предмет значительно легче найти в базе данных.

Вы можете вспомнить о том, что элемент «список» допускает возможность выбора нескольких вариантов. В этом случае переменная будет содержать список ответов. И хотя управлять такими списками совсем не сложно, однако этим мы займемся в отдельной главе, а именно главе 4. Пока что мы будем работать с единственным выбором в списке.

Получаем данные из групп переключателей

С помощью CSS разработчик может установить значения самых разных величин. В данном месте идеально подходит группа переключателей, поскольку в каждый момент времени может изменяться только одна из данных величин. И хотя на странице `borderDemo.html` расположено целых четыре переключателя с именем `sizeType`, PHP-программа в итоге получит только одну переменную `$sizeType`. Значением этой переменной станет значение атрибута `value` выбранного пользователем варианта. Обратите внимание, что, как и в случае с элементами `option`, значение переключателя может отличаться от текста, расположенного рядом с ним.

Возвращаясь к реальности

Как выбрать элемент, который следует использовать?

Сейчас вы наверняка задаете себе вопрос: так ли необходимы все эти разнообразные элементы форм, если к тому моменту, когда дело доходит до интерпретатора PHP, все сводится к имени и значению переменной. Однако использование различных элементов полезно по нескольким причинам.

- Во-первых, многим пользователям проще работать с мышкой, чем вводить текст с клавиатуры. Поэтому везде, где это возможно, лучше использовать различные элементы форм (списки, флажки), предоставляя пользователю готовые варианты выбора, чтобы он мог быстрее перемещаться по вашим формам. Ввод текста часто занимает значительно больше времени, чем управление прочими элементами.
- Во-вторых, элементы интерфейса (особенно выпадающие списки) очень экономно расходуют пространство экрана. При правильном использовании выпадающих списков на одном маленьком экране можно разместить множество элементов. И если вы считаете, что в настоящее время эта проблема потеряла свою актуальность, посмотрите, как много людей выходят в Интернет с карманных компьютеров и сотовых телефонов.
- В-третьих, вы, как программист, сильно облегчите свою жизнь, если всегда будете точно знать, что именно пришлет вам пользователь. Когда пользователям приходится вводить текст, они часто делают грамматические ошибки, опечатываются, используют нелепые сокращения и вообще ведут себя непредсказуемо. Если ограничить пользователям свободу выбора везде, где это только возможно, уменьшится вероятность того, что они останутся недовольны, поскольку ваша программа сможет адекватно обработать все возможные варианты выбора

Возвращаемся к программе, рассказывающей сказки

Программа, рассказывающая сказки, которую мы представляли в начале этой главы, дает вам возможность собрать в единое целое все новые элементы, которые вы изучили. В ней нет ничего нового, но она поможет вам оценить их работу в более крупном проекте.

Планируем программу

Эту программу нельзя назвать чересчур сложной, но если вы попытаетесь немедленно открыть текстовый редактор и углубиться в написание кода, то столкнетесь с трудностями. Сначала необходимо все спланировать. Наиболее важные моменты должны быть продуманы до того, как вы напишете хотя бы строку кода.

В данном случае начните планирование с выбора сказки. Вы можете написать ее сами, а можете изменить существующий текст, сделав его более смешным. Свою сказку я нашел в одной детской книжке. Какой бы путь вы ни выбрали, ваша сказка должна быть готова до того, как вы приступите к программированию. Вначале я набрал в текстовом редакторе оригинальный текст сказки «Little Boy Blue», для того чтобы в полной мере насладиться ее художественными достоинствам,– а потом изменить до неузнаваемости. По мере прочтения исходного текста вашей сказки ищите в ней ключевые слова, которые можно убрать, подобрав взамен описание, которое бы подсказывало, что за слово требуется, не называя его. Например, я распечатал свою сказку и обвел в тексте слово «blue» (синий), написав взамен на отдельном листке бумаги «color» (цвет). Пройдите по тексту всей сказки и найдите несколько подобных слов, которые можно исключить. У вас должен получиться документ, из которого удалены некоторые слова, и список подсказок для этих слов. То, что вышло у меня, показано на рис. 2.15.

Возвращаясь к реальности

На рис. 2.15 план показан в виде документа Microsoft Word. И хотя программисты (особенно профессиональные) зачастую создают именно такой документ, на самом деле я написал свой план на бумаге и только потом перевел его в более понятный формат, чтобы избавить вас от необходимости разбирать мой почерк. Обычно я планирую свои программы на бумаге или доске, но не на компьютере – слишком велик соблазн немедленно погрузиться в программирование. Очень важно сначала написать простым русским языком, чего вы хотите добиться, и только потом это реализовывать. Большинство новичков (и многие профессионалы) начинают программировать слишком рано, и в итоге заходят в тупик. Остаток этой главы посвящен тому, как план постепенно превращается в работающую программу

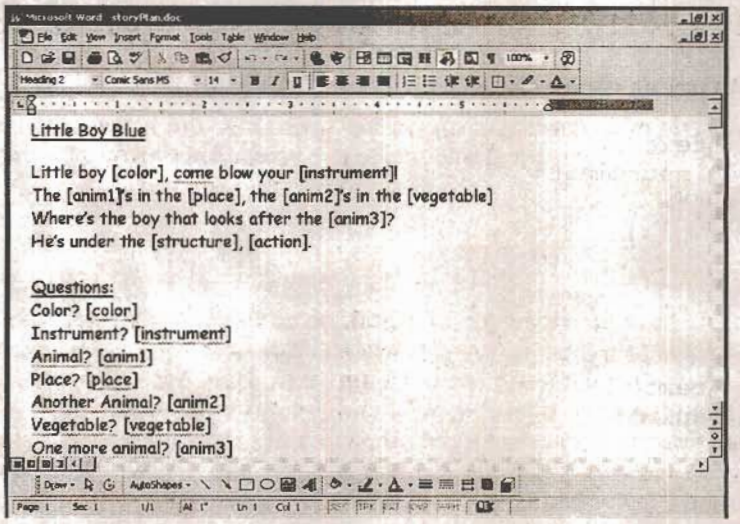


Рис. 2.15. Мой план для написания этой программы. Я выбрал сказку и продумал список слов до начала программирования

Создаем HTML-страницу

После ознакомления с общим планом, показанным на рис. 2.15, становится понятно, как следует создавать нашу программу. Она должна состоять из двух частей. Первая часть – это HTML-страница с подсказками, на которой пользователь будет вводить различные слова. Вот как выглядит код в моей версии.

```
<html>
<head>
<title>Story</title>
</head>
<body>
<h1>Story</h1>
<h3>Please fill in the blanks below, and I'll tell
    you a story</h3>
<form method = "post"
      action = "story.php">
<table border = 1>
<tr>
  <th>Color:</th>
  <th>
    <input type = "text"
           name = "color"
           value = "">
  </th>
```

```
</tr>
<tr>
  <th>Musical Instrument</th>
  <th>
    <input type = "text"
           name = "instrument"
           value = "">
  </th>
</tr>
<tr>
  <th>Animal</th>
  <th>
    <input type = "text"
           name = "anim1"
           value = "">
  </th>
</tr>
<tr>
  <th>Another animal</th>
  <th>
    <input type = "text"
           name = "anim2"
           value = "">
  </th>
</tr>
<tr>
  <th>Yet another animal!</th>
  <th>
    <input type = "text"
           name = "anim3"
           value = "">
  </th>
</tr>
<tr>
  <th>Place</th>
  <th>
    <input type = "text"
           name = "place"
           value = "">
  </th>
</tr>
```

```
<tr>
  <th>Vegetable</th>
  <th>
    <input type = "text"
           name = "vegetable"
           value = "">
  </th>
</tr>
<tr>
  <th>A structure</th>
  <th>
    <input type = "text"
           name = "structure"
           value = "">
  </th>
</tr>
<tr>
  <th>An action</th>
  <th>
    <select name = "action">
      <option value = "fast asleep">fast asleep</option>
        <option value = "drinking cappuccino">drinking cappuc-
cino</option>
       <option value = "wandering around aimlessly">wandering
around aimlessly</option>
     <option value = "doing nothing in particular">doing nothing
in particular</option>
    </select>
  </th>
</tr>
<tr>
  <td colspan = 2>
    <center>
      <input type = "submit"
             value = "tell me the story">
    </center>
  </td>
</tr>
</table>
</form>
</body>
</html>
```

Этот HTML-код не содержит ничего нового или необычного. Фактически, поскольку я составил план, мне было точно известно, что именно я собираюсь спросить у пользователя, и я просто создавал элементы для каждого вопроса. В последнем вопросе я использовал список, чтобы поместить в него несколько интересных вариантов. Обратите внимание на то, что я слегка изменил порядок вопросов, чтобы пользователь не смог догадаться обо всем заранее.

При создании страницы, обращающейся к сценарию, существует несколько моментов, которые необходимо проверять. Во-первых, атрибуту `action` элемента `form` должно быть присвоено правильное значение. (Разумеется, для этого сперва нужно создать этот атрибут.) Для каждого элемента формы необходимо установить атрибут `name`. Если у вас есть объекты-переключатели или элементы `option`, убедитесь, что каждому из них присвоено верное значение. Наконец, не забудьте поместить на форму кнопку типа `submit`.

Проверяем форму

В действительности я создал два разных сценария для обработки этой формы. Первый просто проверял каждый элемент, чтобы убедиться, что тот получает ожидаемое значение. Вот текст первой программы, которая называется `storySimple.php`.

```
<html>
<head>
<title>Little Boy Who?</title>
</head>
<body>
<h1>Little Boy Who?</h1>
<h3>Values from the story page</h3>
<table border = 1>
<tr>
  <th>Variable</th>
  <th>Value</th>
</tr>
<tr>
  <th>color</th>
  <td><? print $color ?></td>
</tr>
<tr>
  <th>instrument</th>
  <td><? print $instrument ?></td>
</tr>
```

```
<tr>
  <th>anim1</th>
  <td><? print $anim1 ?></td>
</tr>
<tr>
  <th>anim2</th>
  <td><? print $anim2 ?></td>
</tr>
<tr>
  <th>anim3</th>
  <td><? print $anim3 ?></td>
</tr>
<tr>
  <th>place</th>
  <td><? print $place ?></td>
</tr>
<tr>
  <th>vegetable</th>
  <td><? print $vegetable ?></td>
</tr>
<tr>
  <th>structure</th>
  <td><? print $structure ?></td>
</tr>
<tr>
  <th>action</th>
  <td><? print $action ?></td>
</tr>
</table>
<form>
</html>
```

Я старался сделать эту программу максимально простой, поскольку знал, что нужна она мне будет недолго. Это просто таблица, в которую помещено имя каждой переменной и ее значение. Я создал ее для того, чтобы убедиться, что каждая переменная получает именно то значение, которое мне необходимо. Нет ни малейшего смысла в создании сказки, если переменные работают не так, как ожидается.

Создаем готовую сказку

Если план написан и переменные работают так, как ожидается, создание самой сказки не представляет ни малейшего труда. Все, что мне нужно было сделать, – это написать текст сказки так, как это было запланировано, и включить в него переменные на подходящих местах. В результате появился следующий код, который можно найти в файле `story.php`.

```
<html>
<head>
<title>Little Boy Who?</title>
</head>
<body>
<center>

<h1>Little Boy Who?</h1>

<?

print <<<HERE
<h3>
Little Boy $color, come blow your $instrument!<br>
The $anim1's in the $place, the $anim2's in the $vegetable.<br>
Where's the boy that looks after the $anim3?<br>
He's under the $structure, $action.
</h3>
HERE;
?>

</center>

</body>
</html>
```

Вы можете удивиться тому, что итоговая программа оказалась существенно проще, чем тестовая. Ни одну из них нельзя назвать очень сложной, но если вы написали сказку, создали переменные и убедились, что значения всех переменных передаются верно, создание программы-сказки становится очень простым делом. Большая часть кода в файле `story.php` представляет собой простой HTML. Единственной написанной на PHP командой является длинный оператор `print`. Он использует синтаксис `print <<<HERE` для вывода длинной строки HTML-текста с включенными в него переменными PHP. Этот текст и есть наша сказка.

Итоги

В этой главе вы познакомились с несколькими необычайно важными понятиями. Вы узнали, что такое переменные и как они создаются на PHP. Вы научились подключать к форме PHP-программу, изменяя атрибуты формы `method` и `action`. Вы научились создавать стандартные ссылки, передающие данные в серверные сценарии. Вы создали программы, которые взаимодействуют с различными типами элементов ввода, включая списки, выпадающие списки и кнопки-переключатели. Вы прошли весь процесс написания программы от начала до конца, включая самую важную стадию планирования, разработали форму для ввода пользовательских данных и на основании введенных данных создали интересный результат.

ДОМАШНЕЕ ЗАДАНИЕ

1. Разработайте веб-страницу, которая будет спрашивать у пользователя его имя и фамилию, а потом с помощью PHP-сценария создавать персональное письмо для этого пользователя. Сообщите ему, что он может стать миллионером.
2. Создайте веб-страницу и разместите на ней ссылки, в которых бы использовалась передача данных по методу `get`. Пусть эти ссылки указывают на ваши излюбленные поисковые системы, сайты местных новостей и погоды и другие интересные вам веб-ресурсы.
3. Напишите собственную программу-сказку. Найдите и измените текст, создайте соответствующую форму для ввода данных и выведите готовую сказку при помощи PHP-сценария.

Глава 3

Управляем работой программы с помощью условий и функций

К настоящему моменту вы уже написали несколько PHP-программ, получающих информацию от пользователя, хранящих ее в переменных и производящих простые действия над этими переменными. Самая интересная работа с компьютером начинается тогда, когда ему позволяют принимать решения. На самом деле нам лишь кажется, что компьютер способен что-либо решить самостоятельно. Программист создает код, с помощью которого сообщает компьютеру, что именно тот должен делать в той или иной ситуации. В этой главе вы узнаете, как можно управлять ходом работы программы.

Вы научитесь:

- генерировать случайные числа;
- использовать структуру `if` для изменения поведения программы;
- сравнивать переменные с помощью операторов условий;
- применять оператор `else` для обработки случаев, когда условие не выполняется;
- использовать оператор `switch` при работе с несколькими вариантами;
- создавать функции, позволяющие легче ориентироваться в коде;
- писать программы, умеющие самостоятельно создавать формы.

Представляем игру «Сколько лепестков у розы»

Игра «Сколько лепестков у розы», представленная на рис. 3.1, иллюстрирует все новые навыки, которые вы приобретете, изучив эту главу.

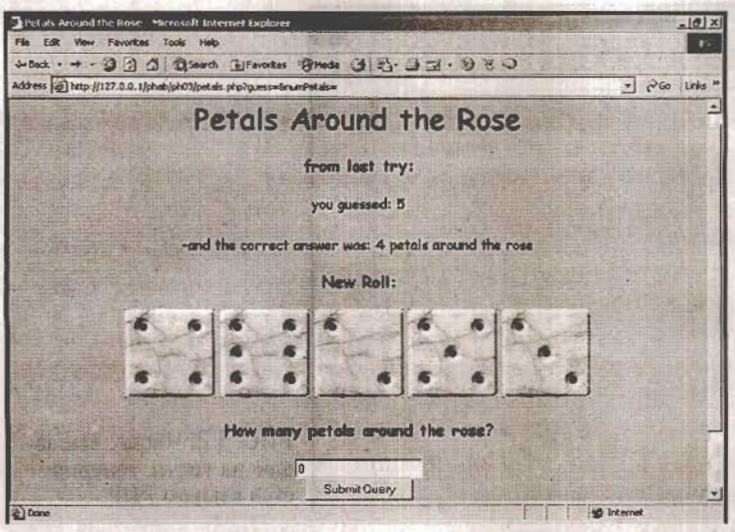


Рис. 3.1. Новая версия старой игры-загадки в кости

Правила этой игры очень просты. Компьютер бросает пять костей и предлагает пользователю угадать, сколько лепестков у розы. Пользователь вводит число и нажимает на кнопку. После этого компьютер сообщает ему, угадал ли он, и снова бросает кости. Стоит пользователю разгадать секрет, и игра окажется очень простой, но для того чтобы в ней разобраться, может потребоваться немало времени. Когда вы будете изучать код программы в конце главы, вы узнаете секрет этой игры, а пока попробуйте самостоятельно разгадать его до того, как заглянете в программу.

Генерируем случайное число

Игра в кости, как и множество других игр, интересна тем, что основана на случайных числах. Большинство языков предоставляют как минимум один способ для их генерации. На PHP случайные числа очень просто получаются с использованием функции `rand`.

Рассматриваем программу «Roll 'em»

Программа «Roll 'em», показанная на рис. 3.2, демонстрирует, как при помощи функции `rand` генерируются виртуальные кости.

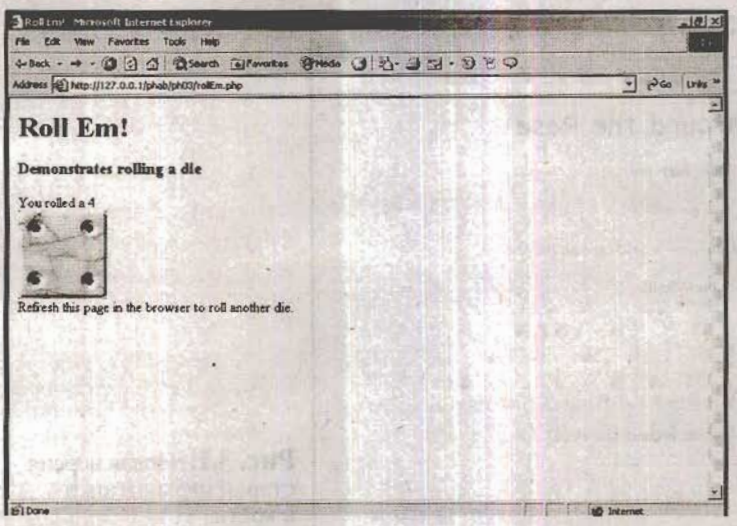


Рис. 3.2. Число, выпавшее на кости, генерируется языком PHP

Код программы «Roll 'em» показывает, насколько легко генерируется случайное число.

```
<html>
<head>
<title>Roll Em!</title>
</head>
<body>
<h1>Roll Em!</h1>
<h3>Demonstrates rolling a die</h3>
<?
$roll = rand(1,6);
print "You rolled a $roll";
print "<br>";
print "<img src = die$roll.jpg>";
?>
<br>
Refresh this page in the browser to roll another die.
</body>
</html>
```

Для создания случайного числа в диапазоне от одного до шести включительно я использовал функцию `rand`, а результат сохранил в переменной `$roll`. Функция `rand` имеет два параметра. Первый должен содержать наименьшее требуемое вам значение числа, а второй – наибольшее. Поскольку я хотел сымитировать обычную шестигранную кость, я сообщил функции `rand`, что возвратить она должна значение между одним и шестью. Поскольку я знал, что `rand` вернет значение, я присвоил это значение переменной `$roll`. Это значение я присвоил переменной `$roll`. К моменту, когда строка

```
$roll = rand(1,6);
```

завершит свое выполнение, в переменную `$roll` будет записано случайное значение. Наименьшим возможным значением будет единица, а наибольшим – шестерка, и это число не будет иметь дробной части (другими словами, оно никогда не будет равным, например, 1.5).

Если вы переходите на PHP с другого языка, вас может удивить то, каким образом в PHP создаются случайные числа. Большинство языков просто генерируют случайное число с плавающей точкой, находящееся в диапазоне от нуля до единицы, а вам потом приходится переводить его в нужный диапазон. PHP позволяет (а фактически – заставляет вас) создавать числа в заранее заданном диапазоне, что, как правило, и требуется. Если вам понадобится число между нулем и единицей, вы можете создать случайное число от нуля до 1000 и затем разделить его на 1000.

Выводим соответствующую картинку

Обратите внимание на то, как я использовал возможность интерполяции переменной. Картинку кости с единицей я назвал `die1.jpg`, с двойкой – `die2.jpg` и так далее. В том месте, где мне нужно вывести картинку на экран, я поместил обычный HTML-тег `image`, атрибуту `source` которого присвоено значение `die$roll.jpg`. Если `$roll` равно трем, будет показана картинка `die3.jpg`. Вскоре вы увидите, как еще компьютер может реагировать на случайные числа, а пока отметьте, что интерполяция переменной может оказать вам большую услугу, если вы знаете, как структурированы имена файлов.

Вероятно вы помните из главы 2, что интерполяцией назвается прием, с помощью которого в строку, заключенную в кавычки, можно подставить значение переменной, просто включив ее имя в эту строку.

ГДЕ НАЙТИ КАРТИНКИ

Игра в кости, обсуждаемая в этой главе, показывает, как с помощью графических изображений можно сделать ваши программы более интересными. Получить картинки можно несколькими способами. Самый простой из них – это найти готовые изображения в Интернете. Но хотя это и совсем не трудно технически, многие изображения в сети имеют владельцев, поэтому старайтесь не нарушать их права на интеллектуальную собственность. Попробуйте получить разрешение на то, чтобы использовать нужные вам картинки.

Другой вариант – нарисовать изображения самостоятельно. Даже если у вас совсем нет художественных способностей, современные программы и технологии позволяют очень просто создавать довольно сносные картинки. Очень многого можно добиться при помощи цифровой камеры и бесплатного графического редактора. Даже если вы захотите нанять профессионального художника, чтобы он нарисовал графику для вашей программы, вам все равно может понадобиться нарисовать эскиз, чтобы объяснить ему, что вы хотите получить в итоге.

Управляем течением работы программы при помощи оператора if

Самое интересное в работе с компьютером начинается тогда, когда пользователю кажется, что машина может принимать решения. На самом деле это не более чем иллюзия. Программист дает компьютеру очень точные инструкции, и тот действует строго в соответствии с ними. Самой простой формой такого поведения является структура, называемая «оператор `if`».

Представляем программу «Ace»

Я немного изменил программу «Roll 'em», чтобы проиллюстрировать, как ее можно улучшить с помощью оператора `if`. На рис. 3.3 показана страница, появляющаяся на экране, если программа выбрасывает любое число, кроме единицы.

Если же выпадает единица, происходит нечто потрясающее (или, как минимум, удивительное), что показано на рис. 3.4.

Включаем в программу условные операторы

На первый взгляд, программа «Ace» ведет себя очень примитивно: что-либо интересное в ней происходит лишь тогда, когда на кости выпадает единица, во всех же остальных случаях ничего необычного не случается. Но хотя такая идея кажется очень простой, она имеет далеко идущие последствия.

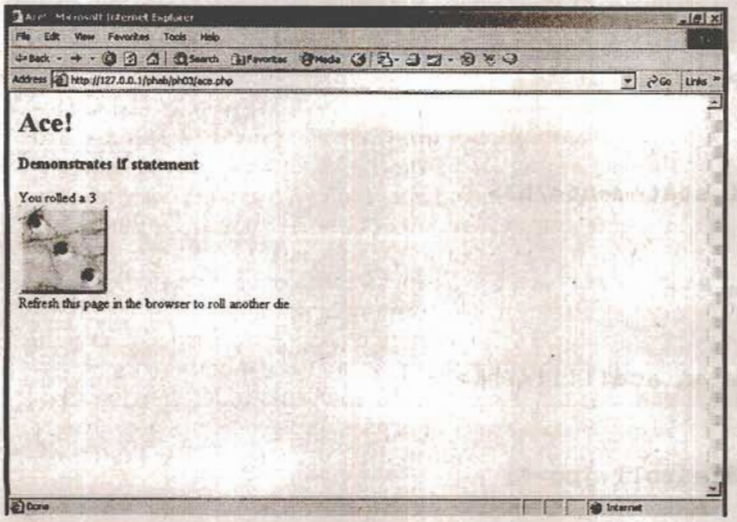


Рис. 3.3. Когда выпадает не единица, ничего интересного не происходит

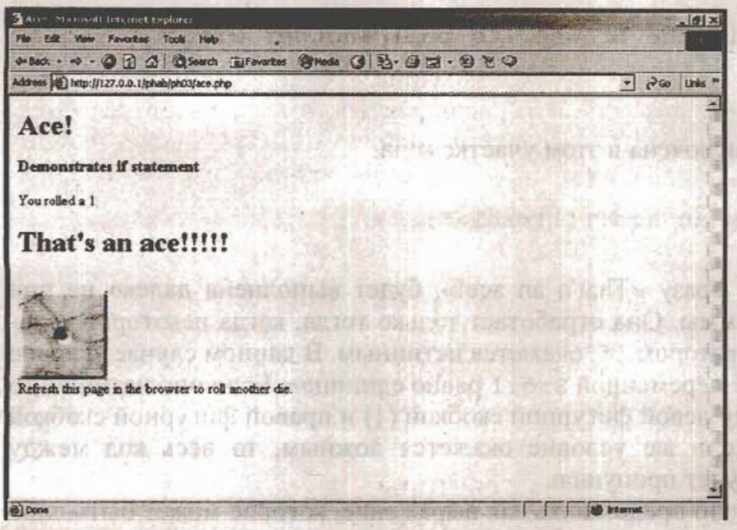


Рис. 3.4. Когда выпадает единица, радости программы нет предела

На основе примитивного механизма, использованного в программе «Ace», построены все сложные компьютерные программы, от игровых симуляторов до кардиологических систем. Взгляните на код программы «Ace», й попробуйте определить новый для нас элемент.

```
<html>
<head>
<title>Ace!</title>
</head>
<body>
<h1>Ace!</h1>
<h3>Demonstrates if statement</h3>
<?
$roll = rand(1,6);
print "You rolled a $roll";

if ($roll == 1){
  print "<h1>That's an ace!!!!!</h1>";
} // завершение if
print "<br>";
print "<img src = die$roll.jpg>";
?>
<br>
Refresh this page in the browser to roll another die.
</body>
</html>
```

Тайна программы заключена в этом участке кода.

```
if ($roll == 1){
  print "<h1>That's an ace!!!!!</h1>";
} // завершение if
```

Строка, выводящая фразу «That's an ace!», будет выполнена далеко не при каждом запуске программы. Она отработает только тогда, когда некоторое условие, установленное оператором `if`, окажется истинным. В данном случае условие читается так: «значение переменной `$roll` равно единице». Если оно истинно, то код, помещенный между левой фигурной скобкой ({) и правой фигурной скобкой (}) будет выполнен. Если же условие окажется ложным, то весь код между фигурными скобками будет пропущен.

Вообще условие можно представить как выражение, которое может быть либо истинным (`true`), либо ложным (`false`). Любое выражение, которое возвращает одно из этих значений, может быть использовано в качестве условия. Большинство условий, применяемых в операторах, похожи на то, что использовано в программе «Ace» – оно проверяет, равно ли значение переменной `$roll` единице. Обратите внимание на то, что оператор равенства обозначается двумя знаками «равно» (=).

Это крайне важно, поскольку компьютерные программы не обладают такой же гибкостью, как люди. Мы, люди, часто используем один и тот же символ в разных целях. И хотя компьютерные языки тоже могут так поступать, обычно это приводит к проблемам. Одинарный знак «равно» зарезервирован для *присваивания*. Строка:

```
$x = 5;
```

читается «переменной x присваивается значение 5» и выполняет операцию присваивания, тогда как фрагмент:

```
$x == 5;
```

читается «x равно пяти» и выполняет *операцию сравнения*. По существу, этот оператор выясняет, равно ли x пяти. Условие, такое, как `$x==5`, не может быть помещено в код в качестве самостоятельного элемента. Наоборот, оно всегда используется как часть другой структуры, например оператора `if`.

Исследуем операторы сравнения

Равенство (==) не единственный оператор сравнения в языке PHP. Существует еще несколько способов сравнить переменную с неким значением или две переменные. Эти операторы сравнения представлены в табл. 3.1.

Операторы сравнения работают для любых типов данных, хотя результаты иногда могу показаться странными. Например, если вы поместите в код подобное условие:

```
"a" < "b"
```

то получите результат `true`, поскольку в алфавите буква «a» идет раньше, чем буква «b», поэтому ее значение «меньше».

Табл. 3.1. Операторы сравнения

Оператор	Описание
==	равно
<	меньше
>	больше
<=	меньше или равно
>=	больше или равно
!=	не равно

Создаем оператор if

Оператор `if` начинается с ключевого слова `if`, за которым следует условие в скобках. После скобок идет открывающая фигурная скобка ({). Между открывающей и закрывающей (}) фигурной скобкой вы можете поместить любое количество строк кода. Код, заключенный в фигурные скобки, будет выполнен только в том случае, если условие окажется истинным. В случае если условие будет ложным, программа сразу перейдет на следующую строчку после закрывающей фигурной скобки. Строку с фигурной скобкой нет необходимости оканчивать точкой с запятой. Обычно принято сдвигать вправо все строки кода, помещенные между открывающей и закрывающей фигурными скобками.

СТИЛЬ ПРОГРАММИРОВАНИЯ

Вам уже наверняка известно, что интерпретатор PHP игнорирует все пробелы и символы возврата каретки, встречающиеся в коде. Поэтому вы можете задать вопрос, зачем при написании кода уделять столько внимания отступам, положению скобок и т. д. Хотя для PHP-процессора нет никакой разницы, как отформатирован код, то для людей, которые будут его читать, разница есть. У программистов временами разгораются жаркие споры о том, как именно должен быть отформатирован код. Если вы создаете программу в группе (например, при работе над большим проектом или во время учебы), то, как правило, вам будут выданы требования к стилю программирования, которых необходимо придерживаться. Если вы работаете сами по себе, то не столько важен стиль, который вы выберете, сколько ваша способность постоянно его придерживаться. Особенности стиля, которых я придерживаюсь в этой книге, встречаются довольно часто, достаточно легко читаются (особенно для начинающих) и легко переносятся на другие языки. Если у вас еще не выработался собственный стиль, то представленный в этой книге может стать для вас хорошей отправной точкой. Однако если руководитель вашей команды разработчиков или ваш преподаватель требует, чтобы вы придерживались другого стиля, вам придется следовать их требованиям. Но независимо от конкретных особенностей стиля, который вы используете, рекомендуется применять отступы, не скупиться на комментарии в тексте программы и разделять участки кода пробелами и символами перевода строки, чтобы впоследствии этот код было проще читать и отлаживать

Не помещайте точку с запятой в конце строки `if`. *Следующий код*

```
if ("day" == "night") ; {
print "we must be near a black hole";
} // завершение if
```

выведет на экран строку `"we must be near a black hole"`. *Когда PHP-процессор обнаружит точку с запятой после условия* `("day" == "night")`, *он посчитает, что кода, который должен быть выполнен, если условие окажется истинным, в данном случае нет вообще, поэтому условие будет, фактически, пропущено. По существу, фигурные скобки используются для того, чтобы показать, что несколько строк кода должны восприниматься как одна структура и что эта структура является частью текущей логической строки.*

Обрабатываем отрицательные результаты

Программа «Ace» показывает, как создать код, использующий условный оператор. Во многих случаях вам потребуется писать программы, которые должны будут выполнять одни действия, когда условие истинно, и другие действия, когда условие ложно. В большинстве языков программирования существует отдельный вариант оператора `if` именно для подобных случаев.

Демонстрируем программу «Ace or Not»

Программа «Ace or Not» создана на основе программы «Ace», однако обладает одним важным отличием, как видно на рис. 3.5 и 3.6.

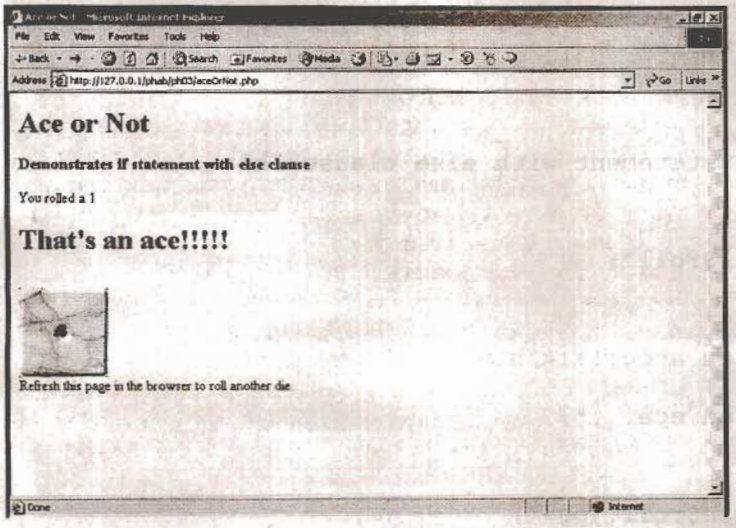


Рис. 3.5. Если программа выбрасывает единицу, она, как и прежде, кричит «Ace!»

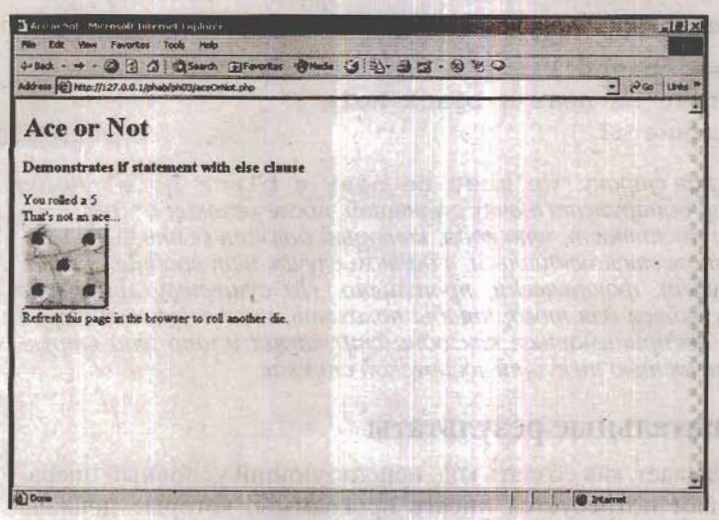


Рис. 3.6. Если программа выбрасывает не единицу, она тоже сообщает об этом пользователю

Другими словами, программа выполняет одни действия, когда условие истинно, и другие – когда условие ложно.

Используем конструкцию else

Из кода программы «Ace or Not» видно, как конструкция `else` может быть использована для реализации различного поведения программы.

```
<html>
<head>
<title>Ace or Not</title>
</head>
<body>
<h1>Ace or Not</h1>
<h3>Demonstrates if statement with else clause</h3>
<?
$roll = rand(1,6);
print "You rolled a $roll";
print "<br>";
if ($roll == 1){
 print "<h1>That's an ace!!!!!</h1>";
} else {
 print "That's not an ace...";
} // завершение if
print "<br>";
print "<img src = die$roll.jpg>";
```

```
?>
<br>
Refresh this page in the browser to roll another die.
</body>
</html>
```

Интересующий нас участок кода начинается после оператора `if`.

```
if ($roll == 1){
 print "<h1>That's an ace!!!!!</h1>";
} else {
 print "That's not an ace...";
} // завершение if
```

Если условие `$roll == 1` истинно, программа выводит фразу «That's an ace!!!!!» Если условие ложно, вместо этого выполняется код, помещенный после оператора `else` до конца структуры `if`. Обратите внимание на структуру и отступы. Первый участок кода (между условием и оператором `else`, заключенный в фигурные скобки) выполняется, когда условие истинно. Если условие ложно, выполняется код, расположенный между оператором `else` и окончанием структуры `if` (он также заключен в фигурные скобки). Любой из этих участков может содержать столько строк, сколько вам необходимо. Только один из двух участков будет выполнен (в зависимости от условия), и в любом случае выполнение одного из них произойдет.

Работаем с несколькими значениями

Часто может оказаться так, что вам придется иметь дело с более сложными даными. Например, вы можете захотеть разные ответы для каждого из шести возможных вариантов броска кости. Программа «Binary Dice», представленная на рис. 3.7 и 3.8, демонстрирует именно такую ситуацию.

Создаем программу «Binary Dice»

Эта программа имеет немного более сложную структуру операторов `if`, чем предыдущие, поскольку для каждого из шести возможных исходов должно быть отображено собственное двоичное значение. Обычного оператора `if` для этого недостаточно, но можно совместить несколько таких операторов, что решит нашу проблему.

```
<html>
<head>
<title>Binary Dice</title>
</head>
<body>
<h1>Binary Dice</h1>
```

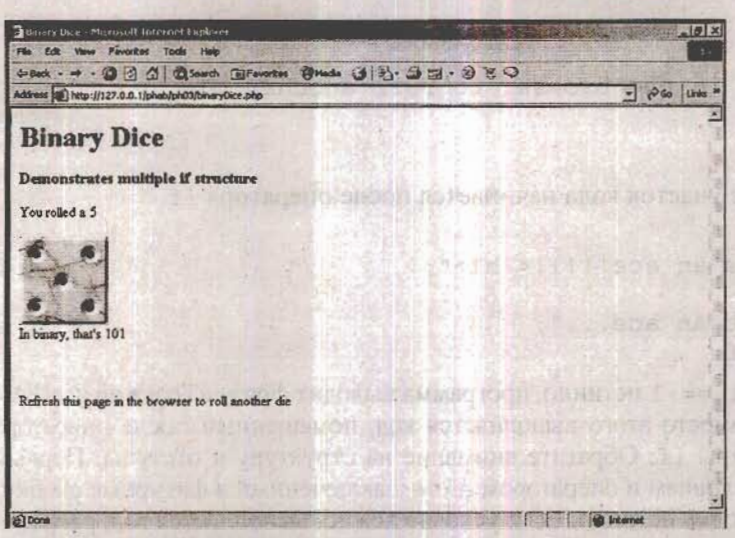


Рис. 3.7. На кости выпало 5, и программа показывает двоичное представление этого числа

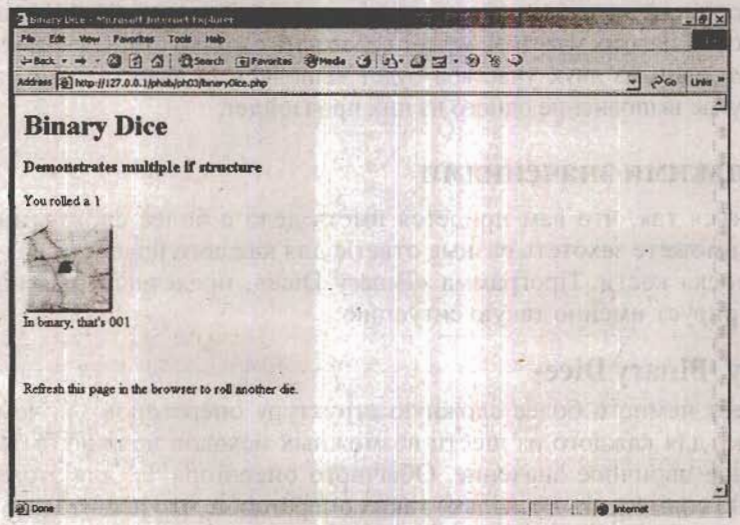


Рис. 3.8. Просле очередного броска программа вновь сообщает его результат в двоичном виде

```
<h3>Demonstrates multiple if structure</h3>
<?
$roll = rand(1,6);
print "You rolled a $roll";
print "<br>";
if ($roll == 1){
```

```
 $binValue = "001";
} else if ($roll == 2){
 $binValue = "010";
} else if ($roll == 3){
 $binValue = "011";
} else if ($roll == 4){
 $binValue = "100";
} else if ($roll == 5){
 $binValue = "101";
} else if ($roll == 6){
 $binValue = "110";
} else {
 print "I don't know that one...";
} // завершение if
print "<br>";
print "<img src = die$roll.jpg>";
print "<br>";
print "In binary, that's $binValue";
print "<br>";
print "<br>";
print "<br>";
?>
<br>
Refresh this page in the browser to roll another die.
</body>
</html>
```

Используем несколько операторов else if

Программа «Binary Dice» по-прежнему содержит единственную структуру `if`, которая, в отличие от предыдущих случаев, имеет несколько операторов `else`. Первое условие просто сравнивает значение `$roll` с единицей. Если это так, то выполняется соответствующий участок кода (тот, что присваивает переменной `$binValue` двоичное представление числа 1). Если первое условие не выполняется, программа последовательно проходит все пары операторов `if else` до тех пор, пока какой-либо условный оператор не вернет значение «истина». Если ни одно из условий не истинно, будет выполнен код последнего оператора `else`.

Вы можете спросить, зачем я поместил в свой код последний оператор `else`. *Мы знаем, что значение переменной* `$roll` *должно находиться между одним и шестью, и мы уже проверили каждое из этих значений, поэтому программа ни в каком случае не должна перейти к выполнению последнего оператора* `else`. *Однако про-*

граммы не всегда работают так, как вы ожидаете, а потому нелишним бывает написать код для оператора `else` *даже если он не должен понадобиться. Будет лучше, если ваша программа сообщит о том, что произошло нечто неожиданное, чем если она внезапно, без объяснения причин, закроется в тот момент, когда с ней работают пользователи.*

Полезно бывает выделить отступами код внутри структуры `if` с несколькими условиями, чтобы вам потом сразу было видно, какие участки кода относятся к структуре `if`, а какие будут выполняться в том случае, если то или иное условие окажется истинным.

Упрощаем код с помощью структуры switch

Ситуации, подобные встретившейся нам в программе «Binary Dice», возникают достаточно часто, поэтому существует отдельная структура, разработанная специально для случаев сравнения одной переменной с несколькими возможными значениями. Программа «Switch Dice», показанная на рис. 3.9, с точки зрения пользователя выглядит точно так же, как и программа «Binary Dice», за исключением того, что показывает число, выпавшее на кости, не в двоичной записи, а римскими цифрами.

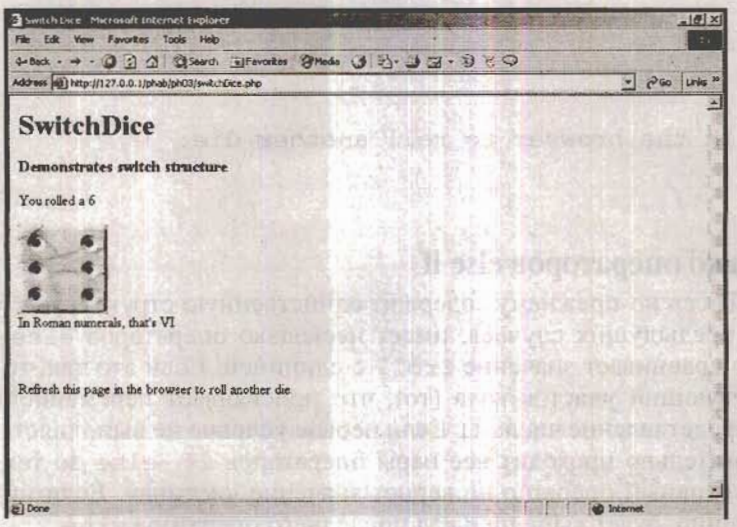


Рис. 3.9. Эта версия программы показывает значение кости римскими цифрами

Хотя внешне две последние программы кажутся очень похожими, при написании кода второй из них была использована другая структура, чтобы продемонстрировать очень удобный оператор, который назвается `switch`.

Создаем программу «Switch Dice»

Код программы «Switch Dice» выглядит иначе, чем код «Binary Dice», но результат их работы одинаков.

```
<html>
<head>
<title>Switch Dice</title>
</head>
<body>
<h1>SwitchDice</h1>
<h3>Demonstrates switch structure</h3>
<?
$roll = rand(1,6);
print "You rolled a $roll";
print "<br>";
switch ($roll){
 case 1:
  $romValue = "I";
  break;
 case 2:
  $romValue = "II";
  break;
 case 3:
  $romValue = "III";
  break;
 case 4:
  $romValue = "IV";
  break;
 case 5:
  $romValue = "V";
  break;
 case 6:
  $romValue = "VI";
  break;
 default:
  print "This is an illegal die!";
} // завершение switch
print "<br>";
print "<img src = die$roll.jpg>";
print "<br>";
print "In Roman numerals, that's $romValue";
print "<br>";
```

```
print "<br>";
print "<br>";
?>
<br>
Refresh this page in the browser to roll another die.
</body>
</html>
```

Используем структуру switch

Структура `switch` создана специально для таких случаев, когда у вас есть одна переменная, которую необходимо сравнить с несколькими возможными значениями. Чтобы получить нужный результат, используйте ключевое слово `switch`, после которого в скобках следует имя сравниваемой переменной. Фигурные скобки указывают на то, что следующий участок кода занимается сравнением этой переменной с возможными значениями.

Для каждого из возможных значений используйте оператор `case`, после которого следует значение, завершаемое двоеточием. Каждый оператор `case` завершается командой `break`, которая указывает на то, что программа должна перестать обрабатывать данное значение и перейти к следующему.

Использование оператора `break`*, пожалуй, является самой сложной частью применения структуры* `switch`*, особенно если вы знакомы с языками наподобие Visual Basic, не требующими подобных конструкций. Очень важно не забыть поместить команду* `break` *после каждого оператора* `case`*. В противном случае программа просто перейдет к обработке следующего значения, даже если оно при сравнении не дало бы результата «истина». Как новичку, вам следует каждый оператор* `case` *завершать командой* `break`*.*

Последнее возможное значение задается ключевым словом `default`. Оно работает точно так же, как оператор `else` в структуре `if` с несколькими сравнениями. После него помещается код, который будет выполнен в том случае, если не отработает ни один из других операторов `case`. Как и в случае со структурой `if`, бывает полезно помещать в код вариант `default`, даже если вам кажется, что компьютер никогда до него не дойдет. Временами случаются очень странные вещи, и лучше быть к ним подготовленным.

Соединяем форму и ее результаты

Большинство написанных вами к настоящему моменту PHP-программ состоят из двух отдельных файлов. HTML-файл содержит форму, которая вызывает PHP-программу. Иногда следить за двумя файлами бывает утомительно. С помощью

оператора `if` вы можете объединить обе функции на одной странице. Программа «hiUser», результат работы которой показан на рис. 3.10 и 3.11, выглядит совсем как ее двойник из главы 2, однако обладает серьезным отличием. Вместо того чтобы состоять из HTML-страницы и отдельной PHP-программы, она вся целиком помещается в одном файле на сервере.

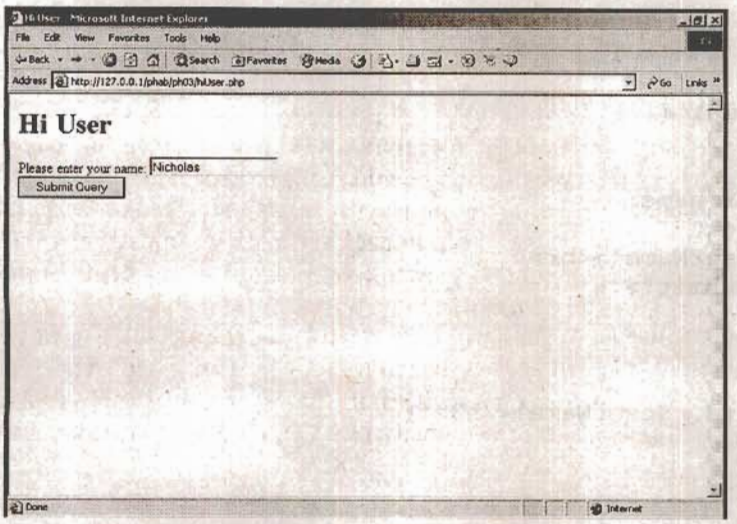


Рис. 3.10. Эта HTML-страница создается с помощью PHP-кода

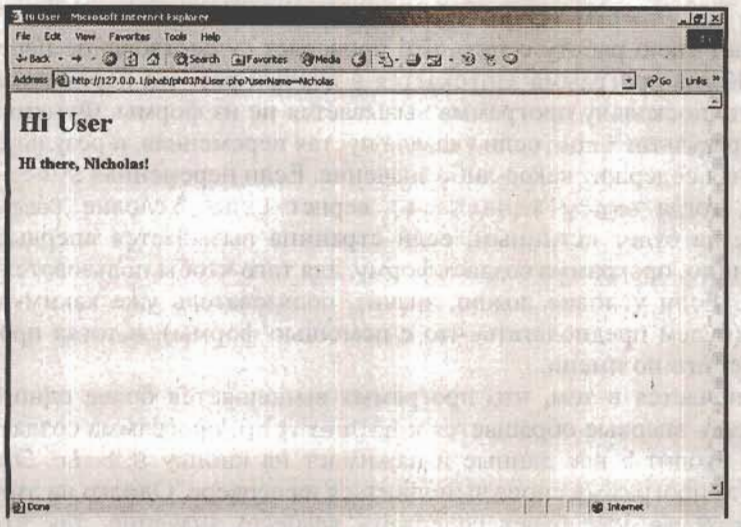


Рис. 3.11. Результат возвращает та же самая программа

Код новой версии программы «hiUser» показывает, как этого достичь.

```
<html>
<head>
<title>Hi User</title>
</head>
<body>
<h1>Hi User</h1>
<?
if (empty($userName)){
 print <<<HERE
 <form>
 Please enter your name:
 <input type = "text"
        name = "userName"><br>
 <input type = "submit">
 </form>
HERE;
} else {
print "<h3>Hi there, $userName!</h3>";
} //завершение
?>
</body>
</html>
```

Программа начинает свою работу с того, что проверяет существование переменной `$userName`. Когда программа запускается в первый раз, эта переменная не будет существовать, поскольку программа вызывается не из формы. Функция `empty()` возвращает результат `true`, если указана пустая переменная, и результат `false`, если переменная содержит какое-либо значение. Если переменная `$userName` не существует, тогда `empty($userName)` вернет `true`. Условие `(empty($userName))` всегда будет истинным, если страница вызывается впервые. Если это условие истинно, программа создаст форму, для того чтобы пользователь мог ввести свое имя. Если условие ложно, значит, пользователь уже каким-то образом вводил имя (будем предполагать, что с помощью формы), и тогда программа поприветствует его по имени.

Главная идея заключается в том, что программа выполняется более одного раза. Когда пользователь впервые обращается к `hiUser.php`, программа создает форму. Пользователь вводит в нее данные и нажимает на кнопку `Submit`. Это заставляет ту же самую программу снова выполниться на сервере. Однако на этот раз переменная `$userName` уже будет содержать какое-то значение, так что вместо создания формы, программа использует это значение в приветствии.

Программирование на стороне сервера часто бывает именно таким. Пользователь в процессе решения какой-то задачи нередко последовательно вызывает одну и ту же программу много раз. Вы будете часто использовать структуры ветвления, такие, как `if` и `switch`, чтобы направлять ход работы программы в зависимости от того, на каком этапе решения задачи находится пользователь.

Выделяем самостоятельные части программы с помощью функций

Прошло совсем немного времени, а ваши программы уже начали усложняться. Как только количество строк кода начинает превышать размеры окна в редакторе, следить за ним становится намного труднее. Программисты предпочитают разбивать код на небольшие участки, называемые функциями, чтобы избежать его чрезмерного усложнения. Функция подобна маленькой программе. Она должна хорошо выполнять одну задачу. Взгляните на простой пример, показанный на рис. 3.12.

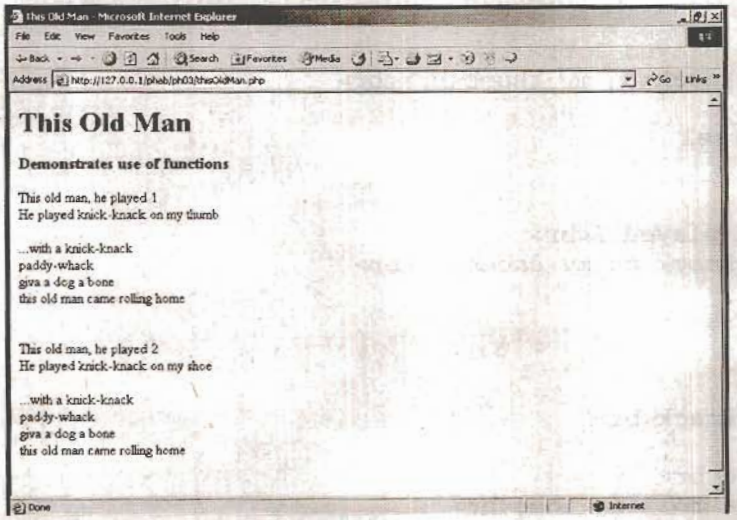


Рис. 3.12. Эта песня следует привычному шаблону: куплет, припев, куплет, припев

Исследуем программу «This Old Man»

В текстах песен часто по несколько раз повторяются одни и те же строки. Песня «This Old Man», текст которой показан на рис. 3.12, являет собой наглядный пример. Каждый куплет отличается от других, но припев всегда один и тот же. Обычно, если приходится записывать текст подобной песни, куплеты пишут полностью, а припев записывают только один раз, после чего в нужных местах просто пишут слово «припев», и всем бывает понятно, что петь нужно не слово

«припев», а припев песни. Именно так устроены подпрограммы в языках программирования. Код программы «thisOldMan» илюстрирует, как это работает.

```
<html>
<head>
<title>This Old Man</title>
</head>
<body>
<h1>This Old Man</h1>
<h3>Demonstrates use of functions</h3>
<?
verse1();
chorus();
verse2();
chorus();

function verse1(){
  print <<<HERE
  This old man, he played 1<br>
  He played knick-knack on my thumb<br><br>
HERE;
} // завершение verse1

function verse2(){
  print <<<HERE
  This old man, he played 2<br>
  He played knick-knack on my shoe<br><br>
HERE;
} // завершение verse1

function chorus(){
  print <<<HERE
  ...with a knick-knack<br>
  paddy-whack<br>
  giva a dog a bone<br>
  this old man came rolling home<br>
  <br><br>
} // завершение chorus

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

Внимательное изучение этого кода подскажет вам, как он работает. Главная часть программы очень проста.

```
verse1();
chorus();
verse2();
chorus();
```

Создаем новые функции

Сперва может показаться, что в тексте программы есть новые PHP-функции. Я вызываю функцию `verse1()`, потом `chorus()` и т. д. Новые функции действительно есть, но они не являются внутренними для языка PHP. Я создал их вместе со страницей. Вы можете написать несколько инструкций и хранить их под неким именем. Это, фактически, создает на время новую команду на PHP, позволяя объединять простые команды для выполнения более сложных задач. Написать функцию очень просто. Сперва идет ключевое слово `function`, затем название функции и пара скобок. Между скобками пока что ничего помещать не надо. В следующем разделе вы узнаете, что может находиться между ними. Строки кода, составляющие функцию, заключаются в фигурные скобки ({}). Не забудьте закончить описание функции закрывающей фигурной скобкой. Полезным будет также сдвинуть вправо весь код, помещенный в тело функции.

Вы можете заметить, что у меня в коде встречается строка, для которой я никогда не делаю отступ. Это метка `HERE`*, отмечающая многостроковый текст. Вспомните, что слово* `HERE` *действует как закрывающая кавычка и должно начинаться с первой позиции строки, в которой оно расположено, поэтому выделять отступом его нельзя.*

Вы можете назначать своим функциям любые имена, однако будьте осторожны. Если вы попытаетесь определить функцию, которая уже существует, программа начнет себя вести странно. PHP имеет огромное количество встроенных функций. Если у вас возникают непонятные ошибки в функции, поищите ее имя в системе контекстной помощи (она поставляется вместе с PHP, кроме того, ее можно скачать с сайта www.php.net) – возможно, функция с таким именем уже есть.

Особенно удобна в этой программе функция `chorus()`, поскольку она может быть использована несколько раз. Нет необходимости повторять код припева каждый раз. Вместо этого можно просто вызвать функцию.

Используем параметры и значения функций

Каждая функция является самодостаточной. Это необходимо потому, что программа целиком может оказаться слишком сложной для понимания. Если же разбить сложную программу на несколько более мелких функций, то каждую из них можно выполнять независимо. Когда работает код, расположенный внутри функции, нет необходимости ориентироваться на что-либо за ее пределами. Если внутри функции создана переменная, она будет уничтожена, как только вы выйдете из тела функции. Это предотвращает многие ошибки, которые в ином случае могли бы проникнуть в ваш код. Однако у подобной независимости есть и свои минусы, поскольку часто бывает необходимо, чтобы функция работала с данными из вызывающей ее программы. Для реализации этого существует несколько путей. Вы можете передать в функцию *параметры*, т. е., одно или несколько значений, поступающих в функцию при ее вызове. Кроме того, каждая функция может *возвращать* значение. Проиллюстрируем это на примере. Программа `param`, показанная на рис. 3.13, демонстрирует еще один вариант песни «This Old Man». Несмотря на то что пользователь об этом может и не знать, существует несколько важных различий между этой, более сложной, программой и первой версией «This Old Man».

Изучаем программу Param.php

Заметьте, что страница-результат, показанная на рис. 3.13, длиннее, чем ее предшественница на рис. 3.12, и в то же время код, создающий эту более длинную страницу, короче и эффективнее.

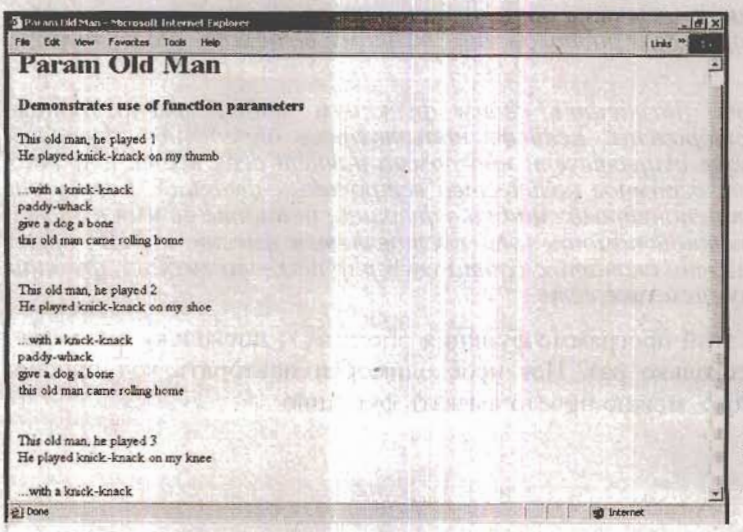


Рис. 3.13. Результат очень похож на рис. 3.12, но программа, создающая эту страницу, работает эффективнее

```
<html>
<head>
<title>Param Old Man</title>
</head>
<body>
<h1>Param Old Man </h1>
<h3>Demonstrates use of function parameters</h3>
<?
print verse(1);
print chorus();
print verse(2);
print chorus();
print verse(3);
print chorus();
print verse(4);
print chorus();
function verse($stanza){
  switch ($stanza){
    case 1:
      $place = "thumb";
      break;
    case 2:
      $place = "shoe";
      break;
    case 3:
      $place = "knee";
      break;
    case 4:
      $place = "door";
      break;
    default:
       $place = "I don't know where";
} // завершение switch
  $output = <<<HERE
    This old man, he played $stanza<br>
    He played knick-knack on my $place<br><br>
HERE;
  return $output;
} // завершение verse
function chorus(){
  $output = <<<HERE
    ...with a knick-knack<br>
```

```
    paddy-whack<br>
    give a dog a bone<br>
    this old man came rolling home<br>
    <br><br>
HERE;
  return $output;
} // завершение chorus
?>
</body>
</html>
```

Разбираем инкапсуляцию в основном коде

По сравнению с предыдущей версией, в этом коде сделан ряд улучшений. Во-первых, посмотрите на основной код программы, который теперь выглядит следующим образом.

```
print verse(1);
print chorus();
print verse(2);
print chorus();
print verse(3);
print chorus();
print verse(4);
print chorus();
```

Разобраться в том, что делает основной код, очень просто. Программа печатает первый куплет, затем припев, второй куплет, снова припев и т. д. Как именно генерируется каждый участок текста, описано в соответствующих функциях. Это – пример *инкапсуляции данных*. Инкапсуляция очень удобна, поскольку позволяет вам рассматривать задачу на нескольких уровнях. На верхнем уровне вы обдумываете основные идеи (распечатать куплеты и припев) и не слишком вдаетесь в подробности. Подобным образом вы стали бы рассказывать о том, как прошел ваш день: «Я поехал на работу, провел несколько встреч, пообедал и прочитал лекцию». Обычно в таких случаях не рассказывают обо всем в деталях. Каждая крупная задача впоследствии может быть разбита на несколько более мелких. (Если кто-нибудь спросит, вы подробно расскажете о том, как прошла встреча: «Я налил себе кофе, сделал вид будто яростно записываю что-то на карманном комьютере, тем временем установив новый рекорд в пасьянс, порисовал каракули на листке с повесткой дня и вздремнул во время презентации».)

Возвращаем значение: функция chorus()

Еще одна интересная деталь, встретившаяся нам в основном коде, касается использования функции `print()`. В предыдущей программе я просто вызывал функцию `chorus()`, и программа печатала припев. Здесь же я слегка поменял подход. Функция `chorus()`, на самом деле, ничего не выводит на экран. Вместо этого она создает длинную строку, содержащую припев, и возвращает ее в основную программу, где над этой строкой могут быть выполнены все необходимые действия. Такое поведение не является для вас новым. Вспомните функцию `rand()`. Она всегда возвращает некое значение в вызывающую программу. Наша функция поступает точно так же. Посмотрите на ее код еще раз, и вы поймете, что я имею в виду.

```
function chorus(){
  $output = <<<HERE
  ...with a knick-knack<br>
  paddy-whack<br>
  give a dog a bone<br>
  this old man came rolling home<br>
  <br><br>
HERE;
  return $output;
} // завершение chorus
```

В самом начале функции я создаю новую переменную с именем `$output`. Переменные внутри функций можно создавать точно так же, как и в основном коде программы: просто обращаясь к ним. Однако переменная, созданная внутри функции, теряет свое значение, как только функция заканчивает работу. Это очень удобно, поскольку означает, что переменная, созданная внутри функции, принадлежит только этой функции. Вам не нужно беспокоиться о том, существует ли эта переменная еще в какой-то части программы. Исключаются также ошибки, которые могли бы возникнуть, если бы вы случайно изменили значение существующей переменной. Своей переменной `$output` я присваиваю в качестве значения длинную строку, используя для этого конструкцию `<<<HERE`.

В последней строке функции содержится оператор `return`, с помощью которого значение переменной `$output` возвращается в программу. Любая функция может завершаться оператором `return`. Любое значение, помещенное после ключевого слова `return`, будет возвращено в программу, вызывавшую функцию. Таким образом, функции могут взаимодействовать с основной программой.

Передаем параметр в функцию verse()

Наиболее эффективной частью обновленной программы является функция `verse()`. Вместо того чтобы писать отдельную функцию для каждого куплета, я написал одну, но такую, которая может вывести любой из них. Тщательно проанализировав текст песни, я пришел к выводу, что каждый из ее куплетов весьма похож на все остальные. Единственное их различие состоит в том, что сыграл старик (а это не что иное, как номер куплета) и где он это сыграл (рифма к номеру куплета). Поэтому достаточно просто знать номер куплета, чтобы правильно составить его текст. Обратите внимание на то, что когда основной код вызывает данную функцию, в скобках всегда указан номер куплета. Например, там есть команды `verse(1)` и `verse(3)`. Они обе вызывают функцию `verse`, но передают в нее разные значения (1 и 3 соответственно). Еще раз посмотрите на код функции `verse()`, и вы увидите, как она реагирует на передачу этих значений.

```
function verse($stanza){
  switch ($stanza){
    case 1:
      $place = "thumb";
      break;
    case 2:
      $place = "shoe";
      break;
    case 3:
      $place = "knee";
      break;
    case 4:
      $place = "door";
      break;
    default:
      $place = "I don't know where";
  } // завершение switch
  $output = <<<HERE
  This old man, he played $stanza<br>
  He played knick-knack on my $place<br><br>
HERE;
  return $output;
} // завершение verse
```

Я указал переменную `$stanza` в качестве параметра в описании функции. *Параметр* – это обычная переменная, связанная с функцией. Если функция имеет параметр, то при каждом ее вызове в нее необходимо передавать некоторое значение. Переменная-параметр автоматически получает это значение из основного кода. Например, если в программе написано `verse(1)`, это значит, будет вызвана функция `verse` и переменная `$stanza` будет содержать значение 1.

Затем я использовал конструкцию `switch` для того, чтобы присвоить переменной `$place` значение, соответствующее значению переменной `$stanza`. Наконец, я создал переменную `$output`, в которой использовал и переменную `$stanza`, и переменную `$place` и возвратил значение переменной `$output`.

Функция может также получать несколько параметров. Для этого необходимо объявить несколько переменных в скобках при описании функции. При вызове такой функции будет нужно указать соответствующее число аргументов. Не забудьте: параметры разделяются запятыми.

Возвращаясь к реальности

Если вы уже являетесь опытным программистом, то, вероятно, понимаете, что сущесвуют способы сделать этот код еще эффективнее. Мы еще вернемся к данной программе после того, как вы в ближайших главах узнаете о циклах и массивах

Задаем область видимости переменной

Вам уже известны некоторые способы организации обмена значениями переменных между основной программой и функциями. Кроме передачи параметров, иногда может потребоваться сделать так, чтобы функция могла обратиться к переменной, созданной в основной программе. Это особенно важно, поскольку автоматически созданные переменные в PHP (например, поступающие из форм) помещаются на уровень основной программы. Вам придется явно сообщать PHP о том, что вы хотите, чтобы функция использовала переменную, созданную на основном уровне.

Если вам приходилось программировать на других языках, то вас неизбежно должно смутить то, как PHP обращается с глобальными переменными. В большинстве языков любая переменная, созданная на основном уровне, доступна из любой функции. На PHP вам придется явно указывать, что глобальная переменная должна быть доступна и внутри функции. Если этого не сделать, то на уровне функции будет создана новая локальная переменная с таким же именем и без значения.

Пример области видимости переменной

Для того чтобы проиллюстрировать понятие глобальной переменной, давайте разберем демонстрационную программу, результат работы которой представлен на рис. 3.14.

Посмотрите на код демонстрационной программы, и вы поймете, в чем отличие.

```
<html>
<head>
<title>Scope Demo</title>
```

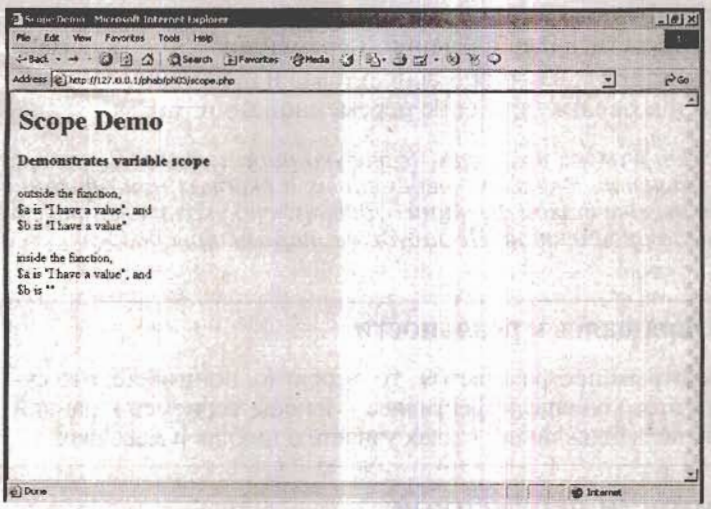


Рис. 3.14. Переменная `$a` сохраняет свое значение внутри функции, а переменная `$b` – нет

```
</head>
<body>
<h1>Scope Demo</h1>
<h3>Demonstrates variable scope</h3>
<?
$a = "I have a value";
$b = "I have a value";
print <<<HERE
  outside the function, <br>
  \$a is "$a", and<br>
  \$b is "$b"<br><br>
HERE;
myFunction();
function myFunction(){
  //сделать $a глобальной, а $b нет
  global $a;
  print <<<HERE
    inside the function, <br>
    \$a is "$a", and<br>
    \$b is "$b"<br><br>
HERE;
} // завершение myFunction
?>
</body>
</html>
```

В этой программе я создаю две переменные с именами `$a` и `$b` и присваиваю обеим значение «I have a value». Для проверки я сразу же вывожу на экран значение той и другой переменной.

Обратите внимание на прием, с помощью которого я вывел в тексте знак доллара. Если PHP встречает этот знак внутри текста, заключенного в кавычки, он по умолчанию считает, что имеет дело с переменной. Иногда (как в нашем случае) вам бывает нужно вывести сам знак доллара. Для того чтобы сообщить интерпретатору о том, что на экране должен появиться данный знак, перед ним необходимо поставить обратную косую черту (\ ). Таким образом, команда `print $a` *выведет значение переменной* `$a`*, а команда* `print \$a` *выведет текст «*`$a`*».*

Возвращаемся к игре «Сколько лепестков у розы»

В начале этой главы я показал вам игру «Сколько лепестков у розы». Для ее создания вам потребуются все приобретеные к этому моменту знания, включая те, которые вы получили в данной главе. Если вы еще не поиграли в эту игру, сейчас самое время разобраться с ней, потому что как только вы узнаете ее секрет, играть станет неинтересно.

Общий план игры выглядит так: каждый раз, когда запрашивается страница, она случайным образом бросает пять костей и рассчитывает число лепестков по суперсекретной формуле. Страница содержит форму с текстовой областью, которая называется `guess`. В эту область пользователь вводит свой ответ, пытаясь угадать правильный. Форма также содержит скрытое поле с названием `numPetals`, в котором для программы хранится правильный ответ.

ВОЗВРАЩАЯСЬ К РЕАЛЬНОСТИ

А нельзя ли сделать так, чтобы программа просто помнила правильный ответ? Поскольку правильный ответ был рассчитан программой в самом начале, вы можете сильно удивиться, узнав, что его необходимо хранить в скрытом виде на веб-странице и извлекать «из тайника» с помощью той же самой программы, которая его рассчитала. Каждое соединение между клиентом и сервером является совершенно новым. Когда пользователь первый раз сыграет в игру, страница будет отправлена в его браузер, и соединение на этом будет разорвано до тех пор, пока пользователь не нажмет на кнопку `Submit`. Когда он отправит форму, программа начнет свою работу с самого начала. Может получиться так, что пользователь сыграет в игру перед самым сном и оставит на ночь страницу открытой на своем компьютере. А за это время к вашей программе может обратиться еще не одна сотня человек. Поэтому пока что для хранения данных о состоянии пользователя мы будем использовать скрытые поля. Позже по ходу этой книги вы познакомитесь с более изощренными способами, позволяющими сделать то же самое

В коде игры «Сколько лепестков у розы» для нас сейчас нет ничего незнакомого, просто этот код чуть длинее, чем у других программ, которые мы разбирали. Рассмотрим этот код по частям. На нашем сайте вы можете найти весь текст этой программы, собранный вместе.

Начальная HTML-страница

Как и в большинстве PHP-программ, в нашей игре все начинается с HTML-страницы. Она не содержит ничего сложного, поскольку самые интересные HTML-участки создаются при помощи PHP-кода.

```
<HTML>
<head>
<title>Petals Around the Rose</title>
</head>
<body bgcolor = "tan">
<center>
<font face = "Comic Sans MS">
<h1>Petals Around the Rose</h1>
```

Я решил, что в данном случае уместно будет использовать затейливый шрифт на желто-коричневом фоне. Это должно придать внешнему виду программы ощущение легкости.

Основной код

Основная часть PHP-кода должна выполнять три задачи. Эти задачи распределены между тремя разными функциями. Одна из целей инкапсуляции состоит в том, чтобы сделать основной текст программы как можно более ясным. В игре «Лепестки» нам удалось достичь этой цели.

```
<?
printGreeting();
printDice();
printForm();
```

Вся настоящая работа передается различным функциям, каждую из которых мы сейчас кратко опишем. Еще до того как вы посмотрите на текст этих функций, вам станет понятно, для чего нужна каждая из них, и вы получите четкое представление о ходе программы в целом.

Если вы используете при написании кода инкапсуляцию и грамотно присваиваете имена функциям, то в нем бывает намного легче разобраться, а также найти и устранить неполадки.

Функция printGreeting()

Задача этой фукнции – напечатать приветствие для пользователя. Мы выделили три возможных вида приветствия. Если пользователь никогда раньше не обращался к этой программе, она скажет ему «Добро пожаловать». Если пользователь уже бывал здесь, то он высказывал свои предположения о количестве лепестков. Предположение могло оказаться верным (и в этом случае необходимо поздравить пользователя) или неверными (и тогда нужно сообщить ему правильный ответ). Функция `printGreeting()` использует конструкцию `if-else` для обработки каждого из возможных случаев.

```
function printGreeting(){
  global $guess, $numPetals;
  if (empty($guess)){
   print "<h3>Welcome to Petals Around the Rose</h3>";
  } else if ($guess == $numPetals){
   print "<h3>You Got It!</h3>";
  } else {
   print <<<HERE
     <h3>from last try: </h3>
     you guessed: $guess<br><br>
      -and the correct answer was: $numPetals petals around the
rose<br>
HERE;
  } // завершение if
} // завершение printGreeting
```

Эта функция обращается к переменным `$guess` и `$numPetals`, которые создаются автоматически. В одном операторе `global` можно указывать несколько глобальных переменных, разделяя их при этом запятыми.

Переменная `$guess` будет пустой в случае, если пользователь вызвал программу в первый раз. Если переменная `$guess` пустая, я вывожу на экран приветствие. Если значение переменной `$guess` равно значению `$numPetals`, то, следовательно, пользователь сделал верное предположение, и в этом случае я вывожу на экран поздравления. Если ни одно из этих условий не выполняется (а такое будет происходить чаще всего), функция выведет на экран немного более сложную строку, сообщающую последнее предположение пользователя о числе лепестков, а также, правильный ответ. Этой информации пользователю должно хватить для того, чтобы в конце концов разгадать нашу загадку.

Использование структуры `else if` оказалось самым простым решением для обработки трех возможных вариантов, которые мне необходимо было проверить.

Функция printDice()

После того как программа выведет приветствие, начинается наиболее важный этап работы – необходимо случайным образом бросить кости. Как вы уже видели ранее в данной главе, сделать это довольно просто. Однако мне хотелось добиться от программы большей эффективности и попутно сосчитать правильное количество лепестков. Чтобы функция `printDice()` выполняла обе задачи, она вызывает несколько других пользовательских функций.

```
function printDice(){
  global $numPetals;
  print "<h3>New Roll:</h3>";
  $numPetals = 0;
  $die1 = rand(1,6);
  $die2 = rand(1,6);
  $die3 = rand(1,6);
  $die4 = rand(1,6);
  $die5 = rand(1,6);
  showDie($die1);
  showDie($die2);
  showDie($die3);
  showDie($die4);
  showDie($die5);
  print "<br>";
  calcNumPetals($die1);
  calcNumPetals($die2);
  calcNumPetals($die3);
  calcNumPetals($die4);
  calcNumPetals($die5);
} // завершение printDice
```

Функции `printDice()` очень важно иметь значение переменной `$numPetals` и совершенно не нужен доступ к переменной `$guess`. Она запрашивает доступ к `$numPetals` у основной программы. После вывода надписи «New Roll» она сбрасывает переменную `$numPetals` в ноль. Значение переменной `$numPetals` будет пересчитываться при каждом броске костей.

Новые значения костей я получаю путем вызова функции `rand(1,6)` шесть раз. Каждый результат сохраняется в отдельной переменной, с именами от `$die1` до `$die6`. Чтобы вывести соответствующее изображение для каждой кости, я вызываю функцию `showDie()` (она описана следующей). Я вывожу символ конца строки и вызываю функцию `calcNumPetals()` (она также описана чуть ниже) один раз для каждой кости.

Функция showDie()

Функция `showDie()` создана для того, чтобы упростить вывод повторяющегося кода. Она принимает в качестве параметра значение кости и создает соответствующий HTML-код для вывода изображения кости с необходимым числом точек.

```
function showDie($value){
  print <<<HERE
  <img src = "die$value.jpg"
       height = 100
       width = 100>
HERE;
} // завершение showDie
```

Одно из преимуществ использования функций для вывода повторяющихся участков HTML-кода состоит в том, насколько легко становится изменять большие участки кода. Например, если вы захотите изменить размеры картинок, вам понадобится исправить один тег `img` *в одной функции, и размеры всех шести костей станут другими.*

Функция calcNumPetals()

Функция `printDice()` вызывает также функцию `calcNumPetals()` по одному разу для каждой из костей. Эта функция принимает в качестве параметра значение, выпавшее на кости. Она также обращается к глобальной переменной `$numPetals`. В ней используется оператор `switch`, в котором определяется, какое число необходимо добавить к `$numPetals`, если выпала та или иная грань кости.

А вот мы и подошли к секрету программы. Центральная точка на кости – это роза. Любые точки вокруг нее – лепестки. На грани с единицей есть роза, но нет лепестков. На гранях 2, 4 и 6 есть лепестки, но нет самой розы. Грань 3 имеет 2 лепестка, а 5 – четыре. Если на кости выпало 3, `$numPetals` необходимо увеличить на 2, а если выпало 5, к `$numPetals` следует прибавить 4.

```
function calcNumPetals($value){
  global $numPetals;
  switch ($value) {
    case 3:
     $numPetals += 2;
     break;
    case 5:
     $numPetals += 4;
     break;
  } // завершение switch
} // завершение calcNumPetals
```

Команда += представляет собой сокращенную запись. Строка

```
$numPetals += 2;
```

полностью тождественна строке

```
$numPetals = $numPetals + 2;
```

Первая запись намного короче, ее проще набирать на клавиатуре, поэтому большинство программистов предпочитают использовать именно эту форму.

Функция printForm()

Задача функции `printForm()` состоит в том, чтобы вывести форму в нижней части HTML-страницы. Форма является очень простой, за исключением одного скрытого поля, в которое помещается значение `$numPetals`.

```
function printForm(){
  global $numPetals;
  print <<<HERE
  <h3>How many petals around the rose?</h3>
  <form method = "post">
  <input type = "text"
         name = "guess"
         value = "0">
  <input type = "hidden"
         name = "numPetals"
         value = "$numPetals">
  <br>
  <input type = "submit">
  </form>
  <br>
  <a href = "petalHelp.html"
     target = "helpPage">
  give me a hint</a>
HERE;
} // завершение printForm
```

Этот код размещает форму на странице. Большую часть формы можно было бы записать простым HTML, а на PHP только добавить скрытое поле, но начинающему программисту всегда хочется написать как можно больше кода на новом для него языке – PHP в нашем случае. Так проще понять ход работы программы (вывести приветствие, вывести кости, вывести форму).

Завершающий HTML-код

Завершающий набор HTML-команд закрывает страницу. Он завершает участок PHP-кода, закрывает теги, устанавливающие шрифт, центрирование текста, тело страницы и, наконец, сам HTML.

```
?>
</font>
</center>
</body>
</html>
```

Итоги

В этой главе вы узнали много нового. Вы изучили несколько видов команд ветвления, включая операторы `if`, `else` и структуру `switch`. Вы теперь знаете, как создаются функции, которые повышают эффективность работы ваших программ и облегчают чтение кода.

Вы узнали, как в функции передаются параметры и как возвращаются значения. Вы научились обращаться к глобальным переменным из тела функции. Вы собрали все эти знания вместе и создали интересную игру. Вам есть, чем гордиться! В следующей главе вы научитесь пользоваться операторами цикла, что позволит сделать ваши программы еще более мощными.

ДОМАШНЕЕ ЗАДАНИЕ

1. Напишите программу, бросающую кости с 4, 10 и 20 гранями (они иногда используюся в различных играх).
2. Напишите программу, позволяющую пользователю решить, сколько граней будет у кости, и выведите на экран случайно брошенную кость с учетом ограничения по количеству граней (не тратьте время на поиск изображений, выводите просто цифру).
3. Создайте программу «шулерская кость», которая будет выбрасывать 1 в половине случаев и какое-либо другое значение – в другой половине.
4. Измените игру из предыдущей главы таким образом, чтобы форма и программа хранились в одном файле.
5. Создайте генератор веб-страниц. Подготовьте форму, в которую бы вводился заголовок страницы, цвет фона, цвет шрифта и текст страницы. Создайте HTML-страницу с помощью этой формы.

Глава 4

Циклы и массивы: покер в кости

Теперь вам известны все основные элементы, из которых состоит программа, однако существуют и другие, призванные значительно облегчить создание программ и повысить их эффективность. В этой главе вы познакомитесь с двумя очень важными инструментами. Массивы являются особыми переменными, с помощью которых создаются списки. Циклы используются для того, чтобы повторить выполнение отдельного участка кода. Как вы, наверное, догадались, массивы и циклы часто применяются вместе. Вы узнаете, как с помощью этих новых элементов можно создать еще более интересные программы. Вы научитесь:

- создавать простые структуры со счетчиком, использующие цикл `for`;
- применять в цикле `for` разные виды счетчиков;
- использовать оператор `while` для создания циклов с более гибкими условиями;
- правильно задавать граничные условия циклов;
- создавать простые массивы;
- писать программы с использованием массивов и циклов;
- хранить данные в скрытых полях.

Представляем игру «Покер в кости»

Основная программа данной главы – это упрощенная игра в кости. В начале этой игры вы получаете $100 виртуальных денег. Ставка в каждом розыгрыше составляет 2 доллара. Компьютер бросает пять костей. Вы решаете, какую кость следует оставить, а какую – бросить снова. После второго броска компьютер проверяет выпавшие комбинации костей. Вы выигрываете деньги, если выпало два, три, четыре или пять одинаковых цифр или «стрит» (пять цифр по порядку). На рис. 4.1 и 4.2 эта игра показана в действии.

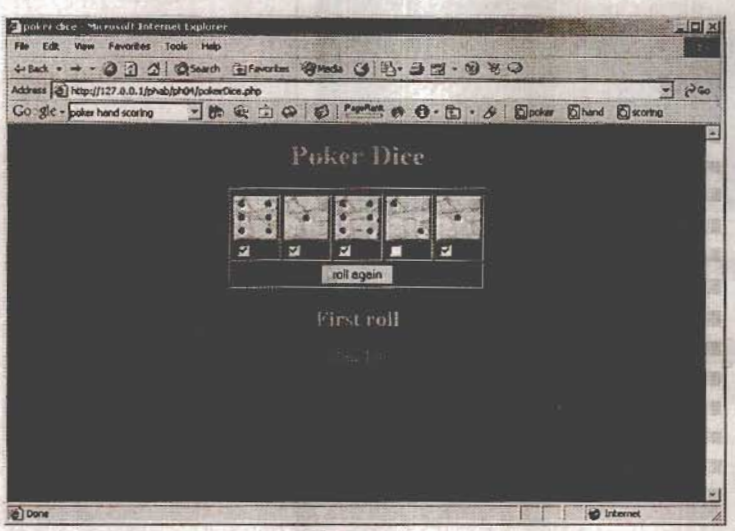


Рис. 4.1. После первого броска вы можете оставить некоторые кости, отметив флажок под каждой из них

Основные идеи этой программы очень похожи на использованные нами ранее при создании предыдущих игр. Хранить данные о каждой из костей может оказаться очень сложно, поэтому в данной программе используются массивы и циклы, позволяющие справиться с таким потоком информации.

Считаем с помощью цикла for

Компьютеры отлично справляются с задачей повторения. Вам часто понадобится сделать так, чтобы компьютер повторил одно и то же действие несколько раз. Например, посмотрите на программу `simpleFor.php`, результат работы которой представлен на рис. 4.3.

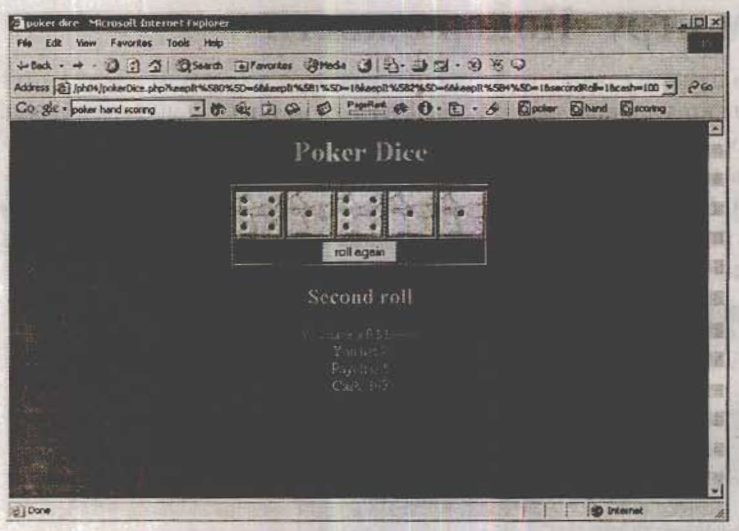


Рис. 4.2. Получив комбинацию «три плюс два», игрок заработал немного денег

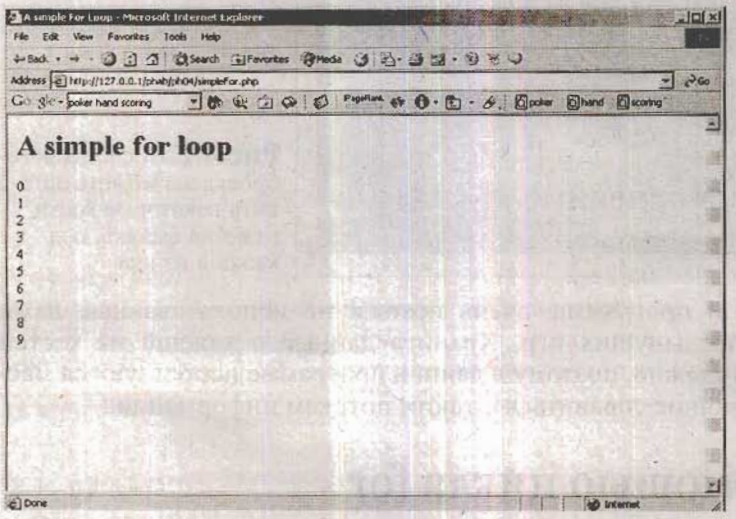


Рис. 4.3. Эта программа считает от нуля до девяти при помощи единственного оператора `print`

Несмотря на то что результат работы программы `simpleFor` ничем не примечателен, сама программа обладает уникальным свойством. Она содержит один-единственный оператор `print`, который выполняется десять раз. Посмотрите на исходный код этой программы, и вы поймете, как она работает.

```
<html>
<head>
<title>
A simple For Loop
</title>
</head>
<body>
<h1>A simple for loop</h1>
<?
for ($i = 0; $i < 10; $i++){
  print "$i <br>\n";
} // завершение цикла for
?>
</body>
</html>
```

Каждое из чисел на экран выводится следующей строкой.

```
print "$i <br>\n";
```

Она может напечатать только одно значение, но в данном случае это происходит десять раз. Такого поведения мы добились посредством оператора `for`. Он состоит из трех основных частей.

Инициализируем переменную цикла

В циклах `for` обычно используют целочисленную переменную. Иногда эту основную переменную цикла называют *счетчиком цикла,* или *управляющей (контрольной)* переменной цикла, поскольку ее значение определяет число выполнений цикла. Первая часть определения цикла – это строка кода, в которой переменная цикла задается и получает некоторое начальное значение. В нашем простом примере инициализация выглядит следующим образом.

```
$i = 0;
```

В данной строке устанавливается, что переменная цикла будет называться `$i` и ее начальным значением будет ноль.

Компьютерные программы, как правило, начинают считать с нуля, поэтому я тоже присваиваю переменной `$i` значение ноль.

Несмотря на то что участок `$i=0;` выглядит как законченная строка кода (каковой он и является), обычно его помещают в одну строку вместе с другими частями описания цикла.

ВОЗВРАЩАЯСЬ К РЕАЛЬНОСТИ

Вы можете спросить, почему переменная цикла называется `$i`. Как и для большинства переменных, лучше всего было бы присваивать переменным цикла имя, отражающее их назначение. Однако бывает так, что в цикле `for` переменная цикла является просто натуральным числом и не выполняет никакой дополнительной функции. В таких случаях в игру вступает старая традиция программистов. На языке Фортран (один из первых широко распространенных языков программирования) имена всех целочисленных переменных должны были начинаться с букв «i», «j» и еще нескольких символов. Поэтому программисты, работающие на Фортране, часто использовали имя «i» для переменных, выполняющих только роль счетчика цикла. И хотя большинство современных программистов не написали на Фортране ни строки кода, традиция сохранилась. Удивительно, насколько сильно уже обросла подобными традициями такая молодая отрасль деятельности, как программирование

Устанавливаем условие завершения цикла

Добиться от компьютера повторения некоторых операций – это обычно самая простая часть работы. Но заставить его остановиться именно тогда, когда это необходимо, – задача более сложная.

Вторая часть цикла `for` представляет собой условие. Пока это условие истинно, цикл продолжает работу. Как только условие становится ложным, происходит выход из цикла. В данном случае я написал условие `$i<10;`. Это означает, что пока значение переменной `$i` меньше 10, цикл продолжает свою работу. Как только программа определяет, что значение `$i` стало равным или больше 10, происходит выход из цикла. Обычно условие в цикле `for` сравнивает переменную цикла с неким граничным значением.

Изменяем значение переменной цикла

Последним из важнейших элементов цикла `for` является механизм изменения значения переменной цикла. В какой-то момент времени значение `$i` должно достичь 10 или превысить этот порог, иначе цикл будет продолжать свою работу вечно. В программе `simpleFor` часть структуры `for`, отвечающая за это, выглядит так: `i++`. Запись `$i++` является командой «добавь единицу к `$i`» и тождественна `$i = $i + 1`. Символ `++` назвается *оператором инкремента (приращения)*, поскольку предоставляет простой путь для *инкрементирования* (увеличения на единицу) переменной.

Создаем цикл

Как только все части цикла for собраны вместе, его становится очень просто использовать. Поместите фигурные скобки ({}) вокруг кода и сдвиньте вправо весь текст программы, находящийся внутри цикла. Между фигурными скобками можно поместить сколько угодно строк кода, в том числе операторы ветвления и другие циклы. Переменная цикла ведет себя внутри него особым образом. Сначала ей присваивается значение, записанное в строке инициации. При каждом повторении это значение будет изменяться, как описано в структуре `for`, и интерпретатор будет проверять условие, чтобы убедиться, осталось ли оно истинным. Если да, код цикла будет выполнен снова. В случае с программой `simpleFor` – в начале работы цикла `$i` равно нулю. При первом вызове команды `print` будет напечатан ноль, потому что таково в этот момент значение переменной `$i`. Когда интерпретатор дойдет до правой фигурной скобки, обозначающей окончание цикла, он увеличит `$i` на единицу (следуя команде `i++` в структуре `for`) и проверит условие `($i < 10)`. Поскольку 1 меньше 10, условие окажется истинным, и код, расположенный в теле цикла, будет выполнен снова. В конце концов, значение `$i` станет равным `10`, и тогда условие `($i < 10)` перестанет быть истинным. Интерпретатор перейдет к первой строке кода, расположенной после окончания цикла, которая в нашем случае и завершает программу.

Изменяем цикл for

Теперь, когда вы понимаете основы применения цикла `for`, можно внести в него несколько интересных изменений. Вы можете создать цикл, который будет изменять значение переменной цикла не на единицу, а на пять, или такой, который считает в обратном порядке.

Считаем в цикле с шагом пять

Программа `countByFive.php`, результат работы которой показан на рис. 4.4, иллюстрирует пример цикла, на каждом шаге изменяющего значение переменной цикла на пять.

Программа очень похожа на `simpleFor`, но содержит пару интересных моментов.

```
<html>
<head>
<title>
Counting By Fives
</title>
```

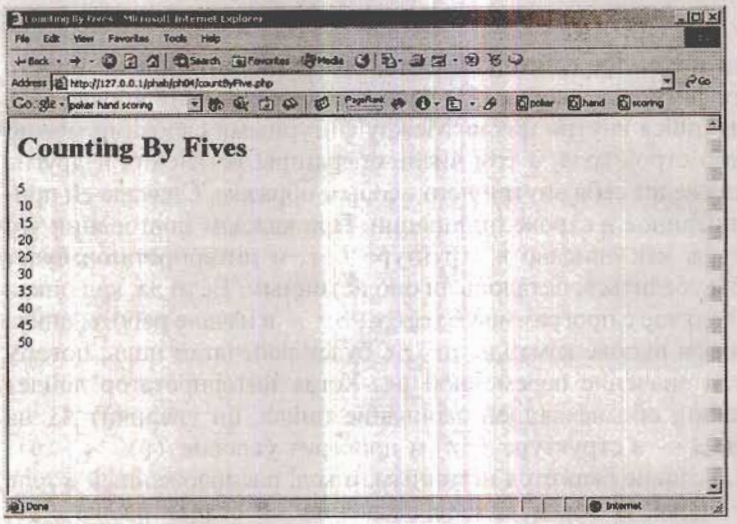


Рис. 4.4. Эта программа перебирает в цикле `for` числа, кратные пяти

```
</head>
<body>
<h1>Counting By Fives</h1>
<?

for ($i = 5; $i <= 50; $i+= 5){
  print "$i <br>\n";
} // завершение цикла for
?>
</body>
</html>
```

Единственное, что здесь изменилось, – это параметры оператора `for`. Поскольку глупо было бы считать с 0, в качестве первоначального значения `$i` я устанавливаю равным 5. Кроме того, я решил, что программа должна остановиться, когда `$i` достигнет значения 50 (после десяти повторений). Каждый раз в цикле `$i` будет увеличиваться на 5. Синтакс `+=` используется для увеличения переменной. Строка

```
$i += 5;
```

полностью соответствует строке

```
$i = $i + 5;
```

Считаем в обратном порядке

Можно очень просто изменить цикл `for` таким образом, чтобы он стал считать в обратном порядке. Рис. 4.5 иллюстрирует такую возможность.

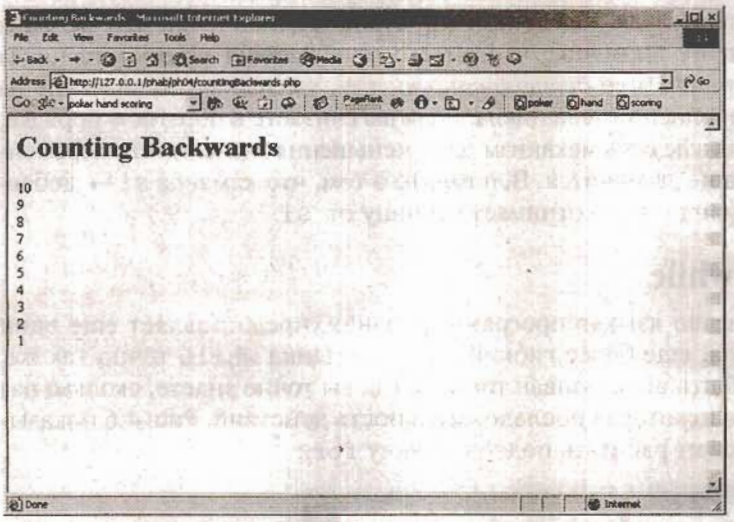


Рис. 4.5. Эта программа считает в обратном порядке от десяти до одного, используя для этого цикл `for`

Как и в предыдущем разделе, основная структура очень похожа на использованную в первой программе с циклом `for`, но, поменяв параметры структуры `for`, я добился того, что поведение программы изменилось. Исходный текст этой программы показывает, как это было сделано.

```
<html>

<head>
<title>
Counting Backwards
</title>
</head>

<body>

<h1>Counting Backwards</h1>

<?

for ($i = 10; $i > 0; $i--){
  print "$i <br>\n";
} // завершение цикла for

?>

</body>
</html>
```

Если вы уже разобрались в том, как действует цикл `for`, то все изменения должны быть вам понятны. На этот раз я считаю в обратном порядке, поэтому `$i` начинает с большего значения (в данном случае с 10). Условие продолжения цикла сейчас выглядит как `$i > 0`, и это означает, что цикл будет продолжать свою работу до тех пор, пока `$i` больше нуля. Как только `$i` станет равной нулю или меньше, цикл завершится. Обратите внимание, что вместо прибавления некоторого числа к переменной `$i`, на этот раз я ее *декрементирую*, т. е., уменьшаю ее значение на единицу при каждом выполнении тела цикла. Если вы считаете в обратном порядке, убедитесь, что в вашем цикле есть механизм для уменьшения значения его переменной, иначе цикл никогда не закончится. Вспомните о том, что команда `$i++` добавляет единицу к `$i`, а команда `$i --` отнимает единицу от `$i`.

Используем цикл while

PHP, как и большинство языков программирования, предоставляет еще один вариант оператора цикла, еще более гибкий, чем `for`. Цикл `while` точно так же, как и цикл `for`, может быть использован тогда, когда вы точно знаете, сколько раз должна быть выполнена некоторая последовательность действий. Рис. 4.6 показывает, что цикл `while` может работать подобно циклу `for`.:

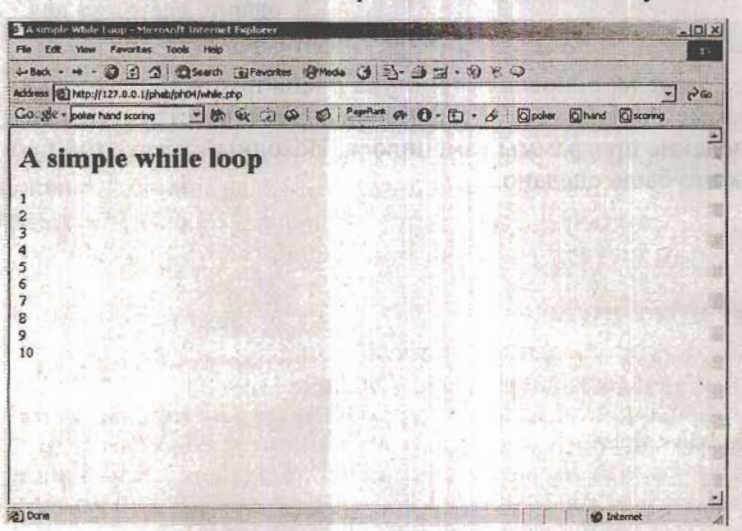


Рис. 4.6. Несмотря на то, что результат работы этой программы похож на результат первой программы с циклом `for`, на этот раз для его достижения использована другая конструкция

Повторяем выполнение кода в цикле while

Код программы `while.php` очень похож на код, демонстрирующий работу цикла `for`, но, как вы заметили, конструкция цикла здесь немного проще.

```
<html>
<head>
<title>
A simple While Loop
</title>
</head>
<body>
<h1>A simple while loop</h1>
<?
$i = 1;
while ($i <= 10){
  print "$i <br>\n";
  $i++;
} // завершение цикла while
?>
</body>
</html>
```

Для цикла `while` требуется только один параметр – условие. Цикл будет повторяться до тех пор, пока условие истинно. Как только условие окажется ложным, цикл завершит свою работу. Рассматриваемая программа начинается с инициализации переменной `$i`, затем она сравнивает эту переменную с числом десять при помощи оператора «меньше или равно» в цикле `while`. В теле цикла программа выводит на экран текущее значение `$i` и увеличивает `$i` на единицу.

Распознаем бесконечные циклы

Гибкая конструкция `while` обладает огромной мощью, но вместе с этой мощью в нее заложены и возможные проблемы. Циклы `while` очень легко строить, но неправильно работающий цикл может доставить немало неприятностей. Может получиться так, что код, расположенный внутри цикла, ни разу не будет выполнен. Или, хуже того, вы можете допустить какую-либо логическую ошибку, из-за которой выполнение цикла будет продолжаться бесконечно. В качестве примера посмотрите на следующий код.

```
<html>
<head>
<title>
```

```
A bad While Loop
</title>
</head>
<body>
<h1>A bad while loop</h1>
<?
$i = 1;
while ($i <= 10){
  print "$i <br>\n";
  $j++;
} // завершение цикла while
?>
</body>
</html>
```

Программа badWhile.php предназначена специально для того, чтобы показать, что происходит, когда в программе появляется бесконечный цикл. Если вы запустите эту программу, она может на какое-то время замедлить работу всего веб-сервера. Убедитесь в том, что ваш сервер настроен так, чтобы останавливать PHP-процесс, когда пользователь нажимает на кнопку `Stop` *в своем браузере. (Для большинства вариантов установки языка PHP эта настройка включена по умолчанию.)*

Программа badWhile.php содержит мелкую, но фатальную ошибку. Посмотрите внимательно на ее исходный код и попробуйте обнаружить ее самостоятельно. Этот код выглядит точь-в-точь как наша первая программа с циклом `while`, за исключением того что вместо инкрементирования `$i` в нем инкрементируется `$j`. Переменная `$j` не имеет никакого отношения к переменной `$i`, поэтому значение `$i` не изменяется. Цикл будет продолжать и продолжать свое выполнение, поскольку он не может закончиться, пока `$i` не станет строго больше десяти, а этого не произойдет никогда. Эта программа представляет собой пример классического «бесконечного цикла». Каждый программист хотя бы раз в жизни случайно создавал нечто подобное, и вас эта участь тоже не минует.

Обычно виновником бесконечного цикла является небрежно придуманное имя переменной, опечатка или ошибка в регистре. Если вы используете в качестве счетчика цикла переменную наподобие `$myCounter`*, а потом инкрементируете* `$MyCounter`*, PHP определит их как две совершенно разные переменные, и ваша программа будет работать неправильно. Вот еще одна причина для того, чтобы быть последовательным в именовании переменных и использовании больших букв.*

Создаем корректный цикл

К счастью, существует несколько основных принципов построения цикла, который будет работать именно так, как вам требуется. Более того, вы уже изучили эти основные положения, поскольку они заложены в структуру цикла `for`. При написании цикла `while`, вы должны четко выполнить три вещи.

1. Создать переменную цикла.
2. Написать условие.
3. Убедиться, что цикл имеет завершение.

Следующие разделы мы посвятим обсуждению каждого из этих положений.

Создаем и инициализируем переменную цикла

Если ваш цикл построен на значении переменной (существуют и другие варианты), убедитесь в том, что вы определили эту переменную, что она имеет соответствующую область видимости и ей присвоено разумное начальное значение. Хорошо бы также проверить, что цикл с этим значением выполняется как минимум один раз (хотя бы в тех случаях, когда вам необходимо именно это). Создание переменной во многом напоминает этап инициализации в цикле `for`.

Создаем условие для продолжения работы цикла

Условие, как правило, будет сравнивать переменную с неким значением. Убедитесь в том, что для вашего условия существуют случаи, когда оно выполняется, и случаи, когда оно нарушается. Самое трудное состоит не в том, чтобы написать цикл, а в том, чтобы программа вышла из него в нужный момент. Это условие очень похоже на условие в цикле `for`.

Убеждаемся, что цикл имеет завершение

В цикле должна существовать некая команда, которая бы изменяла переменную цикла таким образом, чтобы тот в итоге завершился. Код этой команды должен находиться в теле цикла. Убедитесь в том, что переменная цикла имеет возможность принять значение, необходимое для выхода из цикла, т. е., для нарушения условия его продолжения.

Работаем с простыми массивами

Программирование – это совмещение управляющих структур (таких, как циклы) и структур данных (таких, как переменные). Вы изучили очень мощные операторы циклов. Сейчас пришло время обратиться к структуре данных, которая естественным образом взаимодействует с циклами. *Массивы* – это особые пере-

менные, созданные для хранения списков данных. В языке PHP работа с массивами реализована очень просто. Посмотрите на рис. 4.7. Эта программа, называющаяся `basicArray.php`, демонстрирует работу с двумя массивами.

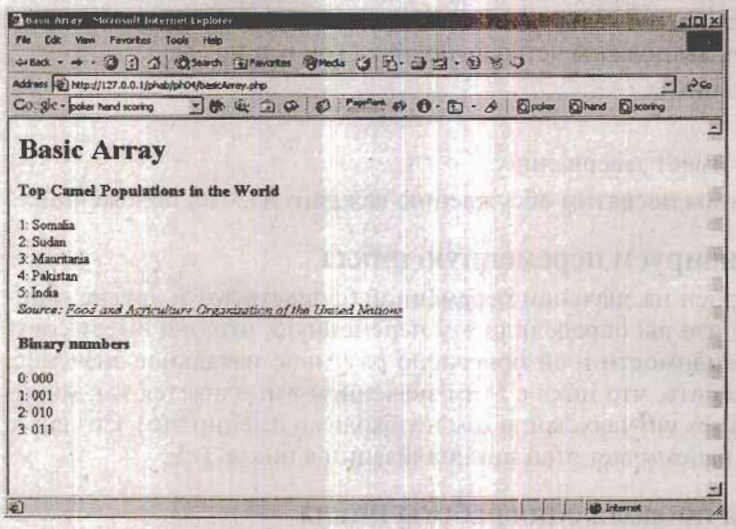


Рис. 4.7. Данные, показанные на этой странице, хранятся в двух массивах

Сначала посмотрите на полный текст программы, а потом я расскажу вам о том, как она работает.

```
<html>
<head>
<title>
Basic Array
</title>
</head>
<body>
<h1>Basic Array</h1>
<?
//просто присвоить значения массиву
$camelPop[1] = "Somalia";
$camelPop[2] = "Sudan";
$camelPop[3] = "Mauritania";
$camelPop[4] = "Pakistan";
$camelPop[5] = "India";
//вывести значения массива
print "<h3>Top Camel Populations in the World</h3>\n";
for ($i = 1; $i <= 5; $i++){
```

```
  print "$i: $camelPop[$i]<br>\n";
} // завершение цикла for

print "<i>Source: <a href = http://www.fao.org/ag/aga/glipha/
index.jsp> Food and Agriculture
Organization of the United Nations</a></i>\n";

//использовать функциюarray для заполнения массива
$binary = array("000", "001", "010", "011");

print "<h3>Binary numbers</h3>\n";
for ($i = 0; $i < count($binary); $i++){
  print "$i: $binary[$i]<br>\n";
} // завершение цикла for

?>

</body>
</html>
```

Создаем простой массив

Посмотрите на строки кода, описывающие переменную `$camelPop`.

```
//просто присвоить значения массиву
$camelPop[1] = "Somalia";
$camelPop[2] = "Sudan";
$camelPop[3] = "Mauritania";
$camelPop[4] = "Pakistan";
$camelPop[5] = "India";
```

Переменная `$camelPop` предназначена для хранения названий пяти стран с наибольшим поголовьем верблюдов в мире. (Если массивы остались для вас загадкой, то, по крайней мере, вы уже подчерпнули в этой главе что-то новое для себя!) Поскольку `$camelPop` должна хранить названия пяти разных стран, понятно, что это массив (это словечко из жаргона компьютерщиков, обозначающее слово «список»), а не просто переменная. Единственное отличие `$camelPop` от переменных, с которыми мы работали до настоящего момента, состоит в том, что она может иметь несколько значений. Для разделения этих значений используется индекс, заключенный в квадратные скобки.

У боксера Джорджа Формана (George Foreman) есть несколько сыновей, каждого из которых тоже зовут Джордж. Мне всегда было любопытно, как поступает его супруга, когда ей нужно, чтобы кто-то вынес мусор. Подозреваю, что каждому из Джорджей она присвоила отдельный номер, исключающий путаницу. Массивы работают именно так. Каждый из элементов имеет одно и то же имя, но собственный цифровой индекс, чтобы их можно было различать.

Во многих языках программирования необходимо явно описывать переменные-массивы, но в PHP все организовано значительно проще. Просто присвойте переменной с индексом в квадратных скобках значение – и массив создан.

Несмотря на то что PHP очень по-доброму реагирует на создание массивов «на лету», некоторые веб-серверы (такие, на которых опции сообщения об ошибках присвоено значение `E_ALL`*) могут в этом случае выдать предупреждение. В этом случае вы можете явно создать пустой массив с помощью функции* `array()`*, описанной ниже, а затем присваивать ему значения.*

Используем цикл для исследования содержимого массива

Массивы самым естественным образом сочетаются с циклами `for`. Если у вас есть переменная-массив, вам очень часто понадобится последовательно пройти по всем ее значениям и совершить какое-либо действие с каждым из них. В данном примере я хочу вывести индекс и соответствующее ему название страны. Вот цикл `for`, выполняющий эту задачу.

```
//вывести значения массива
print "<h3>Top Camel Populations in the World</h3>\n";
for ($i = 1; $i <= 5; $i++){
  print "$i: $camelPop[$i]<br>\n";
} // завершение цикла for
```

Поскольку мне известно, что индексы нашего массива изменяются от 1 до 5 (включительно), я настроил цикл таким образом, чтобы значение переменной `$i` пробегало диапазон от 1 до 5. В теле цикла я просто вывожу на экран индекс (`$i`) и название соответствующей страны (`$camelPop[$i]`). При первом проходе по циклу `$i` будет равна 1, так что `$camelPop[$i]` будет равным `$camelPop[1]`, т. е. «Somalia». При каждом проходе значение `$i` будет увеличено на единицу, поэтому в итоге каждый элемен массива появится на странице.

Преимущество совместного использования массивов и циклов состоит в том, что это удобно. Если вам необходимо что-либо сделать с каждым из элементов массива, достаточно написать код, выполняющий ваше действие, один раз, и поместить его в цикл. Это бывает особенно эффективно при разработке программ, оперирующих с большими объемами данных. Например, если мне понадобится вывести поголовье верблюдов для каждой страны, присутствующей в базе данных ООН, а не только для первых пяти стран, то все, что я должен буду сделать, – это создать больший массив и изменить параметры цикла `for`*.*

Заполняем массив значениями с помощью функкции array()

Очень часто вам будет заранее точно известно, какие именно значения необходимо поместить в массив. В PHP предусмотрен более короткий путь для заполнения массива набором значений.

```
//использовать функцию array для заполнения массива
$binary = array("000", "001", "010", "011");
```

В этом примере я создал массив из четырех первых двоичных чисел (начиная с нуля). Для того чтобы заполнить массив списком значений, вы можете воспользоваться ключевым словом `array`. Заметьте, что при использовании этого приема индексы элементов массива создаются автоматически.

Большинство языков программирования начинают счет не с единицы, к чему привыки мы, люди, а с нуля. Это может вызвать путаницу. Когда PHP создает массив автоматически, его первым индексом будет ноль, а не единица.

Определяем размер массива

Массивы позволяют сделать ваш код более гибким. Вам не обязательно знать, сколько именно элементов содержит тот или иной массив, потому что PHP содержит функцию `count()`, которая может определить количество элементов массива. В следующем коде я использовал функцию `count()` для вычисления размера массива.

```
print "<h3>Binary numbers</h3>\n";
for ($i = 0; $i < count($binary); $i++){
  print "$i: $binary[$i]<br>\n";
} // завершение цикла for
```

Заметьте, что мой счетчик цикла проходит диапазон от нуля до значения, на единицу меньшего, чем количество элементов в массиве. Если в массиве содержится четыре элемента и индексы начинаются с нуля, последний из индексов будет равен трем. Это стандартный способ обработки массива в цикле.

Поскольку очень часто возникает необходимость пробежать по всем элементам массива, PHP предоставляет еще один вид цикла, с помощью которого эта задача решается еще проще. Вы будете изучать данный цикл в главе 5. А пока постарайтесь как следует разобраться в том, как при работе с массивами используется обыкновенный цикл `for`.

Улучшаем программу «This Old Man» с помощью массивов и циклов

Программа `basicArray.php` познакомила вас с тем, как создавать массивы, но ее нельзя считать иллюстрацией всей мощи совместного применения массивов и циклов. Для того чтобы лучше понять, какую помощь они могут оказать программисту, давайте вернемся к нашей старой знакомой – программе из главы 3. Версия программы «This Old Man», показанная на рис. 4.8, выглядит очень похоже на то, что мы видели в главе 3, но ее код занимает значительно меньше места.

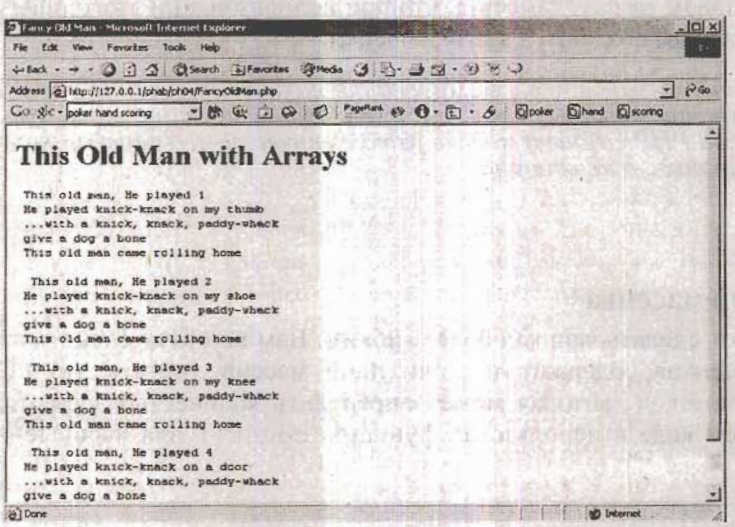


Рис. 4.8. Программа «Fancy Old Man» использует более компактную структуру, которую легче изменять

Улучшения в этой версии программы заметны, только если заглянуть внутрь.

```
<html>
<head>
<title>
Fancy Old Man
</title>
</head>
<body>
<h1>This Old Man with Arrays</h1>
<pre>
<?
$place = array(
  "",
  "on my thumb",
  "on my shoe",
```

```
  "on my knee",
  "on a door");
// вывести текст песни
for ($verse = 1; $verse <= 4; $verse++){
print <<<HERE
  This old man, He played $verse
  He played knick-knack $place[$verse]
  ...with a knick, knack, paddy-whack
  give a dog a bone
  This old man came rolling home
HERE;
} // завершение цикла for
?>
</pre>
</body>
</html>
```

Эта обновленная версия в полной мере использует тот факт, что единственные изменения от куплета к куплету касаются номера куплета и места, где этот куплет исполняется. Вы можете создать массив из мест исполнения, и это сильно упростит запись текста песни.

Создаем массив мест исполнения

Я обратил внимание на то, что каждое место исполнения ассоциируется с определенной цифрой. С помощью директивы `array()` я заполнил массив `$place` соответствующими значениями. Нет никакого места, связанного с нулем, поэтому нулевой элемент я оставил пустым.

```
$place = array(
  "",
  "on my thumb",
  "on my shoe",
  "on my knee",
  "on a door");
```

Заметьте, что, как и в большинстве случаев при программировании на PHP, перевод строки не играет при написании кода никакой роли. Я решил расположить каждое место на отдельной строке только потому, что это лучше смотрится.

Записываем текст песни

Наша песня состоит целиком из повторов. Все куплеты повторяют один другой, за исключением номера куплета и соответствующего ему места. Для каждого куплета значением переменной `$verse` будет текущий номер куплета. Соответствующее ему место хранится в переменной `$place[$verse]`. Код, выводящий полный текст песни, состоит из одного большого оператора `print`, выполняемого в цикле `for`.

```
// вывести текст песни
for ($verse = 1; $verse <= 4; $verse++){

print <<<HERE
 This old man, He played $verse
 He played knick-knack $place[$verse]
 ...with a knick, knack, paddy-whack
 give a dog a bone
 This old man came rolling home
HERE;
} // завершение цикла for
```

Программа «Fancy Old Man» наглядно иллюстрирует выигрыш, который мы получили от использования массивов. Создание программы, правильно применяющей массивы, обычно требует чуть больше времени на планирование, чем это требуется для программы, состоящей из одних управляющих структур (такой, какие мы рассматривали в главе 3). Однако начальное увеличение объема работы очень хорошо себя оправдывает, поскольку такую программу потом легче будет изменять и расширять.

Сохраняем данные между запусками программы

При разработке обычных программ, как правило, исходят из того, что программа постоянно ведет с пользователем диалог. После запуска она может что-либо запросить у пользователя, потом вывести ответ на введенные пользователем данные и продолжать это взаимодействие до тех пор, пока пользователь тем или иным образом не даст понять, что желает завершить работу. Программы, предназначенные для выполнения на веб-серверах, построены по-другому. PHP-программы, которые вы создаете, имеют очень короткое время жизни. Когда пользователь отправляет на веб-сервер запрос для вашей PHP-программы, сервер запускает интерпретатор языка PHP (программу, преобразовывающую PHP-код в команды на машинном языке, с которыми на самом деле работает сервер). Результатом работы программы становится веб-страница, отправляемая в браузер пользователя. После того как страница отправлена пользователю, PHP-программа закрывается, по-

скольку ее работа на этом окончена. Веб-серверы не поддерживают постоянное соединение с браузером после отправки страницы. Каждый запрос пользователя воспринимается ими как совершенно новая операция. Вам может показаться, что программа «Покер в кости», представленная в начале этой главы, может взаимодействовать с пользователем неограниченное время. На самом же деле она просто последовательно вызывает себя раз за разом. Эта программа по-разному действует в разных обстоятельствах, а значит, ей необходимо каким-то образом хранить информацию о том, в каком состоянии она сейчас находится.

ВОЗВРАЩАЯСЬ К РЕАЛЬНОСТИ

Низкоуровневый веб-протокол (HTTP), используемый веб-серверами, не поддерживает открытое соединение дольше, чем необходимо. Это называется *протокол без состояния*. Для такого поведения существуют объективные причины. Представьте себе, что ваша программа выполняется на сервере в течение всего времени, когда кто-либо смотрит на ее веб-страницу. А если кто-то запустит ее, а сам отправится спать? В этом случае серверу пришлось бы поддерживать соединение для этой страницы всю ночь. Не забудьте, что в одно и то же время к вашей программе могуть обращаться тысячи человек. Нагрузка по поддержанию всех этих постоянных соединений оказалась бы для сервера весьма тяжелой. Обмен данными с использованием протокола без состояния оптимизирует работу вашего сервера, но за эту оптимизацию приходится расплачиваться. По существу, после каждого запуска ваши программы забывают обо всем, что было сделано ранее, и необходим какой-то дополнительный механизм для того, чтобы определить, на каком шаге программа находится в данный момент

Считаем в полях формы

Существует несколько путей хранения информации, о которых вы узнаете чуть позже по ходу этой книги. Самый простой подход заключается в том, чтобы спрятать эти данные в странице пользователя. В качестве примера посмотрите на рис. 4.9 и 4.10.

Каждый раз, когда вы нажимаете на кнопку `Submit` программы `persistence`, счетчики увеличиваются на единицу. Кажется, что работа программы `persistence` противоречит самой природе программ, выполняемых на стороне сервера, – она как будто запоминает предыдущее значение счетчика и каждый раз прибавляет к нему единицу. На самом деле если два пользователя станут работать с программой `persistence` одновременно, она будет корректно работать для каждого из них.

Чтобы разобраться в работе этой программы, посмотрите на ее исходный код.

```
<html>
<head>
```

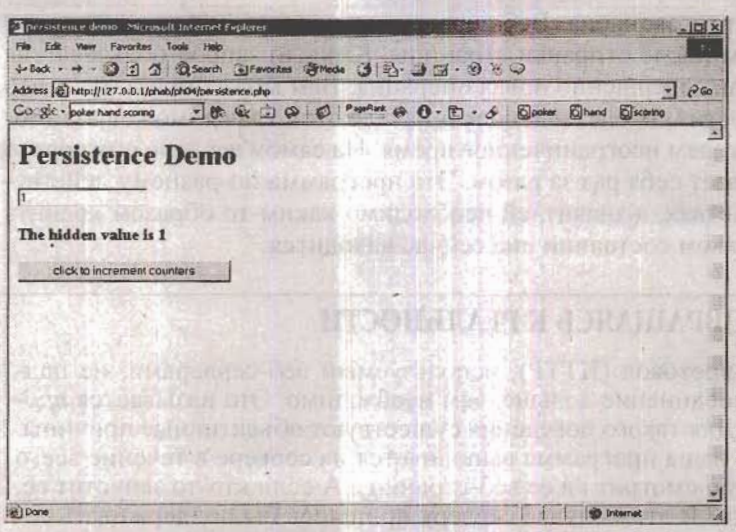


Рис. 4.9. Программа содержит два счетчика, каждый из которых при первом запуске равен единице

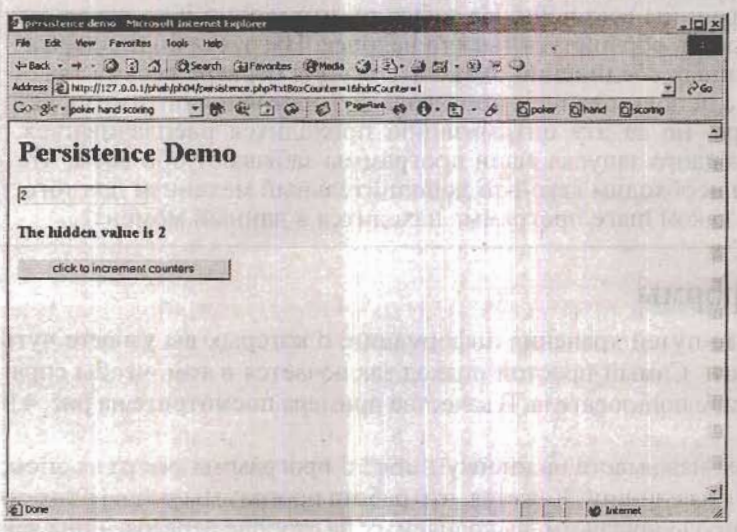


Рис. 4.10. После нажатия на кнопку `Submit` оба значения увеличиваются на единицу

```
<title>
persistence demo
</title>
</head>
<body>
<h1>Persistence Demo</h1>
```

```
<form>
<?
//инкрементируем счетчики
$txtBoxCounter++;
$hdnCounter++;
print <<<HERE
<input type = "text"
       name = "txtBoxCounter"
       value = "$txtBoxCounter">
<input type = "hidden"
       name = "hdnCounter"
       value = "$hdnCounter">
<h3>The hidden value is $hdnCounter</h3>
<input type = "submit"
       value = "click to increment counters">
HERE;
?>
</form>
</body>
</html>
```

Сохраняем данные в текстовом поле

Эта программа содержит две переменные, `$txtBoxCounter` и `$hdnCounter`. Для начала обратим все внимание на `$txtBoxCounter`. Эта переменная связана с текстовым полем. При запуске программа берет значение `$txtBoxCounter` (если эта переменная существует) и добавляет к нему единицу. При выводе текстового поля на экран, программа автоматически помещает в него значение `$txtBoxCounter`. Поскольку в описании формы не определен атрибут `action`, при нажатии на кнопку `Submit` программа вызовет сама себя. На этот раз переменная `$txtBoxCounter` уже будет содержать значение (это будет 1). При очередном запуске программа снова увеличит значение `$txtBoxCounter` на единицу и поместит новое значение (2) в текстовое поле. При каждом запуске программы, она сохранит в текстовом поле значение, которое ей потребуется при *следующем* запуске.

Используем скрытое поле для надежного хранения данных

В предыдущем примере текстовое поле нас полностью устраивало, поскольку сразу видно значение, содержащееся в нем. Однако при написании настоящих программ использование текстового поля может создать серьезные сложности. Пользо-

ватель может изменять содержимое текстового поля, а значит, он может поместить туда любые данные и создать вам уйму неприятностей. Скрытые поля – вот невоспетые герои программирования на стороне сервера. Посмотрите на переменную `$hdnCounter` в исходном коде. Это скрытое поле тоже содержит счетчик, но поскольку оно скрытое, пользователь его не увидит. И тем не менее, значение переменной `$hdnCounter` будет отправлено в PHP-программу, определяемую атрибутом `action` формы. Программа может производить с этим значением любые действия, в том числе и выводить его в теле HTML-страницы.

Очень часто при необходимости сохранить данные между вызовами программы, вы будете пользоваться для этого скрытыми полями на веб-странице, отправляемой прользователю.

ВОЗВРАЩАЯСЬ К РЕАЛЬНОСТИ

Технология использования скрытых полей отлично работает, когда необходимо хранить небольшие объемы информации, но она крайне неэффективна и ненадежна при работе с более серьезными формами данных. По мере прочтения этой книги вы научитесь еще нескольким способам сохранения требуемых данных, включая файлы и базы данных

Создаем программу «Покер в кости»

Настало время снова взглянуть на программу «Покер в кости», дебют которой состоялся в начале этой главы. Как обычно, она не содержит ни одного элемента, который бы вы не изучили к настоящему времени. Она немного сложнее, чем простые примеры, которые я вам показывал по ходу этой главы, но занимает удивительно мало места, учитывая, как много всего она делает. Вряд ли вас удивит то, что секрет успеха этой программы заключен в использовании массивов и циклов.

Создаем начальный HTML-код

Как всегда, основой PHP-программы служит простая HTML-страница. Я решил добавить к ней небольшую таблицу стилей, чтобы вывести желто-коричневые буквы на зеленом фоне.

```
<html>
<head>
<title>poker dice</title>
<style type = "text/css">
body {
 background: green;
```

```
 color: tan;
}
</style>
</head>
<body>
<center>
<h1>Poker Dice</h1>
<form>
<?
```

Разрабатываем основной код программы

Наша программа является достаточно длинной для использования функций. В целях обучения я разбил ее на несколько небольших участков, но вы можете также просмотреть ее полный код, записанный на компакт-диске, поставляемом вместе с книгой.

Основной код используется для описания хода работы программы в целом. Большая часть работы выполняется в функциях, вызываемых из основного кода.

```
//Смотрим, впервые ли запущена программа
if (empty($cash)){
 $cash = 100;
} // завершение if

rollDice();

if ($secondRoll == TRUE){
 print "<h2>Second roll</h2>\n";
 $secondRoll = FALSE;
 evaluate();
} else {
 print "<h2>First roll</h2>\n";
 $secondRoll = TRUE;
} // завершение if

printStuff();
```

Первое, что стоит на повестке дня, – это проверка, впервые ли пользователь пришел на нашу страницу. Очень важно понимать, какую роль в выполнении этой программы играет согласованность действий. Пользователь будет считать, что он играет в одну и ту же игру на протяжении нескольких ходов, тогда как на самом деле при каждом броске костей программа запускается по-новому. Она будет выполнять разные действия в зависимости от того, какие из элементов формы содержат значения (если такие есть). Если пользователь никогда раньше не был на

этой странице, переменная $cash будет содержать пустое значение. Первый оператор if, проверяет данное условие. Если переменная $cash еще не создана, пользователь получит начальную сумму в $100. (Как жаль, что так не поступают в настоящих казино...)

Затем программа вызывает функцию rollDice(), которую мы очень скоро рассмотрим. Эта функция бросает кости и выводит их на экран.

Если вы внимательно посмотрите на то, как выполняется наша программа, то заметите, что она работает в двух разных режимах. Каждая игра состоит из двух возможных бросков. После первого броска пользователь может сохранить ту или иную кость при помощи флажка, а подсчет итогов в этот момент не производится. После второго броска флажков на странице уже нет (потому что новую игру пользователь должен начинать с броска всех костей), а программа подсчитывает результат игры, добавляя пользователю денег за различные выпавшие комбинации.

Переменная $secondRoll используется для того, чтобы программа знала, какой бросок сейчас выполняется – первый или второй. Я решил, что когда выполняется второй бросок, эта переменная должна содержать значение TRUE, а когда выполняется первый бросок – значение FALSE. Если $secondRoll содержит TRUE, программа вызовет функцию evaluate(), которая подсчитает выигрыш или проигрыш. Независимо от этого я сообщаю пользователю о том, какой бросок выполнен, и изменяю значение переменной $secondRoll так, чтобы она показывала, что должно быть сделано, когда программа будет вызвана в следующий раз (это случится, когда пользователь щелкнет по кнопке Submit).

Создаем функцию rollDice()

Работа функции rollDice() заключается, как можно догадаться, в том, чтобы бросить кости. Это довольно-таки длинная функция, и я сначала покажу вам полный ее текст, а потом объясню, разбив на меньшие куски. По существу, эта функция создает HTML-таблицу на основе пяти бросков кости. Она может определить, пожелал ли пользователь сохранить какую-либо из костей предыдущего броска, и совершает новый бросок, только если пользователь не захотел сохранить значение кости. Если бросок является первым, программа выводит флажок, с помощью которого пользователь может указать, что хочет сохранить эту кость.

```
function rollDice(){
 global $die, $secondRoll, $keepIt;
 print "<table border = 1><td><tr>";
 for ($i = 0; $i < 5; $i++){
  if ($keepIt[$i] == ""){
   $die[$i] = rand(1, 6);
  } else {
```

```
    $die[$i] = $keepIt[$i];
  } // завершение if
 $theFile = "die" . $die[$i] . ".jpg";
 //вывести изображения костей
 print <<<HERE
 <td>
 <img src = "$theFile"
      height = 50
      width = 50><br>
HERE;
 //вывести флажки только при первом броске
 if ($secondRoll == FALSE){
  print <<<HERE
  <input type = "checkbox"
         name = "keepIt[$i]"
         value = $die[$i]>
  </td>
HERE;
  } // завершение if
 } // завершение цикла for
 //Вывести кнопку submit и окончание таблицы
 print <<<HERE
 </tr></td>
 <tr>
  <td colspan = "5">
  <center>
  <input type = "submit"
         value = "roll again">
  </center>
  </td>
 </tr>
 </table>
HERE;
} // завершение функции rollDice
```

Флажки, которые то появляются, то исчезают, представляют собой отдельную задачу. Главная стратегия работы с ними звучит так: если это первый бросок, выводим флажок под каждой костью. Флажки называются `keepIt`, и каждый из них имеет индекс. Когда PHP увидит несколько переменных с одним и тем же именем, но разными индексами, он автоматически создаст массив. Работа с флажками на PHP немного отличается от работы с прочими элементами, поскольку флажок

передает значение, только если он отмечен. Значение любого флажка, который не отмечен пользователем, не будет передано в программу. Если флажок отмечен, то значение, назначенное ему, в программу передается.

Если флажок не был отмечен, в программу не передается никакого значения.

Бросаем кости если это необходимо

В программе имеется два массива для хранения значений, выпавших на костях. Массив `$die` используется для хранения текущего значения каждой кости. Массив `$keepIt` содержит значения только в том случае, если пользователь отметил соответствующий флажок (а это можно сделать только после первого броска, поскольку после второго флажки не будут показаны на странице).

```
if ($keepIt[$i] == ""){
    $die[$i] = rand(1, 6);
  } else {
    $die[$i] = $keepIt[$i];
  } // завершение if
  $theFile = "die" . $die[$i] . ".jpg";
```

Значение каждой кости, которую пользователь решил сохранить, будет записано в соответствующий элемент массива `$keepIt`. Если значение сохранено, оно будет переписано в массив `$die`. В противном случае программа выбросит на кости новое случайное значение.

Выводим содержимое таблицы

После того как получены значения каждой кости (копированием из `$keepIt` или новым броском), настало время вывести на страницу изображения, соответствующие полученным значениям костей.

```
//вывести изображения костей
 print <<<HERE
 <td>
 <img src = "$theFile"
      height = 50
      width = 50><br>
HERE;
 //вывести флажки только при первом броске
 if ($secondRoll == FALSE){
```

```
  print <<<HERE
  <input type = "checkbox"
         name = "keepIt[$i]"
         value = $die[$i]>
  </td>
HERE;
 } // завершение if
```

Если бросок является первым, то функция, кроме всего прочего, выводит под каждой костью флажок `keepIt`. Обратите внимание на то, что имя флажка согласовано с именем кости. (Не забудьте о том, что перед выводом HTML-страницы переменная `$i` будет заменена ее значением.) Значение каждой кости хранится в поле `value` флажка `keepIt`.

Не пугайтесь, если окажется, что вам совсем ничего не понятно. Бывает очень трудно осознать, как отдельные куски кода работают вместе. Попробуйте запустить программу несколько раз и внимательно посмотрите на создаваемый ей HTML-код – это должно помочь. Если вы хотите полностью разобраться в PHP-программе, недостаточно видеть только то, что лежит на поверхности и доступно конечному пользователю. Часто бывает необходимо заглянуть в HTML и посмотреть на скрытые элементы, чтобы понять, что происходит.

Выводим окончание таблицы

После цикла, который бросает и выводит на экран кости, совсем нетрудно поместить кнопку отправки формы и окончание HTML-таблицы.

```
//Вывести кнопку submit и окончание таблицы
 print <<<HERE
 </tr></td>
 <tr>
  <td colspan = "5">
  <center>
  <input type = "submit"
         value = "roll again">
  </center>
  </td>
 </tr>
 </table>
HERE;
```

Обратите внимание на то, что, поскольку в описании формы отсутствует тег `action`, PHP по умолчанию обратится к той же самой странице, которая содержит

форму. Это очень удобно при работе с программами, подобными этой, которые последовательно вызывают сами себя.

Создаем функцию evaluate()

Задача функции `evaluate()` состоит в том, чтобы исследовать массив `$die` и определить, присутствуют ли в нем комбинации, достойные выигрыша. И снова я сначала привожу полный текст функции, а ниже прокомментирую наиболее интересные ее участки.

```
function evaluate(){
 global $die, $cash;
 //обнуляем сумму выигрыша
 $payoff = 0;

 //вычитаем ставку на этот кон
 $cash -= 2;

 //считаем кости
 $numVals = array(6);
 for ($theVal = 1; $theVal <= 6; $theVal++){
  for ($dieNum = 0; $dieNum < 5; $dieNum++){
   if ($die[$dieNum] == $theVal){
    $numVals[$theVal]++;
   } // завершение if
  } // завершение цикла for для dieNum
 } // завршение цикла for для theVal

 //выводим результаты
 // for ($i = 1; $i <= 6; $i++){
 // print "$i: $numVals[$i]<br>\n";
 // } // завршение цикла for

 //считаем комбинации по два, три, четыре, пять
 $numPairs = 0;
 $numThrees = 0;
 $numFours = 0;
 $numFives = 0;

 for ($i = 1; $i <= 6; $i++){
  switch ($numVals[$i]){
   case 2:
    $numPairs++;
    break;
   case 3:
    $numThrees++;
    break;
```

```
  case 4:
   $numFours++;
   break;
  case 5:
   $numFives++;
   break;
 } // завершение switch
} // завершние цикла for

//проверяем, есть ли две пары
if ($numPairs == 2){
 print "You have two pairs!<br>\n";
 $payoff = 1;
} // завершение if

//проверяем, есть ли три или три плюс два
if ($numThrees == 1){
 if ($numPairs == 1){
  //три плюс два
  print "You have a full house!<br>\n";
  $payoff = 5;
 } else {
  print "You have three of a kind!<br>\n";
  $payoff = 2;
 } // завершение if для пары
} // завершение if для трех

//ищем четыре одинаковых значения
if ($numFours == 1){
 print "You have four of a kind!<br>\n";
 $payoff = 5;
} // завершение if

//ищем пять одинаковых значений
if ($numFives == 1){
 print "You got five of a kind!<br>\n";
 $payoff = 10;
} // завершение if

//ищем стрит
if (($numVals[1] == 1)
 && ($numVals[2] == 1)
 && ($numVals[3] == 1)
 && ($numVals[4] == 1)
 && ($numVals[5] == 1)){
 print "You have a straight!<br>\n";
 $payoff = 10;
```

```
} // завершение if
if (($numVals[2] == 1)
 && ($numVals[3] == 1)
 && ($numVals[4] == 1)
 && ($numVals[5] == 1)
 && ($numVals[6] == 1)){
 print "You have a straight!<br>\n";
 $payoff = 10;
} // завершение if
print "You bet 2<br>\n";
print "Payoff is $payoff<br>\n";
$cash += $payoff;
} // завершение функции evaluate
```

Основная стратегия функции `evaluate()` заключается в том, чтобы каждый кон получать от игрока ставку в размере двух долларов. (Поменяйте это правило – и игра станет проще или сложнее.) Затем я создаю массив с именем `$numVals`, в котором записывается, сколько раз выпало то или иное значение. При анализе массива `$numVals` проще определить различные комбинации, чем если бы мы искали их непосредственно в массиве `$die`. Остаток функции проверяет наличие каждой из возможных комбинаций и рассчитывает выигрыш.

Подсчитываем значения костей

При определении различных выигрышных комбинаций в этой игре очень важно знать, сколько раз выпало то или иное значение. Пользователь получает очки за две, три, четыре и пять одинаковых костей, а также за стрит (пять значений по порядку). Я создал новый массив с именем `$numVals`, имеющий шесть элементов. `$numVals[1]` содержит количество единиц, выпавших на костях, `$numVals[2]` покажет, сколько выпало двоек, и т. д.

```
//считаем кости
$numVals = array(6);
for ($theVal = 1; $theVal <= 6; $theVal++){
 for ($dieNum = 0; $dieNum < 5; $dieNum++){
  if ($die[$dieNum] == $theVal){
   $numVals[$theVal]++;
  } // конец if
 } // завершение цикла for для dieNum
} // завершение цикла for для theVal
//выводим результаты
// for ($i = 1; $i <= 6; $i++){
// print "$i: $numVals[$i]<br>\n";
// } // заврешение цикла for
```

При построении массива `$numVals` я в цикле `for` прохожу каждое возможное значение кости (от 1 до 6). И для каждого значения с помощью еще одного цикла `for` проверяю все кости, чтобы определить, не выпало ли на них данное значение. (Другими словами, во внешнем цикле я сначала ищу единицы, затем двойки, тройки и т. д.) Если я нахожу текущее значение, то увеличиваю на единицу число, содержащееся в `$numVals[$theVal]`.

Обратите внимание на закомментированные строки в конце этого участка. Код, подсчитывающий выигрыш, потерял бы всякий смысл, если бы массив `$numVals` начал вдруг работать не так, как ожидается, поэтому я на скорую руку написал цикл, который сообщал мне, сколько каких значений обнаружила моя программа. Таким образом, я сразу же убедился, что программа работает как надо, и только затем стал дописывать к ней новые возможности. Очень полезно бывает время от времени проверять уже написанный код подобным образом, с тем чтобы убедиться, что все работает именно так, как вы ожидали. Увидев, что все функционирует должным образом, я решил поместить знак комментария перед каждой строкой, чтобы временно ее отключить. Сделав так, я убрал этот код из программы, но сохранил его под руками на случай, если что-то пойдет не так и мне снова захочется увидеть содержимое массива `$numVals`.

Считаем комбинации по два, три, четыре, пять

Массив `$numVals` содержит основную часть информации, которая мне требуется, но пока еще она представлена в не совсем подходящем формате. Пользователь получает деньги за две, три, четыре и пять выпавших одинаковых костей. Для проверки этих условий я использую еще несколько переменных и новый цикл, просматривающий массив `$numVals`.

```
//считаем комбинации по два, три, четыре, пять
 $numPairs = 0;
 $numThrees = 0;
 $numFours = 0;
 $numFives = 0;
 for ($i = 1; $i <= 6; $i++){
  switch ($numVals[$i]){
   case 2:
    $numPairs++;
    break;
   case 3:
    $numThrees++;
    break;
   case 4:
    $numFours++;
```

```
    break;
   case 5:
    $numFives++;
    break;
  } // завершение switch
 } // завершение for
```

Сначала я создал переменные для подсчета количества комбинаций по две, три, четыре и пять одинаковых костей. Каждой из этих переменных я присвоил значение «ноль». Затем я прошел по массиву `$numVals`, чтобы определить, сколько раз встретилось каждое значение. Например, если на костях выпало 1,1,5,5,5, то `$numVals[1]` будет равно 2, а `$numVals[5]` – 3. После выполнения оператора switch, `$numPairs` будет содержать 1 и `$numThrees` тоже будет равно 1. Все остальные переменные `$num` по-прежнему будут содержать ноль. Создание этих переменных сильно облегчает мне работу по определению того, сколько очков необходимо начислить игроку (и нужно ли их начислять вовсе).

Ищем две пары одинаковых костей

Работа по настройке переменных для подсчета комбинаций полностью себя оправдывает, поскольку после этого очень легко определить, когда настало время начислять выигрыш. Я решил выплачивать пользователю один доллар за две пары (за одну он не получит ничего). Если значение `$numPairs` равно 2, значит, пользователь набрал две пары и переменной `$payoff` присваивается значение 1.

```
//проверяем, есть ли две пары
 if ($numPairs == 2){
  print "You have two pairs!<br>\n";
  $payoff = 1;
 } // завершение if
```

Конечно, вы можете изменить размеры выплат, как пожелаете. Сейчас программа ведет себя необычайно щедро, но так играть с ней только веселее.

Ищем три и три плюс две

Я совместил проверки на три одинаковых кости и комбинацию «три плюс две». Сначала данный код проверяет по значению переменной `$numThrees`, есть ли три одинаковых кости. Если таковые есть, программа ищет, не найдется ли пара. Если оба условия выполняются, значит, выпала комбинация «три плюс две» и пользователь получает соответствующий выигрыш. Если же пары нет, небольшой выигрыш все равно выплачивается за три одинаковых кости.

```
//проверяем, есть ли три или три плюс два
 if ($numThrees == 1){
  if ($numPairs == 1){
   //три плюс два
   print "You have a full house!<br>\n";
   $payoff = 5;
  } else {
   print "You have three of a kind!<br>\n";
   $payoff = 2;
  } // завершение if для пары
 } // завершение if для трех
```

Ищем четыре и пять одинаковых костей

Поиск четырех и пяти одинаковых костей очень прост. Все, что нужно сделать, это проверить соответствующие переменные.

```
//ищем четыре одинаковых кости
 if ($numFours == 1){
  print "You have four of a kind!<br>\n";
  $payoff = 5;
 } // завершение if
 //ищем пять одинаковых костей
 if ($numFives == 1){
  print "You got five of a kind!<br>\n";
  $payoff = 10;
 } // завершение if
```

Ищем комбинацию «стрит»

Поиск комбинации «стрит» представляет немного более сложную задачу, поскольку всего возможны две таких комбинации. У пользователя могут выпасть все кости от 1 до 5 или от 2 до 6. Для того чтобы обнаружить обе эти комбинации, я написал два составных условия.

```
//ищем стрит
 if (($numVals[1] == 1)
  && ($numVals[2] == 1)
  && ($numVals[3] == 1)
  && ($numVals[4] == 1)
  && ($numVals[5] == 1)){
  print "You have a straight!<br>\n";
  $payoff = 10;
 } // завершение if
```

```
if (($numVals[2] == 1)
 && ($numVals[3] == 1)
 && ($numVals[4] == 1)
 && ($numVals[5] == 1)
 && ($numVals[6] == 1)){
 print "You have a straight!<br>\n";
 $payoff = 10;
} // завершение if
```

Обратите внимание на то, что каждый оператор `if` содержит условие, составленное из нескольких подусловий, объединенных при помощи оператора `&&`. Оператор `&&` носит название *«логическое и»* (читается «и»). Только в том случае, если истинным окажется каждое из подусловий, результатом вычисления условия будет `true`.

Эти условия очень похожи одно на другое. Они просто проверяют обе возможные комбинации «стрит».

Выводим результаты

Последняя функция программы выводит на страницу пользователя значения различных переменных. Значение `$cash` показывает, сколько у пользователя осталось денег. Два скрытых элемента хранят данные, которые потребуются программе для следующего запуска. Элемент `secondRoll` содержит значение `true` или `false`, показывая, следует ли очередной бросок считать вторым. Элемент `cash` указывает, сколько денег должно быть у пользователя во время следующего кона.

```
function printStuff(){
 global $cash, $secondRoll;
 print "Cash: $cash\n";
 //сохраняем переменные в скрытых полях
 print <<<HERE
 <input type = "hidden"
        name = "secondRoll"
        value = "$secondRoll">
 <input type = "hidden"
        name = "cash"
        value = "$cash">
HERE;

} // завершение printStuff
```

Итоги

Вы уже приближаетесь к завершению базового курса программирования. К вашему багажу навыков добавились основы создания циклов. Теперь ваши программы могут повторять некоторые действия на основании условий, которые вы им зададите. Вы умеете писать циклы `for`, работающие в прямом, обратном порядке, а также с некоторым шагом. Кроме того, вы умеете создавать циклы `while`. Вы знаете основные принципы написания корректно работающих циклов. Вы умеете формировать массивы как вручную, так и с помощью директивы `array()`. Вы можете в цикле пройти по всем элементам массива. Вы узнали, как ваша программа может сохранять данные между запусками, помещая их в поля форм на выводимой странице. Вы собрали все эти навыки вместе и создали интересную игру. В главе 5 вы научитесь более сложным приемам работы с массивами и циклами, создадите более мощные массивы и будете использовать специализированные циклические структуры.

ДОМАШНЕЕ ЗАДАНИЕ

1. Измените игру «Покер в кости» каким-либо образом. Добавьте собственный фон, поменяйте изображения костей или модифицируйте суммы выплат, сместив игровой баланс по своему желанию
2. Напишите классическую игру «Угадай число». Пусть компьютер создает случайное число и дает пользователю возможность его угадать. Сообщайте пользователю, является ли его ответ слишком большим, слишком маленьким или правильным. После того как пользователь угадает значение, сообщите ему, сколько попыток для этого потребовалось. Для этой игры массивы не нужны, зато понадобится хранить данные в скрытых полях формы.
3. Перепишите игру «Угадай число» наоборот. Пусть пользователь загадывает случайное число от 1 до 100, а компьютер угадывает правильный ответ. Пользователь будет выбирать из вариантов «слишком много», «слишком мало», «угадал». Ваш алгоритм должен угадывать любое число не больше, чем за семь ходов.
4. Напишите программу, случайным образом сдающую карты для игры в покер. Изображения карт можно найти по ссылке http://waste.org/~oxymoron/cards/ или взять из другого источника. Вашей программе не нужно определять, какая комбинация выпала. Она должна просто сдать пять случайных карт. Для хранения колоды используйте массив.

Глава 5

Улучшенная обработка массивов и строк

На данный момент вы немного научились работать с информацией в ваших PHP-программах. В этой главе вы познакомитесь с некоторыми важными концепциями, позволяющими более эффективно работать с данными. Вы узнаете о новых интересных способах работы с массивами и текстовой информацией. В частности, вы научитесь:

- управлять массивами с помощью цикла `foreach`;
- создавать и использовать ассоциативные массивы;
- извлекать полезную информацию из некоторых массивов, встроенных в PHP;
- создавать основные двухмерные массивы;
- создавать двухмерные ассоциативные массивы;
- разбивать строку на более мелкие части;
- искать строки внутри других строк.

Знакомимся с игрой «Поиск слов»

К концу этой главы вы сможете написать забавную программу, которая создает головоломки из введенных пользователем слов, как показано на рис. 5.1.

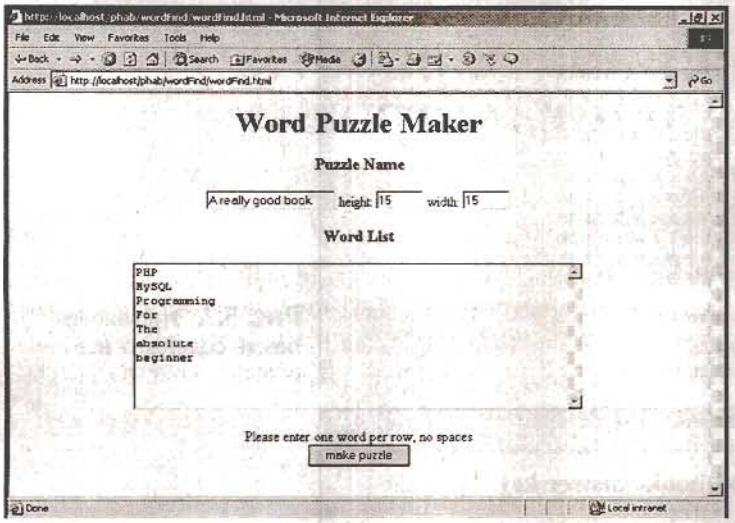


Рис. 5.1. Пользователь вводит список слов и размер головоломки

После этого, используя введенный пользователем список слов, программа пытается создать головоломку. (Это не всегда возможно, однако обычно программе удается создать корректную головоломку). Один из возможных вариантов для слов, указанных на рис. 5.1, показан на рис. 5.2.

По желанию программа также создает решение для головоломки. Эта возможность показана на рис. 5.3.

Секрет программы, создающей головоломки (как и секрет большинства компьютерных программ), заключается в управлении данными. Как только я определил хорошую схему для работы с данными в программе, само программирование было уже достаточно простым.

Используем цикл foreach для работы с массивом

Как я упоминал в главе 4 «Циклы и массивы: Покер в кости», циклы `for` и массивы зачастую используются вместе. Фактически, PHP предоставляет особый тип цикла, который еще больше упрощает последовательное прохождение по элементам массива.

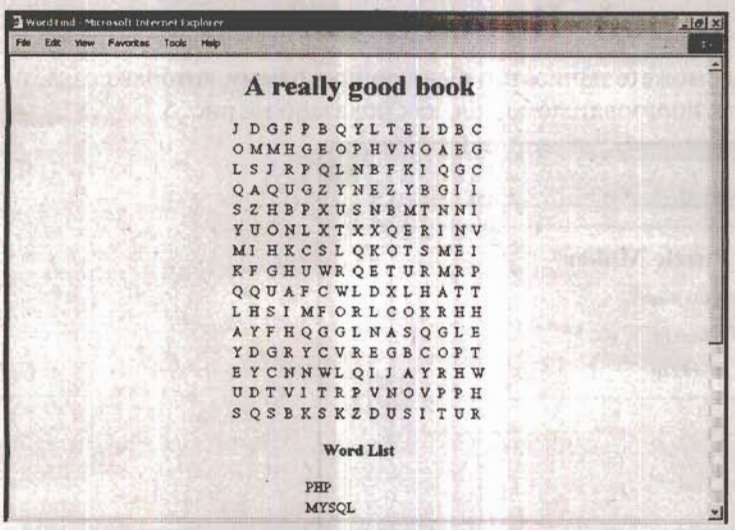


Рис. 5.2. Эта головоломка содержит все слова из списка

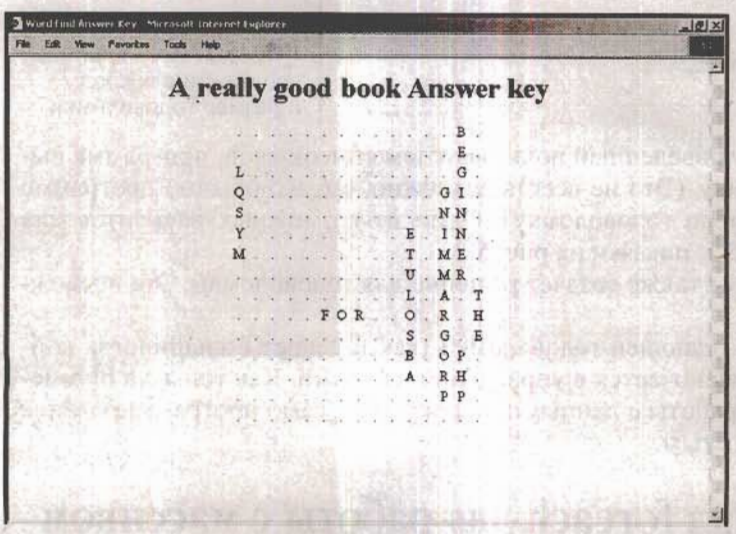


Рис. 5.3. А вот решение головоломки

Знакомимся с программой foreach.php

Показанная на рис. 5.4 программа иллюстрирует работу цикла `foreach`.

Сама по себе эта HTML-страница ничем не примечательна, но она была создана удивительно простым кодом.

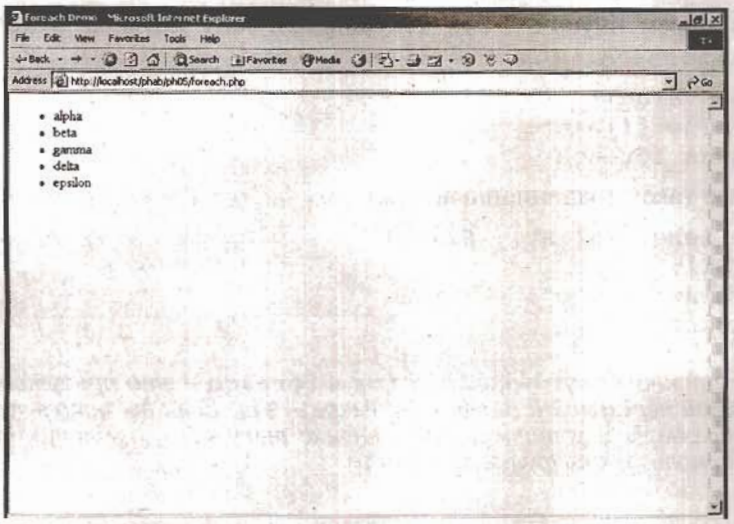


Рис. 5.4. Хотя это и выглядит как обычный HTML, на самом деле эта страница создана с помощью массива и цикла `foreach`

```
<html>
<head>
<title>Foreach Demo</title>
</head>
<body>
<h1>Foreach Demo</h1>
<?
$list = array("alpha", "beta", "gamma", "delta", "epsilon");
print "<ul>\n";
foreach ($list as $value){
  print "  <li>$value</li>\n";
} // завершение цикла foreach
print "</ul>\n";
?>
</body>
</html>
```

Все значения, которые будут в списке, занесены в массив `$list` с помощью функции `array()`.

Цикл `foreach` работает почти так же, как цикл `for`, за исключением того, что он немного проще. Первый параметр конструкции `foreach` – это массив (в нашем случае `$list`). Ключевое слово as указывает на имя переменной, в которой будет содержать каждое значение по очереди. В нашем случае, цикл `foreach` пройдет через массив `$list` столько раз, сколько это необходимо. На каждом витке цикла

функция поместит в переменную `$value` текущий элемент массива `$list`. В сущности, этот цикл `foreach`.

```
foreach ($list as $value){
  print "  <li>$value</li>\n";
} //завершение цикла foreach
```

работает так же, как вот такой традиционный цикл `for`.

```
for ($i = 0; $i < lehgth($list); $i++){
  $value = $list[$i];
  print "  <li>$value</li>\n";
} // завершение цикла for
```

Основное отличие между циклами `for` *и* `foreach` *– это присутствие ключевой переменной (в этом примере –* `$i`*). Если вы используете цикл* `foreach` *и хотите узнать индекс текущего элемента, вы можете использовать функцию* `key()`.

Цикл `foreach` может быть очень полезнен, если необходимо пройтись через каждое значение массива. Поскольку эта задача встречается часто, умение работать с циклом `foreach` очень важно. По мере знакомства с другими типами массивов вы научитесь изменять цикл `foreach` для работы с ними.

Создаем ассоциативный массив

PHP славится своими чрезвычайно гибкими массивами. Вы можете создать значительное количество интересных и полезных типов массивов, помимо тех, которые вы уже использовали. Один из самых удобных типов массивов называется «ассоциативный массив».

Несмотря на такое сложное название, ассоциативный массив очень похож на обычный. В то время как обычный массив использует числовое индексирование, ассоциативные массивы используют строковые индексы. На рис. 5.5 показана страница, созданная с помощью двух ассоциативных массивов.

Изучаем программу assoc.php

Представьте, что вы хотите хранить список столиц. Вы, конечно, можете хранить города в массиве. Однако если самое интересное для вас – это связь между штатом и его столицей, то, используя обычные массивы, управлять этой связью будет сложно. В нашем примере удобно было бы использовать в качестве индекса массива название штата вместо числа.

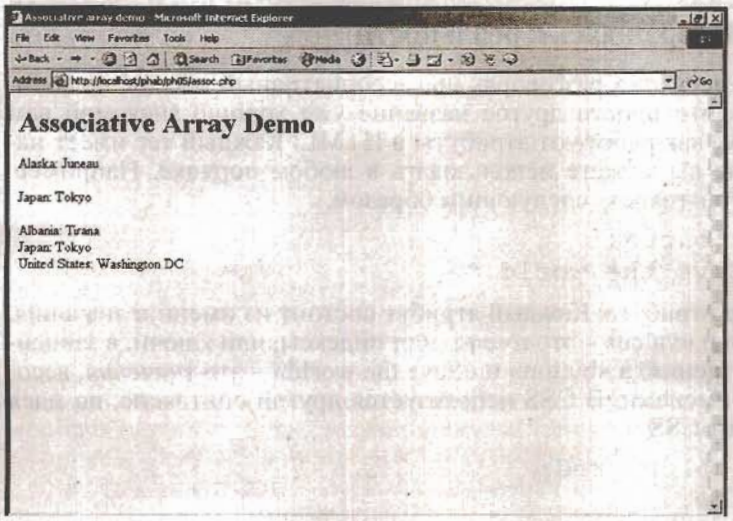


Рис. 5.5. Эта страница использует ассоциативные массивы, чтобы связать страны и штаты с их столицами

Построение ассоциативного массива

Вот код из программы `assoc.php`, который создает массив столиц штатов.

```
$stateCap["Alaska"] = "Juneau";
$stateCap["Indiana"] = "Indianapolis";
$stateCap["Michigan"] = "Lansing";
```

За исключением строковых значений индексов, ассоциативный массив больше ничем не отличается от обычного. Обратите внимание, что индексы должны быть заключены в кавычки. Как только вы создали ассоциативный массив, он используется так же, как и обычный.

```
print "Alaska: ";
print $stateCap["Alaska"];
print "<br><br>";
```

И еще раз обратите внимание, что индекс массива заключен в кавычки. Ассоциативная форма отлично подходит для данных вроде информации о столице штата. В сущности, она позволяет «найти» столицу по названию штата.

Возвращаясь к реальности

Если вы запутались во всех этих разговорах про ассоциативный массив, не паникуйте. На самом деле, это просто другое название уже хорошо знакомой вам вещи. Подумайте о том, как работают атрибуты в HTML. Каждый тег имеет набор атрибутов, которые вы можете использовать в любом порядке. Например, обычная кнопка может выглядеть следующим образом.

```
<input type = "button"
    value = "Save the world.">
```

У этой кнопки есть два атрибута. Каждый атрибут состоит из имени и значения. Ключевые слова «type» и «value» – это *имена* (или индексы, или ключи, в зависимости от вашей точки зрения) а «button» и «Save the world» – это *значения*, ассоциированные с этими именами. В CSS используется другой синтаксис, но идея точно такая же. Элемент CSS

```
P {background-color: red;
   color: yellow;
   font-size: 14pt}
```

указывает на некоторые изменения тега абзаца. И хотя синтаксис здесь другой, все равно применяется та же модель. Самая важная часть определения CSS является списком пар имен и значений.

Есть еще одно место, где можно встретить ассоциативные массивы. Информация, передающаяся из HTML-формы в вашу программу, приходит в виде ассоциативного массива. Имя каждого элемента становится индексом, а значение этого элемента формы переводится в значение элемента массива. Чуть дальше в этой главе вы увидите, как можно выгодно этим воспользоваться.

Ассоциативный массив – это просто структура данных, используемая, когда отношения типа имя/значение являются простейшим способом работы с какими-либо данными

Построение ассоциативного массива с помощью функции array()

Зная значения, которые вы хотите хранить в вашем массиве, вы можете использовать функцию `array()`, чтобы построить ассоциативный массив. Однако создание ассоциативных массивов требует немного другого синтаксиса, в отличие от обычных массивов, которые вы встречали ранее в этой главе. Я создал массив `$worldCap`, используя синтаксис `array()`.

```
$worldCap = array(
  "Albania"=>"Tirana",
  "Japan"=>"Tokyo",
  "USA"=>"Washington DC"
);
```

Когда вы создаете обычный массив, функция `array()` требует данные, но не требует от вас перечисления индексов. Она автоматически создает индекс каждого элемента, используя следующее доступное целое число. В ассоциативном массиве на вас лежит ответственность за предоставление как данных, так и индексов. Общий формат для таких присвоений содержит особый оператор присваивания. Оператор => указывает на то, что элемент содержит какое-либо значение. Я обычно читаю его как «содержит», т. е. вы можете сказать «Japan содержит Tokyo». Другими словами, «Japan» => «Tokyo» указывает, что PHP следует создать элемент массива с индексом «Japan» и присвоить ему значение «Tokyo». Вы можете получить доступ к значению этого массива так же, как к любому другому ассоциативному массиву.

```
print "Japan: ";
print $worldCap["Japan"];
print "<br><br>";
```

Используем цикл foreach в ассоциативных массивах

Цикл `foreach` так же удобен в использовании с ассоциативными массивами, как и с обычными, однако для ассоциативных массивов используется немного другой синтаксис. Посмотрите на этот код из программы `assoc.php`.

```
foreach ($worldCap as $country => $capital) {
  print "$country: $capital<br>\n";
} // завершение цикла foreach
```

При работе с обычными массивами в цикле `foreach` используется только одна переменная, потому что индекс можно легко вычислить. В ассоциативном массиве каждый элемент в массиве имеет уникальный индекс и значение. Ассоциативная форма цикла `foreach` учитывает это и содержит две переменные. В первой переменной содержится индекс, во второй – значение, соответствующее этому индексу. Внутри цикла вы можете обращаться к текущему индексу и значению, используя любые имена переменных, указанные вами в структуре цикла `foreach`. При каждом выполнении цикла вы получаете имя и значение. В этом примере имя будет храниться в переменной `$country`, потому что все индексы в этом массиве являются названиями стран. При каждом выполнении цикла переменная `$country` будет принимать разное значение, а переменная `$capital` будет содержать значение элемента массива, соответствующее значению переменной `$country`.

В отличие от традиционных массивов, ассоциативный массив не дает никаких гарантий относительно того, в каком порядке цикл `foreach` *будет возвращать элементы массива. Так что если вам надо, чтобы элементы появлялись в определенном порядке, вам придется обращаться к ним напрямую.*

Используем встроенные ассоциативные массивы

Ассоциативные массивы чрезвычайно удобны, потому что они воспроизводят очень часто используемый тип хранилища информации. Фактически, вы использовали замаскированные ассоциативные массивы, начиная со второй главы этой книги. Всякий раз, когда PHP-скрипт получает данные из формы, эти данные на самом деле хранятся в нескольких ассоциативных массивах. Для каждого элемента формы PHP автоматически создает переменную. Однако вы не можете всегда надеяться на это «волшебное свойство». Все чаще администраторы серверов отключают такое «автоматическое создание переменных» из соображений безопасности. Фактически, в стандартной настройке PHP эта опция (со странным именем `register_globals`) теперь выключена. Так что знание о том, как PHP получает данные из формы, пригодится вам в качестве хорошего примера использования ассоциативных массивов. Помимо этого, вам наверняка потребуется получать данные из формы, не используя автоматически созданные для вас переменные.

Знакомимся с программой formReader.php

Программа `FormReader.php` на самом деле является одной из первых программ, написанных мной на PHP, и я до сих пор часто ее использую. Она очень удобна, потому что может получить данные из любой HTML-формы и вывести имена и значения всех элементов формы на странице. Например, на рис. 5.6 изображена типичная веб-страница с формой.

Когда пользователь нажимает на кнопку `Submit Query` («Отправить запрос»), `formReader` выдает простую диагностику, как показано на рис. 5.7.

Используем массив $_REQUEST

Программа `formReader` делает свою работу, используя ассоциативный массив, встроенный в PHP. До сих пор вы просто полагались на то, что PHP создает для вас переменную по входным элементам любой формы, которая вызвала вашу программу. Такое автоматическое создание переменных называется `register_globals`. Хотя, конечно, это очень удобное средство, оно может быть опасным, поэтому некоторые администраторы его отключают. Однако даже если `register_globals` активно, вам может пригодиться знание других путей доступа к информации, получаемой из формы.

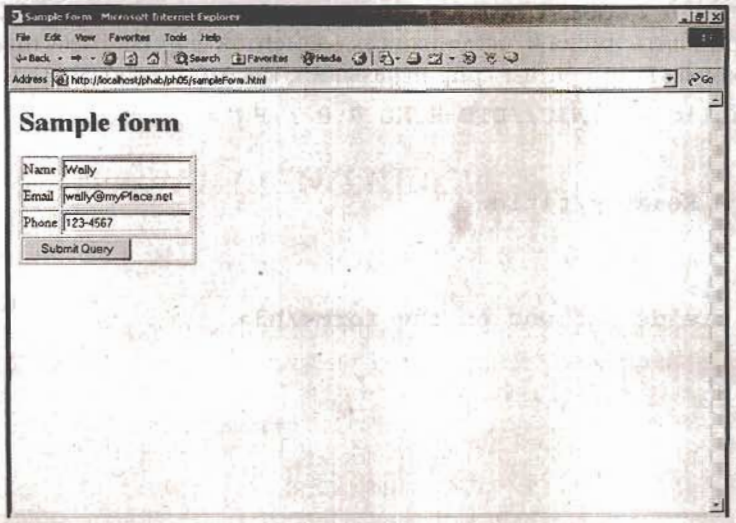


Рис. 5.6. Эта форма содержит три простых поля. Она вызовет программу `formReader.php`

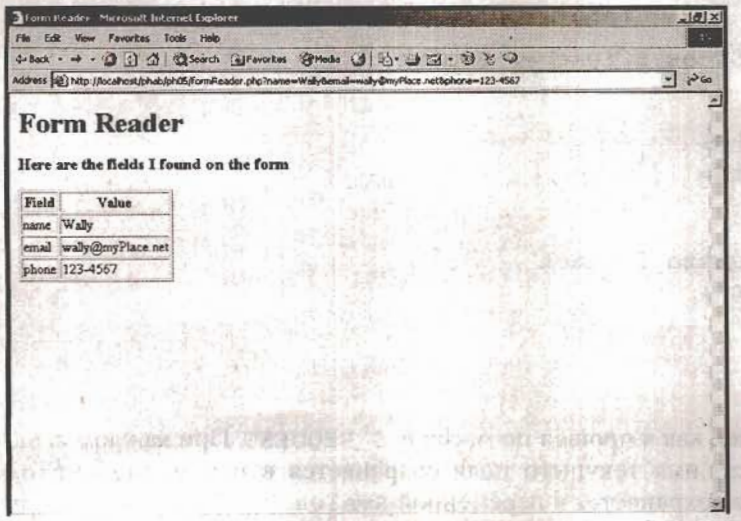


Рис. 5.7. Программа `formReader` определяет каждое поле и его значение

Все поля, отправленные вашей программе, автоматически сохраняются в специальном ассоциативном массиве, имеющем имя `$_REQUEST`. Имя каждого поля в исходной форме становится ключом, а значение этого поля становится значением, ассоциированным с этим ключом. Если у вас в форме было поле userName, вы можете получить значение этого поля, вызвав `$_REQUEST["userName"]`.

Помимо этого, массив $_REQUEST полезен тем, что, используя цикл foreach, вы можете быстро определить имена и значения всех элементов формы, известных программе. Код программы formReader.php показывает, как это можно сделать.

```
<!doctype html public "-//W3C//DTD HTML 4.0 //EN">
<html>
<head>
       <title>Form Reader</title>
</head>
<body>
<h1>Form Reader</h1>
<h3>Here are the fields I found on the form</h3>
<?
print <<<HERE
<table border = 1>
<tr>
  <th>Field</th>
  <th>Value</th>
</tr>
HERE;
foreach ($_REQUEST as $field => $value){
  print <<<HERE
  <tr>
    <td>$field</td>
    <td>$value</td>
  </tr>
HERE;
} // завершение цикла foreach
print "</table>\n";
?>
</body>
</html>
```

Обратите внимание, как я прошел по массиву $_REQUEST. При каждом выполнении цикла foreach имя текущего поля сохраняется в переменной $field, а значение этого поля сохраняется в переменной $value.

Я использую этот скрипт для отладки своих программ. Если я не получаю элементы, которых я ожидал получить от формы, я добавляю такой цикл в начале моей программы, чтобы убедиться в том, что я точно знаю, какие данные были переданы программе. Такая методика часто помогает найти опечатки или другие ошибки в программе.

Возвращаясь к реальности

PHP предоставляет несколько других переменных, похожих на `$_REQUEST`. Массив `$HTTP_POST_VARS` содержит все имена и значения, отправленные через запрос POST, а `$HTTP_GET_VARS` содержит имена и значения, отправленные через запрос GET. Вы можете использовать эти массивы, чтобы сделать свой код более безопасным. Если вы создаете переменную только с помощью массива `$HTTP_POST_VARS`, например, то все данные, посланные методом GET, будут проигнорированы. Это помешает пользователям подменить данные, вбивая их в адресную строку браузера. Конечно, сообразительный пользователь все равно сможет создать форму с поддельными полями, так что, работая с данными, полученными от пользователя, надо быть всегда немного недоверчивым

Создаем многомерный массив

Массивы – очень удобная структура для хранения разных видов данных в памяти компьютера. Обычные массивы очень похожи на списки. Ассоциативные массивы похожи на пары имен и значений. Третий, особый тип массива похож на таблицу с данными. Например, представьте, что вы хотите написать программу, которая помогает пользователям определять расстояние между крупными городами. Для начала вы можете начертить на бумаге таблицу вроде табл. 5.1.:

Табл. 5.1. Расстояние между крупными городами

	Индианаполис	**Нью-Йорк**	**Токио**	**Лондон**
Индианаполис	0	648	6476	4000
Нью-Йорк	648	0	6760	3470
Токио	6476	6760	0	5956
Лондон	4000	3470	5956	0

Работа с такой разновидностью табличных данных достаточно распространена в компьютерных программах. PHP (как и большинство других языков) предоставляет особый тип массива для помощи в работе с таким видом информации. Программа `basicMultiArray`, показанная на рис. 5.8 и 5.9, является примером того, как программа может инкапсулировать таблицу.

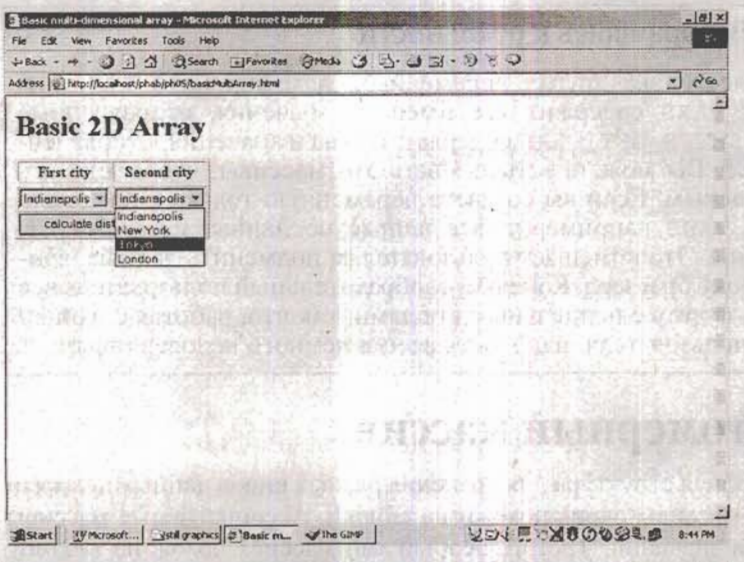


Рис. 5.8. Пользователь может выбрать начальный и конечный города с помощью выпадающих списков

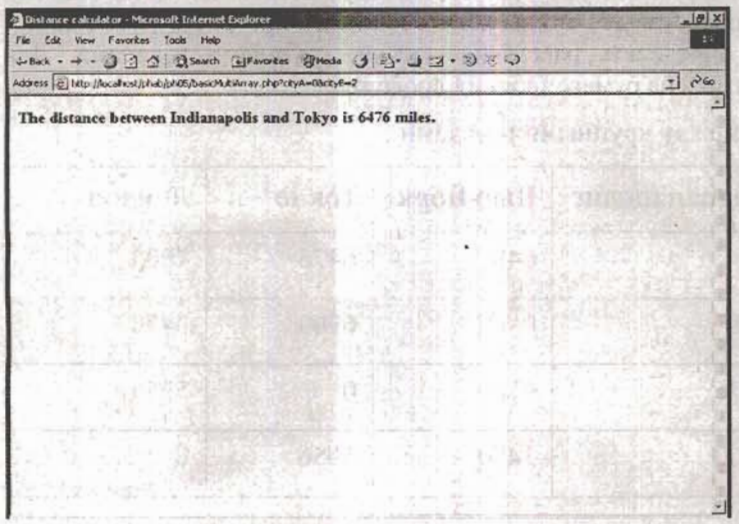


Рис. 5.9. Программа определит расстояние между двумя городами и выведет соответствующее значение

Создаем HTML для простого многомерного массива

Использовать двумерный массив на самом деле довольно легко, если все хорошо спланировать. Сначала я нарисовал свою таблицу с данными на бумаге (на самом деле, у меня в офисе висит доска специально для таких ситуаций). Каждому городу я присвоил численное значение следующим образом.

Индианаполис = 0
Нью-Йорк = 1
Токио = 2
Лондон = 3

Так будет проще следить за городами в последствии.

Вот HTML-код, который создает два выпадающих списка и кнопку «Отправить» в форме.

```
<!doctype html public "-//W3C//DTD HTML 4.0 //EN">
<html>
<head>
      <title>Basic multi-dimensional array</title>
</head>
<body>
<h1>Basic 2D Array</h1>
<form action = basicMultiArray.php>
<table border = 1>
<tr>
  <th>First city</th>
  <th>Second city</th>
<tr>
<!-- Обратите внимание, что значение каждого элемента option -
числовое -->
<tr>
  <td>
    <select name = "cityA">
      <option value = 0>Indianapolis</option>
      <option value = 1>New-York</option>
      <option value = 2>Tokyo</option>
      <option value = 3>London</option>
    </select>
  </td>
  <td>
    <select name = "cityB">
      <option value = 0>Indianapolis</option>
      <option value = 1>New York</option>
      <option value = 2>Tokyo</option>
      <option value = 3>London</option>
    </select>
  </td>
</tr>
```

```
<tr>
  <td colspan = 2>
    <input type = "submit"
           value = "calculate distance">
  </td>
</tr>
</table>
</body>
</html>
```

Когда пользователь отправляет эту форму, она посылает две переменные. В переменной cityA будет храниться свойство *value*, ассоциированное с начальным городом, который выберет пользователь, а в переменной `cityB` – конечный город. Я аккуратно установил свойства value таким образом, что они соответствуют численному индекс каждого города. Если пользователь выберет Нью-Йорк как начальный город, значение $cityA будет равно 1, потому что я решил, что Нью-Йорку будет соответствовать значение 1. Причина для создания численных значений состоит в том, что информация будет храниться в массивах, а обычные массивы имеют численные индексы (в следующем разделе я покажу вам, как сделать то же самое с помощью ассоциативных массивов).

Отвечаем на запрос определения расстояния

Код программы, которая определяет расстояние между городами, на самом деле довольно прост, стоит только разобраться с массивами.

```
<!doctype html public "-//W3C//DTD HTML 4.0 //EN">
<html>
<head>
      <title>Distance calculator</title>
</head>
<body>
<?
$city = array (
  "Indianapolis",
  "New York",
  "Tokyo",
  "London"
  );
$distance = array (
  array (0, 648, 6476, 4000),
  array (648, 0, 6760, 3470),
  array (6476, 6760, 0, 5956),
```

```
  array (4000, 3470, 5956, 0)
  );
$result = $distance[$cityA][$cityB];
print "<h3>The distance between ";
print "$city[$cityA] and $city[$cityB]";
print " is $result miles.</h3>";
?>
</body>
</html>
```

Храним названия городов в массиве $city

В этой программе есть два массива. Массив `$city` – абсолютно обычный массив строковых значений. Он содержит список названий городов. Я аккуратно создал массив таким образом, чтобы численные значения, которые я присвоил городам, соответствовали индексам массива. Помните – индексирование массива обычно начинаются с нуля, так что Индианаполис – это ноль, Нью-Йорк – это один и т. д.

Пользователь не знает, что Indianapolis – это 0, поэтому я использовал массив `$city`, чтобы назначить названия городам. Если пользователь выберет город ноль (Индианаполис) в поле `$cityA`, я могу обратиться к названию города как к `$city[$cityA]`, потому что `$cityA` будет равен 0, а `$city[0]` – это «Indianapolis».

Храним расстояния в массиве $distance

Расстояния нельзя расположить списком, потому что для определения расстояния требуется две переменных. Чтобы определить расстояние, вам надо знать, из какого города вы выходите и в какой направляетесь. Эти два значения соответствуют строкам и столбцам начальной таблицы. Взгляните еще раз на код, который создает массив `$distance`.

```
$distance = array (
  array (0, 648, 6476, 4000),
  array (648, 0, 6760, 3470),
  array (6476, 6760, 0, 5956),
  array (4000, 3470, 5956, 0)
  );
```

Массив `$distance` на самом деле является массивом, содержащим другие массивы! Каждый из внутренних массивов соответствует расстоянию от определенного начального города. Например, поскольку Индианаполис – это 0, первый (нулевой?) внутренний массив содержит расстояния между Индианаполисом и другими городами. Для лучшего понимания представьте себе каждый внутренний массив как

строку таблицы, а таблицу как массив строк. Хотя на первый взгляд построить двумерный массив сложно, на самом деле это куда естественнее чем вы, возможно, думаете. Если вы сравните табл. 5.1 и код, который создает двумерный массив, вы увидите, что все числа на тех же местах.

Вовсе не обязательно останавливаться на двух измерениях. Можно создать массивы с тремя, четырьмя или с любым другим количеством измерений. Однако становится трудно представить, как работают данные в таких сложных массивах. Как правило, одно- и двумерные массивы являются самыми сложными из массивов, которые вам понадобятся. Для более сложных типов данных вам потребуется обратиться к инструментам работы с файлами и реляционным структурам данных, которых мы еще коснемся в этой книге.

Получение данных из массива $distance

Как только информация попадает в двумерный массив, ее достаточно легко оттуда получить. Чтобы найти информацию в таблице, вам надо знать строку и столбец. Двумерный массив требует двух индексов – один для строки и один для столбца. Чтобы найти расстояние между Токио (город номер 2) и Нью-Йорком (город номер 1), вам надо просто обратиться к `$distance[2][1]`. Код для демонстрационной программы получает значения индексов из следующей формы.

```
$result = $distance[$cityA][$cityB];
```

Это значение сохраняется в переменной `$result`, а затем отсылается пользователю.

Создаем двухмерный ассоциативный массив

Вы можете также создавать двухмерные ассоциативные массивы. Требуется немного больше работы, чтобы создать такой массив, но она того стоит, потому что отношения имя-значение устраняет необходимость следить за числовыми идентификаторами каждого элемента. Другая версия программы `multiArray` показывает, как использовать ассоциативные массивы для создания такой же программы подсчета расстояний между городами.

Поскольку для пользователя эта программа выглядит так же, как и `basicMultiArray`*, я не показываю рисунков, иллюстрирующих ее работу. Все интересные особенности этой программы находятся в ее коде.*

Создаем HTML для ассоциативного массива

HTML-страница для «ассоциативной» версии программы очень похожа на программу для обычного массива, однако имеет одно существенное отличие. Посмотрите, удастся ли вам найти это отличие в коде программы.

```
<!doctype html public "-//W3C//DTD HTML 4.0 //EN">
<html>
<head>
       <title>2D Array</title>
</head>
<body>
<h1>2D Array</h1>

<form action = multiArray.php>
<table border = 1>
<tr>
  <th>First city</th>
  <th>Second city</th>
<tr>
<!-- обратите внимание, что значение каждого объекта option
строковое  -->
<tr>
  <td>
    <select name = "cityA">
      <option value = "Indianapolis">Indianapolis</option>
      <option value = "New York">New York</option>
      <option value = "Tokyo">Tokyo</option>
      <option value = "London">London</option>
     </select>
   </td>
  <td>
    <select name = "cityB">
      <option value = "Indianapolis">Indianapolis</option>
      <option value = "New York">New York</option>
      <option value = "Tokyo">Tokyo</option>
      <option value = "London">London</option>
     </select>
  </td>
</tr>
<tr>
  <td colspan = 2>
```

```
    <input type = "submit"
           value = "calculate distance">
  </td>
</tr>
</table>
</body>
</html>
```

Единственное отличие между этой HTML-страницей и предыдущей заключается в свойстве `value` объектов `select`. В этом случае массив расстояний будет ассоциативным, поэтому числовые индексы нам не понадобятся. Поскольку индексы могут быть текстовыми, я посылаю настоящие имена городов в качестве значений `$cityA` и `$cityB`.

Отвечаем на запрос

Код для ассоциативного ответа интересен, потому что он проводит большой объем работ, чтобы построить хороший ассоциативный массив. Как только массив создан, работать с ним очень легко.

```
<!doctype html public "-//W3C//DTD HTML 4.0 //EN">
<html>
<head>
<title>Distance Calculator</title>
</head>
<body>
<h1>Distance Calculator</h1>
<?
//создаем массивы
$indy = array (
  "Indianapolis" => 0,
  "New York" => 648,
  "Tokyo" => 6476,
  "London" => 4000
  );
$ny = array (
  "Indianapolis" =>648,
  "New York" => 0,
  "Tokyo" => 6760,
  "London" => 3470
  );
$tokyo = array (
```

```
  "Indianapolis" => 6476,
  "New York" => 6760,
  "Tokyo" => 0,
  "London" => 5956
  );
$london = array (
  "Indianapolis" => 4000,
  "New York" => 3470,
  "Tokyo" => 5956,
  "London" => 0
  );
//устанавливаем главный массив

$distance = array (
  "Indianapolis" => $indy,
  "New York" => $ny,
  "Tokyo" => $tokyo,
  "London" => $london
  );
$result = $distance[$cityA][$cityB];
print "<h3>The distance between $cityA and $cityB is $result
miles.</h3>";
?>
</body>
</html>
```

Построение двумерного ассоциативного массива

Основной принцип построения двухмерного массива одинаков и для обычных, и для ассоциативных массивов. По существу, вы создаете каждую строку как массив, а затем строите массив из созданных массивов. В обычных массивах индексы создавались автоматически. Создание ассоциативного массива немного сложнее, потому что вам придется указывать ключ для каждого значения. В качестве примера посмотрите на код, который я использовал для создания массива `$indy`.

```
$indy = array (
  "Indianapolis" => 0,
  "New York" => 648,
  "Tokyo" => 6476,
  "London" => 4000
  );
```

Внутри массива я использовал названия городов в качестве индексов. Значение каждого индекса указывает на расстояние от текущего города (Индианаполис) до определенного места назначения. Расстояние между Индианаполисом и Индианаполисом равно нулю, между Индианаполисом и Нью-Йорком – 648, и т. д.

Я создал ассоциативный массив для каждого города, а затем собрал эти ассоциативные массивы в нечто вроде гигантского ассоциативного массива.

```
$distance = array (
  "Indianapolis" => $indy,
  "New York" => $ny,
  "Tokyo" => $tokyo,
  "London" => $london
  );
```

Этот новый массив тоже является ассоциативным, но каждый из его индексов указывает на массив расстояний.

Получаем данные из двумерного ассоциативного массива

Как только двухмерный массив создан, он становится чрезвычайно прост в использовании. Названия городов использованы в качестве индексов, так что нет необходимости в отдельном массиве для хранения названий городов. Данные можно вывести двумя строчками кода.

```
$result = $distance[$cityA][$cityB];
print "<h3>The distance between $cityA and $cityB is $result
miles.</h3>";
```

Если вы хотите, вы можете совмещать ассоциативные и обычные массивы. Можно создать несколько ассоциативных массивов и собрать их в один обычный массив, или наоборот. Возможности работы с массивами в PHP допускают феноменальный уровень контроля над вашими структурами данных.

Управляем строковыми значениями

Программа «Поиск слов», описанная в начале этой главы, использует массивы для части своей функциональности, однако одних массивов недостаточно, чтобы программа работала корректно. Эта программа использует несколько особых функций управления строками для обширной работы с текстовыми значениями. В PHP есть огромное количество строковых функций, которые дают вам невероятные возможности для работы со строками.

Демонстрация работы со строками на примере переводчика на забавную латынь

Для лучшего понимания функций работы со строками обратите внимание на программу, показанную на рис. 5.10 и 5.11. Эта программа позволяет пользователю ввести фразу в текстовое поле и конвертирует фразу в имитацию латыни.

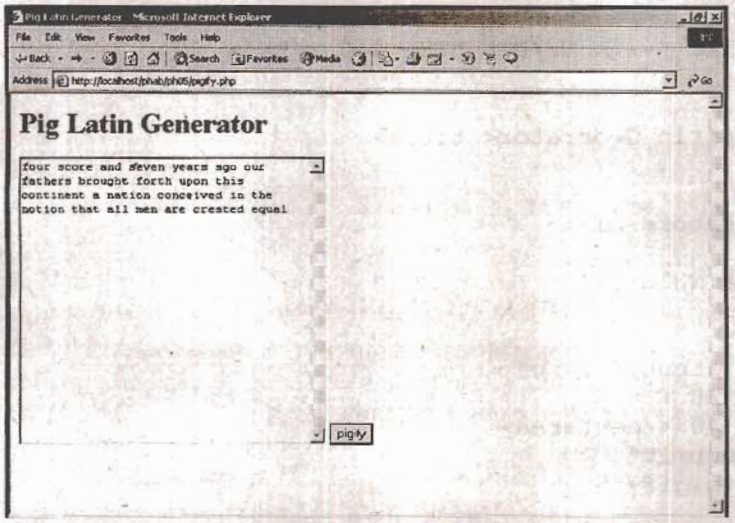


Рис. 5.10. Программа `pigify` позволяет пользователю ввести какой-нибудь текст в текстовую область

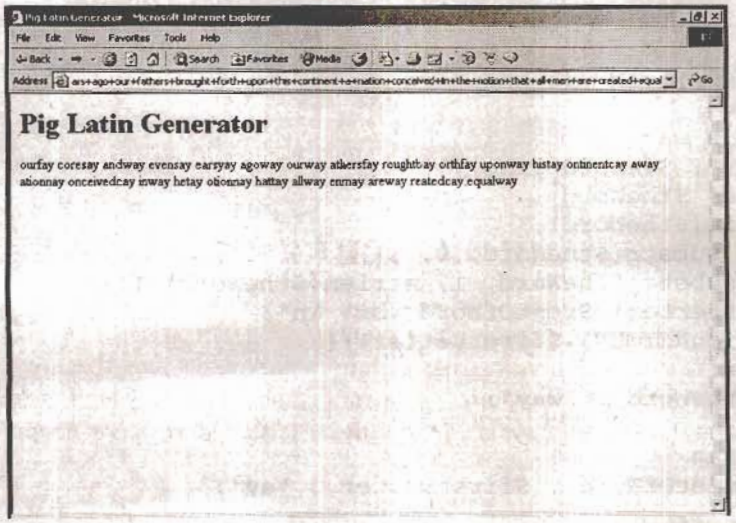


Рис. 5.11. Программа превращает бессмертную прозу в невероятную глупость

Забавная латынь – это глупая детская игра. По существу, вы берете первую букву каждого слова, переставляете ее в конец слова и добавляете "ay". Если слово начинается с гласной, вы просто добавляете в конец слова "way".

Программа `pigify` использует несколько строковых функций для работы с текстом.

```
<!doctype html public "-//W3C//DTD HTML 4.0 //EN">
<html>
<head>
       <title>Pig Latin Generator</title>
</head>
<body>
<h1>Pig Latin Generator</h1>
<?
if ($inputString == NULL){
  print <<<HERE
  <form>
  <textarea name = "inputString"
            rows = 20
            cols = 40></textarea>
   <input type = "submit"
          value = "pigify">
   </form>
HERE;
} else {
  //передано значение, так что займемся им

  //разобьем фразу в массив
  $words = split(" ", $inputString);
  foreach ($words as $theWord){
    $theWord = rtrim($theWord);
    $firstLetter = substr($theWord, 0, 1);
    $restOfWord = substr($theWord, 1, strlen($theWord)-1);
    //print "$firstLetter) $restOfWord <br> \n";
    if (strstr("aeiouAEIOU", $firstLetter)){
      //это гласная
      $newWord = $theWord . "way";
    } else {
      //это согласная
      $newWord = $restOfWord . $firstLetter . "ay";
    } // завершение if
```

```
    $newPhrase = $newPhrase . $newWord . " ";
  } // завершение цикла foreach
  print $newPhrase;
} // завершение if
?>
</body>
</html>
```

Построение формы

Эта программа использует PHP-страницу и для создания формы ввода, и для вывода обработанных данных. Вначале она проверяет наличие переменной `$inputString`. Когда пользователь первый раз попадет на страницу, эта переменная еще не будет создана. В этом случае программа создаст соответствующую HTML-страницу и дождется ввода данных пользователем. После того как пользователь нажмет на кнопку `Submit` (Отправить), программа запустится снова, но на этот раз переменная `$inputString` будет иметь какое-то значение. Оставшаяся часть программы использует функции работы со строками для создания версии введенной строки на забавной латыни.

Используем функцию Split для помещения строки в массив

Одна из первых задач – разбить строку, введенную пользователем, на отдельные слова. Для этой цели PHP предлагает несколько интересных функций. Функция `split()` получает строку и разбивает ее в массив, основываясь на каком-либо разделителе. Функция `split()` имеет два аргумента. Первый аргумент – это разделитель, второй – строка, которую надо разбить. Я хочу, чтобы каждое слово было отдельным элементом массива, поэтому я использую пробел (" ") в качестве разделителя. Приведенная ниже строка кода берет переменную `$inputString` и разбивает ее в массив `$words`. Каждое слово будет новым элементом массива.

```
$words = split(" ", $inputString);
```

Создав массив `$words`, я прошел по нему с помощью цикла `foreach`, временно сохраняя каждое слово в переменную `$theWord`.

Обрезаем строки с помощью функции rtrim()

Иногда после разделения строки в массив в каждом элементе массива остаются символы-разделители на конце. В программе забавной латыни в конце каждого слова будет пробел, что может породить некоторые проблемы позже. PHP предоставляет функцию `rtrim()`, которая автоматически удаляет пробелы, символы

табуляции, переводы каретки и другие аналогичные символы с конца строки. Я использовал функцию `rtrim()`, чтобы убрать все пробелы, добавленные после операции `split()`, и записал результат обратно в `$theWord`.

```
$theWord = rtrim($theWord);
```

Помимо функции `rtrim()`, *в PHP есть еще* `ltrim()`, *которая отрезает пробелы с начала строки, и* `trim()`, *которая очищает и начало и конец строки одновременно. Помимо этого существует еще вариация «обрезающих» команд, которая позволяет указать, какие именно символы требуется удалить.*

Находим подстроки с помощью функции substr()

Поведение алгоритма зависит от первого символа каждого слова. Помимо этого, нам должно быть известно все слово без первого символа. Функция `substr()` полезна, когда требуется получить часть строки. Она имеет три параметра. Первый аргумент – это строка, часть которой вы хотите получить. Второй аргумент – это порядковый номер символа, с которого начинается подстрока (как правило, начинается с нуля), и третий аргумент – количество символов, которые вы хотите получить.

Я получил первую букву слова вот такой строкой кода.

```
$firstLetter = substr($theWord, 0, 1);
```

Этот код получает одну букву из переменной `$theWord`, начиная с первого символа (позиция 0). Затем я сохранил это значение в переменной `$firstLetter`.

Не намного сложней получить и остальное слово.

```
$restOfWord = substr($theWord, 1, strlen($theWord) - 1);
```

Мне снова надо получить значение из переменной `$theWord`. На этот раз я начну с символа 1 (который простой человек считал бы вторым). Я не знаю, сколько символов я получу, но я могу подсчитать их количество. Мне надо взять на один символ меньше, чем общее количество символов в строке[1]. Функция `strlen()` превосходно подходит для этой операции, потому что она возвращает количество символов в любой строке. Я могу подсчитать необходимое мне количество символов таким образом: `$strlen($theWord)-1`. Это новое укороченное слово я сохранил в переменной `$restOfWord`.

---

[1] Есть и другой способ получить нужную часть строки: для этого достаточно задать для функции `substr()` второй аргумент `-1`. Когда второй аргумент является отрицательным числом, функция `substr()` возвращает подстроку, начиная с символа, указанного в первом аргументе, и заканчивая символом, порядковый номер которого равен длине строки минус второй аргумент. – *Примеч. пер.*

Используем strstr() для поиска одной строки внутри другой

Следующая задача – определить, является ли первая буква гласной. К этой проблеме можно подойти с разных сторон, но, наверное, поисковая функция является простейшим вариантом. Я создал строку со всеми гласными ("aeiouAEIOU"), а затем выполнил поиск значения переменной `$firstLetter` в этой строке. Функция `strstr()` идеальна для выполнения этой задачи. Она принимает два параметра. Первый параметр – это строка, в которой вы осуществляете поиск (в документации, которую можно найти в Интернете, ее называют haystack – "стог сена"). Второй параметр – это строка, которую вы ищете (она называется needle – "иголка"). Чтобы найти значение переменной `$firstLetter` в строковой константе "aeiouAEIOU", я использовал следующий код.

```
$if (strstr("aeiouAEIOU", $firstLetter)) {
```

Функция `strstr()` возвращает значение `FALSE`, если иглу в стоге сена найти не удалось. Если игла была найдена, она возвращает позицию иглы в параметре "стог сена". В нашем случае, все что меня волнует – нашлось ли значение переменной `$firstLetter` в списке букв. Если нашлось, то оно является гласным, а значит, изменять слово мы будем по-другому.

Используем оператор конкатенации

В большинстве случаев в PHP вы можете использовать интерполяцию строк для совмещения строковых значений. Однако иногда вам все же придется пользоваться формальной операцией для совмещения строк. Процесс объединения двух строк называется *конкатенация* (обожаю, когда у простых идей такие сложные названия). Оператор конкатенации в PHP – это точка (.). В забавной латыни, если слово начинается с гласной, оно просто должно закончиться строкой "way". Я использовал конкатенацию строк для решения этой задачи.

```
$newWord = $theWord . "way";
```

Когда слово начинается с согласной, формула для создания нового слова немного сложней, но все равно выполняется с помощью конкатенации строк.

```
$newWord = $restOfWord . $firstLetter . "ay";
```

Недавнее тестирование показало, что построение строк методом конкатенации работает гораздо быстрее, чем интерполяция. Если для вас скорость имеет большое значение, вам следует использовать конкатенацию строк вместо интерполяции.

Завершаем программу забавной латыни

Создав новое слово, я добавляю его и хвостовой пробел к переменной `$newPhrase`. Когда цикл `foreach` перестанет выполняться, в переменной `$newPhrase` будет содержаться перевод оригинальной фразы на забавную латынь.

Переводим ASCII-коды в символы

Хотя для программы забавная латынь это не требуется, программа поиска слов потребует возможности создания случайно выбранного символа. Я добился этого генерацией случайного ASCII-кода (ASCII – это код, используемый для хранения символов в виде двоичных чисел в памяти компьютера) и переводом этого кода в соответствующий символ. В такой ситуации полезна функция `ord()`. Заглавные буквы представлены в ASCII числами между 65 и 90. Чтобы получить случайную заглавную букву, я воспользовался следующим кодом.

```
$theNumber = random(65, 90);
$theLetter = ord($theNumber);
```

Возвращаемся к программе «Поиск слов»

Теперь вы умеете все, что вам потребуется для создания программы-головоломки, которая дебютировала в начале этой главы. Программа хранится в трех файлах. Сначала пользователь вводит список слов и другую информацию о головоломке в HTML-страницу. Эта страница вызывает основную программу `wordFind.php`, которая анализирует список слов, создает головоломку и выводит ее. Наконец, у пользователя есть возможность получить решение головоломки, которое создается простой PHP-программой.

Получаем данные головоломки от пользователя

Страница `wordFind.html` – это точка входа пользователя в систему поиска слов. Эта страница является обычной HTML-формой с несколькими базовыми элементами.

```
<!doctype html public "-//W3C//DTD HTML 4.0 //EN">
<html>
<head>
      <title>Word Puzzle Maker</title>
</head>
<body>
<center>
<h1>Word Puzzle Maker</h1>
```

```
<form action = "wordFind.php"
      method = "post">
<h3>Puzzle Name</h3>
<input type = "text"
       name = "name"
       value = "My Word Find">
height: <input type = "text"
             name = "height"
             value = "10"
             size = "5">
width: <input type = "text"
             name = "width"
             value = "10"
             size = "5">
<br><br>
<h3>Word List</h3>
<textarea rows=10 cols=60 name = "wordList"></textarea>
<br><br>
Please enter one word per row, no spaces
<br>
<input type="submit" value="make puzzle">
</form>
</center>
</body>
</html>
```

Свойство формы `action` указывает на программу `wordFind.php`, которая является основной программой в системе. Обратите внимание, что я использовал метод `post` для отправки данных в программу, потому что я собираюсь отправлять в программу большие строковые значения, а метод `get` позволяет отправлять серверу только малое количество данных.

Форма состоит из простых текстовых полей для названия, высоты и ширины головоломки. Эти данные будут использованы для построения головоломки. Текстовая область `wordlist` предназначена для ввода списка слов, которые будут использованы для создания головоломки.

Настраиваем страницу вывода

Основная масса работы системы `wordFind` выполняется в странице `wordFind.php`. Эта станица содержит небольшое количество HTML-кода для подготовки к выводу результатов, но большая часть файла является PHP-кодом.

```
<html>
<head>
<title>
Word Find
</title>
</head>
<body>
<?
// Поиск слов
// автор - Энди Харрис (Andy Harris), 2003
// для книги «Программирование на PHP/MySQL для новичков»
// Генерирует головоломку для поиска слов на основе списка слов,
// введеных пользователем. Пользователь также может задать
размер
// головоломки и, если захочет, напечатать ключ ответа
```

Обратите внимание на комментарии в начале кода. Поскольку код этой программы немного сложнее, чем большинство программ, которые вы встречали ранее в этой книге, я решил закомментировать его более тщательно. Вообще всегда стоит добавлять к своим программам комментарии, чтобы проще было определять, что они делают. Вы будете удивлены, насколько сложно разобраться в своем коде, отвлекшись от него на пару дней. Хорошие комментарии значительно упрощают работу с вашим кодом и позволяют другим легко исправлять и улучшать ваши программы в будущем. Здесь мои комментарии обрисовывают общий план программы.

Работаем с пустым набором данных

В целях тестирования я написал PHP-страницу поиска слов до того, как озаботился созданием HTML. Для этого я просто добавил значения по умолчанию для списка слов и других основных переменных, которые определяют формат поля (высота, ширина и название). Я полагаю, что в финальной версии программы PHP-код вряд ли когда-нибудь будет вызван без HTML-страницы, но я оставил значения по умолчанию просто для того, чтобы вы увидели, как они работают.

```
if ($wordList == NULL){
  //создаем
  $word = array(
    "ANDY",
    "HEATHER",
    "LIZ",
    "MATT",
    "JACOB"
```

```
  );
$boardData = array(
  width => 10,
  height => 10,
  name => "Generic Puzzle"
);
```

Этот код создает два массива, которые определяют целую программу. Массив `$word` содержит список слов, которые надо спрятать в головоломке, а `$boardData` – это ассоциативный массив, хранящий важную информацию о том, как будет создаваться поле.

Конечно, это не те значения, которые я собираюсь использовать, потому что в основном эта программа будет вызвана HTML-формой, которая и передаст значения. Следующая часть кода заполнит эти переменные, если программа вызвана из соответствующей формы.

Создаем основную логику программы

Основная логика программы начинается с получения списка слов и параметров головоломки из пользовательской формы. Затем она пытается превратить этот список в массив. Такой тип исследования текста иногда называется *анализом* (*parsing*).

После этого программа пытается циклически построить поле. Когда программа успешно создала поле, она создает решение головоломки, затем добавляет случайные буквы с помощью функции `addFoils()`. Наконец, программа выводит законченную головоломку.

```
//получаем данные о головоломке из HTML-формы
$boardData = array(
  width => $width,
  height => $height,
  name => $name
);

//пытаемся получить список слов из полученных данных
if (parseList() == TRUE){
  $legalBoard = FALSE;
  //продолжаем пытаться построить поле до удачного результата
  while ($legalBoard == FALSE){
    clearBoard();
    $legalBoard = fillBoard();
  } // завершение цикла while
  //создаем решение
```

```
    $key = $board;
    $keyPuzzle = makeBoard($key);

    //создаем окончательную головоломку
    addFoils();
    $puzzle = makeBoard($board);

    //выводит страницу результатов
    printPuzzle();

  } // завершение if для проанализированного списка
} // завершение if для проверки существования списка слов
```

Посмотрев на этот код, вы должны понять общий ход программы, даже если вы не понимаете, как именно все происходит. Основная часть хорошо продуманной программы должна позволять вам понять основные действия, выполняемые в ней. Большинство мелких деталей доверены функциям. Оставшаяся часть этой главы в основном посвящена объяснению того, как эти функции работают. Постарайтесь убедиться, что вы разбираетесь в основных частях программы; если вы преуспеете в этом, то поймете, как работает все остальное.

Анализируем список слов

На первых порах важной задачей является анализ списка слов, полученных от пользователя. Список слов приходит в виде одной длинной строки, разделенной символами переноса строки (\n). Функция `parselist()` превращает эту строку в массив слов. Она делает еще некоторые важные вещи, включая перевод каждого слова в верхний регистр, проверку того, поместятся ли слова в указанный размер головоломки, и удаление ненужных возвратов каретки.

```
function parseList(){
  //получает список слов, создает из него массив слов
  //или возвращает false, если это невозможно

  global $word, $wordList, $boardData;

  $itWorked = TRUE;

  //переведем весь список слов в верхний регистр
  $wordList = strtoupper($wordList);

  //разделим список слов на массив
  $word = split("\n", $wordList);

  foreach ($word as $currentWord){
    //уберем «хвостовые» символы новой строки
    $currentWord = rtrim($currentWord);
```

```
      //остановимся, если какое-то из слов слишком длинное для
головоломки
    if ((strLen($currentWord) > $boardData["width"]) &&
        (strLen($currentWord) > $boardData["height"])){
      print "$currentWord is too long for puzzle";
      $itWorked = FALSE;
    } // завершение if
  } // завершение цикла foreach
  return $itWorked;
} // завершение parseList
```

Первое, что я сделал, – использовал функцию `strtoupper()`, чтобы перевести весь список слов в верхний регистр. В головоломках поиска слов обычно используют заглавные буквы, так что я решил перевести все в этот формат.

Длинная строка символов с переносами строк вряд ли является удобным форматом для наших целей, поэтому я превратил длинную строку в массив `$word`. Функция `split()` отлично подходит для этой задачи. Обратите внимание, что я разделял по строке `"\n"`. Это символ новой строки, т. е. я превращаю каждую строку из текстовой области в элемент нового массива `$word`.

Следующая задача – исследовать каждое слово в массиве с помощью цикла `foreach`. Когда я тестировал эту часть программы, стало ясно, что кое-где еще оставались символы новой строки, так что я использовал функцию `rtrim()`, чтобы отрезать любые ненужные пустые символы.

Если пользователь введет слово, которое больше ширины или высоты поля головоломки, будет невозможно создать головоломку, поэтому я сравниваю длину каждого слова с высотой и шириной поля. Обратите внимание, что если слово слишком длинное, я просто задаю переменной `$itWorked` значение `FALSE`. Ранее в этой функции я присвоил `$itWorked` значение `TRUE`. К тому моменту, когда функция закончит свою работу, значение `$itWoked` останется `TRUE` только если все слова достаточно малы, чтобы поместиться в головоломку. Если хотя бы одно из слов слишком большое, то значение `$itWorked` будет `FALSE` и программа не будет выполняться дальше.

Очищаем поля

Программа поиска слов использует грубую, но эффективную технику для создания корректных игровых полей. Она создает поля случайным образом, пока не найдет корректное. Хотя это может показаться довольно расточительным подходом, такую программу составить гораздо легче, чем многие другие, более сложные способы, и она выдает довольно хорошие результаты для таких простых задач, как головоломка поиска слов.

Возвращаясь к реальности

Хотя эта программа использует метод "грубой силы" для поиска подходящего решения, вы увидите несколько способов оптимизации кода для увеличения шансов достичь подходящего решения. Один пример вы уже видели: программа прекращает работу, если одно из слов слишком велико для головоломки. Это предотвращает долгие попытки программы втиснуть слово в головоломку, когда это никак не возможно. Проходя далее через код программы, вы увидите еще несколько мест, в которых я немного поработал, чтобы направить алгоритм к правильному решению и оградить от ловушек. Благодаря этим усилиям вы увидите, что программа на самом деле достаточно хорошо создает головоломки поиска слов, если, конечно, слов не слишком много или поле не слишком маленькое

В процессе работы программы игровое поле будет несколько раз создаваться заново, поэтому мне нужна была функция, которая легко создавала бы игровое поле или очищала его. Игровое поле хранится в двумерном массиве `$board`. Когда поле "пустое", в каждой ячейке находится символ точки (.). Я использую такие обозначения, потому что они, с одной стороны, позволяют хранить некоторое значение в каждой ячейке, а с другой стороны, позволяют увидеть пустые ячейки. Функция `clearBoard()` задает или обнуляет массив `$board`, записывая точку в каждую клетку.

```
function clearBoard(){
  //создадим поле с точкой (.) в каждой ячейке
  global $board, $boardData;
  for ($row = 0; $row < $boardData["height"]; $row++){
    for ($col = 0; $col < $boardData["width"]; $col++){
      $board[$row][$col] = ".";
    } // завершения цикла for для col
  } // завершение цикла for для row
} // завершение clearBoard
```

Этот код – классический пример вложенных циклов `for`, часто используемых с двумерными массивами. Обратите внимание, что я использовал цикл `for` вместо цикла `foreach`, потому что мне нужны были индексы циклов. Внешний цикл `for` проходит по строкам. Внутри каждой строки еще один цикл проходит по каждому столбцу. Я присвоил значение "." ячейке, расположенной по индексам `$row` и `$col` в массиве `$board`. В конце концов, в каждой ячейке массива будет содержаться значение ".".

Я определил размер циклов `for`*, обращаясь к ассоциативному массиву* `$boardData`*. Есть несколько разных способов, которыми можно было бы этого добиться, но я выбрал ассоциативный массив по нескольким причинам. Самая важная из них – ясность. Так мне проще видеть, что я работаю с высотой и шириной, относящимися к информации о поле. Еще одно преимущество использования ассоциативного массива в этом случае – это удобство. Поскольку высота, ширина и название поля хранятся в массиве* `$boardData`*, я могу сделать глобальную ссылку на переменную* `$boardData` *и все ее значения будут доступны. Т.е. я получил три переменных по цене одной.*

Заполняем поля

Конечно, смысл очищения поля в том, чтобы заполнить его словами из списка слов. Это произойдет в два этапа. Функция `fillBoard()` контролирует весь процесс заполнения целого поля, но за внесение каждого слова в поле отвечает функция `addWord()` (которую вы увидите следующей).

Поле создано удачно только в том случае, если каждое слово добавлено правильно. Каждое слово добавляется, только если каждая из его букв добавлена без проблем. Программа повторно вызывает функцию `fillBoard()` так часто, как это необходимо, чтобы достичь удачного решения. Каждый раз, когда запускается функция `fillBoard()`, она может вызвать функцию `addWord()` столько раз, сколько необходимо, пока все слова не будут добавлены. Функция `addWord()`, в свою очередь, следит за тем, возможно ли добавить каждый символ слова на поле.

Общий план функции `fillBoard()` состоит в том, чтобы создать случайное направление для каждого слова, а затем попросить функцию `addWord()` расположить на поле указанное слово в указанном направлении. Циклическая структура функции `fillBoard()` уникальна, потому что цикл может закончиться двумя разными способами. Если любое из слов не может быть расположено указанным способом, создание головоломки немедленно остановится и функция вернет значение `FALSE`. Однако если весь список слов успешно расположен на игровом поле, функция также выйдет из цикла, но вернет значение `TRUE`. Есть несколько способов достичь подобного эффекта, но я чаще всего предпочитаю использовать для этой цели особую булеву переменную. Логические переменные – это переменные, предназначенные для хранения только значений true (истина) или false (ложь). Конечно, PHP довольно беспечен в отношении типов переменных, но вы всегда можете сделать переменную булевой, назначая ей только значения `TRUE` или `FALSE`. В функции `fillBoard()` обратите внимание на то, как используется переменная `$keepGoing`. Она инициализирована со значением `TRUE`, и основной цикл функции продолжает выполняться, пока это условие верно. Однако два условия могут заставить цикл перестать выполняться (функции `addWord()` не удалось корректно разместить слово, или список слов закончился), и оба условия заставят

переменную $keepGoing принять значение FALSE. Когда это произойдет, цикл остановится и функция вскоре завершит работу.

```
function fillBoard(){
  //заполнить поле, вызывая функцию addWord() для каждого слова
списка,
  //или вернуть FALSE в случае неудачи
  global $word;
  $direction = array("N", "S", "E", "W");
  $itWorked = TRUE;
  $counter = 0;
  $keepGoing = TRUE;
  while($keepGoing){
    $dir = rand(0, 3);
    $result = addWord($word[$counter], $direction[$dir]);
    if ($result == FALSE){
      //print "failed to place $word[$counter]";
      $keepGoing = FALSE;
      $itWorked = FALSE;
    } // завершение if
    $counter++;
    if ($counter >= count($word)){
      $keepGoing = FALSE;
    } // завершение if
  } // завершение цикла while
  return $itWorked;
} // завершение fillBoard
```

Функция начинается с определения массива для направлений. Я решил поддерживать расположение слов только по четырем основным направлениям, хотя было бы довольно легко добавить и диагонали (кстати, это похоже на первоклассное упражнение по окончанию главы!). Массив $direction содержит инициалы четырех поддерживаемых мной направлений. Булева переменная $itWorked будет сообщать, удалось ли успешно заполнить поле. Она инициализирована со значением TRUE. Если функции addWord() не удастся разместить слово, значение $itWorked будет изменено на FALSE.

Переменная $counter будет использована для определения слова, которое я в данный момент пытаюсь расположить. Я увеличиваю значение $counter при каждом выполнении цикла. Если значение $counter больше, чем размер массива $word, значит, функция успешно добавила все слова и может победоносно закончиться.

Чтобы выбрать направление, я просто создал случайное значение между 0 и 3 и обратился к соответственному значению массива `$direction`.

Последняя строка функции возвращает значение переменной `$itWorked`. Основная программа вызывает функцию `fillBoard()`, пока та не достигнет успеха. Успех или неудача передается в основную программу возвращением переменной `$itWorked`.

Добавляем слова

Функция `fillBoard()` управляет глобальным процессом добавления списка слов на игровое поле, однако процесс добавления каждого слова на поле выполняется функцией `addWord()`. Эта функция требует двух параметров – слово для добавления и направление.

Функция очищает слово и выполняет немного другую работу в зависимости от направления, в котором слово будет расположено. Она помещает каждую букву слова в соответствующую ячейку, следя, чтобы буквы не выходили за границы игрового поля. Помимо этого, она проверяет, не содержится ли уже в ячейке буква другого слова (если, конечно, эта буква не совпадает с той, которую функция пытается вставить). Функция может выглядеть длинной и сложной на первый взгляд, но, присмотревшись поближе, вы увидите, что она крайне однообразна.

```
function addWord($theWord, $dir){
   //попытаемся добавить слово на поле или вернем false, если
попытка не удалась
  global $board, $boardData;
  //удалим замыкающие символы
  $theWord = rtrim($theWord);
  $itWorked = TRUE;
  switch ($dir){
      case "E":
        //колонки от 0 до ширины поля - ширина слова
      //строки от 0 до высоты поля
            $newCol = rand(0, $boardData["width"] - 1 -
strlen($theWord));
        $newRow = rand(0, $boardData["height"]-1);
      for ($i = 0; $i < strlen($theWord); $i++){
        //новый символ, та же строка, начальная колонка + $i
        $boardLetter = $board[$newRow][$newCol + $i];
        $wordLetter = substr($theWord, $i, 1);
        //проверяем, свободно ли значение текущей ячейки поля
        if (($boardLetter == $wordLetter) ||
```

```
            ($boardLetter == ".")){
          $board[$newRow][$newCol + $i] = $wordLetter;
        } else {
          $itWorked = FALSE;
        } // завершение if
      } // завершения цикла for
      break;

      case "W":
        //колонка от ширины слова до ширины поля
        //строка от 0 до высоты поля
      $newCol = rand(strlen($theWord), $boardData["width"] -1);
      $newRow = rand(0, $boardData["height"]-1);
      //print "west:\tRow: $newRow\tCol: $newCol<br>\n";
      for ($i = 0; $i < strlen($theWord); $i++){
        //проверим, можно ли продвинуться
        $boardLetter = $board[$newRow][$newCol - $i];
        $wordLetter = substr($theWord, $i, 1);
        if (($boardLetter == wordLetter) ||
            ($boardLetter == ".")){
          $board[$newRow][$newCol - $i] = $wordLetter;
        } else {
          $itWorked = FALSE;
        } // завершение if
      } // завершение цикла for
      break;

    case "S":
        //колонка от 0 до ширины поля
        //строка от 0 до высоты поля - длина слова
      $newCol = rand(0, $boardData["width"] -1);
                $newRow = rand(0, $boardData["height"]-1 -
strlen($theWord));
      //print "south:\tRow: $newRow\tCol: $newCol<br>\n";
      for ($i = 0; $i < strlen($theWord); $i++){
        //проверим, можно ли продвинуться
        $boardLetter = $board[$newRow + $i][$newCol];
        $wordLetter = substr($theWord, $i, 1);
        if (($boardLetter == $wordLetter) ||
            ($boardLetter == ".")){
          $board[$newRow + $i][$newCol] = $wordLetter;
          } else {
```

```
            $itWorked = FALSE;
               } // завершение if
        } // завершение цикла for
        break;
    case "N":
        //колонка от 0 до ширины поля
        //строка от длины слова до высоты поля
        $newCol = rand(0, $boardData["width"] -1);
        $newRow = rand(strlen($theWord), $boardData["height"]-1);
        for ($i = 0; $i < strlen($theWord); $i++){
          //проверим, можно ли продвинуться
          $boardLetter = $board[$newRow - $i][$newCol];
          $wordLetter = substr($theWord, $i, 1);
          if (($boardLetter == $wordLetter) ||
              ($boardLetter == ".")){
            $board[$newRow - $i][$newCol] = $wordLetter;
              } else {
            $itWorked = FALSE;
              } // завершение if
        } // завершение цикла for
        break;
  } // завершение switch
  return $itWorked;
} // азвершение addWord
```

Главная часть функции `addWord()` – это структура `switch`, основанная на направлении слова. Код внутри каждой ветви switch в общих чертах очень похож.

Внимательно изучаем восточный код

В Западных языках обычно пишут слева направо, так что код для "E"[1], что означает "пишем в сторону Востока", наверное, будет самым простым для понимания. Я объясню, как этот код работает, а затем покажу, чем отличаются другие направления. Вот часть кода, которая пытается записать слово в «восточном направлении».

```
case "E":
      //колонки от 0 до ширины поля - ширина слова
      //строки от 0 до высоты поля
            $newCol = rand(0, $boardData["width"] - 1 -
strlen($theWord));
      $newRow = rand(0, $boardData["height"]-1);
```

[1] East – Восток. – *Примеч.пер.*

```
for ($i = 0; $i < strlen($theWord); $i++){
  //новый символ, та же строка, начальная колонка + $i
  $boardLetter = $board[$newRow][$newCol + $i];
  $wordLetter = substr($theWord, $i, 1);

  //проверяем, свободно ли значение текущей ячейки поля
  if (($boardLetter == $wordLetter) ||
      ($boardLetter == ".")){
    $board[$newRow][$newCol + $i] = $wordLetter;
  } else {
    $itWorked = FALSE;
  } // завершение if
} // завершение цикла for
break;
```

Определяем начальные значения для символов

В сущности, код проходит через каждое слово по одной букве за раз, помещая каждую букву в ячейку справа от предыдущей. Я мог бы выбрать случайную ячейку и проверить, когда код выйдет за границу поля, но это потребовало бы сложного и неуклюжего кода. Более элегантное решение – аккуратно определить диапазон подходящих начальных ячеек и выбирать ячейки только в этом диапазоне. Например, я помещаю слово "elephant" (состоящее из восьми букв) слева направо в головоломку с шириной десять символов, т. е. единственными подходящими колонками будут нулевая и первая (помните, компьютеры обычно начинают считать с нуля). Если я помещаю это же слово в ту же головоломку, но справа налево, единственными подходящими были бы последние две колонки (8 и 9). Как только я это понял, мне осталось только определить, как воплотить эту идею в коде, чтобы я мог работать со словами любого размера в головоломке любого размера.

Возвращаясь к реальности

Самая важная часть этого кода – комментарии в его начале. Хотя я довольно опытный программист, даже мне довольно легко запутаться, решая проблемы определенной сложности. Просто чтобы напоминать себе, я поместил эти комментарии, чтобы объяснять самому себе, какие именно параметры у этого куска кода.
Я обращался к этим комментариям много раз в процессе написания и отладки кода. Если бы я не задал себе четких указаний о том, что я пытаюсь сделать, я бы, наверное, настолько запутался, что не смог бы написать эту программу

Чтобы определить, где поместить каждое слово, мне нужно случайное значение строки и столбца, но это случайное значение должно быть в пределах определенного диапазона, основанного на длине слова и ширине поля. С помощью проб, ошибок и некоторых набросков на доске, я определил, что `$boardData["width"] - 1` является последней колонкой на игровом поле, а `strlen($theWord)` – это длина текущего слова в символах. Отняв длину слова от ширины поля, я получу наибольшее дозволенное стартовое значение для расположения «слева направо». Вот так я получил немного пугающую формулу.

```
$boardData["width"] - 1 - strlen($theWord);
```

Наименьшее дозволенное стартовое значение для такого расположения равняется нулю, потому что нулевая колонка всегда подходит, если слово помещать слева направо и если слово такого же размера или меньше, чем головоломка (что уже установлено).

В «восточном расположении» номер строки не имеет значения, потому что все буквы будут размещены в одной строке.

Узнав наибольшую и наименьшую дозволенную точку начала слова, я могу случайным образом создать начальную точку, зная, что от этой точки может быть успешно расположено целое, если оно не пересечется с другими словами.

Я использовал цикл `for` для помещения одного символа из слова с помощью функции `substr()`. Каждый символ расположен в той же строке, что и начальный символ, но в колонке, смещенной на позицию символа в слове. Снова обратимся к примеру "elephant": если начальной колонкой выбрана колонка 1, то символ "E" будет расположен в колонку 1, потому что "E" находится на нулевой позиции в слове "elephant", а `1 + 0 = 1`. Когда счетчик (`$i`) доберется до буквы "L", у него будет значение 1, так что буква будет помещена в колонку 2, и т. д.

Если формула для выбора стартовой позиции и план для расположения последующих букв работают правильно, будет невозможно добавить букву за границы головоломки. Однако может случиться еще одна неприятность, если символ из ранее расположенного слова находится в ячейке, которую облюбовало текущее слово. Код проверяет состояние текущей ячейки поля. Если ячейка содержит значение ".", т. е. она пуста, то в нее можно поместить новый символ. Если в ячейке содержится такая же буква, какую текущее слово пытается в нее поместить, то программа продолжит выполняться не прерываясь. Однако если в ячейке содержится любой другой символ, циклу придется прерваться, а программе обнулить игровое поле и попробовать еще раз. Это сделано с помощью присвоения переменной `$itWorked` значения `FALSE`.

Печатаем слова в других направлениях

После того как вы поймете, как печатать слова в восточном направлении, вы увидите, что печать во всех остальных направлениях очень похожа. Однако для каждого направления мне необходимо узнать, какими были подходящие начальные значения и в какой ячейке помещать букву. Все эти значения объединены в таблицу, которая приведена ниже.

Табл. 5.2. Минимальные и максимальные индексы для каждого направления

	Восточное	Западное	Южное	Северное
Минимальное значение столбца	0	Ширина слова	0	0
Максимальное значение столбца	Ширина доски-1– ширина слова	Ширина доски-1	Ширина доски-1	Ширина доски-1
Минимальное значение строки	0	0	0	Ширина слова
Максимальное значение строки	Высота доски-1	Высота доски-1	Высота доски-1-ширина слова	Высота доски-1
Столбец буквы	Начало+i	Начало-i	Начало	Начало
Строка буквы	Начало	Начало	Начало+i	Начало-I

Приведем небольшое объяснение таблицы 5.2. В таблице я определил минимальные и максимальные индексы столбцов и строк для каждого направления. Фактически, проще всего это было определить, написав несколько примеров на бумаге. Расположение каждой буквы зависит от индексов начальной строки и столбца, причем i является индексом текущей буквы в слове. Поэтому в западном направлении я помещу вторую букву слова в ранее случайно выбранную строчку, но в столбец, имеющий индекс начального минус два. Это позволит напечатать буквы справа налево. Попробуйте рассмотреть другие примеры на бумаге, чтобы понять, как они работают.

Возвращаясь к реальности

Это является как раз тем случаем, когда для большинства людей программирование начинает казаться чем-то загадочным. До сих пор вы, вероятно, просто повторяли все, что я делал, однако эта задача расположения букв требует знаний математики и вы не видели, как я с ней боролся. Вам может показаться, что я просто знал нужные формулы. Это не так. Мне пришлось подумать, и сделать это *без* компьютера. Я взял доску (мое любимое средство разработки), бумагу и попытался выяснить что представляет из себя математическое толкование фразы «записать символы снизу вверх». Это трудно, однако вы можете это сделать. Начинающие всегда забывают сделать одну вещь – выключить компьютер. Возьмите листок бумаги и выясните, что вы пытаетесь заставить делать компьютер. После этого вы можете начинать писать код. Однако все равно он будет с ошибками (по крайне мере был у меня). Однако если вы записали алгоритм, вы можете сравнивать то, что происходит в действительности, с тем, что вы ожидали увидеть, и сможете решить даже такую сложную математическую задачу

Создаем доску головоломки

К тому моменту, когда функция `fillBoard()` закончит добавление всех слов при помощи вызовов `addWord()`, ключ ответа будет завершен. Каждое слово будет на своем месте, а ячейки, не содержащие слов, будут по-прежнему содержать точки. Основная программа скопирует текущую переменную `$board` в массив `$key`. После этого ключ ответа будет готов к преобразованию в форму, которую сможет использовать пользователь. Однако вместо того чтобы создать функцию, выводящую ключ ответа, и функцию, отображающую законченную головоломку, я написал одну функцию, которая принимает в качестве параметра массив и создает длинную строку кода HTML, помещающего головоломку в таблицу.

```
function makeBoard($theBoard){
// используя заданный массив доски, вернуть таблицу HTML
  global $boardData;
  $puzzle = "";
  $puzzle .= "<table border = 0>\n";
  for ($row = 0; $row < $boardData["height"]; $row++){
    $puzzle .= "<tr>\n";
    for ($col = 0; $col < $boardData["width"]; $col++){
      $puzzle .= "  <td width = 15>{$theBoard[$row][$col]}</td>\n";
    } // end col for loop
    $puzzle .= "</tr>\n";
  } // завершение цикла for для строки
  $puzzle .= "</table>\n";
  return $puzzle;
} // завершение printBoard;
```

Большая часть функции предназначена для создания таблицы HTML, которая будет храниться в переменной `$puzzle`. Каждая строка, которой соответствует головоломка, начинается в тега HTML `<tr>` и создания пары `<td></td>` для каждого элемента в таблице.

Иногда PHP испытывает трудности правильной интерполяции двумерных массивов. Если вы обнаружите, что массив был неправильно интерполирован, вы можете попробовать сделать две вещи. Прежде всего, вы можете просто заключить ссылку на массив в фигурные скобки, как я сделал это в коде `makeBoard()`. *Кроме этого, вы можете использовать конкатенацию вместо интерполяции. Например, вы могли бы создать все ячейки с помощью следующего кода.*

```
$puzzle .= "<td> width = 15>" . $theBoard[$row][$col] . "</td>\n";
```

Добавляем контрастные буквы

Головоломка может быть легко решена при помощи ключа ответа. Как только слова помещены в список, все, что необходимо сделать для генерации головоломки, – это заменить точки на любые случайные буквы. Я называю такие символы «контрастными буквами», потому что их задача – сбить пользователя с толка. Добавление этих букв является более простой задачей, чем процесс добавления слов.

```
function addFoils(){
  //добавить случайные символы на доску
  global $board, $boardData;
  for ($row = 0; $row < $boardData["height"]; $row++){
    for ($col = 0; $col < $boardData["width"]; $col++){
      if ($board[$row][$col] == "."){
        $newLetter = rand(65, 90);
        $board[$row][$col] = chr($newLetter);
      } // завершение if
    } // завершение цикла for для столбцов
  } // завершение цикла for для строк
} // завершение addFoils
```

Функция использует стандартную пару вложенных циклов для просмотра всех ячеек массива. При этом для каждой ячейки, содержащей точку, она генерирует случайное число в диапазоне от 65 до 90. Эти числа соответствуют численным кодам ASCII для прописных букв. После этого я использую функцию `chr()` для получения буквы, соответствующей этому числу, и сохраняю ее в массиве.

Выводим головоломку

Последним шагом основной программы является отображение результатов пользователю. Пока что мы занимались фоновой работой. Теперь пришло время создать страницу HTML, содержащую результаты. Этой цели служит функция `printPuzzle()`. Сама головоломка и таблицы ключей ответов уже были отформатированы в HTML функцией `printBoard()`. HTML-код головоломки хранится в переменной `$puzzle`, а ключ ответа хранится в `$keyPuzzle`.

```
function printPuzzle(){
  //отобразить страницу, содержащую головоломку, пользователю
  global $puzzle, $word, $keyPuzzle, $boardData;
  //отобразить головоломку
  print <<<HERE
  <center>
  <h1>{$boardData["name"]}</h1>
  $puzzle
  <h3>Word List</h3>
  <table border = 0>
HERE;
  //отобразить список слов
  foreach ($word as $theWord){
    print "<tr><td>$theWord</td></tr>\n";
  } // завершение цикла foreach
  print "</table>\n";
  $puzzleName = $boardData["name"];
  //отобразить форму запроса ключа ответа
  //послать ключ ответа этой форме (ловкий ход!)
  print <<<HERE
  <br><br><br><br><br><br><br><br>
  <form action = "wordFindKey.php"
        method = "post">
  <input type = "hidden"
         name = "key"
         value = "$keyPuzzle">
  <input type = "hidden"
         name = "puzzleName"
         value = "$puzzleName">

  <input type = "submit"
         value = "show answer key">
  </form>
```

```
  </center>
HERE;
  } // завершение printPuzzle
```

Эта функция в основном печатает стандартный HTML из переменных, которые были созданы в ходе выполнения программы. Название головоломки хранится в `$boardData["name"]`. Сама головоломка является значением переменной `$puzzle`. Я напечатал список слов соответствующим вызовов цикла `foreach`, создающим список из массива `$word`.

Самой хитрой частью кода является работа с ключом ответов. Было бы достаточно просто отобразить его непосредственно на той же самой странице. На самом деле, я так и сделал при тестировании программы. Однако разгадывание головоломки не принесет никакого удовольствия, если ответ будет показан сразу, поэтому для получения ответа пользователю необходимо нажать кнопку. Ключ относится только к текущей головоломке. Если бы тот же самый список слов был бы заново использован в программе поиска слов, из него бы получилась совсем другая головоломка с совершенно иным ответом. Секретом является хранение текущего ключа ответов в форме с атрибутом «hidden» и передаче его в другую программу. Я создал форму с двумя скрытыми полями. Я храню название головоломки в поле `puzzleName`, а весь HTML ключа ответа в поле `key`. Когда пользователь нажимает кнопку `Submit`, это инициирует вызов программы `wordFindKey`.

Возвращаясь к реальности

Передача ключа ответа в другую программу является «грязным» приемом. Он работает по нескольким причинам. Во-первых, так как поле `key` является скрытым и форма отсылает данные с помощью метода `post`, пользователь не будет знать, что ответ головоломки находится у него под носом. Так как я предполагаю, что эта программа будет использоваться в основном учителями, которые в любом случае посмотрят ответ, в этом нет ничего страшного. Даже не беря во внимание вопросы секретности, необходимо передавать данные ключа с помощью метода `post`, так как их размер более 256 символов, разрешенных в методе `get`.
Отправка отформатированного в HTML ключа ответов в следующую программу делает ее достаточно простой, однако у этого подхода есть еще одно преимущество: очень сложно полностью передать массив в виде переменной, однако при создании HTML-таблицы все данные массива преобразуются в строку, которую можно легко передать в другую программу посредством формы

Печатаем ключ ответов

Программа wordFindKey является очень простой, потому что вся работа по генерации ключа ответов была проделана программой поиска слов. Все, что необходимо сделать данной программе, – получить название головоломки и ключ ответов из переменных формы и отобразить их. Так как ключ был отформатирован в виде таблицы, программе wordFindKey не надо делать ничего сложного.

```
<!doctype html public "-//W3C//DTD HTML 4.0 //EN">
<html>
<head>
<title>Word Find Answer Key</title>
</head>
<body>

<?
//ключ ответов для программы поиска слов
//вызывается из wordFind.php

print <<<HERE
<center>
<h1>$puzzleName Answer key</h1>
$key
</center>

HERE;
?>
</body>
</html>
```

Итоги

В этой главе вы начали понимать, насколько важно делать данные легко читаемыми. Вы познакомились с несколькими основными типами массивов и инструментов для управления ими. Вы научились использовать циклы foreach для последовательного просмотра каждого элемента массива. Вы можете использовать строковые индексы для создания ассоциативных массивов. Вы позакомились с основными операциями со строками, включающими поиск строки внутри другой строки, извлечение подстрок и разделение строк в массивы. Вы применили все эти знания для создания интересной и нетривиальной программы. Вы должны гордиться достигнутым.

Домашнее задание

1. Добавьте возможность использования диагоналей в ваших головоломках. (Подсказка: вам просто необходимо соединить формулы, которые я вам показал ранее. Вам нет нужды придумывать новые.)
2. Создайте игру Морской бой (BattleShip) для двух игроков за одним компьютером. Игра будет отображать сетку. (Заранее определенное расположение флотилий сделает ее проще.) Позвольте пользователю выбирать расположение на сетке с помощью флажков. Сообщите о результатах выстрела пользователю и передайте ход другому игроку.
3. Напишите версию программы Conway's Life. Эта программа имитирует клеточную жизнь в сетке с тремя простыми правилами.
   - Каждая клетка, имеющая трех соседей, станет или останется живой.
   - Каждая клетка, которая жива и имеет двух соседей, остается живой.
   - Все остальные клетки умирают.
4. Случайно создайте первую клетку и позвольте пользователю нажать кнопку для генерации следующего поколения.

Глава 6

Работаем с файлами

По мере того как будет расти ваш опыт программирования, вам будет становиться понятна вся значимость данных. Вы начали свой путь постижения данных с простых переменных, после чего узнали, как простые и более сложные массивы могут сделать ваши программы более гибкими и мощными. Однако данные хранятся в памяти компьютера временно, особенно в серверной среде. Зачастую необходима возможность более длительного хранения данных, чем с помощью способов, уже изученных вами ранее. PHP предоставляет несколько мощных функций для работы с текстовыми файлами. Используя эти возможности, вы сможете создавать исключительно полезные программы. В частности, вы научитесь:

- открывать файлы для чтения, записи и добавления данных;
- использовать дескрипторы файлов для управления текстовыми файлами;
- записывать данные в текстовый файл;
- считывать данные из текстового файла;
- открывать весь файл в массив;
- изменять текстовые данные «налету» в реальном времени;
- получать информацию обо всех файлах в заданной директории;
- получать набор файлов по их именам.

Предварительный обзор машины тестирования

Основная программа этой главы является забавным, но в то же время мощным инструментом, имеющим множество областей применения. Это не просто программа, а целая система совместно работающих программ, позволяющая вам создавать, администрировать и оценивать многовариантные тесты автоматически.

Возвращаясь к реальности

Достаточно просто создать страницу HTML, представляющую собой тест, и программу PHP, оценивающую только этот тест. Однако если вы хотите сделать несколько тестов, наверное, лучше будет потратить время и силы на создание системы, которая может автоматизировать создание и администрирование тестов. Настоящая сила программирование заключается не в том, что вы можете решить какую-либо конкретную проблему, а в том, что вы создаете такое решение, которое может быть применено ко всем аналогичным проблемам. Машина тестирования как раз является примером такой системы. Для создания такой системы вначале требуется больше усилий, но они окупаются сторицей, когда вы можете заново использовать систему

Введение в систему машины тестирования

На рис. 6.1 изображена основная страница системы. Чтобы пройти тестирование, пользователю нужен пароль, а для редактирования нужен дополнительный пароль администратора. В данном случае я ввел пароль администратора (им является слово *«absolute»*) в соответствующее поле для пароля и собираюсь отредактировать тест Monty Python.

Редактирование теста

Если пользователь ввел верный пароль, появится экран, изображенный на рис. 6.2, отображающий в специальном формате запрошенный тест.

Администратор теста может редактировать его несколькими способами. У каждого теста есть название, адрес электронной почты инструктора и пароль. Каждый вопрос хранится в одной строке, содержащей вопрос, четыре возможных ответа и правильный ответ, отделенный знаком двоеточия (:).

Прохождение тестирования

Пользователи, знающие соответствующий пароль, могут пройти любой из известных системе тестов. Если пользователь выберет для прохождения тест Monty Python, появится экран, изображенный на рис. 6.3.

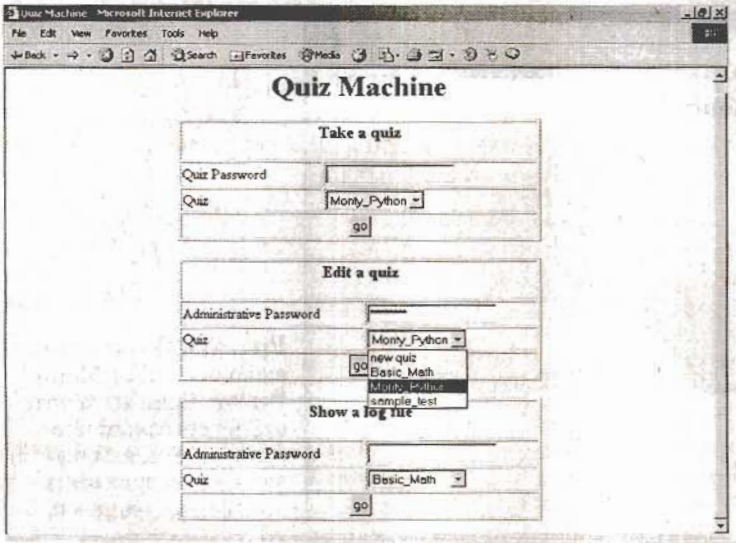


Рис. 6.1. Пользователь является администратором, готовящимся к редактированию теста

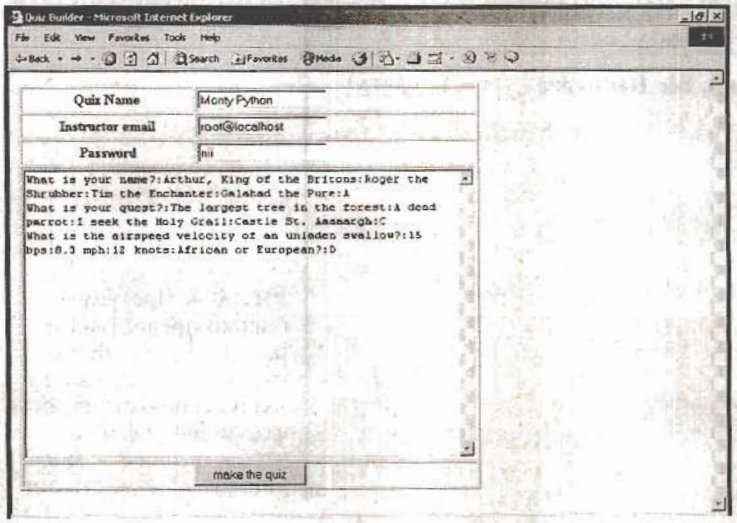


Рис. 6.2. Пользователь выбрал для редактирования тест Monty Python

Просмотр результатов

Когда пользователь проходит тестирование, его ответы посылаются программе, которая оценивает тест и немедленно сообщает о результатах тестирования, как показано на рис. 6.4.

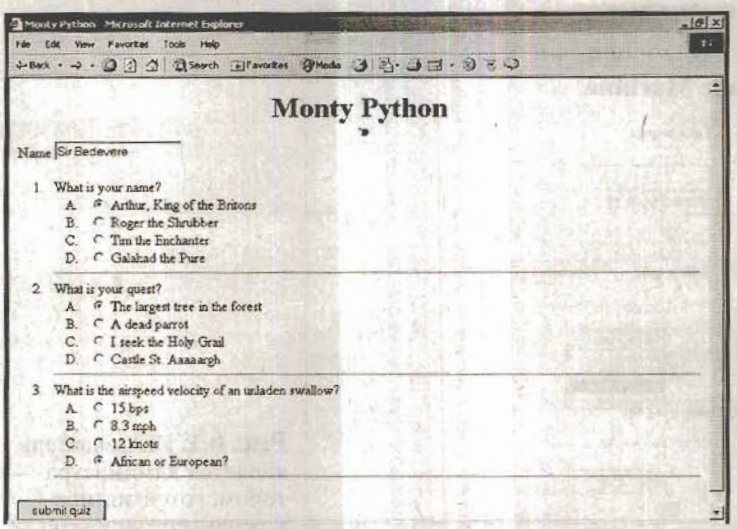


Рис. 6.3. Пользователь выполняет тест Monty Python. Если вы хотите стать серьезным программистом, вам, наверное, следует взять этот фильм напрокат

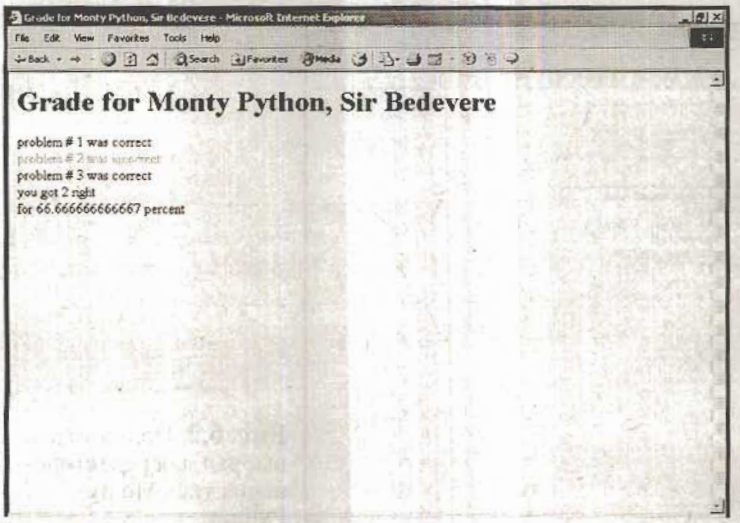


Рис. 6.4. Программа оценки предоставляет немедленную обратную связь пользователю и сохраняет эту информацию в файле, чтобы позднее ее мог увидеть администратор

Просмотр журнала событий теста

Система ведет журнал событий для каждого теста, чтобы администратор мог сообщить результаты каждому человеку, прошедшему тестирование. На рис. 6.5 показаны результаты людей, выполнивших тест Monty Python.

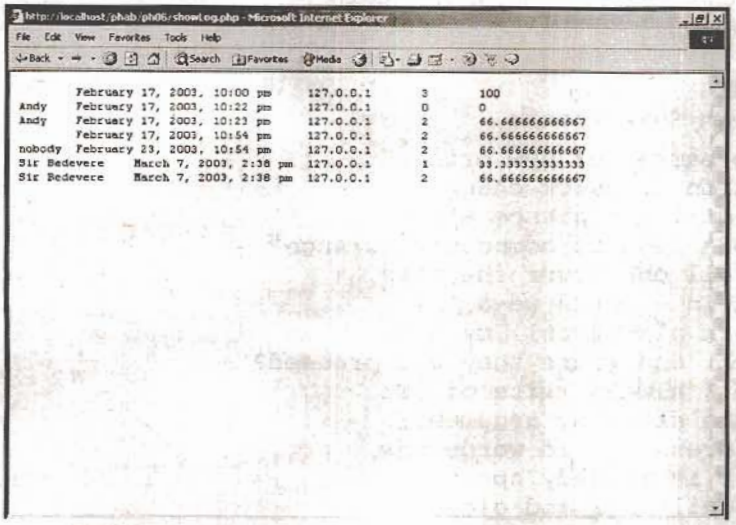


Рис. 6.5. Программа получения журнала событий отображает его для каждого теста

Хотя журнал результатов и выглядит очень просто, он генерируется в виде, который может быть легко импортирован в большинство программ записи оценок и таблиц. Это очень полезно, если тест будет использоваться в учебных целях.

Сохраняем файл в файловой системе

Ваши PHP-программы могут получать доступ к файловой системе сервера для сохранения и извлечения информации. Ваши программы могут создавать новые файлы, добавлять данные в файлы и считывать информацию из файлов. Вы начнете с написания программы, создающей файл и добавляющей в него данные.

Знакомимся с программой saveSonnet.php

Программа `saveSonnet.php`, показанная в следующем ниже листинге, открывает файл на сервере и записывает в него один из сонетов Шекспира.

Обычно я привожу снимки экрана всех программ, однако так как эта программа ничего не отображает, такой снимок был бы бесполезен. Следующие несколько программ будут считывать и отображать этот файл на экране, и при их рассмотрении я покажу вам, как они выглядят.

```
head>
<title>SaveSonnet</title>
</head>
```

```
<body>
<?
$sonnet76 = <<<HERE
Sonnet # 76, William Shakespeare

Why is my verse so barren of new pride,
So far from variation or quick change?
Why with the time do I not glance aside
To new-found methods, and to compounds strange?
Why write I still all one, ever the same,
And keep invention in a noted weed,
That every word doth almost tell my name,
Showing their birth, and where they did proceed?
O! know sweet love I always write of you,
And you and love are still my argument;
So all my best is dressing old words new,
Spending again what is already spent:
For as the sun is daily new and old,
So is my love still telling what is told.

HERE;

$fp = fopen("sonnet76.txt", "w");
fputs($fp, $sonnet76);
fclose($fp);

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

Большая часть кода сохраняет содержимое 76-го сонета Шекспира в переменную `$sonnet76`. Оставшиеся три строчки сохраняют данные, содержащиеся в этой переменной, в текстовый файл.

Открываем файл с помощью fopen

Команда `fopen()` используется для открытия файла. Заметьте, что вы можете создавать файлы только на веб-сервере. Вы не можете создавать файлы непосредственно на машине клиента, так как не имеете доступа к файловой системе этой машины. (Если бы вы имели такой доступ, любая программа на серверной стороне могла бы очень просто создавать вирусы.) Однако поскольку вы является программистом серверной стороны, у вас есть возможность создавать файлы на сервере. Программы, создаваемые вами, сами по себе являются файлами. Ваши программы могут создавать файлы, как если бы они были вами.

На самом деле, владение файлами, созданными вашими PHP программами, может быть сложнее, в зависимости от операционной системы, сервера и настроек PHP. В общем случае, владельцем любого файла, созданного вашей программой, будет пользователь с именем PHP или учетная запись, которая была активна в момент написания программы. Это является очень важным в UNIX-подобных операционных системах, потому что владение файлами является основной частью защитного механизма. Лучшим способом выяснить, как работает этот механизм, является написание программы, создающей файл, и просмотр его свойств.

Первым параметром функции `fopen()` является имя файла. Это имя может содержать в себе информацию о директориях или быть относительной ссылкой.

Вы должны всегда тестировать ваши программы, в особенности если они используют относительные ссылки для названий файлов. Возможен вариант, когда текущая директория не будет совпадать с заданной по умолчанию. Кроме того, название файла основывается на фактической файловой системе сервера, а не унифицированном указателе ресурсов файла.

Вы можете создать файл в любом месте сервера, к которому у вас есть доступ. Ваши файлы могут располагаться в любой части директории `system`, доступной веб-серверу (обычно это поддиректории `public_html` или `htdocs`). Однако в некоторых ситуациях вы можете захотеть запретить непосредственный доступ к файлам с помощью их унифицированных указателей ресурсов. Вы можете управлять доступом к этим файлам, помещая их вне открытого пространства html и устанавливая разрешения таким образом, чтобы только вы (и ваши программы) могли получать доступ к этим файлам.

Создаем дескриптор файла

Когда вы создаете файл с помощью команды `fopen()`, она возвращает целое число, называемое *дескриптором файла* (иногда также называемое *указателем на файл*). Это специальное число будет использовано для обращения к файлу в соответствующих командах. Обычно нет нужды заботиться о фактическом значении этого дескриптора, однако его необходимо сохранить в переменной (обычно я использую `$fp`), чтобы остальные команды работы с файлами знали, какой файл использовать.

Рассмотрение модификаторов доступа к файлам

Последним параметром команды `fopen()` является модификатор доступа. PHP поддерживает несколько модификаторов доступа, определяющих способ взаимодействия вашей программы с файлом. Обычно файлы открываются для

чтения, записи или добавления. Вы также можете использовать файл одновременно для чтения и записи, однако такие файлы обычно не требуются в PHP, потому что технологии реляционных баз данных, которые вы изучите в следующих главах, предоставляют те же самые возможности, обеспечивая при этом большую гибкость и требуя выполнения меньшего объема работы. Однако остальные формы доступа к файлам (чтение, запись и вывод) очень полезны, так как предоставляют очень простой доступ к информации, хранящейся в файле. (Список модификаторов доступа к файлам и их описание смотрите в табл. 6.1.)

Табл. 6.1. Модификаторы доступа к файлам

Модификатор	Тип	Описание
«r»	Только чтение	Программа может считывать данные из файла
«w»	Запись	Программа может записывать данные файлы, при этом если файл существовал, то его содержимое стирается
«a»	Добавление	Программа может записывать данные в конец файла
«r+»«w+»	Чтение и запись	

Модификаторы доступа к файлам определяют, какие операции могут выполняться с файлами. В режиме чтения файл открывается для ввода данных, что позволяет вашей программе считывать информацию из файла. Вы не можете записывать информацию в файл, открытый для чтения. Вы увидите пример использования режима чтения в следующем разделе «Загрузка файла из дисковой системы». Режим записи позволяет вам открывать файл для вывода данных. Если заданный файл не существует, PHP автоматически создает его.

Будьте очень осторожны при открытии файлов в режиме записи. Если открываемый файл уже существует и вы открываете его для записи, PHP создаст новый файл, перезаписав старый и уничтожив его содержимое.

Режим добавления позволяет вам записывать данные в файл, не уничтожая при этом его содержимого. Когда вы записываете данные в файл, открытый в режиме добавления, все новые данные записываются в конец файла.

Возвращаясь к реальности

Модификаторы «r+» и «w+» используются для другого способа доступа к файлам, называемого «произвольным доступом», который позволяет одновременно производить чтение и запись в одном и том же файле. Хотя это и очень полезная возможность, я не буду тратить на нее время в этой вводной книге. Методы последовательного доступа, которые вы изучите в этой главе, являются достаточными для рассмотрения вопроса сохранения информации в файлах, а реляционные базы данных, о которых вы узнаете в оставшейся части книги, не сложнее чем модель произвольного доступа, однако предоставляет значительно больше возможностей

Записываем данные в файл

Программа `saveSonnet` открывает файл `sonnet76.txt` для записи. Если в текущей директории уже существует такой файл, он уничтожается. Указатель на файл для текстового файла сохраняется в переменной `$fp`. После этого я могу использовать функцию `fputs()` для записи данных в файл.

Вы могли заметить здесь закономерность. Большинство функций работы с файлами начинаются с буквы `f` *(*`fopen()`*,* `fclose()`*,* `fputs()`*,* `fgets()`*,* `feof()`*). Это соглашение пришло из языка C. Это поможет вам запомнить, что определенная функция работает с файлами. Конечно же, в PHP не каждое выражение, начинающиеся с* `f`*, обязательно относится к файлам (*`foreach` *является хорошим примером), однако большинство функций, названия которых начинаются с* `f`*, предназначены для работы с файлами.*

Функция `fputs()` имеет два параметра. Первым является указатель на файл. Это позволяет сообщить PHP, куда записывать данные. Вторым параметром является текст, записываемый в файл.

Закрываем файл

Функция `fclose()` сообщает системе, что ваша программа завершила работу с файлом и система может его закрыть.

Дисковые системы намного медленнее памяти компьютера, и для достижения приемлемой скорости требуется гораздо больше времени. По этой причине, когда программа обнаруживают команду `fputs()`*, она не сразу же сохраняет данные в файл на диск. Вместо этого она добавляет данные в специальный буфер и записывает данные только тогда, когда в буфере находится достаточное*

количество информации или программа вызывала команду `fclose()`. *Поэтому очень важно закрывать открытые файлы. Если программа завершается, так и не вызывав* `fclose()`, *PHP должен автоматически закрыть файл, однако происходящее на самом деле не всегда соответствует этому.*

Загружаем файл из дисковой системы

Очень важной является возможность получения данных из файловой системы. Если вы откроете файл с модификатором доступа «r», вы сможете считать информацию из него.

Знакомимся с программой loadSonnet.php

Программа `loadSonnet.php`, показанная на рис. 6.6, загружает сонет, сохраненный программой `saveSonnet.php`, и отображает его.

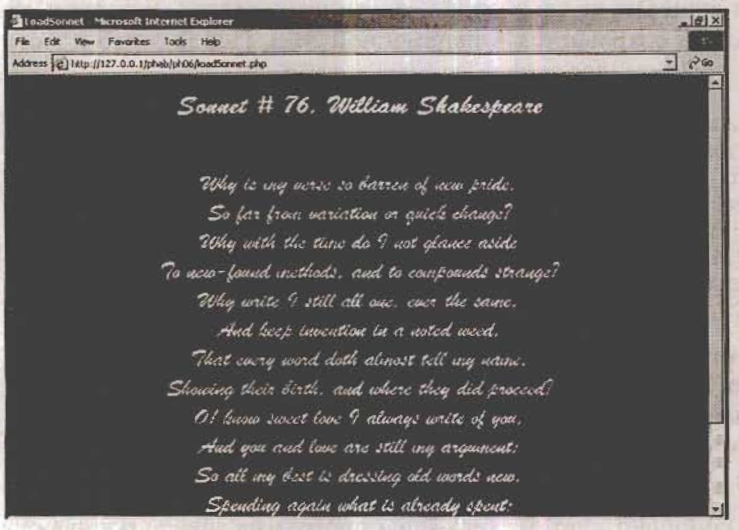


Рис. 6.6. Файл был загружен из дисковой системы и немного улучшен с помощью приемов CSS

Вот код программы `saveSonnet.php`.

```
<html>
<head>
<title>LoadSonnet</title>
<style type = "text/css">
body{
  background-color:darkred;
  color:white;
  font-family:'Brush Script MT', script;
```

```
  font-size:20pt
}
</style>
</head>
<body>
<?
$fp = fopen("sonnet76.txt", "r");
//первая строка является названием
$line = fgets($fp);
print "<center><h1>$line</h1></center>\n";
print "<center>\n";
//вывести оставшуюся часть сонета
while (!feof($fp)){
  $line = fgets($fp);
  print "$line <br>\n";
} // завершение while
print "</center>\n";
fclose($fp);
?>
</body>
</html>
```

Используем таблицы CSS для улучшения вывода

Лучшим способом улучшения внешнего вида текста является использование стилей CSS. Установив простое стилевое оформление, я смог очень быстро улучшить внешний вид сонета, не изменяя при этом его текст. Особое внимание обратите на то, как я указал несколько шрифтов, на тот случай, если выбранный мною шрифт не установлен в системе пользователя.

Используем модификатор доступа «r»

Чтобы считать данные из файла, вы должны получить указатель на файл, открыв его с помощью модификатора «r». Если файл не существует, вместо указателя на файл вы получите FALSE.

На самом деле вы можете открывать файлы для чтения где угодно в Интернете. Если вы укажете унифицированный указатель на ресурс в качестве имени файла, вы сможете считывать данные по этому указателю, как если бы он был файлом. Однако вы не можете открывать такие URL-файлы для вывода.

Я открыл файл «sonnet76.txt» с помощью команды `fopen()` и модификатора доступа «r», после чего скопировал возвращенное число в переменную файлового указателя `$fp`.

Проверяем достижение конца файла с помощью feof()

При считывании данных из файла ваша программа в общем случае не знает, каким будет его размер. Команда `fgets()`, часто используемая для получения данных из файла, за раз считывает одну строку файла. Так как вы не знаете, сколько строк в файле, пока не считаете их, PHP предоставляет специальную функцию `feof()`, которая означает «файл конец файла»[1] (вероятно, это название предложено министерством сокращения министерств). Эта функция возвращает `FALSE`, если в файле еще остались строки с данными, и `TRUE`, если программа находится в конце данных файла. В большинстве случаев, когда вы считываете данные из файла, вы используете цикл `while`, который выполняется, если `feof()` ложно. Самым простым способом создание такого цикла является код следующего вида.

```
while (!feof($fp)){
```

Функция `feof()` принимает указатель на файл в качестве своего единственного параметра.

Считываем данные из файла с помощью fgets()

Функция `fgets()` получает одну строку данных из файла и возвращает ее значение в виде строки, сдвигая при этом специальный указатель на следующую строку файла. Эта функция обычно вызывается в теле цикла, который выполняется до тех пор, пока `feof()` истинно.

Считываем файл в массив

Зачастую полезно работать с файлом как с массивом данных в памяти. Часто вы можете заметить, что выполняете какие-либо операции над каждой строкой массива. PHP предоставляет функции для упрощения таких операций. Программа `cartoonifier.php` демонстрирует один из способов управления всем файлом без использования указателя на файл.

---

[1] file end of file – файл конец файла. – *Прим.ечеч. пер.*

Представляем программу cartoonifier.php

Программа `cartoonifier.php`, показанная на рис. 6.7, является действительно серьезной и широко использует современные серверные технологии.

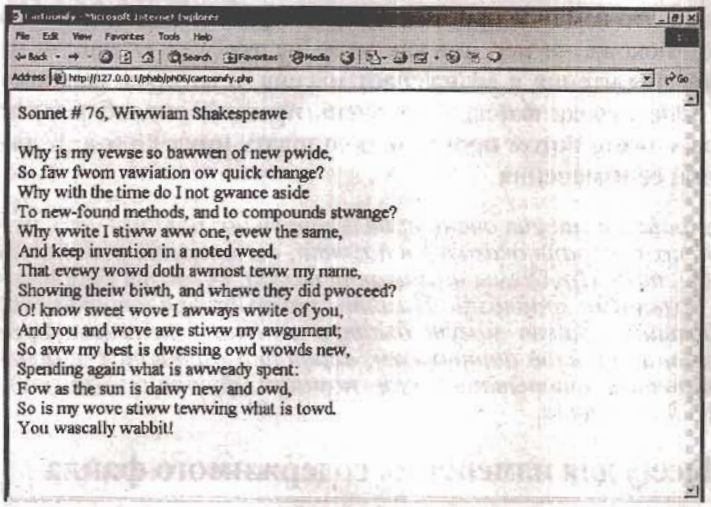


Рис. 6.7. Программа `cartoonifier` показывает, чтобы случилось, если бы Шекспир был персонажем мультфильма

Эта программа загружает весь сонет в массив, просматривает каждую строчку стиха и преобразует ее в уникальный мультипликационный диалект, выполняя операции поиска и замещения.

```
<html>
<head>
<title>Cartoonify</title>
</head>
<body>
<?
$fileName = "sonnet76.txt";
$sonnet = file($fileName);
$output = "";
foreach ($sonnet as $currentLine){
  $currentLine = str_replace("r", "w", $currentLine);
  $currentLine = str_replace("l", "w", $currentLine);
  $output .= rtrim($currentLine) . "<br>\n";
} // завершение цикла foreach
  $output .= "You wascally wabbit!<br>\n";
print $output;
?>
</body>
</html>
```

Загружаем файл в массив с помощью file()

Существуют также обходные способы работы с файлами, которые не требуют создания указателя на файл. Возможно, вы вспомните команду `readFile()` из первой главы этой книги. Это команда просто считывает файл и записывает его в вывод. Команда `file()` похожа на нее, так как тоже не требует указателя на файл. Она открывает файл для чтения и возвращает массив, в котором каждому элементу соответствует одна строка файла. Это очень простой способ работы с файлами, потому что вы можете потом просто использовать цикл `foreach` для просмотра каждой строки и ее изменения.

Считывание файла в массив очень привлекательно, потому что это просто и как только файл оказался в памяти, вы можете работать с ним очень быстро. Проблемы возникают тогда, когда вы работаете с очень большими файлами. Память компьютера ограничена, и поэтому большие файлы могут быстро стать причинами проблем. Для больших файлов данных вам, вероятно, придется использовать построчное считывание при помощи функции `fgets()`*, помещенной в тело цикла.*

Используем str_replace() для изменения содержимого файла

Внутри цикла `foreach` очень просто заменить все буквы «r» и «l» на «w» с помощью функции `str_replace()`. После этого полученная строка добавляется в переменную `$output`, которая в конечном счете будет выведена на экран.

Возвращаясь к реальности

Это приложение является очень простым и довольно бессмысленным, однако возможность замены всех вхождений строки на другую строку в тексте является очень полезной. Например, вы могли бы заменить все вхождения символа «<» в документе HTML на последовательность `<`. Это приведет к тому, что исходный код можно будет просматривать непосредственно в браузере. Кроме того, вы сможете использовать такую технологию для формирования букв, беря информацию из шаблона текста и замещая ее на значения, введенные пользователем или полученные из другого файла

Работаем с информацией о директориях

Когда вы работаете с файловыми системами, зачастую вам приходится работать и со структурой директорий, которые содержат файлы. PHP содержит несколько команд, которые помогают управлять директориями.

Знакомимся с программой imageIndex.php

Программа `imageIndex.php`, изображенная на рис. 6.8, является простой вспомогательной программой, генерирующей список всех файлов jpg и gif, содержащихся в заданной директории.

Когда пользователь щелкает мышью по макету изображения, отображается его полная версия. Технологии, использованные для отображения файлов изображения, могут быть использованы для получения определенных наборов файлов из любой директории.

```
<html>
<head>
<title>imageIndex</title>
</head>
<body>
<?
// список изображений
// генерирует файл списка, содержащего названия всех изображений,
// расположенных в заданной директории
//задайте директорию, список изображений которой хотите получить
//список запишется в эту директорию в файл imageIndex.html
$dirName = "C:\csci\mm";
$dp = opendir($dirName);
chdir($dirName);
//добавить все файлы, находящиеся в директории, в массив $theFiles
while ($currentFile !== false){
  $currentFile = readDir($dp);
  $theFiles[] = $currentFile;
} // завершение while
//извлечь все изображения gif и jpg
$imageFiles = preg_grep("/jpg$|gif$/", $theFiles);
$output = "";
foreach ($imageFiles as $currentFile){
  $output .= <<<HERE
<a href = $currentFile>
  <img src = "$currentFile"
       height = 50
       width = 50>
</a>
HERE;
} // завершение цикла foreach
```

```
// сохранить список в локальную файловую систему
$fp = fopen("imageIndex.html", "w");
fputs ($fp, $output);
fclose($fp);
//readFile("imageIndex.html");
print "<a href = $dirName/imageIndex.html>image index</a>\n";
 ?>
</body>
</html>
```

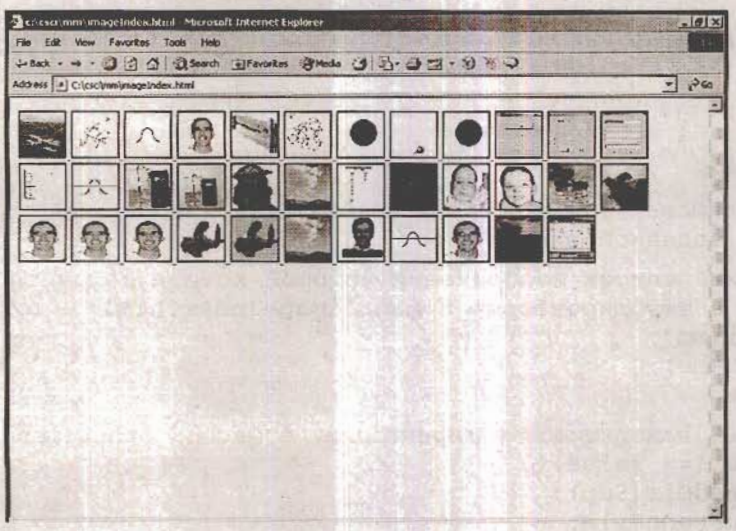


Рис. 6.8. Файл HTML был автоматически сгенерирован программой `imageIndex.php`

Создание дескриптора директории с помощью openDir()

Очевидно, что операциям работы с директориями необходима конкретная директория, с которой можно работать. Зачастую будет полезно хранить имя директории в переменной, чтобы его можно было легко изменить при переносе вашей программы в различные системы. В программе `imageIndex` я сохранил целевую директорию в переменной `$dirName`. Директория может быть сохранена в виде относительной ссылки (в этом случае она будет расположена относительно текущей директории программы) или абсолютного пути (в текущей файловой системе).

Получаем список файлов с помощью readdir()

Функция `readdir()` используется для считывания файла из корректного указателя на директорию. Каждый раз, когда вы вызываете функцию `readdir()`, она возвращает название следующего найденного файла, пока в директории не останется файлов. Если функция не может найти следующий файл, она возвратит значение `FALSE`. Я думаю, что очень полезно хранить названия всех файлов директории в массиве, поэтому я обычно использую следующий цикл.

```
while ($currentFile !== false){
  $currentFile = readDir($dp);
  $theFiles[] = $currentFile;
} // завершение цикла while
```

Этот цикл выполняется, пока значение переменной `$currentFile` не станет равным `FALSE`, что произойдет, когда в директории не останется файлов. На каждой итерации она использует функцию readdir() для загрузки нового значения в `$currentFile`, после чего добавляет это значение в массив $theFiles. Заметьте, что когда я присваиваю значение массиву, не указывая индекса, добавляемый элемент располагается по ближайшему доступному индексу. Это очень простой способ заполнения массива в PHP.

Специальный оператор !== немного отличается от оператора сравнения, виденного вами ранее. Он используется здесь для предотвращения появления очень специфичных типов ошибок. Возможна ситуация, когда в заданной пользователем директории находится файл с именем «false». Если это действительно так, обычное сравнение `$currentFile != false` *приведет к странным результатам, потому что PHP может спутать название файла «false» с буквенным значением* `false`. *Оператор !== определяет сравнение между самими объектами, а не их значениями, и поэтому нормально работает в этом нетривиальном случае.*

Выбираем заданные файлы с помощью preg_grep()

Как только все файлы директории были сохранены в массиве, зачастую вам нужно выбрать из них некоторый набор файлов, с которым предстоит работать. В данном случае меня интересуют графические файлы, которые имеют расширения «gif» или «jpg». Имеющая странное название функция `preg_grep()` замечательно подходит для данной ситуации. В ней реализованы некоторые интересные идеи из оболочки UNIX и языка программирования perl. Grep – это название команды UNIX, которая позволяет фильтровать файлы в соответствии с заданным шаблоном. Часть «preg» означает, что такая форма просмотра использует регулярные выражения языка perl. Несмотря на странное имя, эта функция очень полезна. Если вы посмотрите на код `imageIndex.php`, вы увидите следующую строчку.

```
$imageFiles = preg_grep("/jpg$|gif$/", $theFiles);
```

Этот код выбирает все файлы, которые имеют расширение «jpg» или «gif», и копирует их в другой массив, имеющий название `$imageFiles`.

Используем основные регулярные выражения

Хотя и возможно использовать функции работы со строками для определения файлов, которые нужно скопировать в новый массив, существует множество ситуаций, когда вам необходима более детальная работа со строками. В данной ситуации, я хотел найти все файлы, содержащие «gif» *или* «jpg». При использовании обычных функций работы со строками не существует простого способа сравнения с двумя возможными значениями. Кроме этого, мне нужно *не любое* название файла, содержащего эти строки, а только те названия файлов, которые *заканчиваются* на «gif» или «jpg». Регулярные выражения – это специальное соглашение, используемое как раз в таких ситуациях. В качестве примера я объясню, как работает регулярное выражение `"/jpg$|gif$/"`. Регулярные выражения обычно выделяют косой чертой в начале и конце. Первым и последним символом этого выражения является косая черта. Символ «|» означает *или*, поэтому я ищу «jpg» или «gif». В регулярных выражениях знак доллара ($) означает конец строки, поэтому «jpg$» будет совпадать с «jpg», только если он расположен в конце строки. В результате, выражение `"/jpg$|gif$/"` будет соответствовать любой строке, оканчивающейся на «jpg» или «gif».

Регулярные выражения являются очень мощными и немного загадочными. В дополнение к `preg_grep` PHP поддерживает несколько специальных функций, использующих регулярные выражения. Посмотрите в интерактивной помощи раздел «Regular Expression Functions – Perl compatible»[1] для получения списка этих функций и дополнительной информации о работе с регулярными выражениями в PHP. Если вас пугают регулярные выражения, вы можете найти функцию или две для работы со строками, которые выполняют ту же самую работу. (Список основных регулярных выражений смотрите в табл. 6.2.)

Храним вывод

Как только массив `$imageFiles` был завершен, программа использует эти данные для создания списка изображений HTML и сохраняет эти данные в файл. Так как этот код был приведен несколько страниц назад, я воспроизведу его часть.

```
foreach ($imageFiles as $currentFile){
  $output .= <<<HERE
<a href = $currentFile>
  <img src = "$currentFile"
```

[1] «Функции регулярных выражений – совместимость с Perl». – *Примеч. пер.*

```
        height = 50
        width = 50>
HERE;
} // завершение цикла foreach
// сохранить список в локальную файловую систему
$fp = fopen("imageIndex.html", "w");
fputs ($fp, $output);
fclose($fp);
print "<a href = $dirName/imageIndex.html>image index</a>\n";
```

Табл. 6.2. Основные операторы регулярных выражений

Оператор	Описание	Пример шаблона	Совпадение	Несовпадение
.	Любой символ за исключение перевода на новую строку	.	a	\n
^	Начало строки	^a	apple	banana
$	Конец строки	a$	banana	apple
[символы]	Любой символ из заключенных в скобки	[abcABC]	a	d
[диапазон символов]	Описывает диапазон символов	[a-zA-Z]	r	9
\d	Любая цифра	\d\d\d-\d\d\d\d	123-4567	The-thing
\b	Границы слова	\bthe\b	the	Theafter
+	Одно или более вхождение предыдущего символа	\d+	1234	Text
*	Ни одного или несколько вхождений предыдущего символа	[a-zA-Z]\d*	a1, a234	2, 555
{цифра}	Повторить предыдущий символ заданное количество раз	\d{3}-\d{4}	123-4567	999-99-9999

Табл. 6.2. Основные операторы регулярных выражений (продолжение)

Оператор	Описание	Пример шаблона	Совпадение	Несовпадение
|	Оператор или	apple|banana	apple, banana	peach
(фрагмент шаблона)	Сохраняет результат в шаблоне памяти, возвращенном численным кодом	(^.).*/1	gig, blab (любое другое слово начинающееся и заканчивающееся одной и той же буквой)	Любое другое слово

Я использовал цикл `foreach` для просмотра всех элементов массива `$imageFiles`. Я добавил HTML для генерации макета каждого изображения в переменную $output. После чего в текущей директории я открыл для записи файл `imageIndex.html`, поместил значение `$output` в этот файл и закрыл его. Наконец, я добавил ссылку на файл.

Возможно, вы захотели использовать команду `readFile()`, *чтобы сразу же просмотреть содержимое файла. (Лично я захотел.) Она может работать неправильно, потому что веб-браузер предполагает, что директория файла* `imageList.php` *является текущей. В программе я изменил текущую директорию, однако веб-браузер об этом не знает. Когда я выполнил* `readFile()`, *в HTML содержалось множество нарушенных ссылок, потому что все относительные ссылки на странице HTML указывали на файлы, расположенные в другой директории. Когда вместо этого я добавил ссылку на страницу, веб-браузер смог самостоятельно найти изображения, потому что он искал их в корректной директории.*

Работаем с форматированным текстом

Работать с текстовыми файлами очень просто, однако они чрезвычайно неупорядоченные. Возможно, иногда вам будет нужно немного отформатировать текстовый файл, чтобы работать с данными в нем. В следующих главах вы изучите несколько методов работы с данными, однако немного подумав, вы можете многое сделать и при помощи обычных файлов.

Введение в программу mailMerge.php

Чтобы показать, как текстовые файлы могут быть использованы для хранения данных, я создал простую программу добавления почты. Результат работы этой программы показан на рис. 6.9.

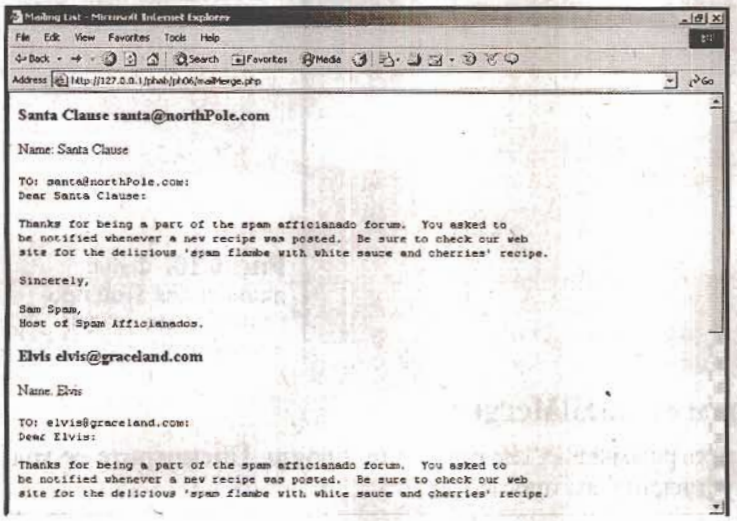


Рис. 6.9. Программа создала несколько форм писем по заданному списку имен и адресов электронной почты

Вы можете видеть, что одно и то же письмо было использовано много раз, однако каждый раз в нем использовались различные имена и адреса электронной почты. Имена и адреса электронной почты были получены из файла.

Определяем формат данных

Файл данных (показанный на рис. 6.10) для этой программы является обычным файлом, созданным в Блокноте (Notepad). Каждая строка состоит из имени и адреса электронной почты, разделенных символом табуляции.

Такой формат (для каждой записи используется одна строка, поля которой разделены табуляциями) называется файлом, элементы данных в котором разделены символом табуляции (tab-delimited file), и является очень популярным, потому что вы можете очень легко создать его в текстовом редакторе, таблице или в любой другой программе. Кроме этого, в качестве разделителя легко использовать любой другой символ. Программы таблиц обычно сохраняют данные в формате с разделяющими запятыми (файлы с расширением CSV – от англ. «comma-separated values»), однако использовать такой формат для строковых данных не удобно, потому что в строках могут содержаться запятые.

Рис. 6.10. Файл данных для этой программы был создан в Блокноте

Изучаем код программы mailMerge.php

Основная стратегия программы `mailMerge` очень проста. Посмотрите ее код, и, возможно, вы будете удивлены его простотой.

```
<html>
<head>
<title>Mailing List</title>
</head>
<body>
<form>
<?
// Простое добавление почты
// предполагается наличие файла maillist.dat, элементы которого
разделены
// символами табуляции
$theData = file("maillist.dat");
foreach($theData as $line){
  $line = rtrim($line);
print "<h3>$line</h3>";
  list($name, $email) = split("\t", $line);
  print "Name: $name";
  $message = <<<HERE
TO: $email:
```

```
Dear $name:
Thanks for being a part of the spam afficionado forum. You asked to
be notified whenever a new recipe was posted. Be sure to check our
web
site for the delicious 'spam flambe with white sauce and cherries'
recipe.
Sincerely,
Sam Spam,
Host of Spam Afficionados.
HERE;
  print "<pre>$message</pre>";
} // завершение цикла foreach
?>
</body>
</html>
```

Загружаем данные с помощью команды file()

Прежде всего необходимо загрузить данные в форму. Вместо того чтобы использовать указатель на файл, рассмотренный ранее в этой главе, я использовал очень привлекательный обходной путь. Команда `file()` принимает в качестве указателя имя файла и автоматически загружает его в массив. Каждая строка файла становится элементом массива. Это особенно полезно, когда ваши текстовые файлы содержат данные, потому что каждая строка моего файла данных представляет отдельную часть информации.

Команда `file()` *настолько проста, что, возможно, вы попробуете использовать ее всегда, однако она загружает весь файл в память, поэтому вы должны использовать ее только для относительно небольших файлов. При использовании* `fgets()` *вам необходимо хранить в памяти только одну строку файла, поэтому метод* `fgets()` *можно очень эффективно использовать для работы с файлами любого размера. Использование* `file()` *для очень больших файлов может сильно снизить производительность.*

Разделяем строки массива в скалярные значения

Возможно, вы помните функцию `split()` из главы 5 «Улучшенная обработка массивов и строк». Эта функция использовалась для разделения элементов строки на основе некоторого критерия. Я использовал функцию `split()` в теле цикла `foreach` для разбиения каждой строки на составляющие значения. Однако в данной ситуации

мне не нужен массив. Вместо этого я хочу, чтобы первый элемент строки был считан в переменную `$name`, а вторая переменная была сохранена в `$email`. Функция `list()` позволяет вам выделить элементы массива в скалярные (не массивы) переменные. В данном случае я не сохранял результаты выполнения функции `split()` в массиве, а сразу записывал содержимое в скалярные переменные. Как только все данные были записаны, они могут быть легко вставлены в сообщение добавления почты.

Возвращаясь к реальности

Следующим очевидным этапом этой программы будет автоматическая отправление каждого сообщения в виде электронного письма. PHP предоставляет функцию mail(), которая делает добавление такой функциональности очень простым. Однако работа этой функции сильно зависит от настроек сервера, и на каждом сервере она будет работать по-разному.

Существуют как плюсы, так и минусы в программном отправлении электронных писем. Совершенно оправдано послать электронное письмо человеку, который попросил это, или сделать программу, которая будет отсылать электронные письма, если в этом возникнет необходимость. Например, моя собственная, более безопасная версия программы тестирования посылает мне электронные письма при возникновении подозрения, что кто-то жульничает. Программа, которая самопроизвольно шлет письма людям, создаст плохое мнение о вашем сайте

Созданиеем программу машины тестирования

Инструмент тестирования, о котором говорилось в начале этой главы, на самом деле является целой системой взаимодействующих программ, в данном случае, пяти различных программ. Каждый тест хранится в двух различных файлах, которые автоматически генерируются программами. На рис. 6.11 показано, как различные программы связаны между собой.

Программа `QuizMachine.php` является точкой входа в систему, как для администратора теста, так и для человека, проходящего тестирование. Она состоит из трех форм, которые позволяют получить доступ к другим частям программы. Чтобы обеспечить хотя бы минимальный уровень безопасности, все программы системы требуют различного рода паролей для получения доступа к ним. Программа `QuizMachine` служит «шлюзом» для доступа к другим частям системы. Если у пользователя есть администраторский доступ (определяемый паролем), он или она может выбрать тест и вызвать страницу `editQuiz.php`. Эта страница загружает фактический основной файл теста (если он уже существует) или уста-

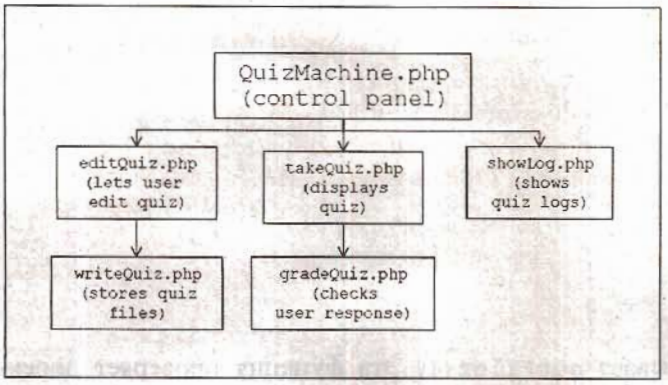


Рис. 6.11. Эта диаграмма иллюстрирует передвижения пользователя по системе машины тестирования

навливает прототип теста и помещает его данные на веб-страницу в виде простого редактора. Программа `editQuiz` вызывает программу `writeQuiz.php`, которая берет результаты формы `editQuiz` и записывает их в основной файл наряду с записью HTML страницы.

Если пользователь хочет выполнить тест, система переходит на страницу `takeQuiz.php`, которая проверяет пароль пользователя, и если проверка прошла успешно, отображает тест. После того как пользователь закончит выполнение теста, программа `gradeQuiz.php` оценивает результаты и сохраняет оценки в текстовом файле.

Наконец, администратор может просмотреть файлы журналов результатов для любого из тестов, указав желаемый тест на странице `QuizMachine`. Программа `showLog.php` отобразит соответствующий файл журнала.

Создаем страницу управления QuizMachine.php

Основной частью системы тестирования является страница `quizMachine.php`. Это единственная страница, на которую пользователь заходит непосредственно. Все остальные части системы вызываются из этой страницы или одной из других страниц, вызываемых из нее. Эта страница служит чем-то вроде панели управления. Она состоит из трех частей, соответствующих трем основным задачам, выполняемым системой: создание или редактирование тестов, прохождение тестирования и анализ результатов тестирования. В каждом из этих случаев пользователь будет знать тест, поэтому панель управления может автоматически предоставить список соответствующих файлов в каждом сегменте. Кроме того, каждая из этих задач требует пароля, что предоставляет хоть какую-то безопасность.

Основная часть программы `QuizMachine.php` просто содержит основные теги HTML и вызывает набор функций, которые и выполняют всю работу.

```
<html>
```

```
<head>
<title>Quiz Machine</title>
</head>
<body>
<center>
<h1>Quiz Machine</h1>
<?
getFiles();
showTest();
showEdit();
showLog();
```

Программа сначала вызывает `getFiles()`. Эта функция проверяет директорию и получает список содержащихся в ней файлов. Этот список названий файлов будет использован в других функциях. Следующие три функции генерируют HTML формы. Каждая из этих форм содержит список выбора, который динамически генерируется из списка файлов. Кнопка, соответствующая каждой форме, направляет форму соотвествующей PHP-странице системы.

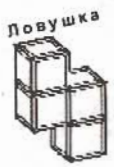
Возможно, вы захотите сделать другой вариант этой основной страницы для людей, проходящих ваши тесты. На этой станице вы не будете отображать административные настройки. Такую страницу очень просто сделать. Просто скопируйте программу `QuizBuilder.php` *в другой файл и закомментируйте вызовы функций* `showEdit()` *и* `showLog()`.

Получаем список файлов

Так как большая часть кода программы `QuizBuilder` работает со списком файлов, функция `getFiles()` предназначена для выполнения этой важной работы.

```
function getFiles(){
  //получить список всех файлов, используемых в других функциях
  global $dirPtr, $theFiles;
  chdir(".");
  $dirPtr = openDir(".");
  $currentFile = readDir($dirPtr);
  while ($currentFile !== false){
    $theFiles[] = $currentFile;
    $currentFile = readDir($dirPtr);
  } // завершение цикла while
} // завершение getFiles
```

Первым делом эта функция изменяет файловую систему, чтобы она указывала на текущую директорию, и устанавливает переменную-указатель в эту директорию.



Директория, в которой находятся программы PHP, открыта для всех. Возможно, вы захотите поместить ваши файлы тестов в другое место. Для упрощения этого примера я хранил все файлы тестов в той же директории, что и программы, однако вы можете хранить все файлы данных в отдельной директории. Из соображений безопасности вы можете хранить все файлы данных в директории, которая не доступна из Интернета (подальше от вашей структуры `public_html`*, например), чтобы люди не могли посмотреть ключ ответа. Если вы захотите сделать это, вам будет необходимо изменить все ссылки на директории во всей системе.*

После этого я создал массив `theFiles`, который содержит названия всех файлов в директории. Переменная `theFiles` является глобальной, поэтому она разделяется между программой и всеми функциями, которые объявляют ссылки на нее.

Отображаем список «Пройдите тестирование»

Большинство пользователей не будут создавать или редактировать тесты, они будут их выполнять. Чтобы выполнить тест, пользователь должен выбрать его и знать пароль для выполнения этого теста. Чтобы упростить выбор теста, функция `showTest()` берет все HTML файлы в директории с тестами и помещает их в список выбора. Пароль вводится в виде обычного поля пароля. Код функции `showTest()` создает форму, которая вызывает программу `takeQuiz.php` при отправке соответствующего запроса.

```
function showTest(){
  //печатает список тестов, которые может выполнить пользователь
  global $theFiles;
  print <<<HERE
<form action = "takeQuiz.php"
      method = "post">
<table border = 1
       width = 400>
<tr>
  <td colspan = 2><center>
    <h3>Take a quiz</h3>
  </td>
</tr>
<tr>
```

```
  <td>Quiz Password</td>
  <td>
    <input type = "password"
           name = "password">
  </td>
</tr>
<tr>
  <td>Quiz</td>
td>
    <select name = "takeFile">
HERE;
  //выбрать только html файлы тестов
  $testFiles = preg_grep("/html$/", $theFiles);
  foreach ($testFiles as $myFile){
    $fileBase = substr($myFile, 0, strlen($myFile) - 5);
    print "    <option value = $fileBase>$fileBase</option>\n";
  } // завершение цикла foreach
  print <<<HERE
    </select>
  </td>
</tr>
<tr>
  <td colspan = 2><center>
    <input type = "submit"
           value = "go">
  </center></td>
</tr>
</table>
</form>
HERE;
} // завершение showTest
```

Хотя код длинный, большая его часть является чистым HTML. Единственной частью, использующей PHP, является часть, выбирающая файлы HTML и помещающая их в группу выбора. Этот фрагмент кода использует `preg_grep()` для выбора файлов, оканчивающихся на «HTML», и создает тег option для таких файлов. Заметьте, что я выделил только часть названия `.html`, потому что на самом деле мне оно не нужно и будет только затруднять некоторые моменты кода программы `takeQuiz`, которую мы рассмотрим ниже.

Отображаем список редактирования

Функция `showEdit()` работает почти как `showTest()`. Эта функция используется для отображения списка основных файлов системы. Хотя зачастую он будет точно таким же, как и список тестов, это будет не всегда так, потому что могут быть основные файлы, которые еще не были переделаны в HTML.

```
function showEdit(){
  // позволяет пользователю выбрать основной файл для редактирования
  global $theFiles;
  //получить только основные файлы тестов
  $testFiles = preg_grep("/mas$/", $theFiles);
  //редактировать тест
  print <<<HERE
<form action = "editQuiz.php"
      method = "post">
<table border = 1
       width = 400>
<tr>
  <td colspan = 2><center>
    <h3>Edit a quiz</h3>
  </center></td>
</tr>
<tr>
  <td>Administrative Password</td>
  <td>
    <input type = "password"
           name = "password"
          value = "">
  </td>
</tr>
<tr>
  <td>Quiz</td>
  <td>
    <select name = "editFile">
      <option value = "new">new quiz</option>
HERE;
  foreach ($testFiles as $myFile){
    $fileBase = substr($myFile, 0, strlen($myFile) - 4);
    print "  <option value = $myFile>$fileBase</option>\n";
  } // завершение цикла foreach
  print <<<HERE
```

```
    </select>
  </td>
</tr>
<tr>
  <td colspan = 2><center>
    <input type = "submit"
           value = "go">
  </center></td>
</tr>
</table>
</form>
HERE;
} // завершение showEdit
```

Функция `showEdit()` очень похожа на `showQuiz()`, однако форма указывает на программу `editQuiz.php`, а список файлов содержит файлы, заканчивающиеся на «mas».

Есть еще одно небольшое, но важное отличие. Посмотрите на код элемента `select`, и вы увидите, как я добавляю опцию «new quiz» («новый тест»). Если пользователь выберет эту опцию, функция `editQuiz()` не будет пытаться загружать тест в память, а выполнит необходимые настройки для создания нового теста.

Отображаем список журнала

Последний фрагмент предназначен для администратора теста. Он позволяет пользователю с правами администратора получить доступ к журналу любого теста системы. Эта часть показывает кто, где и когда выполнил тест, а также результаты тестирования. Когда пользователь нажимает на кнопку `Submit`, ассоциированную с этой частью страницы, управление передается программе `showLog.php`.

```
function showLog(){
  //позволяет пользователю выбрать журнал из списка
  global $theFiles;
  print <<<HERE
<form action = "showLog.php"
      method = "post">
<table border = 1
       width = 400>
<tr>
  <td colspan = 2><center>
```

```
    <h3>Show a log file</h3>
  </td>
</tr>
<tr>
  <td>Administrative Password</td>
  <td>
    <input type = "password"
           name = "password"
value = "">
  </td>
</tr>
<tr>
  <td>Quiz</td>
  <td>
    <select name = "logFile">
HERE;
  //выбрать только файлы журналов
  $logFiles = preg_grep("/log$/", $theFiles);
  foreach ($logFiles as $myFile){
    $fileBase = substr($myFile, 0, strlen($myFile) - 4);
    print "      <option value = $myFile>$fileBase</option>\n";
  } // завершение цикла
  print <<<HERE
    </select>
  </td>
</tr>
<tr>
  <td colspan = 2><center>
    <input type = "submit"
           value = "go">
  </td>
</tr>
</table>
</form>
HERE;
} // end showLog
?>
</center>
</body>
</html>
```

Я решил, что все файлы журналов должны заканчиваться на `.log`, поэтому программа может с легкостью получить список файлов журналов, помещаемых в группу выбора.

Редактируем тест

В целях упрощения я решил использовать очень простой формат теста. Первые три строчки файла теста содержат название теста, адрес электронной почты инструктора и пароль для теста. После этого следуют данные теста. Каждый вопрос занимает одну строку (хотя его длина может меняться – длина строки определяется символом перехода на следующую строку). Вопрос будет содержать непосредственно вопрос, после которого следуют четыре возможных ответа и правильный ответ. Все эти элементы разделяются символом двоеточия (:).

Возвращаясь к реальности

Если вы скажете, что для форматирования вопросов используется слишком много правил, то я с вами соглашусь. Это ограничение, накладываемое технологиями последовательного доступа к файлам, используемым для хранения данных в этой главе. В следующих главах вы изучите другие способы работы с данными, которые не являются такими требовательными. Однако это достаточно простой способ хранения данных, поэтому я написал программу, которая облегчает вашу работу. В общем случае, вы хотите написать программу так, чтобы пользователь не знал ничего об используемых структурах данных

Программа `editQuiz.php` создана, чтобы помочь пользователю создавать и редактировать тесты. На самом деле она является очень простой программой, потому что основная работа выполняется после того, как пользователь завершает редактирование и нажимает кнопку `Submit`.

Получаем данные существующего теста

Прежде всего, необходимо определить тест, с которым работает пользователь. Помните, что значение «new» означает, что пользователь хочет создать новый тест, поэтому это значение обрабатывается отдельно. Все другие значения будут основываться на названии файла, поэтому я открываю соответствующий основной файл и загружаю его данные в соответствующие элементы формы.

```
<html>
<head>
<title>Quiz Builder</title>
</head>
```

```
<body>
<?
if ($password != "absolute"){
  print <<<HERE
<font color = "red" size = +3>
Invalid Password!
</font>
HERE;
} else {
  //проверить, выбрал ли пользователь форму для редактирования
  if ($editFile == "new"){
    //если это новый файл, поместить пустые значения
    $quizName = "sample test";
    $quizEmail = "root@localhost";
    $quizData = "q:a:b:c:d:correct";
    $quizPwd = "php";
  } else {
    //открыть файл и получить из него данные
    $fp = fopen($editFile, "r");
    $quizName = fgets($fp);
    $quizEmail = fgets($fp);
    $quizPwd = fgets($fp);
    while (!feof($fp)){
      $quizData .= fgets($fp);
    } // завершение цикла while
    fclose($fp);
} // завершение 'new form' if
```

Я решил задать значение «absolute» (из названия этой серии книг) в качестве административного пароля. У каждого теста будет собственный пароль и у каждой административной функции (как, например, редактирование теста) также будет собственный пароль. Если в поле пароля будет введено значение, отличное от выбранного пароля, программа сообщит об этом и не будет продвигаться дальше.

Такое использование административного пароля не исключает возможности его утечки вне системы, однако не существует абсолютных систем защиты. Такая система не подходит для случаев, когда вы должны быть уверены в абсолютной безопасности тестов.

Как только вы узнали, что пользователь ввел правильный пароль для редактирования тестов, вам необходимо определить, выбрал ли он новый тест или уже существующий. Если был выбран новый тест, я просто добавляю некоторые данные в переменные, которые используются следующей формой. Если пользователь хочет увидеть существующий тест, я открываю файл для чтения и считываю первые три строчки из него, которые соответствуют полям `$quizName`, `$quizEmail` и `$quizPwd`.

После этого я использую цикл `foreach` для загрузки оставшейся части файла в переменную `$quizData`.

Вам, наверное, интересно, зачем тесту нужно поле пароля, если пароль уже был введен при получении доступа к этой форме. Система тестирования содержит несколько уровней защиты. Любой может попасть на страницу `quizBuilder.php`. *Однако чтобы перейти на другую страницу, пользователь должен иметь соответствующий пароль. Однако только администратор должен использовать программы* `editPage` *и* `showLog`, *поэтому эти программы требуют дополнительного пароля администратора. Кроме того, с каждым тестом ассоциирован пароль. Этот пароль хранится в основном файле теста, чтобы вы могли ассоциировать различные пароли с различными тестами. Таким образом, пользователь, имеющий доступ к одному тесту, не сможет выполнять остальные (и запутывать тем самым файлы журналов).*

Печатаем форму

Как только переменные заполнены соответствующими значениями, то для того чтобы позволить пользователю редактировать тест, необходимо просто напечатать HTML-форму. Эта форма почти полностью состоит из HTML, в котором на соответствующих местах расположены переменные теста.

```
print <<<HERE
<form action = "writeQuiz.php"
method = "post">

<table border = 1>
<tr>
  <th>Quiz Name</th>
  <td>
    <input type = "text"
           name = "quizName"
           value = "$quizName">
  </td>
</tr>
```

```
<tr>
  <th>Instructor email</th>
  <td>
    <input type = "text"
           name = "quizEmail"
           value = "$quizEmail">
  </td>
</tr>

<tr>
  <th>Password</th>
  <td>
    <input type = "text"
           name = "quizPwd"
           value = "$quizPwd">

<tr>
  <td rowspan = 1
      colspan = 2>
    <textarea name = "quizData"
              rows = 20
              cols = 60>
$quizData</textarea>
  </td>
</tr>

<tr>
  <td colspan = 2><center>
    <input type = "submit"
           value = "make the quiz">
  </center></td>
</tr>

</table>
</form>
HERE;
} // завершение if

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

Сохраняем тест

Как только администратор закончит редактирование файла теста, этот файл должен быть сохранен в файловой системе, а также должна быть сгенерирована HTML страница для этого теста. Для этой цели предназначена программа `writeQuiz.php`.

Настраиваем основную логику

Прежде всего, необходимо создать два файла. Название теста может быть основой для имени файла, однако во многих файловых системах не принято использовать пробелы в названиях файлов, поэтому я использую функцию `str_replace()` для замены всех пробелов в `$quizName` на символ подчеркивания (_). После этого я создаю имя основного файла, заканчивающегося на .mas, и имя файла для самого теста, оканчивающееся на .html. Чтобы создать файл HTML, я открываю его для записи. После этого я использую функцию `buildHTML()` (позже она будет кратко описана) для создания кода HTML, который я записываю в файл html, и закрываю его. Основной файл создается практически так же, за исключением того, что он вызывает функцию `buildMas()` для создания соответствующего текста.

```
<html>
<head>
<title>Write Quiz</title>
</head>
<body>
<?
//по файлу, полученному из editQuiz,
//генерирует основной файл и файл HTML для теста

//открыть выходной файл
$fileBase = str_replace(" ", "_", $quizName);
$htmlFile = $fileBase . ".html";
$masFile = $fileBase . ".mas";

$htfp = fopen($htmlFile, "w");
$htData = buildHTML();
fputs($htfp, $htData);
fclose($htfp);

$msfp = fopen($masFile, "w");
$msData = buildMas();
fputs($msfp, $msData);
fclose($msfp);

//сделать предварительный просмотр основного файла
print <<<HERE
```

```
<pre>
$msData
</pre>
HERE;
```

Чтобы убедиться, что все было сделано успешно, я добавил проверку в конец страницы, которая печатает содержимое основного файла. Сгенерированные этой программой данные позволяют администратору узнать, правильно ли работает тест. На самом деле, администратор может выполнить тест и передать его программе оценки из этой страницы. Если в тесте были ошибки, очень полезно видеть содержимое файла .mas на странице. Конечно же, сама страница HTML не будет содержать этих данных, потому что на ней отображаются ответы.

Создаем основной файл

Функция создания основного файла очень проста.

```
function buildMas(){
  //создание основного файла
  global $quizName, $quizEmail, $quizPwd, $quizData;
  $msData = $quizName . "\n";
  $msData .= $quizEmail . "\n";
  $msData .= $quizPwd . "\n";
  $msData .=  $quizData;
  return $msData;
} // завершение buildMas
```

Самым важным является помнить правила оформление структуры этого файла, чтобы любая программа, использующая его, работала корректно. Название теста располагается в первой строке, и после него идет символ перевода на новую строку. Далее располагаются переменные `$quizEmail` и `$quizPwd`, каждая на своей строке, и, наконец, далее следует `$quizData` (обычно состоящая из нескольких строк), которой и заканчивается файл. Заметьте, что функция не сохраняет данные в файл, а возвращает их программе. Это предоставляет большую гибкость, поэтому я могу записывать данные как в файл, так и на страницу.

Создаем HTML файл

Функция, создающая HTML, требует большей работы, но все равно достаточно проста. Вот алгоритм ее функционирования: создать форму HTML, содержащую все вопросы. Для каждой строки основного файла создать группу переключателей. Поместить вопрос и все возможные ответы в набор вложенных элементов `<ol>`. В конце страницы должна располагаться кнопка `Submit`. Когда пользователь нажмет на нее, система вызовет страницу `gradeQuiz.php`, которая оценит ответы пользователя.

```
function buildHTML(){
  global $quizName, $quizData;
  $htData = <<<HERE
<html>
<head>
<title>$quizName</title>
</head>
<body>
HERE;
  //получить данные теста
  $problems = split("\n", $quizData);
  $htData .= <<<HERE
<center>
<h1>$quizName</h1>
</center>
<form action = "gradeQuiz.php"
      method = "post">
Name
<input type = "text"
       name = "student">
<ol>
HERE;
  $questionNumber = 1;
  foreach ($problems as $currentProblem){
       list($question, $answerA, $answerB, $answerC, $answerD,
$correct) =
      split (":", $currentProblem);
      $htData .= <<<HERE
<li>
  $question
  <ol type = "A">
    <li>
      <input type = "radio"
             name = "quest[$questionNumber]"
             value = "A">
      $answerA
    </li>
    <li>
      <input type = "radio"
             name = "quest[$questionNumber]"
```

```
                value = "B">
        $answerB
      </li>
      <li>
        <input type = "radio"
               name = "quest[$questionNumber]"
               value = "C">
        $answerC
      </li>
      <li>
        <input type = "radio"
               name = "quest[$questionNumber]"
               value = "D">
        $answerD
    </li>
    </ol>

</li>
HERE;
    $questionNumber++;
  } // завершение цикла foreach
  $htData .= <<<HERE
</ol>
<input type = "hidden"
       name = "quizName"
       value = "$quizName">

<input type = "submit"
       value = "submit quiz">

</form>
HERE;
  print $htData;
  return $htData;
} // завершение buildHTML
?>
</body>
</html>
```

Большая часть важной информации, необходимой этой функции, хранится в $quizData. Каждая строка $quizData содержит один вопрос, поэтому я использую функцию split() для разделения $quizData в массив $problems. Я использую цикл foreach для просмотра каждого вопроса. Каждый вопрос содержит список значений, которые разделяются на набор скалярных переменных при помощи вызовов split() и list().

В цикле foreach я также добавил код HTML, необходимый для печати информации текущего вопроса. Внимательно посмотрите на код для группы переключателей. Вспомните, что группы кнопок, которые взаимодействуют, как одно целое, должны иметь одно и тоже название. Я выполнил это требование, назвав их quest[$questionNumber]. Переменная $questionNumber будет содержать номер текущего вопроса, и это значение будет изменяться до записи HTML кода. У первого вопроса будет четыре различных кнопки, называемых quest[1]. Программа gradeQuiz будет видеть это как массив $quest.

В конце HTML я добавил название теста в качестве скрытого поля и кнопку Submit.

Проходим тестирование

Целью всей этой работы является получение набора тестов, которые пользователь может выполнять, поэтому будет полезно иметь программу, отображающую тесты. На самом деле, так как тесты сохранены в виде страниц HTML, вы могли бы просто предоставить ссылку на тест и довольствоваться этим, однако я хотел обеспечить больший уровень безопасности. Я хотел получить возможность хранить файлы теста вне обычного файлового пространства public_html, а также хотел использовать простейшую систему паролей, чтобы люди не могли проходить тесты, пока я не разрешу им этого. (Я не сообщу им пароль, пока не буду готов, чтобы люди выполняли тесты). Кроме этого, я могу очень просто «выключить» тест, поменяв пароль.

Единственной задачей страницы takeQuiz является сравнение пароля, введенного пользователем, с паролем выбранного теста и разрешение прохождения теста, если пароли совпали.

```
<?
//takeQuiz.php
//по заданному файла теста печатает сам тест

//получить пароль из файла
$masterFile = $takeFile . ".mas";
$fp = fopen($masterFile, "r");
//пароль хранится в третьей строке, поэтому считываем первые две
строчки
```

```
//и игнорируем их
$dummy = fgets($fp);
$dummy = fgets($fp);
$magicWord = fgets($fp);
$magicWord = rtrim($magicWord);
fclose($fp);

if ($password == $magicWord){
  $htmlFile = $takeFile . ".html";
    //напечатать страницу, если пользователь ввел правильный
пароль
  readFile($htmlFile);
} else {
  print <<<HERE
  <font color = "red"
        size = +3>
Incorrect Password.<br>
You must have a password in order to take this quiz
</font>

HERE;
} // завершение if
?>
```

Пароль, ассоциированный с тестом, хранится в файле теста, поэтому как только я узнаю, какой тест хочет пройти пользователь, я открываю этот файл и извлекаю из него пароль. Пароль хранится в третьей строке файла, и единственным способом его получения из файла с последовательным доступом является считывание первых двух строк во временную переменную, а третья строка, содержащая пароль, считывается в переменную `$magicWord`. Если пользователь указал пароль, совпадающий с `$magicWord`, я использую функцию `readFile()` для отправки содержимого HTML страницы теста браузеру. Если же указанный пароль не верен, я посылаю сообщение об этом.

Это может быть очень привлекательным местом для добавления дополнительной защиты. В готовой версии системы я веду файл журнала всех доступов, поэтому я буду знать, если кто-то попробует войти в мою систему с одной и той же машины 1000 раз за секунду (верный знак автоматической атаки) или о любой другой странности. Я также могу проверить, была ли страница отправлена с той же машины, которая затребовала ее первый раз. Если я захочу, я могу также проверить время запроса и время отправления, чтобы не принимать тесты, выполнявшиеся больше определенного промежутка времени.

Оцениваем тест

Одним из преимуществ такого типа системы является возможность немедленного предоставления результата пользователю. Как только пользователь нажмет кнопку Submit, тест будет автоматически оценен программой gradeQuiz.php, которая также сохраняет результаты студента в журнале, который может затем просмотреть администратор.

Открытие файлов

Программа gradeQuiz, как и все программы системы, полагается на файлы при работе. В данном случае, программа будет использовать основной файл для получения ключа ответов к тесту, а также записывать результаты в файл журнала.

```
<?
print <<<HERE
<html>
<head>
<title>Grade for $quizName, $student</title>
</head>
<body>
</body>
<h1>Grade for $quizName, $student</h1>
HERE;
//открыть соответствующий основной файл для чтения
$fileBase = str_replace(" ", "_", $quizName);
$masFile = $fileBase . ".mas";
$msfp = fopen($masFile, "r");
$logFile = $fileBase . ".log";
//первые три строки содержат название, адрес электронной почты
//инструктора и пароль
$quizName = fgets($msfp);
$quizEmail = fgets($msfp);
$quizPwd = fgets($msfp);
```

Основной файл открыт для чтения. Первые три строки нам не важны, но их необходимо считать для получения данных теста.

Создаем ключ ответов

Я начну с генерации ключа ответов из основного файла. Я просмотрю каждый вопрос в файле и извлеку все нормальные переменные из них (хотя мне на самом деле нужна только переменная $correct). После этого я сохраняю значение $correct в массиве $key. В конце цикла массив $key будет содержать правильные ответы для каждого вопроса теста.

```
//просмотреть вопросы, создавая при этом ключ ответов
$numCorrect = 0;
$questionNumber = 1;
while (!feof($msfp)){
  $currentProblem = fgets($msfp);

  list($question, $answerA, $answerB, $answerC, $answerD, $cor-
rect) =
  split (":", $currentProblem);
  $key[$questionNumber] = $correct;
  $questionNumber++;
} // завершение цикла while
fclose($msfp);
```

Проверяем ответы пользователя

Ответы пользователя поступят из HTML формы в массиве `$quest`. Правильные ответы располагаются в массиве `$key`. Чтобы оценить тест, я могу просто одновременно просмотреть оба массива, сравнивая при этом ответ пользователя с правильным. Каждый раз, когда их значения совпадут, пользователь дал правильный ответ. Если же значения разные, пользователь ответил неправильно (или была ошибка в самом тесте, не забывайте об этом).

```
//проверить каждый ответ пользователя
for ($questionNumber = 1; $questionNumber <= count($quest);
$questionNumber++){
  $guess = $quest[$questionNumber];
  $correct = $key[$questionNumber];
  $correct = rtrim($correct);
  if ($guess == $correct){
    //пользователь ответил правильно
    $numCorrect++;
    print "problem # $questionNumber was correct<br>\n";
  } else {
   print "<font color = red>problem # $questionNumber was incor-
rect</font><br>\n";
  } // завершение if
} // завершение цикла for
```

Я сделал так, чтобы пользователь знал, на какие вопросы он ответил правильно, однако решил не отображать правильный ответ. Возможно, вы захотите предоставить пользователю больше или меньше информации, в зависимости от того, как вы используете программу тестирования.

Отображаем результаты на экране и в файле журнала

После проверки каждого ответа программа сообщает о результатах пользователю в виде обычного счета и процента правильных ответов. Программа также открывает для добавления файл журнала и дописывает в него текущие данные. Доступ для добавления очень похож на доступ для записи, однако вместо перезаписи исходного файла такой тип доступа позволяет дописывать данные в конец файла.

```
print "you got $numCorrect right<br>\n";
$percentage = ($numCorrect /count($quest)) * 100;
print "for $percentage percent<br>\n";

$today = date ("F j, Y, g:i a");
//напечатать "Date: $today<br>\n";
$location = getenv("REMOTE_ADDR");
//напечатать "Location: $location<br>\n";

//добавить результаты в файл журнала
$lgfp = fopen($logFile, "a");
$logLine = $student . "\t";
$logLine .= $today . "\t";
$logLine .= $location . "\t";
$logLine .= $numCorrect . "\t";
$logLine .= $percentage . "\n";

fputs($lgfp, $logLine);
fclose($lgfp);

?>

</html>
```

Я добавил еще несколько элементов в файл журнала, которые могут быть полезны администратору теста. Конечно же, я добавил имя студента и текущую дату. Возможно, вы захотите посмотреть интерактивную помощь для функции `date()`, чтобы увидеть различные способы отображения текущей даты. Я также добавил переменную расположения, которая использует переменную среды `$REMOTE_ADDR` для определения того, на какой машине работал пользователь, когда он или она отправлял результат теста. Это очень важная информация, потому что она может предупредить вас о некоторых случаях обмана. (Например, человек, прошедший один и тот же тест на одной и той же машине, но под разными именами.) Программа `gradeQuiz` добавляет число правильных ответов и их процент от общего числа вопросов в файл журнала, после чего закрывает его. Заметьте, что данные в файле журнала разделены символами табуляции. Это сделано для того, чтобы программе анализа было проще работать с этим файлом посредством команды `split`. Кроме того, большинство программ таблиц могут без

труда считывать файлы, информация в которых разделена символами табуляции, поэтому файл журнала может быть легко импортирован в таблицу для дальнейшего анализа.

Вы можете улучшить функциональность записи в журнал, если вы хотите проводить более глубокие анализы тестов. Например, вы могли бы хранить ответ каждого пользователя на каждый вопрос теста. Это позволило бы вам получить базу данных ответов на каждый вопрос, с помощью которой вы могли бы легко определить, какие вопросы вызывают затруднения.

Просмотр журнала

Программа `showLog.php` на самом деле очень похожа на программу `takeQuiz`. Она проверяет пароль, чтобы убедиться, что у пользователя есть права администратора, после чего открывает журнал при помощи функции `file()`. Она печатает результаты, хранящиеся в файле, заключая их в пару `<pre></pre>`, поэтому символы табуляции сохранятся.

```
<?
//showLog.php
//отображает файл журнала
//требует пароля администратора
if ($password == "absolute"){
  $lines = file($logFile);
  print "<pre>\n";
  foreach ($lines as $theLine){
    print $theLine;
  } // завершение цикла foreach
  print "</pre>\n";
} else {
  print <<<HERE
<font color = "red"
      size = +2>
You must have the appropriate password to view this log
</font>
HERE;
} // завершение if
?>
```

Вы можете улучшить эту программу, записав данные в HTML таблицу. Однако не все таблицы могут работать с данными HTML-таблицы, поэтому я предпочитаю использовать формат с табуляцией. В программу просмотра журнала будет не очень сложно добавить дополнительный анализ данных, включая среднюю успеваемость, стандартное отклонение и приблизительный график успеваемости.

Итоги

В этой главе было рассмотрено использование последовательных файлов в качестве механизма хранения и получения информации. Вы научились открывать файлы для чтения, записи и добавления данных. Вы узнали, как указатели на файлы используются для адресации файлов. Вы записали данные в файлы и загрузили их из него с помощью соответствующих функций. Вы узнали, как загрузить целый файл в массив. Вы можете изучить директорию и определить находящиеся в ней файлы. Вы узнали, как использовать основные регулярные выражения в функции `preg_greq()` для отображения набора файлов директории. Наконец, вы объединили все это в многопрограммной системе, имеющей несколько уровней доступа к интересующему набору данных. Вы должны гордиться своими успехами.

Домашнее задание

1. Улучшите программу тестирования одним из способов, которые я предлагал в этой главе. Добавьте возможность отправки результатов тестирования по электронной почте, анализ результатов тестирования, улучшите страницу редактирования тестов или добавьте что-нибудь свое.
2. Несколько значений этой системы должны быть глобальными для каждой из PHP программ. Наиболее очевидными из них являются корневая директория и административный пароль. Напишите программу, которая сохраняет эти значения в файле .ini, и измените программу тестирования, чтобы эти значения брались из файла.
3. Создайте просмотрщик исходного кода. Программа должна по заданному имени файла считывать его и преобразовывать все экземпляры < в `<`, после чего сохранять этот новый файл с другим именем. Это позволит вам показывать свой исходный код другим людям.
4. Создайте простую гостевую книгу. Позвольте пользователю вводить информацию в форму, и когда он или она нажмет на кнопку `Submit`, добавьте его или ее комментарий в конец страницы. Для этого вы можете использовать один или два файла.

Глава 7

Используем MySQL для создания баз данных

Вы начали программировать на PHP с применения простых переменных. Вскоре вы научились создавать более интересные вещи с помощью массивов и ассоциативных массивов. Вы использовали мощь файлов для получения новых невероятных возможностей. Теперь вы увидите, как можно использовать реляционные базы данных для управления информацией. В этой главе вы узнаете, как создать простую базу данных и использовать ее в своих PHP программах. В частности, вы узнаете:

- как запустить выполняемый файл MySQL;
- как создать простую базу данных;
- важные операторы определения данных SQL;
- как возвратить простой SQL запрос;
- как использовать SQLyog для управления базами данных;
- как использовать базы данных в ваших PHP программах.

Представляем программу создания приключений

Базы данных являются серьезным инструментом, однако они могут быть и забавными. Программа генерации приключений, изображенная на рис. 7.1–7.4, иллюстрирует, как база данных может быть использована в генераторе игровых приключений. Генератор приключений является системой, позволяющей пользователям создавать и затем участвовать в простых «многовыборных» приключениях. Такой тип игры состоит из нескольких узлов. Каждый узел описывает определенного рода решение. Всякий раз у пользователя будет возможность сделать выбор из нескольких (до трех) вариантов. В результате этого выбора принимается новое решение. Если пользователь сделает последовательность правильных выборов, то он или она выигрывает игру.

Это программа интересна в качестве игры, однако действительно захватывающим является то, как пользователь может ее изменять. Пользователь может использовать одну и ту же систему для создания и редактирования приключений. На рис. 7.3 показаны данные, используемые в головоломке. Заметьте, что вы можете редактировать любой узел игры, щелкнув по соответствующей кнопке.

Если пользователь решит редактировать фрагмент, появится станица, изображенная на рис. 7.4.

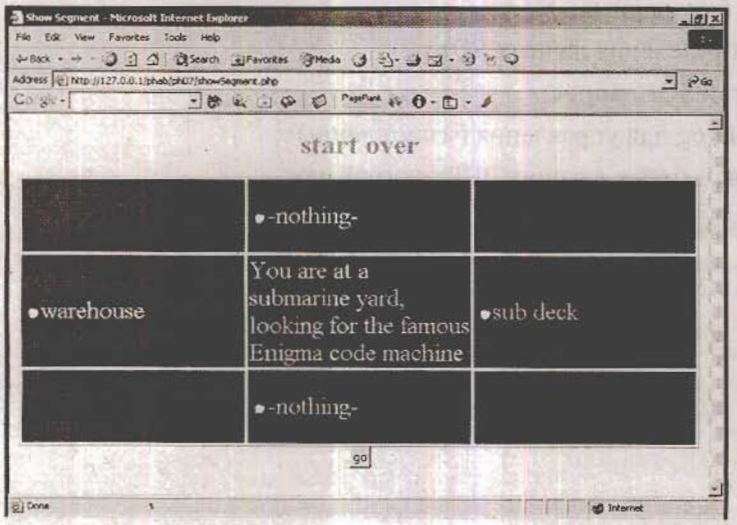


Рис. 7.1. Пользователь может выбрать один из вариантов. Давайте попробуем выбрать это…

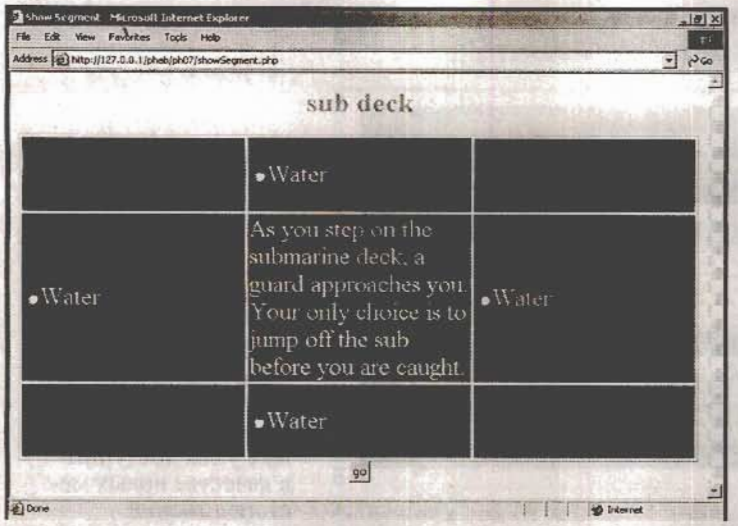


Рис. 7.2. В конце концов, склад может быть наиболее удачным выбором

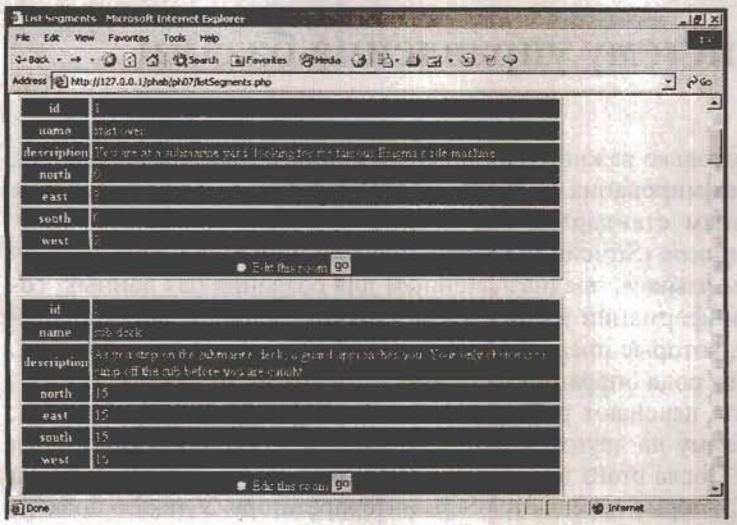


Рис. 7.3. Эта страница предоставляет информацию о каждом фрагменте игры, включая ссылки для непосредственного редактирования фрагмента

Как вы можете видеть, структура данных является наиболее важным элементом этой игры. Вы уже знаете некоторые способы работы с данными, однако в этой главе вводится понятие *систем управления реляционными базами данных* (СУРБД). СУРБД – это система, которая помогает программистам работать с данными. Программа генератора приключений использует базу данных для хранения и управления данными.

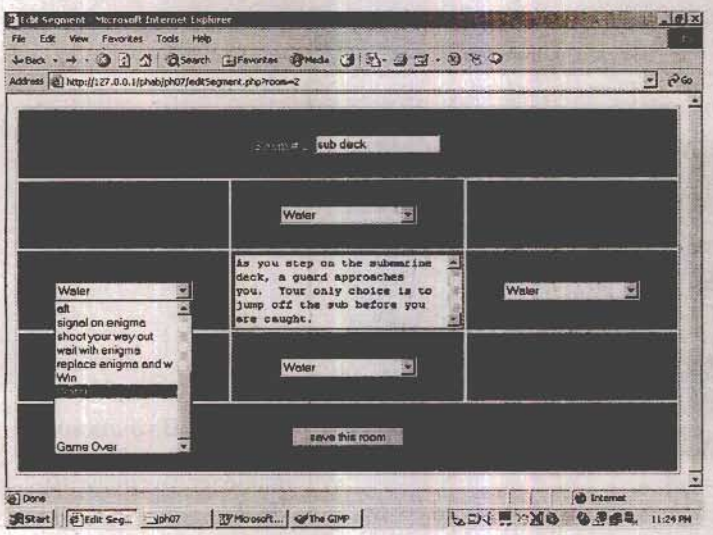


Рис. 7.4. На этой странице можно изменять все характеристики узла. Все уже созданные узлы доступны в качестве новых местоположений

Используем систему управления базами данных

Данные являются настолько важной частью современного программирования, что целые языки программирования полностью предназначены для управления базами данных. Основным стандартом для языков баз данных является *язык структурированных запросов* (Structured Query Language, или SQL). SQL является стандартизированным языком, предназначенным для создания баз данных, сохранения и получения информации из них. Существуют специальные приложения и среды разработки, которые предназначены для интерпретации данных SQL и выполнения различного рода операций над ними.

Обычно программисты начинают работу с создания структуры данных в SQL, после чего пишут программу на другом языке (как, например, PHP) для получения доступа к этим данным. После этого программа PHP может сформировать запросы или обновления данных, которые передаются SQL интерпретатору. У такого подхода есть несколько преимуществ. Во-первых, как только вы изучите SQL, вы сможете легко использовать его в новом языке программирования. Кроме этого, вы можете очень просто добавить множество интерфейсов для существующих данных, потому что многие языки программирования предоставляют средства доступа к SQL интерпретатору. Существует множество систем управления реляционными базами данных, однако среда MySQL лучше всего приспособлена для использования совместно с PHP. Однако основные концепции SQL остаются одними и теми же независимо

от используемого типа базы данных. Большинство SQL команд, описанных в этой главе, работают без их изменения в Microsoft Access, Microsoft SQL Server, Oracle, а также во многих других пакетах СУРБД.

Я начну эту главу с объяснения того, как можно создать простую базу данных в MySQL. Существует несколько вариантов работы с этим пакетом, однако я начну с объяснения того, как написать скрипт, создающий базу данных в текстовом файле. Я буду использовать язык SQL, который по синтаксису и стилю отличается от PHP. После этого я покажу вам SQLyog – замечательный интерфейс для работы с базами данных MySQL. В главе 8 «Соединение с базами данных при помощи PHP» я покажу вам, как связаться и управлять вашими базами данных MySQL из PHP.

Работаем с MySQL

Существует множество пакетов СУРБД. Эти программы обладают различными возможностями, гибкостью и ценой. Однако все они работают приблизительно одинаково. Большинство примеров этой книги используют базы данных MySQL. Эта программа очень часто используется вместе с PHP по нескольким причинам. Во-первых, MySQL обладает очень широкими возможностями. Она обладает большинством возможностей, предоставляемых в дорогих и мощных пакетах для работы с базами данных. Базы данных MySQL используют стандартную форму широко известного языка SQL. MySQL выпускается с открытым кодом и распространяется бесплатно, работает с большим количеством операционных систем и может использоваться со многими языками программирования. Она работает очень быстро даже с большими наборами данных. Кроме того, в PHP встроено множество функций для поддержи баз данных MySQL.

Инсталлируем MySQL

Если вы использовали PHPTriad для инсталляции Apache и PHP, то, скорее всего, вы установили и MySQL. Если вы работаете не со своим веб-сервером, вам необходимо будет спросить администратора сервера, установлена ли на него MySQL. Если ваш сервер поддерживает PHP, то скорее всего он поддерживает и MySQL, так как эти две программы зачастую устанавливаются вместе.

Используем выполняемый файл MySQL

MySQL на самом деле состоит из нескольких программ. Этот пакет программ содержит компонент сервера, который всегда запущен, и несколько вспомогательных программ. Консоль командной строки MySQL, изображенная на рис. 7.5, является основной программой, запускаемой из командной строки. Работать с ней не очень удобно, однако она предоставляет огромные возможности для работы с движком баз данных.

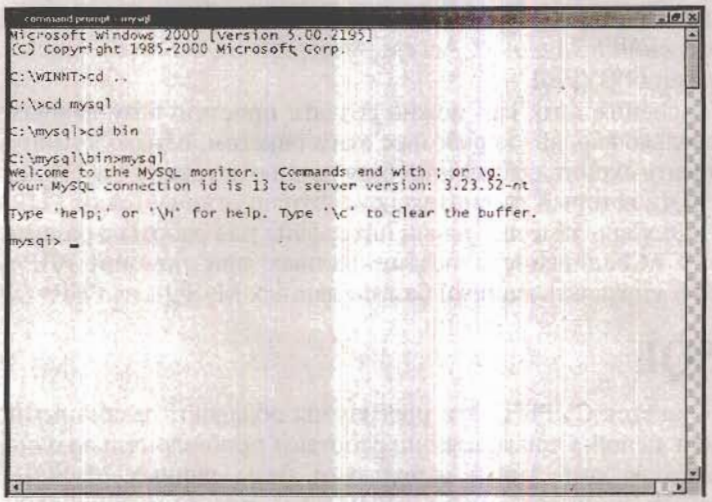


Рис. 7.5. Программа MySQL, соединяющаяся с базой данных

Существует несколько способов использования MySQL, однако типичная процедура работы состоит в следующем: соединение с сервером MySQL, выбор базы данных и использования языка SQL для управления ею, а именно создания таблиц, просмотра данных и т. д.

Консоль MySQL.exe, поставляемая с MySQL, является наиболее распространенным инструментом для работы с базами данных MySQL. Хотя эта программа и не блещет пользовательским интерфейсом, она предоставляет низкоуровневый доступ к базам данных. Однако ее интерфейс необходимо изучить, потому что он очень похож на то, как программы будут взаимодействовать с базой данных.

Создаем базу данных

Базы данных описываются с помощью очень своеобразной схемы организации. Чтобы проиллюстрировать основные понятия баз данных, я создам и просмотрю простой список телефонов. Основная структура такого телефонного списка показана в табл. 7.1.

Табл. 7.1. Список телефонов[a]

id	firstName	LastName	e-mail	phone
0	Andy	Harris	aharris@cs.iupui.edu	123-4567
1	Joe	Slow	jslow@myPlace.net	987-6543

a. id – идентификатор, firstName – имя, lastName – фамилия, e-mail – адрес электронной почты, phone – телефон

Список телефонов представляет собой типичную простую таблицу данных. Люди, занимающиеся базами данных, любят давать специальные названия частям таблицы, поэтому я использую этот простой список телефонов для их иллюстрации. Каждая строка таблицы называется *записью*. Записи описывают дискретные сущности. Список записей называется *таблицей*. Все записи таблицы содержат одинаковые элементы, называемые *полями* (или иногда просто *столбцами*). У всех записей таблицы одинаковое определение полей, однако записи могут содержать различные значения в этих полях. Поля таблицы определяются по-особому. Из-за способа хранения таблиц базы данных в файле, компьютер должен точно знать, сколько места необходимо выделить для каждого поля, поэтому размер и тип каждого поля очень важен. В этой конкретной базе данных определено пять полей. Поле `id` является целочисленным. Все остальные поля содержат строковые данные.

Создаем таблицу

Программы СУРБД используют специальный язык, называемый языком *структурированных запросов* (SQL) для создания и управления базами данных. В сравнение с полнофункциональными языками программирования, SQL достаточно легок для понимания. Зачастую вы можете догадаться о происходящих событиях, даже не имея соответствующих знаний. В качестве примера посмотрите на приведенный ниже код SQL.

```
USE chapter7;

CREATE TABLE phoneList (
  id INT PRIMARY KEY,
  firstName VARCHAR(15),
  lastName VARCHAR (15),
  email VARCHAR(20),
  phone VARCHAR(15)
);

DESCRIBE phonelist;
```

Этот код являет SQL скриптом. Он похож на программу PHP в том смысле, что является набором инструкций, которые выполняет компьютер. Однако интерпретатор PHP не взаимодействует напрямую с языком SQL. Вместо этого эти команды посылаются другой программе. Являясь программистом PHP, вы также сможете создавать код, который будет отсылать команды языку баз данных. Так же, как ваш PHP-код создает данные в HTML формате, интерпретируемом броузером, вы будете создавать код SQL, используемый интерпретатором MySQL.

Когда этот код отсылается в программу, поддерживающую базы данных SQL (как, например, MySQL), он создаст базу данных, чья структура показана в табл. 7.1.

Преимущества SQL

С самого начала базы данных были важной частью программирования, однако процесс работы с данными претерпел несколько изменений. Появление общего языка, который можно использовать во многих приложениях, был очень важным шагом. SQL является языком *четвертого поколения*. В общем случае эти языки были разработаны для решения проблем определенного типа. Некоторые языки четвертого поколения (такие, как SQL) не являются полнофункциональными языками программирования, потому что не поддерживают такие структуры управления, как ветвления и циклы. Однако эти языки выполняют очень важную работу. SQL является очень привлекательным в силу его широкой поддержки. Команды SQL, которые вы изучите в этой главе, могут использоваться в большинстве современных программах управления базами данных практически без изменений. Вы можете взять скрипт из MySQL и использовать его в базах данных Oracle или MS SQL Server (два наиболее распространенных формата), и во всех трех случаях программы создадут одинаковые базы данных. Если вы захотите использовать более мощный пакет работы с данными, вы сможете использовать уже существующий код для работы с данными. Наверное, самой главной причиной использования скриптового языка для управления базами данных является программирование в традиционных языках. Вы можете написать программу, генерирующую код SQL, на любом языке (например, на PHP). После этого вы можете использовать этот код для управления базой данных. Это позволяет вам получить невероятную гибкость и использовать ваши программы в качестве интерфейса к базе данных

Используем базы данных

Существует возможность, что у вас будет несколько проектов баз данных, функционирующих в одной и той же СУРБД. В моем случае у каждой главы этой книги, использующей SQL, имеется собственная база данных. Иногда ваш системный администратор может предоставить базу данных целиком в ваше распоряжение. В любом случае, вам, наверное, придется активизировать эту базу данных с помощью команды USE.

Синтаксис SQL не совсем совпадает с синтаксисом PHP. У SQL есть собственная культура, и необходимо уважать способы написания кода SQL, сложившиеся исторически. В общем случае SQL не чувствителен к регистру букв, однако большинство программистов SQL пишут команды SQL заглавными буквами. Кроме этого, после каждого набора команд, помещаемого в файл, как будет в этом коде, необходимо поместить точку с запятой.

Если у вас еще нет базы данных, которую можно выбрать с помощью USE, *вы можете создать ее с помощью команды* CREATE. *Например, чтобы создать базу данных, имеющую название "myStuff", используйте следующие команды.*

```
CREATE DATABASE mystuff;
USE mystuff;
```

Создаем таблицу

Чтобы создать таблицу, вам необходимо указать ее название, а также название всех полей таблицы. Для каждого поля вы должны указать тип данных, содержащийся в нем, и (по крайне мере для текстовых данных) задать максимальное количество символов, содержащихся в поле. В качестве примера следующий код иллюстрирует создание таблицы `phoneList`.

```
CREATE TABLE phoneList (
  id INT PRIMARY KEY,
  firstName VARCHAR(15),
  lastName VARCHAR (15),
  email VARCHAR(20),
  phone VARCHAR(15)
);
```

Вы можете считать поля переменными, однако в то время как в PHP не надо заботиться о типе переменной, SQL очень требователен к типам данных, содержащихся в полях. Для создания эффективной базы данных MySQL должен точно знать, сколько байтов памяти выделять для каждого поля базы данных. В основном это выполняется с помощью требования от разработчика базы данных указания типа и размера каждого поля таблицы. В табл. 7.2 содержится список основных типов данных, поддерживаемых MySQL.

Табл. 7.2. Основные типы данных, используемые в MySQL

Тип данных	Описание
INT	Стандартное целое число +/- 2 миллиарда (приблизительно)
BIGINT	Большое целое число +/- 9x10^18
FLOAT	Десятичное вещественное число 38 цифр
DOUBLE	Вещественное число с двойной точностью 308 цифр
CHAR(n)	Текст, содержащий n символов. Если текст содержит меньше, чем n символов, оставшееся место дополняется пробелами

Табл. 7.2. Основные типы данных, используемые в MySQL (продолжение)

Тип данных	Описание
VARCHAR(n)	Текст, содержащий n цифр. Оставшееся место автоматически обрезается
DATE	Дата в формате ГГГГ-ММ-ДД
TIME	Время в формате ЧЧ:ММ:СС
YEAR	Год в формате ГГГГ

Хотя типы данных, приведенные в табл.е 7.2, являются наиболее широко используемыми, MySQL поддерживает множество других типов данных. Если вам необходима более подробная информация о каком-либо типе, ищите ее в интерактивной помощи, поставляемой вместе с MySQL. Остальные базы данных используют очень похожий список типов.

Вы могли заметить, что для численных типов не обязательно задавать длину поля (хотя вы можете определить максимальный размер численных типов и количество цифр, которые будут хранится в полях float и double). Эти требования для хранения численных типов основываются на самих типах.

Работаем со строковыми данными в MySQL

Текстовые значения обычно хранятся в полях типа VARCHAR. Эти поля должны содержать столько символов, сколько было для них аллоцировано. Поля CHAR и VARCHAR имеют фиксированную длину. Основная разница между ними в том, что происходит с полем, если в нем содержится меньшее значение, чем заданная длина. Если вы объявите поле CHAR, имеющее длину 10

```
firstName VARCHAR(10);
```

и после этого сохраните в нем значение 'Andy', на самом деле поле будет содержать `'Andy'` (т. е. строку Andy, после которой дописано шесть пробелов). Поля CHAR заполняют оставшиеся символы пробелами. Поле VARCHAR удаляет все ненужное пространство. Обычно вы будете использовать поле типа VARCHAR для хранения строковых данных.

Определение длины поля VARCHAR

Проектирование данных является одновременно и искусством, и наукой. Определение корректной длины ваших текстовых полей является одной из самых старых проблем работы с данными.

Если вы не выделите достаточно места для текстовых данных, это может вызвать множество проблем у пользователей. Однажды я преподавал курс, который назывался CLT SD WEB PRG, потому что база данных, содержащая названия курсов, не содержала достаточно места для названия курса (Client-Side Web Programming (Веб-программирование на стороне клиента)). Мои студенты переименовали этот курс в «Купите гласные».

Однако вы не можете просто сделать каждое текстовое поле длиной тысячу символов, потому что это будет бесполезной тратой системных ресурсов. Если у вас есть поле, которое будет в среднем содержать пять символов, а вы выделите для него сто символов, вам будет необходимо место для хранения лишних 95 символов. Если в вашей базе данных содержатся тысячи записей, это может потребовать значительного количество места дисковой системы. В распределенной среде такое хранение лишней информации приведет к уменьшению пропускной способности. Для определения корректных размеров строковых полей необходимо экспериментировать с ними. Вам также придется протестировать ваше приложение в реальных условиях, прежде чем вы сможете быть уверенным в том, что сделали правильный выбор

Завершаем выражения CREATE TABLE

Как только вы усвоите типы данных полей, синтаксис `CREATE TABLE` станет намного более понятным. Необходимо рассмотреть еще несколько понятий. Как только вы напишете `CREATE TABLE`, используйте пару круглых скобок для обозначения списка полей. Каждое поле имеет название, после которого следует его тип (и длина, если это `CHAR` или `VARCHAR`). Поля разделяются запятыми. Не обязательно помещать каждое поле на отдельную строку или делать отступы от определения поля, однако я предпочитаю делать это, потому что такой намного проще читать и отлаживать.

Создаем первичный ключ

Возможно, вам любопытно назначение самого первого поля базы данных списка телефонов. Напомню, что строка, определяющее это поле, выглядит следующим образом.

```
id INT PRIMARY KEY,
```

Большинство таблиц баз данных содержат поле, похожее на это, которое хранит численное значение. Это специальное поле называется *первичным ключом*.

Возвращаясь к реальности

Очень простые базы данных, такие, как список телефонов, теоретически могут не содержать первичного ключа, однако такие поля настолько важны для более сложных баз данных, что вы можете начать использовать первичный ключ даже в своей первой таблице. Уже стало традицией использовать первичный ключ в каждой таблице. В главе 9 «Нормализация данных» вы получите дополнительную информацию о реляционной модели данных. В этой главе вы узнаете, как ключи используются для создания мощных баз данных, а также как правильно создавать первичные ключи. На самом деле, программа приключений, которую вы уже видели, очень активно использует ключевое поле, хотя в базе данных имеется только одна таблица

Рассмотренный пока код может быть использован непосредственно в программе MySQL. Вы можете увидеть код и результаты его выполнения на рис. 7.6.

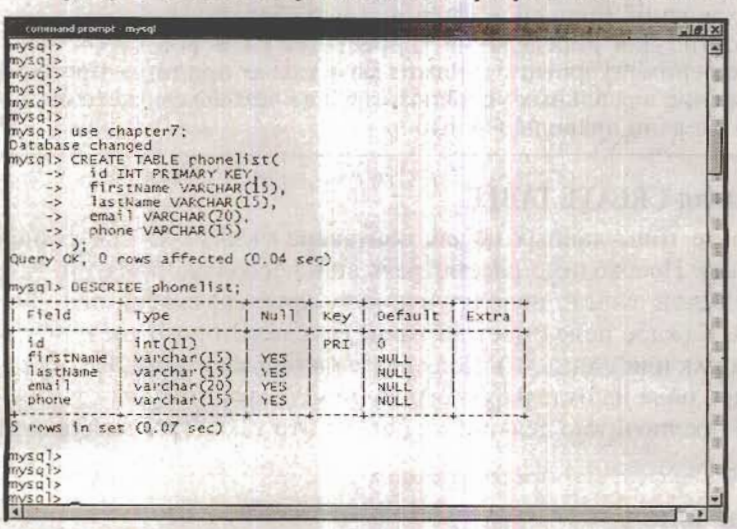


Рис. 7.6. Инструмент командной строки MySQL, после того как я создал таблицу списка телефонов

Используем команду DESCRIBE для проверки структуры таблицы

Иногда очень полезно проверить структуру таблицы, особенно если кто-то другой создал ее или вы не помните точно типы и размеры полей, содержащихся в таблице. Команда `DESCRIBE` позволяет вам просмотреть структуру таблицы.

Вставляем значения

После того как вы создали таблицу, вы можете добавлять в нее данные. Основным инструментом добавления записей в таблицу является команда INSERT.

```
INSERT INTO phoneList
VALUES (
  0, 'Andy', 'Harris', 'aharris@cs.iupui.edu', '123-4567'
);
```

Команда INSERT позволяет вам добавить запись в базу данных. Значения должны приводиться в том же порядке, в котором были определены поля. Значения отделяются запятыми, а все значения типа VARCHAR и CHAR должны быть заключены в одинарные кавычки.

Если вам необходимо загрузить в базу данных большое количество данных, вы можете использовать команду LOAD DATA. Эта команда принимает текстовый файл, элементы которого разделены символами табуляции, каждая строка которого соответствует записи, а поля разделяются символами табуляции. После этого эта команда загружает файл в базу данных. Обычно это самый быстрый способ заполнения базы данных тестовыми данными. Следующая строка кода загружает данные из файла addresses.txt в таблицу phoneList.

```
LOAD DATA LOCAL INFILE "addresses.txt" INTO TABLE phonelist;
```

На рис. 7.7 изображен инструмент MySQL после того, как я добавил в таблицу одну запись.

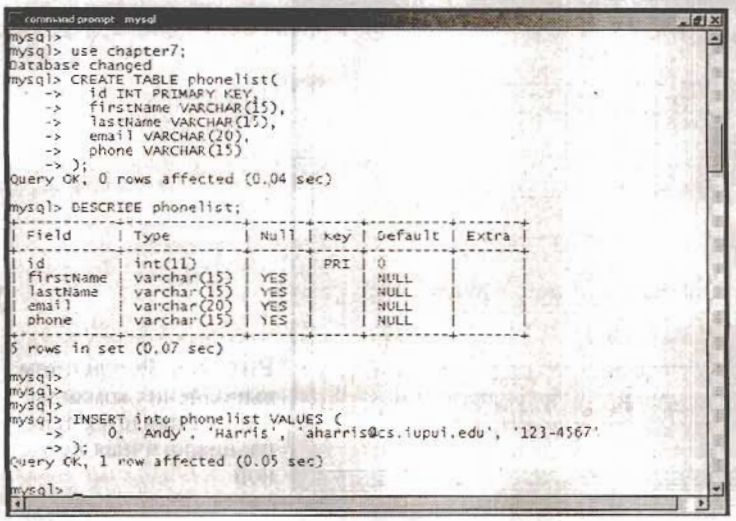


Рис. 7.7. MySQL говорит вам, что операция выполнена успешно, однако больше полезной информации вы не получите

Возвращаясь к реальности

По мере создания базы данных вам будет необходимо заполнять ее тестовыми значениями. На данный момент вам не нужны фактические данные, потому что база данных не будет корректно работать, пока вы ее не настроите. Однако тестовые значения должны быть подобны тем данным, которые будут затем содержаться в базе. Это поможет вам выявить некоторые проблемы, такие, как длины полей, которые слишком маленькие, или отсутствующие поля

Выбираем результаты

Конечно же, вы захотите увидеть результаты своих манипуляций с таблицей. Если вы захотите посмотреть данные, находящиеся в таблице, вы можете использовать команду `SELECT`. Наверное, это одна из самых мощных команд SQL, однако она очень проста в использовании. Чтобы просмотреть все данные базы данных `phonelist`, используйтеследующую команду.

```
SELECT * FROM phonelist
```

Эта команда берет все поля всех записей базы данных `phonelist` и отображает их в табличном формате.

На рис. 7.8 показано что произошло, когда я добавил вызов команды `SELECT` для просмотра списка телефонов.

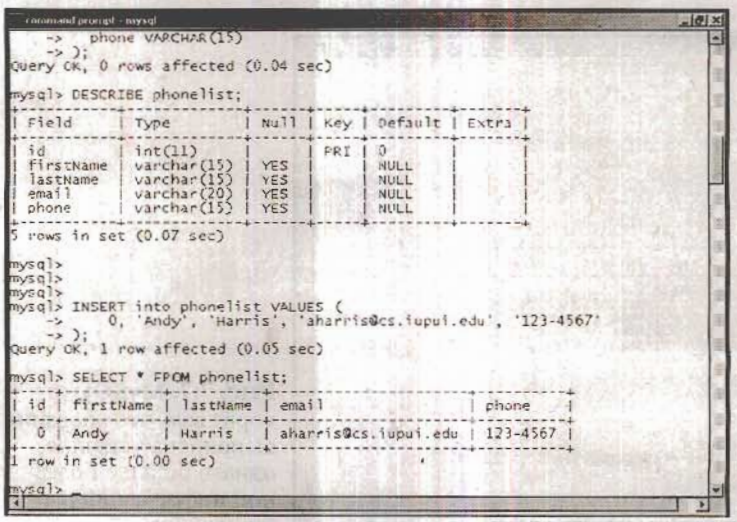


Рис. 7.8. Результатом выполнения команды `SELECT` является таблица, аналогичная исходной

Написание скрипта для создания таблицы

Очень важно понимать, как создавать таблицы в SQL вручную, потому что ваши программы будут делать как раз это. Однако очень утомительно писать код непосредственно в окне MySQL. Обычно при создании реальных приложений вам приходится несколько раз создавать таблицы данных, прежде чем вы будете удовлетворены результатом, и все это будет очень затруднительно сделать из интерфейса командной строки. Кроме этого, по мере того как вы будете создавать программы, работающие с базой данных, вы будете делать в них ошибки, которые портят исходные данные. Хорошо иметь готовый скрипт, который может очень легко заново создать базу данных, содержащую тестовые данные, даже если что-то пошло не так. Большинство программистов создают скрипт, содержащий команды SQL, с помощью текстового редактора (вы можете использовать тот же самый редактор, в котором пишете программы PHP) и используют команду SOURCE для загрузки этого кода. Ниже приведен скрипт SQL, который создает базу данных phonelist.

```
## создать список телефонов
## для mySQL
USE chapter7;
DROP TABLE IF EXISTS phoneList;
CREATE TABLE phoneList (
  id INT PRIMARY KEY,
  firstName VARCHAR(15),
  lastName VARCHAR (15),
  email VARCHAR(20),
  phone VARCHAR(15)
);
INSERT INTO phoneList
VALUES (
  0, 'Andy', 'Harris', 'aharris@cs.iupui.edu', '123-4567'
);
SELECT * FROM phoneList;
```

Этот код не в точности совпадает с кодом, который я использовал в интерактивном режиме, потому что здесь присутствует несколько особенностей, которые очень полезны при создании кода SQL в скрипте.

Создаем комментарии в SQL

На самом деле SQL – это язык. Хотя технически он и не является языком программирования, он обладает многими характеристиками такового. Так же, как PHP и другие языки, SQL поддерживает несколько типов символов комментария. Знак # обычно использует для обозначения комментария в SQL. Комментарии становятся особенно важными, когда вы сохраняете набор команд SQL в текстовом файле для дальнейшего использования. Эти комментарии могут помочь вам вспомнить, какую базу данных вы хотели создать. Очень важно помещать хотя бы основные комментарии в ваши скрипты, чтобы вы могли понимать, что они делают.

Удаляем таблицу

Может показаться странным, что мы говорим об удалении таблицы из базы данных, не создав таковой, однако зачастую (как и в данном случае) база данных создается с помощью скрипта. Прежде чем вы создадите новую таблицу, необходимо проверить, существует ли она, и, если да, то удалить ее с помощью команды `DROP`.

```
DROP TABLE IF EXISTS phoneList;
```

Эта команда как раз и предназначена для удаления таблицы. Если таблица `phoneList` уже существует, она будет удалена, чтобы не возникло никакой путаницы.

Выполняем скрипт с помощью команды SOURCE

Вы можете создать скрипт SQL с помощью любого текстового редактора. Такие скрипты принято сохранять в файлах с расширением `.sql`. В MySQL вы можете использовать команды `SOURCE` для загрузки и выполнения внешнего скрипта. На рис. 7.9 показана командная строка MySQL, после того как я запустил скрипт `buildPhonelist.sql`.

Часто в Windows я перетаскиваю файл из директории в консольные программы, типа MySQL. Windows копирует полное название файла, однако добавляет двойные кавычки, что порождает проблемы с интерпретатором MySQL. Если вы перетаскиваете название файла в MySQL, вам необходимо удалить символы двойных кавычек, чтобы программа MySQL смогла корректно считать файл.

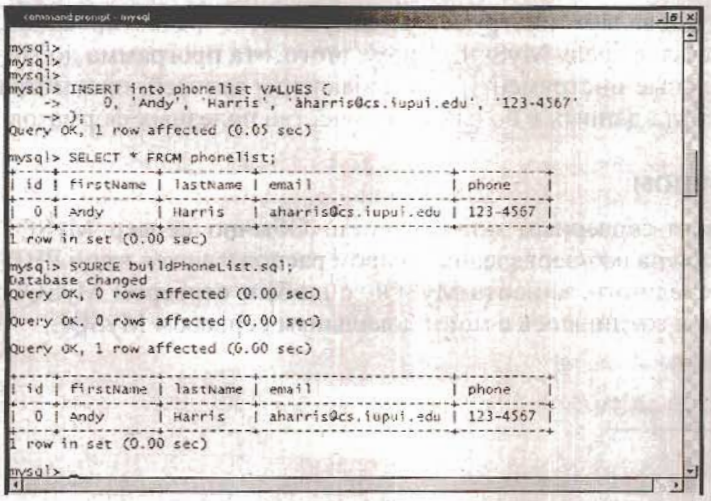


Рис. 7.9. Команда SOURCE позволяет вам считывать инструкции SQL из файла

Работаем с базой данных в SQLyog

Очень важно понимать язык SQL, однако иногда вам может захотеться воспользоваться графическим интерфейсом для создания и просмотра базы данных. Если вы работаете в Windows, вы можете использовать замечательный внешний интерфейс SQLyog. Эта бесплатная программа намного упрощает создание, редактирования и управление базами данных.

Возвращаясь к реальности

SQLyog настолько замечательная программа, что вы захотите пользоваться ею все время. В этом нет ничего страшного, однако убедитесь, что понимаете соответствующий код SQL, потому что ваши программы PHP будут использовать исключительно команды SQL. Вы можете спокойно использовать SQLyog во время создания и управления данными, однако ваши пользователи не будут работать с этой программой. Ваше приложение будет пользовательским интерфейсом к базе данных, поэтому вам необходимо иметь возможность выполнять все команды SQL из PHP. Я использую SQLyog, однако я всегда смотрю на сгенерированный ею код, чтобы я мог написать его самостоятельно

SQLyog в основном добавляет инструменты визуального редактирования, такие как, в Microsoft Access, в среду MySQL. Кроме этого, эта программа добавляет некоторые замечательные инструменты для добавления записей, просмотра структуры данных и экспорта данных в большое количество полезных форматов.

Соединяемся с сервером

MySQL является клиент-серверным приложением. Обычно сервер MySQL автоматически выполняется на веб-сервере, на котором располагаются ваши PHP-программы. Вы можете соединить клиента MySQL с любым сервером MySQL. На рис. 7.10 показано, как я соединяюсь с моим локальным сервером MySQL.

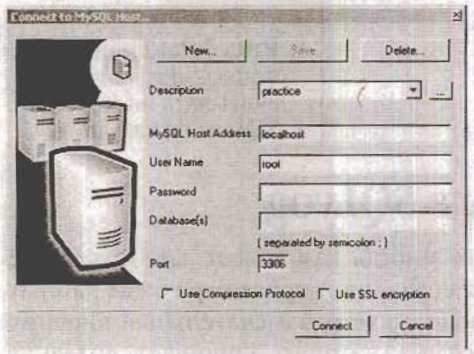


Рис. 7.10. Этот экран помогает вам соединиться с сервером данных

Очень важно знать, что вы можете соединиться с любым сервером данных, который вам разрешено использовать. Этот сервер данных не обязательно должен располагаться на той же физической машине, которую вы используете. Например, эта возможность очень полезна, если вы хотите использовать SQLyog для просмотра данных на удаленном веб-сервере, который вы содержите. Однако большинство удаленных веб-серверов не поддерживает такой тип доступа, поэтому вы должны знать, как работать с консолью MySQL.

Создаем и изменяем таблицу

SQLyog предоставляет визуальные инструменты, помогающие вам создавать и изменять ваши таблицы. По-моему, пример со списком телефонов слишком прост, поэтому я создам новую таблицу для иллюстрации возможностей SQLyog. Эта новая таблица содержит несколько случайно сгенерированных супергероев. На рис. 7.11 показано диалоговое окно, используемое для создания таблицы или изменения ее структуры.

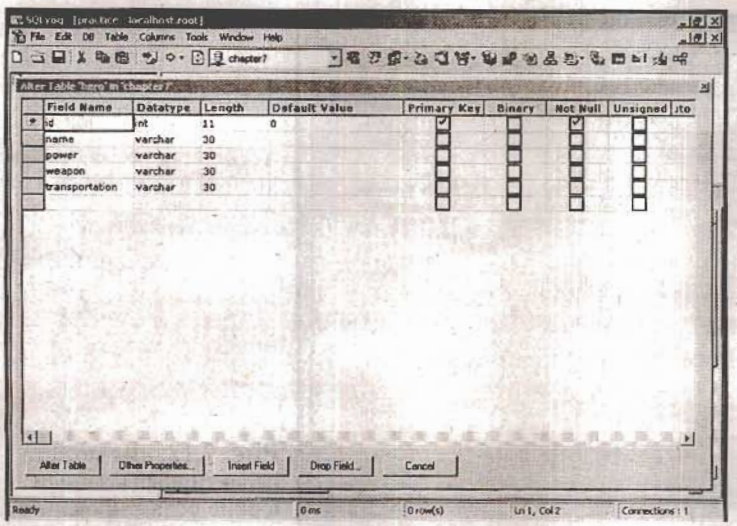


Рис. 7.11. С помощью SQLyog легко создать таблицу и изменить ее структуру

В SQLyog вы можете изменить типы переменных с помощью «выпадающего» списка, а многие свойства полей доступны в виде флажков. Большинство этих настроек пока что не важны. Заметьте, что `id` установлено в качестве первичного ключа. После того как вы закончите создание или изменение таблицы, будет сгенерирован и выполнен корректный код транзакции SQL.

Отдельное спасибо Ли Ситцу (Lee Seitz) за его истеричный генератор супергероев, доступный по адресу http://home.hiwaay.net/~lkseitz/comics/herogen/.

Посмотрите этот сайт как-нибудь на досуге.

Редактируем данные таблицы

Вы можете использовать SQLyog для редактирования таблицы так же, как и в крупноформатных таблицах. На рис. 7.12 приведен пример этого.

Чтобы редактировать таблицу в SQLyog, выберите таблицу из списка таблиц, расположенного в левой части экрана SQL. Для редактирования таблицы вы можете нажать F11, либо выбрать «Вставить/Обновить данные» (Insert/Update Data) из меню Таблица (Table). После того как вы завершите редактировать данные, вы можете нажать кнопку Завершить (Done), и SQLyog автоматически создаст и выполнит код SQL, необходимый для изменения данных таблицы. Добавление данных в последнюю строку приведет к созданию новой записи.

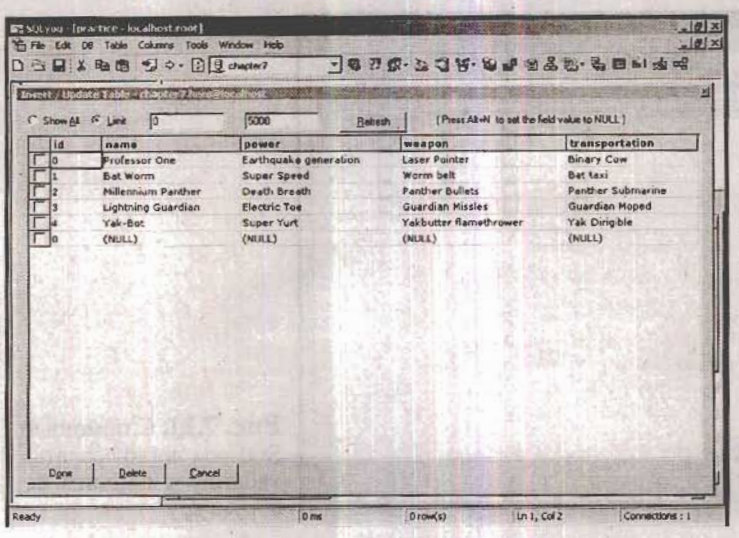


Рис. 7.12. Вы можете редактировать записи в окне редактирования

Экспортируем таблицу

Наиболее интересные особенности SQLyog включают в себя способы экспорта информации о таблицах. Вы можете сгенерировать форматы, отображающие данные различными способами. Как только вы перешли в окно таблицы (выбрав ее и нажав клавишу `Enter`), вы можете перейти в меню инструментов и выбрать «Экспортировать результирующий набор» (Export Result Set). Вы увидите диалоговое окно, похожее на изображенное на рис. 7.13.

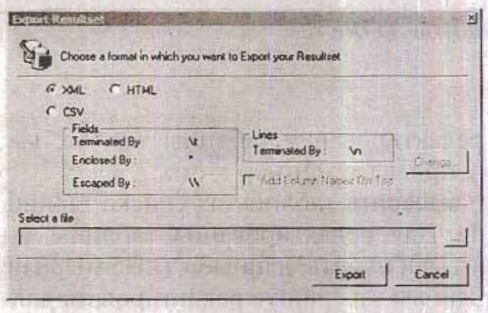


Рис. 7.13. Диалоговое окно Export Result Set позволяет вам сохранить данные в нескольких различных форматах

Вы можете очень легко сгенерировать HTML-сводку ваших данных, выбрав опцию HTML и указав название файла. На рис. 7.14 показана страница HTML, сгенерированная для таблицы героев.

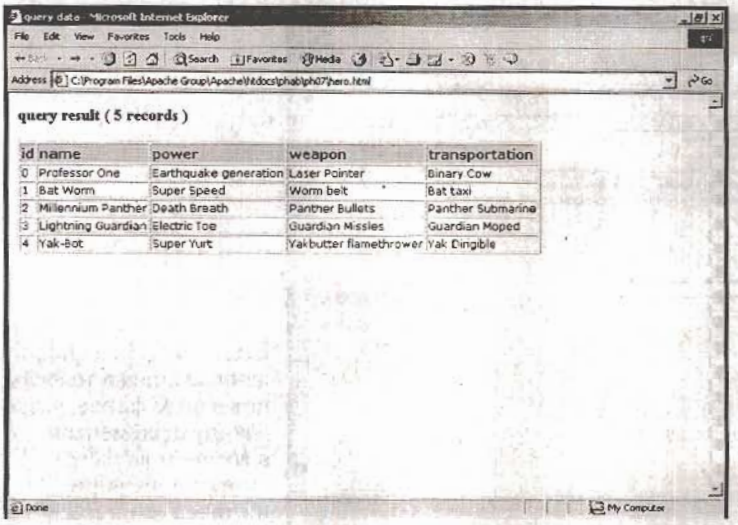


Рис. 7.14. Вы можете очень легко печатать HTML-сводку ваших данных

Возможно, вы предпочтете сохранить ваши результаты в каком-нибудь другом формате с разделителями, обсужденными в главе 6 «Работа с файлами». Вы можете очень легко сгенерировать такой формат, выбрав опцию CSV (значения, разделенные запятыми) и выбрав собственный разделитель. Это хороший выбор, если вы хотите, чтобы ваши данные могли быть прочитаны крупноформатной таблицей, или если вы создаете программу, которая может обрабатывать такой формат, но не имеет непосредственного доступа к базе данных. На рис. 7.15 показана CSV версия набора данных героев.

Кроме этого, вы можете создать XML файл для хранения данных. Как вы можете видеть из рис. 7.16, XML очень похож на HTML и описывает информацию в самодокументированном виде.

Когда вы начнете создавать сложные базы данных, вы быстро поймете ценность описаний каждой таблицы. Вы можете использовать команду «Создать схему» (Create Schema) в меню «База данных» (DB) для генерации HTML страницы, описывающей вашу таблицу. Эта схема может быть важной частью вашей стратегии программирования и документирования. На рис. 7.17 изображена схема таблицы героев.

Последним очень полезным инструментом является функция «экспортировать в виде скрипта» (export as batch script), расположенная в меню DB. Вы можете использовать этот инструмент для автоматической генерации скрипта SQL, создающего и заполняющего таблицу. Эта возможность является очень полезной, если вы используете визуальные инструменты для создания и редактирования таблицы, но хотите иметь возможность создать ее заново с помощью скрипта. Диалоговое окно, изображенное на рис. 7.18, иллюстрирует различные настройки этого инструмента.

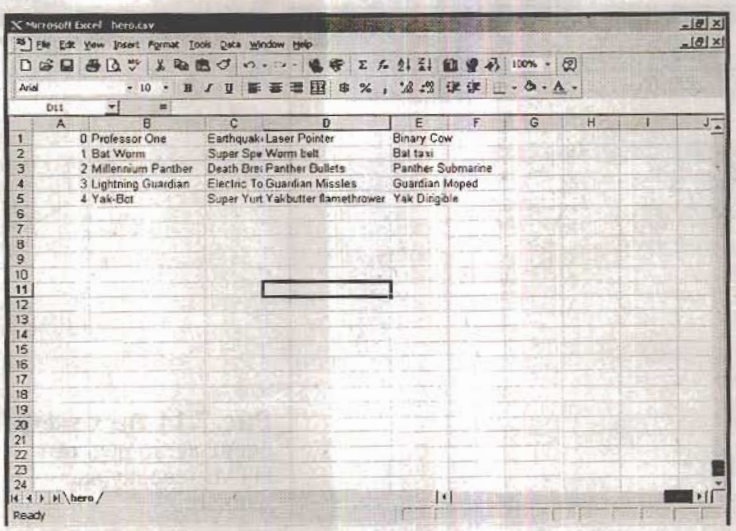


Рис. 7.15. Я сохранил данные списка телефонов в виде файла, разделяющим элементом в котором является символ табуляции, и считал его в Excel

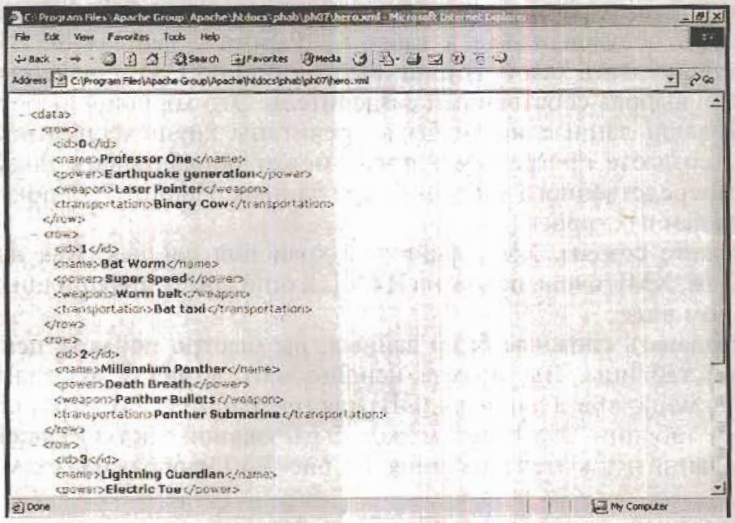


Рис. 7.16. XML форма данных генерирует теги, схожие с HTML, для описания полей таблицы

Вы можете задать, будет ли скрипт генерировать только структуру таблицы или генерировать структуру и добавлять в нее данные. Кроме этого, вы можете задать, чтобы результирующий скрипт содержал код для выбора базы данных, удаления ее таблицы, если она уже существует, и название результирующего скрипта.

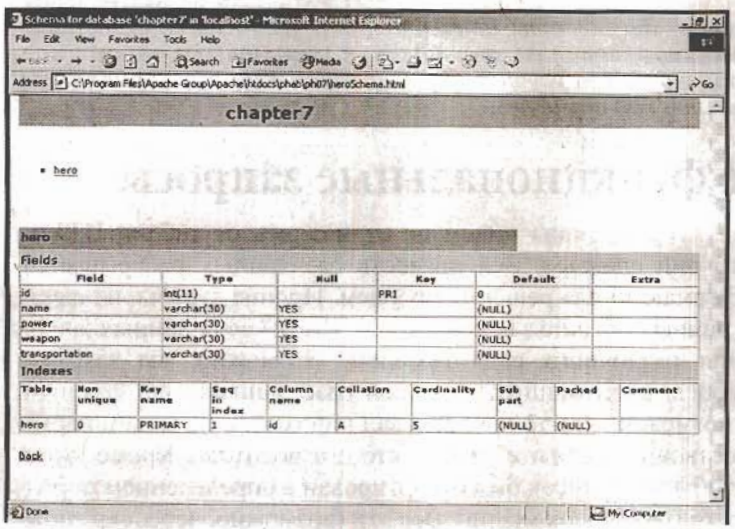


Рис. 7.17. Схема таблицы содержит важную информацию об ее структуре

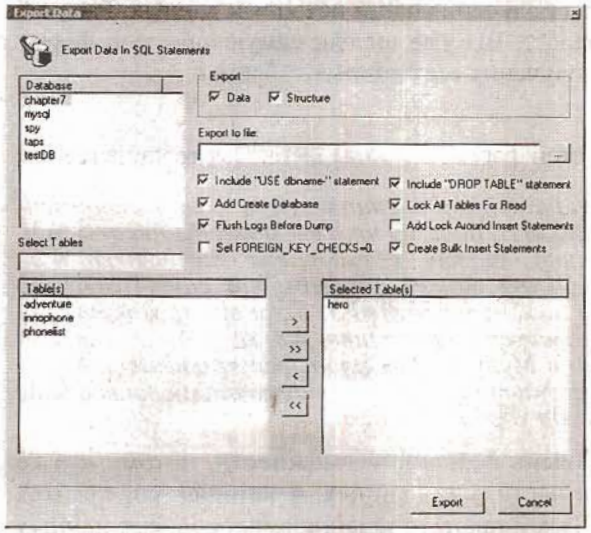


Рис. 7.18. С помощью этого диалогового окна вы можете генерировать код, который будет создавать копии любой базы данных, созданной или просмотренной в SQLyog

Возможность автоматического генерирования скриптов SQL предоставляет огромные возможности. Это может сэкономить много времени, и вы можете многому научиться, просмотрев скрипты, созданные с помощью этой функциональности. Однако не забывайте о том, что вы являетесь программистом, а поэтому несете ответственность за код своих проектов, даже если вы не

писали его непосредственно. Вы должны понимать, для чего нужен сгенерированный код. Большинство кода, который вы видели, я уже описал, однако, возможно, вам придется поискать его улучшенный вариант. Как я говорил ранее, вам нужно знать, как писать код вручную.

Создаем более функциональные запросы

Пока что создаваемые вами таблицы ничем не отличались от таблиц HTML, за исключением того, что они сложнее. Вся прелесть баз данных раскрывается, когда вы используете информацию для решения проблем. Ирония судьбы, но самая важная часть работы баз данных обычно заключается не в *получении* данных, а в их *фильтрации* для решения различного рода проблем. Возможно, вы захотите получить список всех героев, содержащихся в вашей базе данных, чьи фамилии начинаются на «Е», или, возможно, кто-то припарковал дирижабль на вашем парковочном месте, и вы, естественно, захотите узнать, кто его водитель. Кроме этого, возможно, вы захотите, чтобы этот список был отсортирован в определенном порядке, или захотите получить только список машин. Все эти (возможно, чересчур сложные) примеры иллюстрируют способы получения части исходных данных. Рабочей лошадкой SQL является команда `SELECT`. Вы уже видели самую простую форму этой команды, использованную для получения всех данных таблицы.

```
SELECT * FROM hero;
```

На рис. 7.19 показано применение этой формы команды `SELECT` к таблице героев.

SQLyog является замечательным инструментов для экспериментирования с командой `SELECT`*, потому что вы можете писать SQL код вручную и мгновенно видеть результаты его выполнения. Чтобы использовать SQLyog таким образом, напишите SQL-код в редакторе, после чего нажмите Shift+F5. Если вы не хотите использовать SQLyog, вы можете проводить те же самые эксперименты непосредственно в MySQL. Вам придется проделать аналогичную работу, однако результаты будут отформатированы в виде текста, который не всегда удобно читаем.*

Команда `SELECT` предоставляет очень большие возможности, потому что ее можно использовать для получения части всех данных, в которых содержится только необходимые поля и записи. Такой процесс задания вопросов базе данных обычно называется *запросом*.

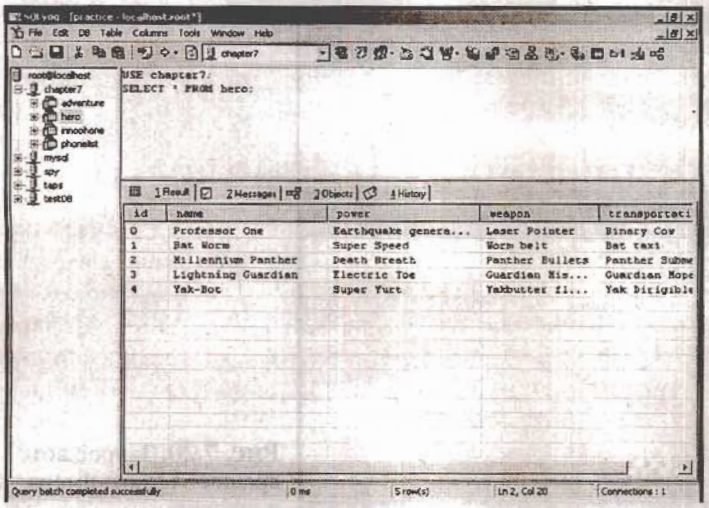


Рис. 7.19. Запрос `SELECT` располагается справа, а результаты его выполнения помещены под ним

Выбираем столбцы

Во многих случаях вам могут требоваться не все поля (или столбцы) таблицы. Например, вам может требоваться только имя и оружие всех записей списка. Вы можете задать это, используя приведенное ниже выражение `SELECT` (показанное на рис. 7.20).

```
SELECT name, weapon
FROM hero;
```

Это может показаться ненужным для такой простой базы данных, как список героев, однако зачастую реальные таблицы очень сложны и содержат множество полей и вам будет необходимо оставить только некоторые из них. Например, я использую базу данных для хранения информации о своих студентах. Информация о каждом студенте содержит множество данных, однако мне может потребоваться только список имен студентов и их адреса электронной почты. Возможность выделять только необходимые поля является одним из способов получения такой информации из базы данных.

Результаты выполнения такого запроса очень похожи на новую таблицу. Результат выполнения запроса можно представить как временную таблицу.

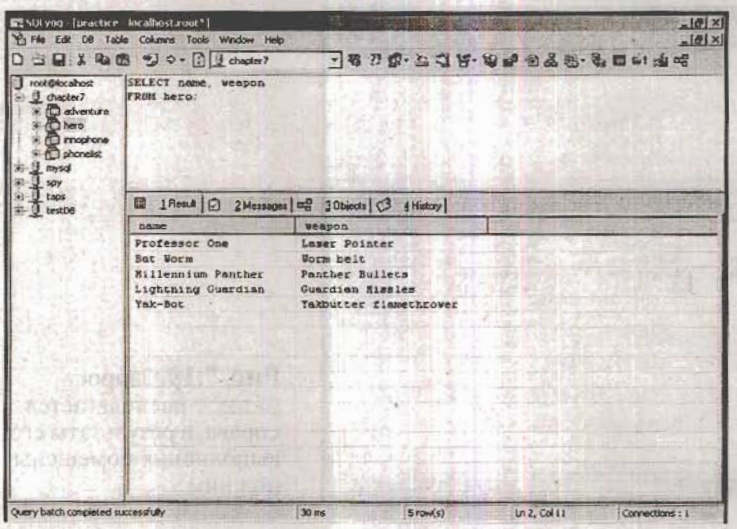


Рис. 7.20. Запрос возвращает только имена и оружия

Выбираем строки с помощью команды WHERE

Кроме выбора столбцов, которые возвращаются запросом, зачастую вам необходимо выбрать и строки. Например, вы можете встретить злодея, которого можно убить только лазерной указкой. Запрос, изображенный на рис. 7.21, является иллюстрацией решения этого вопроса.

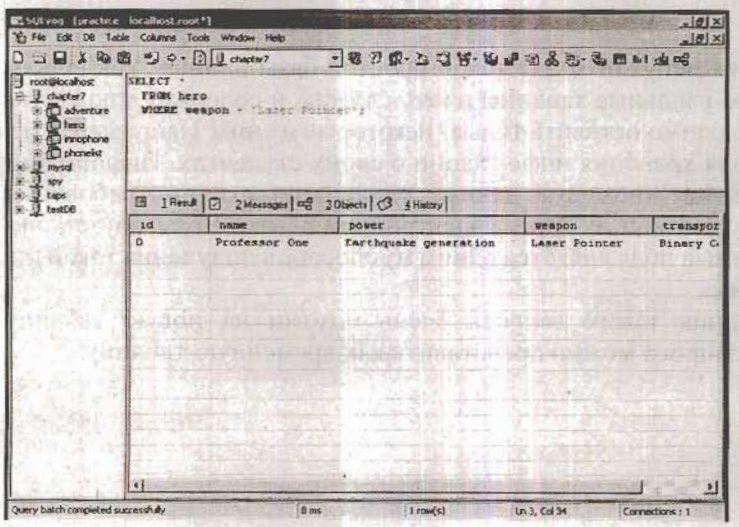


Рис. 7.21. Если вы знаете, как настроить запрос, вы можете получить очень специфичные результаты. В данном случае запрос выбирает только тех героев, у которых есть лазерная указка

```
SELECT *
  FROM hero
  WHERE weapon = 'Laser Pointer';
```

Данный запрос возвращает только строки, удовлетворяющие заданному условию.

Добавляем условия с помощью команды WHERE

Для определения того, какую строку или строки вы хотите увидеть, вы можете добавить в запрос команду WHERE. Эта команда позволяет вам задать условие. Менеджер базы данных проверит все записи в таблице. Если для записи условие выполняется, она будет включена в результирующий набор. Условия в команде WHERE аналогичны тем, которые используются в коде PHP, однако они не *в точности* такие же. В SQL одиночный знак равенства используется для обозначения равенства. Кроме этого, текстовые элементы заключаются в одиночные кавычки ('). Вы можете также использовать условия <, > и <= или >= для ограничения поиска.

Использование операторов сравнения для численных данных, таких, как целые и вещественные числа, понять просто. Однако совсем не очевидно, как язык будет воспринимать сравнения текста. В SQL разработаны стандартные правила, однако в каждой реализации они могут отличаться. В общем случае SQL не чувствителен к регистру букв, поэтому строка "Yak-Bot" будет совпадать с "yak-bot" или "yAK-bOT". Кроме этого, операторы «<» и «>» предназначены для сортировки по алфавиту, поэтому

```
SELECT *
  FROM hero
  WHERE name < 'D';
```

выберет все записи, в которых имена героев начинаются с «A», «B» или «C».

Используем команду LIKE для поиска частичных совпадений

Зачастую вы не знаете точного значения поля, которое хотите найти. Команда LIKE позволяет вам находить частичные совпадения. Например, вы можете захотеть узнать, у каких героев есть супер способности. Этот запрос

```
SELECT *
  FROM hero
  WHERE power LIKE 'Super%';
```

вернет всех героев, чьи способности начинаются со значения «Super». Знак процента (%) может использоваться в качестве специального символа для обозначения любого количества символов. Вы можете использовать различные варианты команды LIKE для поиска информации о всех героях с помощью способа транспортировки, начинающегося с буквы «B».

```
SELECT name, transportation
  FROM hero
  WHERE transportation LIKE 'B%';
```

Вы также можете использовать символ подчеркивания (_) для обозначения одного символа.

Простая поддержка специальных символов в SQL достаточна для многих целей. Если вам нравятся регулярные выражения, вы можете использовать команду REGEXP *для определения совпадения поля с регулярным выражением. Это очень мощный инструмент, однако он является расширением стандарта SQL. Он нормально работает в MySQL, однако он не поддерживается всеми базами данных SQL.*

Создаем множественные условия

Вы можете комбинировать условия с помощью ключевых слов AND, OR и NOT, для создания более сложных выражений. Например,

```
SELECT *
  FROM hero
  WHERE transportation LIKE 'B%'
    AND power LIKE '%super%';
```

выбирает тех героев, чьи транспортные средства начинаются с «B» и в названии способностей которых есть слово «super».

Такая возможность создания сложных выражений будет очень полезна в сложных базах данных, содержащих множество таблиц.

Используем команду ORDER BY для сортировки результатов

Одной очень остроумной особенностью команды SELECT является возможность сортировки результата по любому из полей. На рис. 7.22 и рис. 7.23 показано, как с помощью команды ORDER BY можно определять сортировку таблиц.

Команда ORDER BY позволяет вам определить, как будут отсортированы данные. Вы можете задать любое поле в качестве поля для сортировки. Как вы можете видеть на рис. 7.23, команда DESC определяет, что данные будут отсортированы в порядке убывания.

Изменяем данные с помощью команды UPDATE

Кроме этого, вы можете использовать SQL для обновления данных, содержащихся в базе данных. Ключом к этой функциональности является команда UPDATE. Вот пример, который поможет вам понять, о чем я говорю.

```
UPDATE hero
```

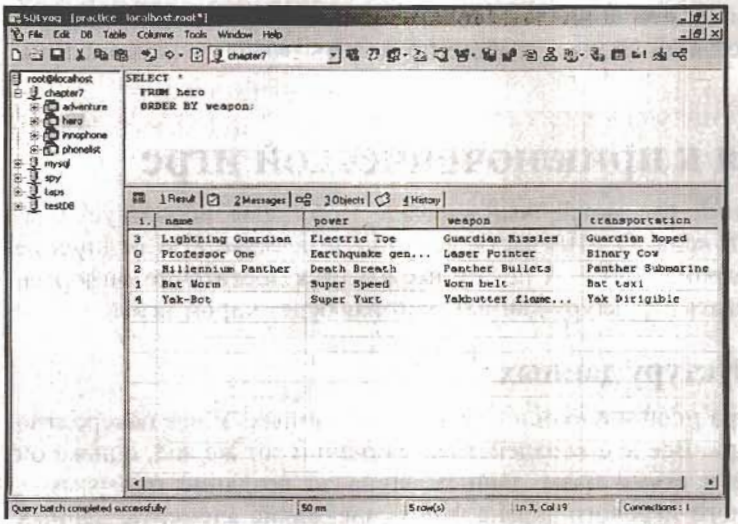


Рис. 7.22. Этот запрос отображает всю базу данных, отсортированную по названию оружия

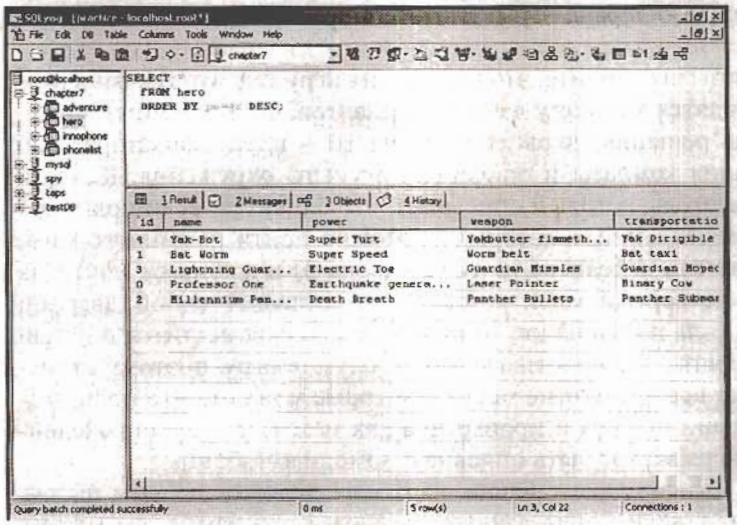


Рис. 7.23. Этот запрос сортирует по способностям в убывающем (с конца алфавита) порядке

```
SET power = 'Super Electric Toe'
WHERE name = 'Lightning Guardian';
```

Этот код изменяет способность Lightning Guardian на Super Electric Toe (который, вероятно, намного лучше, чем обычный Electric Toe).

В общем случае, вы будете обновлять только одну запись за один раз. Вы можете использовать команду WHERE, чтобы определить, какая запись таблицы будет обновляться.

Возвращаемся к приключенческой игре

Генератор приключений, рассмотренный в начале этой главы, использует комбинацию MySQL и PHP кода. О PHP части вы узнаете в главе 8 «Соединение с базами данных из программ PHP». А пока у вас имеется достаточно информации, чтобы начать создавать структуру данных, которая будет ядром игры.

Разрабатываем структуру данных

Приключенческая игра полностью основывается на данных. У нее невероятно повторяющаяся структура. Все время задействуется один и тот же код, однако он работает с различными частями базы данных. Я начал создание программы, сделав набросок основного игрового экрана и подумав, какие элементы данных мне потребуются для каждого экрана игры. В итоге я пришел к созданию таблицы, наподобие табл. 7.3.

После изучения таблицы вы увидите, что я упростил игру так, что все решения, принимаемые в игре, сводятся к одному из семи вариантов, определенных мною. Каждый узел (или точка решения) игры состоит из id – идентификатора (или номера комнаты), названия комнаты и описания текущего окружения. Каждый узел также содержит указатель, который описывает, что случится, если пользователь выберет одно из направлений, в которых возможно пойти из данного узла. Например, если пользователь находится на складе (узел 3) и выбирает пойти на восток, он перейдет в четвертый узел, который представляет собой дверной проем. Если из третьего узла пойти на юг, то пользователь переместится в пятый узел, которым является ящик. Я очень тщательно продумал игру, поэтому структура данных представляет все доступные места, в которые может пойти пользователь. Я определил критерии победы и проигрыша как нахождение в определенных узлах, поэтому вся игра может быть описана с помощью таблицы.

Очень важно понять, что создание таблицы на бумаге является первым шагом. После того, как вы решили, какие типы данных требуются программе, вы можете думать о том, как объединить эти данные. Как вы можете видеть, выбор базы данных предоставляет мне невероятные возможности и позволяет очень легко работать с данными. Наверное, самым замечательным является то, что программа может быть использована совсем для другой игры путем простой замены базы данных. Мне не надо изменять ни одной строки кода, чтобы сделать игру абсолютно другой. Все, что мне необходимо сделать, – это указать другую базу данных, или изменить имеющуюся.

Табл. 7.3. Структура данных приключенческой игры Enigma Adventure

Идентификатор	Название	Описание	Север	Восток	Юг	Запад
0	-нет-	Вы не можете туда идти!	1	0	0	0
1	Начало	Вы находитесь около подводной лодки и ищете знаменитую машину для шифрования Enigma	0	3	0	2
2	Палуба	Как только вы ступили на палубу подводной лодки, к вам подошел охранник. Вам остается только спрыгнуть с субмарины, прежде чем вам поймают	15	15	15	15
3	Склад	Вы ждете внутри склада. Вы видите проход на восток и коробку на юге	0	4	5	0
4	Проход	Вы вышли прямо на группу охранников. Это не выглядит обнадеживающе…	0	19	0	15
5	Коробка	Вы заползли в коробку и ждете. Вскоре вы почувствовали, что коробку подняли и перенесли через пристань!	6	0	0	7
6	Ждать	Вы ждете, пока коробка не будет установлена в темном месте. Вы можете двигаться вперед или назад …	8	0	9	0
7	Выпрыгнуть	Вы решили выпрыгнуть из коробки, однако вы оказались в конце пристани.	15	19	15	15

Табл. 7.3. Структура данных приключенческой игры Enigma Adventure

Идентификатор	Название	Описание	Север	Восток	Юг	Запад
8	Двигаться вперед	Когда вы двигались вперед, два грубых моряка схватили и выбросили вас из рубки	15	15	15	15
9	Двигаться назад	В темной комнате вы увидели устройство Enigma. Как вы вынесите его с подводной лодки?	13	11	10	12
10	Отправить сигнал с помощью Enigma	Вы используете устройство шифрования для отправки сигнала. Союзные силы распознали ваш сигнал и окружили корабль, когда он показался на поверхности	14	0	0	0
11	Перестрелка	Перестрелка в субмарине, находящейся на глубине – это плохая идея…	19	0	0	0
12	Ждать с устройством Enigma	Вы ждете, однако моряки заметили отсутствие устройства и обыскали всю подводную лодку. Вы были обнаружены и выброшены через торпедную шахту	15	0	0	0
13	Заменить Enigma и ждать	Вы поместили устройство обратно на его место и терпели во ждете, однако другого шанса у вас не будет. Вас обнаружили, когда подводная лодка зашла в гавань	19	0	0	0

Табл. 7.3. Структура данных приключенческой игры Enigma Adventure

Идентификатор	Название	Описание	Север	Восток	Юг	Запад
14	Победа	Поздравляем! Вы похитили устройство и прекратили войну!	1	0	0	0
15	Вода	Вы в воде. Подводная лодка движется от вас. Похоже, ваши дела плохи...	19	0	0	0
16			0	0	0	0
17			0	0	0	0
18			0	0	0	0
19	Конец игры	Игра окончена. Вы проиграли	1	0	0	0

Как только я определился со структурой данных, я создал скрипт SQL, который создает набросок базы данных. Вот этот скрипт.

```
## создает SQL файл Adventure
## для MySQL
## Энди Харрис (Andy Harris)
DROP TABLE IF EXISTS adventure;
CREATE TABLE ADVENTURE (
  id int PRIMARY KEY,
  name varchar(20),
  description varchar(200),
  north int,
  east int,
  south int,
  west int
);
INSERT INTO adventure values(
  0, 'lost', 'You cannot go that way!',
  1, 0, 0, 0
);
INSERT INTO adventure values(
```

```
  1, 'start', 'You are at a submarine yard, looking for the
famous Enigma code machine',
  0, 3, 0, 2
);
INSERT INTO adventure values(
  2, 'sub deck', 'As you step on the submarine deck, a guard
approaches you.  Your only choice is to jump off the sub before
you are caught.',
  15, 15, 15, 15
;
INSERT INTO adventure values(
  3, 'warehouse', 'You wait inside the warehouse.  You see a
doorway to the south and a box to the east.',
  0, 4, 5, 0
);
INSERT INTO adventure values(
  4, 'doorway', 'You walked right into a group of guards. It does
not look good...', 0, 19, 0, 15
);
INSERT INTO adventure values(
  5, 'box', 'You crawl inside the box and wait. Suddenly, you
feel the box being picked up and carried across the wharf!', 6,
0, 0, 7
);
INSERT INTO adventure values(
  6, 'wait', '..You wait until the box settles in a dark space.
You can move forward or aft...', 8, 0, 9, 0
);
INSERT INTO adventure values(
  7, 'jump out', 'You decide to jump out of the box, but you are
cornered at the end of the wharf.', 15 ,19, 15, 15
);
INSERT INTO adventure values(
  8, 'forward', 'As you move forward, two rough sailors grab you
and hurl you out of the conning tower.', 15 ,15, 15, 15
);
INSERT INTO adventure values(
  9, 'aft', 'In a darkened room, you see the enigma device. How
will you get it out of the sub?', 13 ,11, 10, 12
);
```

```
INSERT INTO adventure values(
  10, 'signal on enigma', 'You use the enigma device to send a
signal. Allied forces recognize your signal and surround the
ship when it surfaces', 14 ,0, 0, 0
    );
INSERT INTO adventure values(
  11, 'shoot your way out', 'A gunfight on a submerged sub is a
bad idea...', 19 ,0, 0, 0
);
INSERT INTO adventure values(
  12, 'wait with enigma' ,'You wait, but the sailors discover
that enigma is missing and scour the sub for it. You are discov-
ered and cast out in the torpedo tube.', 15 ,0, 0, 0
);
INSERT INTO adventure values(
  13, 'replace enigma and wait','You put the enigma back in place
and wait patiently, but you never get another chance. You are
discovered when the sub pulls in to harbor.', 19 ,0, 0, 0
);
INSERT INTO adventure values(
  14, 'Win', 'Congratulations! You have captured the device and
shortened the war!', 1, 0, 0, 0
);
INSERT INTO adventure values(
  15, 'Water', 'You are in the water. The sub moves away. It
looks bad...', 19 ,0, 0, 0
);
INSERT INTO adventure values(
  16,'','', 0, 0, 0, 0
;
INSERT INTO adventure values(
  17,'','', 0, 0, 0, 0
);
INSERT INTO adventure values(
  18,'','', 0, 0, 0, 0
);
INSERT INTO adventure values(
  19, 'Game Over' ,'The game is over. You lose.', 1, 0, 0, 0
);
SELECT id, name, north, east, south, west FROM adventure;
```

```
SELECT id, description FROM adventure;
```

Я написал этот код вручную, хотя мог разработать его с помощью SQLyog. Заметьте, что я создал таблицу, вставил в нее значения и написал несколько команд SELECT для проверки этих значений. Мне бы хотелось иметь скрипт для создания базы данных, даже если я делаю ее с помощью такого инструмента, как SQLyog, потому что я умудрился несколько раз испортить эту базу данных, пока писал код для этой главы. Очень удобно иметь скрипт, который может сразу же создать заново базу данных, не прилагая при этом никаких усилий.

Итоги

Хотя в этой главе вы и не писали PHP код, вы узнали, как создать простую структуру данных при помощи языка SQL. Вы научились работать с консолью MySQL при создании и использовании баз данных. Вы узнали, как получить данные из базы данных при помощи команды SELECT. Вы знаете, как изменить команду SELECT, чтобы получить более специфические результаты. Вы увидели, как с помощью SQLyog можно упростить создание и управление базами данных MySQL. Вы создали структуру данных для приключенческой игры.

Домашнее задание

1. Разработайте простую базу данных. Начните с чего-нибудь простого, например списка телефонов.
2. Создайте вашу базу данных в SQL.
3. Напишите программу для создания и заполнения вашей базы данных.
4. Используйте SQLyog для управления вашей базой данных и просмотра ее результатов в различных форматах.

Глава 8

Соединяемся с базой данных при помощи PHP

После всех этих разговорах о базах данных вам, наверное, не терпится соединиться с базой данных из программы PHP. PHP хорошо известен своей поддержкой баз данных, и в особенности MySQL. И действительно, соединение с базой данных MySQL из PHP происходит очень просто. Как только вы соединились с базой данных, вы можете посылать ей команды SQL и получать данные результатов их выполнения, которые можно использовать в PHP программах. В конце этой главы вы доделаете приключенческую игру, описанную в начале главы 7 «Использование MySQL для создания баз данных». Вы увидите, что если структура данных спроектирована правильно, программирование становится очень простым. В частности, в этой главе вы узнаете, как:

- создать соединение с базой данных MySQL из PHP;
- использовать определенную базу данных;
- посылать запросы базе данных;
- анализировать результаты выполнения запроса;
- проверять данные на наличие ошибок;
- создавать HTML страницы, основываясь на полученных данных.

Соединяемся с базой данных героев

Чтобы показать, как это работает, я написал простую PHP-программу, которая возвращает все значения, содержащиеся в базе данных героев, созданной в главе 7. На рис. 8.1 показан результат выполнения программы «Show Hero».

Я решил использовать эту простую базу данных вместо более сложной базы данных приключенческой игры. При изучении нового материала лучше всего рассматривать его на очень простых примерах, после чего переходить к изучению более сложных случаев. База данных приключений содержит множество информации, причем взаимосвязи между записями достаточно замысловаты. Я хотел начать с более простой базы данных, чтобы убедиться, что я понял основы соединения данных, прежде чем переходить к реальным базам данных, которые сложны сами по себе.

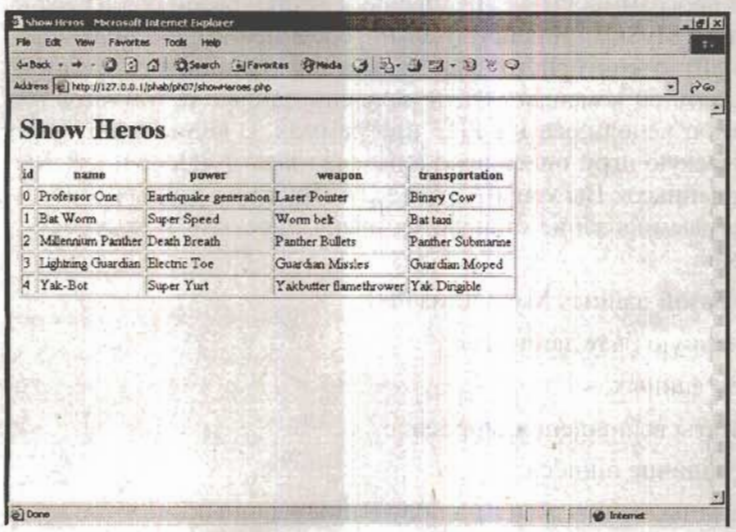


Рис. 8.1. Эта HTML таблица сгенерирована программой PHP, которая считывает данные из базы данных

Код программы, генерирующей эту страницу, показан ниже.

```
<body>
<h1>Show Heros</h1>
<?
//соединиться с базой данных
$conn  = mysql_connect("localhost", "", "");
mysql_select_db("chapter7", $conn);

//создать запрос
$sql = "SELECT * FROM hero";
```

```
$result = mysql_query($sql, $conn);
print "<table border = 1>\n";
//получить названия полей
print "<tr>\n";
while ($field = mysql_fetch_field($result)){
  print "  <th>$field->name</th>\n";
} // завершение цикла while
print "</tr>\n\n";
//получить данные строк и сохранить их в ассоциативный массив
while ($row = mysql_fetch_assoc($result)){
  print "<tr>\n";
  //просмотреть каждое поле
  foreach ($row as $col=>$val){
    print "  <td>$val</td>\n";
  } // завершение цикла foreach
  print "</tr>\n\n";
}// завершение цикла while
print "</table>\n";
?>
</body>
</html>
```

Просмотрите код, и вы увидите, что большая его часть вам уже знакома, за исключением нескольких новых функций, начинающихся с "`mysql_`". Эти функции позволяют вам получать доступ к базам данных MySQL. Если вы посмотрите документацию PHP, вы увидите очень похожие функции для нескольких других типов баз данных, включая Oracle, Informix, mSQL и ODBC. Вы обнаружите, что процессы соединения и использования баз данных очень похожи между собой и не зависят от используемой базы данных.

Устанавливаем соединение

Прежде всего, необходимо установить соединение между вашей PHP-программой и сервером MySQL. Вы можете соединиться с любым сервером, использовать который вам разрешено. Функция `mysql_connect` устанавливает соединение между MySQL и PHP. Вот выражение, производящее соединение, из программы `showHero`.

```
$conn  = mysql_connect("localhost", "", "");
```

Функция `mysql_connect()` требует задания трех параметров: имя сервера, имя пользователя и пароль. Имя сервера – это имя или унифицированный указатель ресурса сервера MySQL, с которым вы хотите соединиться (этот параметр

может быть установлен в `localhost`, если PHP и сервер MySQL располагаются на одном компьютере, что является достаточно распространенным случаем). Имя пользователя соответствует имени пользователя в MySQL. Большинство пакетов баз данных поддерживают учетные записи пользователей.

Вероятно, вам придется изменить поля userName и password, если вы будете использовать этот код на сервере. Я использовал значения, приводимые по умолчанию, которые нормально работают на изолированном тестовом сервере, однако вам придется изменить их на ваше имя пользователя и пароль, если вы будете использовать этот код на действующих серверах.

Вы можете использовать то же самое имя пользователя и пароль, которые использовали для входа в MySQL, и ваши программы будут иметь те же самые права доступа, что и у вас. Конечно, возможно, вы захотите ограничить права доступа ваших программ, для чего вы можете создать специальную учетную запись, которая имеет только соответствующие права доступа для пользователей вашей программы.

Возвращаясь к реальности

Безопасность баз данных является очень важной и непростой проблемой. Существует несколько простых способов защиты ваших данных от большинства взломщиков. Прежде всего, при публикации кода необходимо скрывать ваше имя пользователя и пароль. Я убрал свое имя пользователя и пароль из кода, который привел здесь. На практике вы можете оставить эти значения незаполненными, однако прежде убедитесь, что ваш код не допускает публичного доступа к данным. Если вам необходимо продемонстрировать ваш код (например, на занятии), не забудьте изменить настоящий пароль на какой-нибудь другой

Функция `mysql_connect()` возвращает целое число, ссылающееся на соединение с базой данных. Этот идентификатор можно считать чем-то похожим на указатели на файлы, изученные вами в главе 6 «Работа с файлами». Это соединение необходимо сохранить в переменной (обычно я называю ее `$conn`), потому что многим функциям для работы с базами данных необходим доступ к соединению.

Выбираем базу данных

Соединение может содержать множество баз данных. Функция `mysql_set_db()` позволяет выбрать базу данных, с которой вы будете работать. Она работает точно так же, как команда SQL `USE`. Функция `mysql_set_db()` требует указания в качестве параметров названия выбираемой базы данных и соединение. Эта функция возвращает значение `FALSE`, если не удалось соединиться с указанной базой данных.

Создаем запрос

Создать запрос очень просто. Вот соответствующий код из `showHero.php`.

```
//создать запрос
$sql = "SELECT * FROM hero";
$result = mysql_query($sql, $conn);
```

Прежде всего, вы помещаете код SQL в переменную.

Когда вы вводили команды SQL в консоли SQL или в SQLyog, они должны были оканчиваться точкой с запятой. Когда ваша PHP-программа посылает команду в СУБД, точка с запятой добавляется автоматически, поэтому вам не надо завершать команды SQL точкой с запятой. Конечно же, вы будете присваивать эти команды в строке кода PHP, которая должна оканчиваться точкой с запятой.

Функция `mysql_query()` позволяет вам передавать команды SQL через соединение базе данных. С помощью `mysql_query()` вы можете отправить базе данных любую команду SQL, включая выражения для создания, обновления таблиц и запросы. Она возвращает специальный элемент, называемый *результирующий набор*. Если команда SQL была запросом, результирующая переменная будет содержать указатель на данные, которые мы подробно изучим в следующем шаге. Если команда является определением данных (команды, используемые для создания и изменения таблиц), результирующий объект обычно будет содержать строку, означающее успешное или неуспешное выполнение операции.

Получаем названия полей

Я буду печатать данные в таблицу HTML. Я мог бы создать «шапку» таблицы вручную, потому что я знаю, какие в ней содержаться поля, однако намного лучше получить информацию о полях непосредственно с помощью запроса, потому что не всегда известно, какие поля возвращает определенный запрос. Следующий фрагмент кода предназначен для выполнения этой работы.

```
print "<table border = 1>\n";
//получить названия полей
print "<tr>\n";
while ($field = mysql_fetch_field($result)){
  print "  <th>$field->name</th>\n";
} // завершение цикла while
print "</tr>\n\n";
```

Функция `mysql_fetch_field()` принимает в качестве одного из параметров результат выполнения запроса. После этого она получает следующее поле и сохраняет его в переменной `$field`. Если в результате выполнения запроса больше

не осталось полей, функция возвращает значение FALSE. Это позволяет использовать функцию получения поля в качестве условного выражения.

На самом деле переменная $field является объектом. Вы еще не использовали объекты PHP, однако на самом деле они не очень сложны. В данном случае объект $field очень похож на ассоциативный массив. У него есть несколько свойств (в качестве которых можно взять атрибуты поля). Объект поля имеет несколько атрибутов, приведенных в табл. 8.1.

Табл. 8.1. Наиболее часто используемые свойства объекта-поля

Свойство	Атрибут
max_length	Определяет длину поля (особенно важно для полей типа VARCHAR)
name	Название поля
primary_key	TRUE, если поле является первичным ключом
table	Название таблицы, которой принадлежит это поле
type	Тип данных этого поля

Наиболее частым видом использования объекта поля является определение названий всех полей с помощью запроса. Однако остальные атрибуты также могут использоваться в некоторых ситуациях. Вы можете найти полный список атрибутов в интерактивной помощи MySQL.

Для обращения к свойству объекта вы используете немного другой синтаксис. Заметьте, что я напечатал $field->name в таблицу HTML. Такой синтаксис позволяет обращаться к свойствам объекта по их названиям. На данный момент, если вы будете считать это воображаемым ассоциативным массивом, это в принципе будет соответствовать действительности.

Анализируем результирующий набор

Оставшаяся часть кода проверяет результирующий набор. Я воспроизведу ее здесь, чтобы освежить вашу память.

```
//получить данные строк и сохранить их в ассоциативный массив
while ($row = mysql_fetch_assoc($result)){
  print "<tr>\n";
  //просмотреть каждое поле
  foreach ($row as $col=>$val){
    print "  <td>$val</td>\n";
  } // завершение цикла foreach
  print "</tr>\n\n";
```

```
}// завершение цикла while
```

Функция `mysql_fetch_assoc()` получает следующую строку из результирующего набора. Она принимает указатель на результирующий набор и возвращает ассоциативный массив.

Существует несколько других аналогичных функций для извлечения строк из результирующего набора. `mysql_fetch_object()` *сохраняет строку в виде объекта аналогично тому, как это делает функция* `mysql_fetch_fields()`. *Функция* `mysql_fetch_array()` *возвращает массив, который можно использовать как обычный массив, ассоциативный массив или и то, и другое. Я в основном использую* `mysql_fetch_assoc()`, *потому что я думаю, что это самый простой способ работы для тех, кто не знаком с объектно-ориентированным синтаксисом. Конечно же, вы должны исследовать все возможные виды функций и пользоваться теми, которые наиболее удобны для вас.*

Если в результирующем наборе не осталось строк, `mysql_fetch_assoc()` вернет значение `FALSE`. Она обычно используется в качестве условия в цикле `while`, как я сделал это для получения каждой строки результирующего набора. Каждая строка будет соответствовать строке таблицы HTML, поэтому я печатаю HTML код для начала каждой строки в цикле `while`.

После того как вы получили строку, она сохраняется в ассоциативный массив. Вы можете проанализировать этот массив с помощью стандартного цикла `foreach`. Я присваиваю каждый элемент переменным `$col` и `$val`. На самом деле, в данном случае мне не нужна переменная `$col`, однако иногда очень полезно ее использовать. В цикле `foreach` я поместил код, который печатает текущее поле в ячейку таблицы.

Возвращаемся к программе приключенческой игры

Вспомните, в конце главы 7 вы создали базу данных для приключенческой игры. После того, как вы узнали, как соединиться с базой данных MySQL из программы PHP, вы готовы к написанию самой игры.

Соединяемся с базой данных приключений

После того как я создал базу данных, первая PHP программа, которую я написал, пыталась установить простейшее соединение с базой данных. Я хотел убедиться, что в ней содержаться корректные данные. Вот код этой программы.

```
<html>
<head>
<title>Show Adventure</title>
</head>
<body>
<?
$conn  = mysql_connect("localhost", "", "");
mysql_select_db("chapter7", $conn);
$sql = "SELECT * FROM adventure";
$result = mysql_query($sql);
while ($row = mysql_fetch_assoc($result)){
  foreach($row as $key=>$value){
    print "$key: $value<br>\n";
  } // завершение цикла foreach
  print "<hr>\n";

} // завершение цикла while
?>
</body>
</html>
```

Эта простая программа использовалась для соединения с базой данных и проверки, что все содержащиеся в ней значения корректны. Всякий раз, когда я создаю программу для работы с данными, я также пишу программу наподобие этой, которая быстро просматривает мои данные и убеждается, что все работает корректно. Нет никакого смысла продолжать, пока вы не будете знать, что можете создать самое простое соединение.

Я не привожу снимков экрана этой программы, потому что на них нет ничего особенного. Основной смысл такого метода заключается в создании простых вещей, которые затем постепенно усложняются.

Отображаем один сегмент

Сама игра состоит из повторяющихся вызовов программы `showSegment.php`. Эта программа получает в качестве одного из параметров идентификатор (id) сегмента и использует его для создания страницы на основе этой записи базы данных. Единственным сюрпризом является простота кода этой программы.

```
<html>
<head>
<title>Show Segment</title>
<style type = "text/css">
```

```
body {
  color:red
}
td {
  color: white;
  background-color: blue;
  width: 20%;
height: 3em;
  font-size: 20pt
}
</style>
</head>
<body>
<?
if (empty($room)){
  $room = 1;
} // завершение if

//соединение с базой данных
$conn  = mysql_connect("localhost", "", "");
$select = mysql_select_db("chapter7", $conn);
$sql = "SELECT * FROM adventure WHERE id = '$room'";
$result = mysql_query($sql);
$mainRow = mysql_fetch_assoc($result);

$theText = $mainRow["description"];
$northButton = buildButton("north");
$eastButton = buildButton("east");
$westButton = buildButton("west");
$southButton = buildButton("south");
$roomName = $mainRow["name"];

print <<<HERE
<center><h1>$roomName</h1></center>
<form method = "post">
<table border = 1>
<tr>
  <td></td>
  <td>$northButton</td>
  <td></td>
</tr>

<tr>
  <td>$eastButton</td>
      <td>$theText</td>
  <td>$westButton</td>
```

```
</tr>
<tr>
  <td></td>
  <td>$southButton</td>
  <td></td>
</tr>
</table>
<center>
<input type = "submit"
       value = "go">
</center>
</form>
HERE;
function buildButton($dir){
  //создает кнопку для заданного направления
  global $mainRow, $conn;
  $newID = $mainRow[$dir];
  //напечатать "newID is $newID";
  $query = "SELECT name FROM adventure WHERE id = $newID";
  $result = mysql_query($query, $conn);
  $row = mysql_fetch_assoc($result);
  $roomName = $row["name"];
  $buttonText = <<< HERE
  <input type = "radio"
         name = "room"
         value = "$newID">$roomName
HERE;
  return $buttonText;
} // завершение создания копки
?>
</body>
</html>
```

Создаем стиль CSS

Я начал HTML со стиля CSS. Моя программа визуально выглядит непривлекательно, однако добавление стиля CSS является ответом на отсутствие у меня дизайнерских способностей. Все, что мне необходимо сделать, – найти кого-нибудь, у кого все нормально с воображением, и попросить его переделать мою таблицу CSS, в результате чего я получу привлекательно выглядящую страницу.

Создаем соединение

Как обычно, программа начинается с некоторых настроек. Если пользователь не указал номер сегмента, я начну с комнаты номер 1, которая будет стартовой.

```
if (empty($room)){
  $room = 1;
} // завершение if
//соединение с базой данных
$conn  = mysql_connect("localhost", "", "");
$select = mysql_select_db("chapter7", $conn);
$sql = "SELECT * FROM adventure WHERE id = '$room'";
$result = mysql_query($sql);
$mainRow = mysql_fetch_assoc($result);
$theText = $mainRow["description"];
```

После этого я просто соединяюсь с базой данных и выбираю запись, соответствующую текущему номеру комнаты. Результат выполнения этого запроса сохраняется в переменной `$mainRow` в качестве ассоциативного массива.

Генерируем переменные для кода

Большая часть программы пишет код HTML для текущей записи на экран. Для упрощения я решил на всякий случай создать несколько переменных.

```
$theText = $mainRow["description"];
$roomName = $mainRow["name"];
$northButton = buildButton("north");
$eastButton = buildButton("east");
$westButton = buildButton("west");
$southButton = buildButton("south");
```

Я сохранил поле описания текущей строки в переменной `$theText`. Я сделал аналогичную переменную для названия комнаты.

Возвращаясь к реальности

Не обязательно хранить поле описания в переменной, однако я буду преобразовывать это значение в код HTML, и я обнаружил, что преобразование значений ассоциативного массива может быть нетривиальным. В общем случае, если я хочу преобразовать ассоциативное значение, я копирую его во временную переменную. Используемый мною способ намного проще

Переменные для кнопки немного отличаются. Я решил создать HTML кнопку опции для представления мест, куда может пойти пользователь. Я буду использовать специальную функцию `buildButton()` для создания каждой кнопки.

Создаем функцию buildButton()

Процедура создания кнопок повторяется достаточно часто, чтобы вынести ее в отдельную функцию. Каждая кнопка является переключателем, соответствующим направлению движения. Переключатель будет содержать значение, получаемое из соответствующего значения направления движения из текущей записи. Если северное поле текущей записи содержит число 12 (это означает, что если пользователь пойдет на север, необходимо загрузить данные записи 12), значение переключателя должно быть равным 12. Самым сложным является получение соответствующей метки. В текущей записи хранится только идентификатор следующей комнаты. Если вы хотите отобразить название следующей комнаты, вам необходимо сделать еще один запрос базе данных. Как раз это и делает функция `buildButton()`.

```
function buildButton($dir){
  //создает кнопку для заданного направления
  global $mainRow, $conn;
  $newID = $mainRow[$dir];
  //напечатать "newID is $newID";
  $query = "SELECT name FROM adventure WHERE id = $newID";
  $result = mysql_query($query, $conn);
  $row = mysql_fetch_assoc($result);
  $roomName = $row["name"];
  $buttonText = <<< HERE
  <input type = "radio"
         name = "room"
         value = "$newID">$roomName
HERE;
  return $buttonText;
} // завершение создания кнопки
```

Эта функция заимствует массив `$mainRow` (который содержит значение основной записи этой таблицы) и соединение, записанное в `$conn`. Я получаю идентификатор (ID) для этой кнопки из массива `$mainRow` и сохраняю его в локальную переменную. `buildButton()` принимает в качестве параметра название направления движения. Это направление должны быть названием одного из полей направления.

Я повторяю процесс построения запроса, создавая запрос, который требует только строку, ассоциированную с новым идентификатором. После этого я

получаю название комнаты из этого массива. Как только это сделано, очень просто создать текст переключателя. Переключатель называется так же, как и комната, поэтому во время следующего вызова программы, переменная `$room` будет соответствовать выбранному переключателю.

Завершаем создании HTML

Все, что осталось сделать, – это добавить на форму кнопку `Submit` и закрыть форму и HTML. Самым потрясающим является то, что это все, что надо сделать. Этого кода достаточно, чтобы пользователь мог играть в игру. Необходимо настроить структуру данных, однако затем все, что необходимо сделать, – это предоставить ссылку на первую запись (вызывав программу `showSegment.php` без параметров), после чего программа будет вызывать сама себя.

Просматриваем и выбираем записи

Я предполагаю, что вы могли бы остановиться здесь, потому что игра уже работает, однако всей прелестью такой структуры данных является ее гибкость. Потребуется приложить минимум усилий, чтобы создать редактор, позволяющий вам добавлять и изменять записи.

На самом деле редактор требует создания двух программ. Первая (показанная на рис. 8.2) печатает сводку всей игры и позволяет пользователю редактировать любой узел.

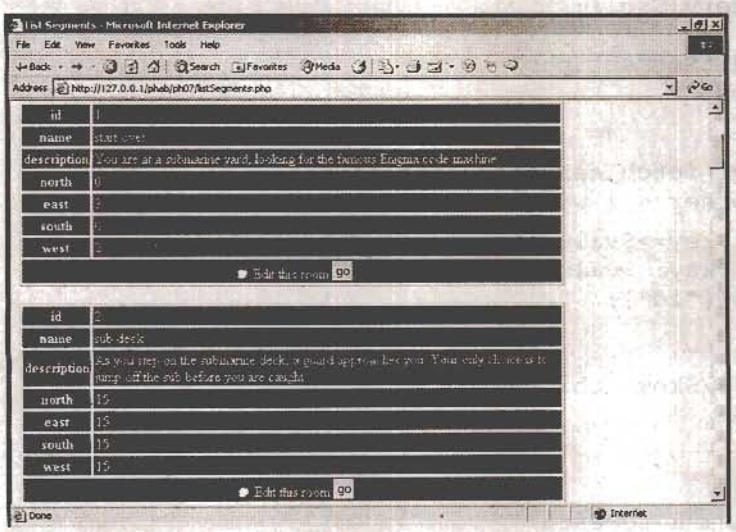


Рис. 8.2. Программа `listSegments` приводит все данные и позволяет пользователю редактировать записи

На самом деле код для программы `listSegments.php` очень похож на код программы `showAdventure.php`, виденный вами ранее. Он просто немного изменен, чтобы иметь возможность записывать данные в таблицу, и содержит форму для вызова редактора, когда пользователь выберет запись, которую хочет редактировать.

```
<html>
<head>
<title>List Segments</title>
<style type = "text/css">
body {
  color:red
}
td, th {
  color: white;
  background-color: blue;
}
</style>
</head>
<body>
<?
$conn  = mysql_connect("localhost", "", "");
$select = mysql_select_db("chapter7", $conn);
$sql = "SELECT * FROM adventure";
$result = mysql_query($sql);
print <<<HERE
<form action = "editSegment.php"
      method = "post">
HERE;
while ($row = mysql_fetch_assoc($result)){
  print "<table border = 1 width = 80%>\n";
  foreach($row as $key=>$value){
    //напечатать "$key: $value<br>\n";
    $roomNum = $row["id"];
    print <<<HERE
  <tr>
    <th width = 10%>$key</th>
    <td>$value</td>
  </tr>
HERE;
  } //завершение цикла foreach
  print <<<HERE
```

```
  <tr>
    <td colspan = 2><center>
      <input type = "radio"
             name = "room"
             value = "$roomNum">
      Edit this room
      <input type = "submit"
             value = "go">
    </center></td>
  </tr>
</table><br>
HERE;
} // завершение цикла while
?>
<center>
<input type = "submit"
       value = "edit indicated room">
</center>
</form>
</body>
</html>
```

Вся программа находится в форме, которая при активации вызовет `editSegment.php`. Программа открывает соединение с базой данных и получает все ее элементы. Она создает таблицу HTML для каждой записи. Каждая таблица содержит переключатель, имеющий название «room» и значение, соответствующее номеру текущей комнаты. Каждая таблица также содержит копию кнопки `Submit`, поэтому пользователю нет нужды прокручивать всю страницу для отправки формы.

Редактируем запись

После того как пользователь выбрал запись в программе `listSegments.php`, в действие вступает программа `editSegment.php` (показанная на рис. 8.3).

Очень важно понимать, что программа `editSegment` на самом деле *не* изменяет запись в базе данных. Вместо этого она получает форму, содержащую текущие значения выбранной записи, и разрешает пользователю определить ее новые значения. Страница `editSegment` является еще одной формой. Когда пользователь отправляет эту форму, управление передается еще одной программе, которая непосредственно изменяет базу данных. Код `editSegment` на самом деле очень похож на аналогичный код, используемый для отображения сегмента в игре. Основной разницей является то, что все данные записи располагаются в редактируемых полях.

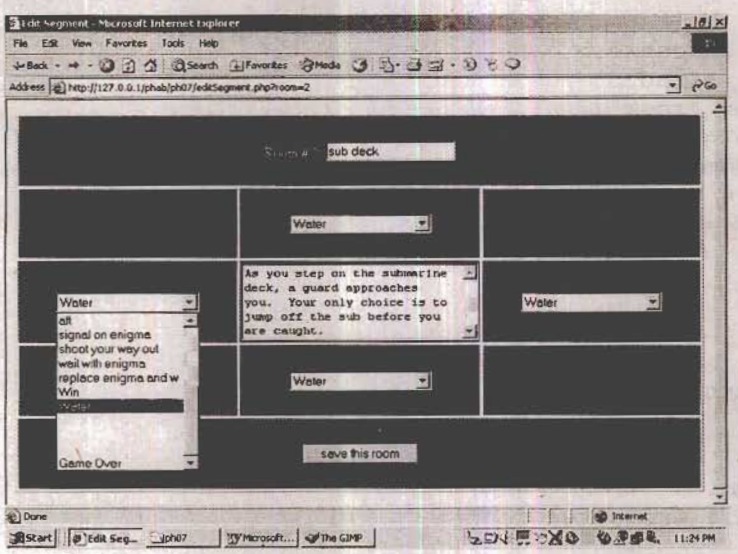


Рис. 8.3. Программа редактирования записи отображает данные выбранной записи и позволяет пользователю изменять их

Внимательно посмотрите на то, как разработчик игры может выбрать комнату, в которую можно перейти, для каждого положения. «Выпадающий» список отображает названия всех существующих комнат. Такое устройство позволяет разработчику игры работать непосредственно с названиями комнат, даже если в базе данных хранятся только их номера.

```
<html>
<head>
<title>Edit Segment</title>
<style type = "text/css">
body {
  color:red
}
td {
  color: white;
  background-color: blue;
  width: 20%;
  height: 5em;
  text-align: center;
}
</style>
</head>
<body>
<?
```

```
if (empty($room)){
  $room = 0;
} // завершение if

//соединиться с базой данных
$conn  = mysql_connect("localhost", "", "");
$select = mysql_select_db("chapter7", $conn);

$sql = "SELECT * FROM adventure WHERE id = '$room'";
$result = mysql_query($sql);
$mainRow = mysql_fetch_assoc($result);
$theText = $mainRow["description"];
$roomName = $mainRow["name"];
$northList = makeList("north", $mainRow["north"]);
$westList = makeList("west", $mainRow["west"]);
$eastList = makeList("east", $mainRow["east"]);
$southList = makeList("south", $mainRow["south"]);
$roomNum = $mainRow["id"];

print <<<HERE

<form action = "saveRoom.php"
      method = "post">
<table border = 1>
<tr>
  <td colspan = 3>
    Room # $roomNum:
    <input type = "text"
           name = "name"
           value = "$roomName">
    <input type = "hidden"
           name = "id"
           value = "$roomNum">
  </td>
</tr>

<tr>
  <td></td>
  <td>$northList</td>
  <td></td>
</tr>

<tr>
  <td>$westList</td>
  <td>
    <textarea rows = 5 cols = 30 name = "description">$theText</tex-
tarea>
      </td>
  <td>$eastList</td>
```

```
</tr>
<tr>
  <td></td>
  <td>$southList</td>
  <td></td>
</tr>
<tr>
  <td colspan = 3>
    <input type = "submit"
           value = "save this room">
  </td>
</table>
</form>
HERE;
function makeList($dir, $current){
  //создать список всех доступных мест системы
  global $conn;
  $listCode = "<select name = $dir>\n";
  $sql = "SELECT id, name FROM adventure";
  $result = mysql_query($sql);
  $rowNum = 0;
  while ($row = mysql_fetch_assoc($result)){
    $id = $row["id"];
  $placeName = $row["name"];
    $listCode .= "  <option value = $id\n";
    //выбрать эту опцию, если она помечена
    if ($rowNum == $current){
      $listCode .= "          selected\n";
    } // завершение if

    $listCode .= ">$placeName</option>\n";
    $rowNum++;
  } // завершение цикла while
  return $listCode;
} // завершение makeList
?>
</body>
</html>
```

Генерируем переменные

После установления стандартного соединения с базой данных, код создает несколько переменных. Некоторые из этих переменных (`$theText`, `$roomName` и `$roomNum`) являются упрощениями ассоциативного массива. Остальные переменные заполняются в результате вызова функции `makeList()`. Задачей этой функции является возврат HTML списка, содержащего названия комнат для каждого сегмента базы данных. Этот список будет настроен так, чтобы номер комнаты, ассоциированный с отображаемым полем, будет задан в качестве значения по умолчанию.

Печатаем код HTML

Основная часть этой программы предназначена для создания HTML кода. В данном случае этот код является большой таблицей, находящейся в форме. Каждое поле записи имеет соответствующий ему элемент формы. Когда пользователь отправляет эту форму, она должна содержать все данные, необходимые для обновления записи в базе данных. Единственное, что пользователь не может изменять непосредственно, – это номер комнаты. Он хранится в скрытом поле. Номера комнат для каждого направления хранятся в списках. Все остальные данные располагаются в соответствующим образом названных текстовых полях.

Создаем списки

Чтобы создать списки, необходимо немного подумать.

Функция `makeList()` принимает два параметра. Параметр `$dir` содержит название поля направления текущего списка. Параметр `$current` содержит информацию о том, какая комната выбрана в данный момент для этого конкретного поля текущей записи. Единственной глобальной переменной является дескриптор соединения `$conn`. Переменная `$listCode` будет содержать фактический код HTML для списка, который будет возвращен основной программе.

Эта функция делает запрос базе данных для получения всех названий комнат. Каждое название добавляется в код списка в соответствующий момент времени и с соответствующим численным значением. Если номер записи соответствует текущему значению записи, в коде HTML обозначается, что текущий элемент должен быть выбран в списке.

Записываем изменения в базу данных

Необходима еще одна программа. Программа `editSegment.php` позволяет пользователю редактировать данные, однако когда пользователь закончит работу, он или она отправит форму, что приведет к вызову программы `saveRoom.php`. Я не буду еще раз приводить снимок экрана этой программы, потому что то, что видно на экране, не важно. Однако эта программа фактически обновляет базу данных с помощью выбранных пользователем значений.

```
<head>
<title>SaveRoom.php</title>
</head>
<body>
<?
//после того, как комната была отредактирована в editSegment,
эта программа
 //соответствующим образом обновляет базу данных
//соединиться с базой данных
$conn  = mysql_connect("localhost", "", "");
$select = mysql_select_db("chapter7", $conn);
$sql = <<<HERE
UPDATE adventure
SET
  name = '$name',
  description = '$description',
  north = $north,
  east = $east,
  south = $south,
  west = $west
WHERE
  id = $id
HERE;
//напечатать $sql;
$result = mysql_query($sql);
if ($result){
  print "<h3>$name room updated successfully</h3>\n";
  print "<a href = \"listSegments.php\">view the rooms</a>\n";
} else {
  print "<h3>There was a problem with the database</h3>\n";
} // завершение if
?>
</body>
</html>
```

Эта программа начинается со стандартного соединения с базой данных. После этого создается выражение SQL UPDATE. Это выражение достаточно простое, потому что вся работа была выполнена предыдущей программой. После этого я просто посылаю запрос базе данных и проверяю результат его выполнения. Команда UPDATE не возвращает набор записей, как это делает команда SELECT. Вместо этого она возвращает значение FALSE, если в результате выполнения команды про-

изошли ошибки. Если запрос на обновление был выполнен успешно, я сообщаю об этом пользователю и предоставляю ссылку на программу `listSegments`. Если были какие-либо ошибки, я предоставляю некоторую (не очень полезную) информацию пользователю.

Итоги

В этой главе вы начали использовать внешние программы для управления данными. Вы увидели, как можно использовать MySQL для интерпретации простых выражений SQL, позволяющих определять данные и управлять ими. Вы создали базу данных непосредственно в консоли MySQL и научились создавать и управлять базами данных с помощью SQLyog. Вы использовали все приобретенные навыки для создания интересной и расширяемой игры.

Домашнее задание

1. Добавьте команду «новая комната» в генератор приключений. *Подсказка:* Подумайте о том, как я создал новый тест в программе машины тестирования из главы 6.
2. Напишите программы PHP, позволяющие просматривать, добавлять и редактировать записи в списке телефонов.
3. Напишите программу, которая запрашивает имя пользователя и ищет в базе данных этого пользователя.
4. Создайте интерфейс для другой простой базы данных.

Глава 9

Нормализация данных

В последних двух главах вы узнали, как создать простую базу данных и соединиться с ней из PHP-программы. PHP и MySQL замечательно подходят для работы с простыми базами данных. Однако большинство проблем, возникающих в действительности, настолько сложны, что их данные не могут быть помещены в одну таблицу. Проектировщики баз данных разработали стандартные методы обработки сложных данных, которые сокращают избыточность, улучшают эффективность и предоставляют гибкость. В этой главе вы узнаете, как использовать реляционную модель для создания сложных баз данных, содержащих в себе множество таблиц. В частности, вы узнаете:

- как работает реляционная модель данных;
- как применять деловые правила для данных;
- как создавать диаграммы взаимодействия таблиц для моделирования данны;.
- как создавать многотабличные базы данных;
- как использовать связи для соединения таблиц;
- как создать таблицу связи для моделирования отношения «многие ко многим»;
- как оптимизировать проектирование таблицы для последующего программирования.

Введение в базу данных шпионов

В этой главе вы создадите базу данных, которая управляет вашей международной сетью шпионов. (У вас же *есть* международная сеть шпионов, разве нет?) Спасение мира является сложной задачей, поэтому чтобы следить за всеми вашими агентами, вам потребуется база данных. Секретным агентам поручаются различные задания по всему миру, и у каждого агента есть определенные способности. Примеры этой главы иллюстрируют все аспекты создания такой базы данных. Вы увидите, как создать такую базу данных в MySQL. В последней главе вы будете использовать эту базу данных для создания действительно мощной системы шпионажа в PHP.

База данных шпионов отражает некоторые особенности моей шпионской организации (называющейся Пантеон Гуманных Исполнителей (Pantheon of Humanitarian Performance), или PHP).

- У каждого агента есть кодовое имя.
- Каждый агент может иметь несколько способностей.
- Несколько агентов могут иметь одну и ту же способность.
- Каждый агент в определенный момент времени может выполнять только одну операцию.
- Каждую операцию могут выполнять несколько агентов.
- Местоположение шпиона определяется выполняемой им операцией.
- У каждой операции есть только одно местоположение.

Этот список правил помогает объяснить некоторые характеристики данных. На языке баз данных они называются *деловыми правилами*. Мне необходимо спроектировать базу данных таким образом, чтобы она соответствовала этим правилам.

Возвращаясь к реальности

Я задал этот набор правил довольно произвольным образом, но они помогут сделать мою базу данных настолько простой, насколько это возможно, и в то же время позволяют проиллюстрировать большинство основных проблем, возникающих при проектировании данных. Обычно вам не приходится создавать деловые правила. Вместо этого вам придется их узнавать, разговаривая с теми, кто повседневно использует данные

База данных badSpy

Как вы видели в главе 7 «Использование MySQL для создания баз данных», создать таблицу данных не очень сложно, особенно если у вас есть такой инструмент, как SQLyog. На рис. 9.1 показана схема первого варианта базы данных шпионов.

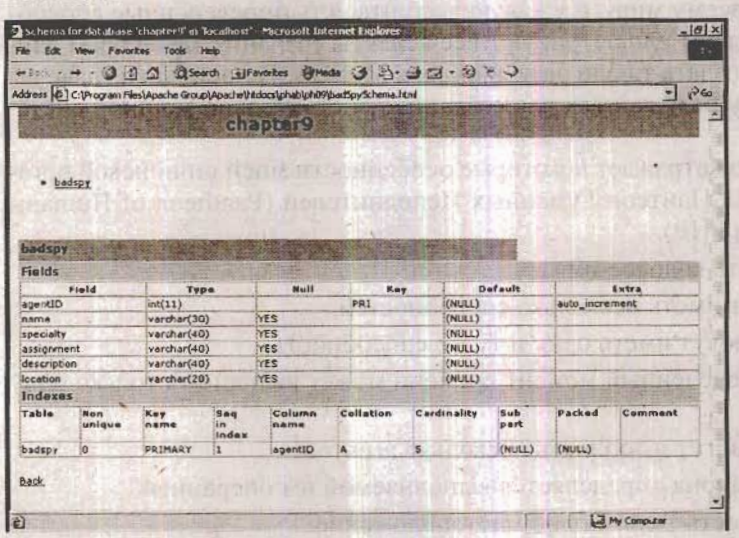


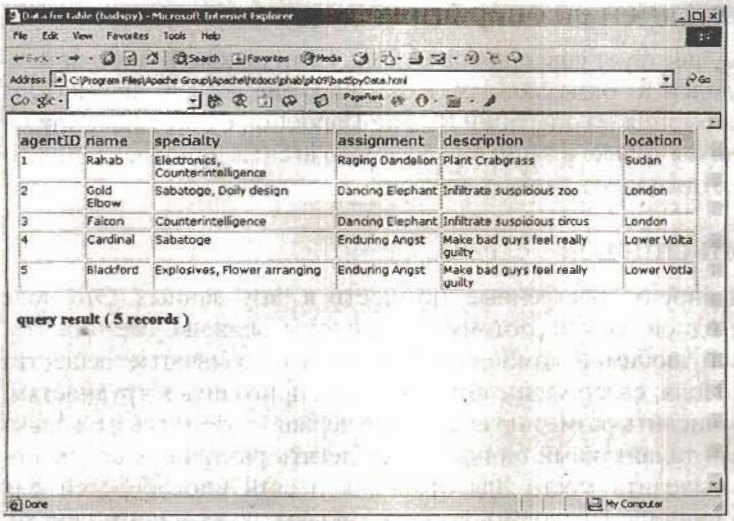
Рис. 9.1. Схема базы данных `badSpy` выглядит достаточно корректно

На первый взгляд архитектура базы данных `badSpy` выглядит достаточно «рабочей», но как только вы начнете добавлять в таблицу данные, вы столкнетесь с проблемами. На рис. 9.2 показана база данных `badSpy` после того, как я добавил в нее информацию о некоторых агентах.

После того как вы начнете добавлять данные в таблицу, вы увидите, что возникает несколько проблем. Посмотрите внимательно на рис. 9.2, и вы увидите потенциальные недостатки.

Проблема противоречивости данных

Запись Gold Elbow говорит, что операция Dancing Elephant заключается во внедрении в подозрительный зоопарк. Запись Falcon говорит, что та же самая операция заключается во внедрении в подозрительный цирк. В целях этого примера я ожидаю, что у задания есть только одно описание, поэтому одно из этих описаний неверно. Не существует способа определить, какое из описаний неправильно, посмотрев на данные таблицы, поэтому обе записи подозрительны. Кроме этого, сложно сказать, пройдет ли операция Enduring Angst в Lower Volta или в Lower



agentID	name	specialty	assignment	description	location
1	Rahab	Electronics, Counterintelligence	Raging Dandelion	Plant Crabgrass	Sudan
2	Gold Elbow	Sabotoge, Doily design	Dancing Elephant	Infiltrate suspicious zoo	London
3	Falcon	Counterintelligence	Dancing Elephant	Infiltrate suspicious circus	London
4	Cardinal	Sabatoge	Enduring Angst	Make bad guys feel really guilty	Lower Volta
5	Blackford	Explosives, Flower arranging	Enduring Angst	Make bad guys feel really guilty	Lower Votla

query result (5 records)

Рис. 9.2. База данных `badSpy` после того, как я добавил в нее несколько агентов

Votla, потому что две записи, описывающие данное задание, имеют разное написание. Противоречие цирк/зоопарк и проблема в написании Volta/Votla возникают по одной и той же причине. В обоих случаях человек, вводящий данные (вероятно, какой-нибудь неопытный гражданский служащий, потому что мастера международного шпионажа слишком заняты, чтобы заниматься вводом данных), должен был ввести одни и те же данные в базу данных несколько раз. Такого рода несоответствие и приводит к появлению разнообразных проблем. Если необходимо, чтобы человек, вводящий данные, ввел несколько раз одни и те же данные, в результате получатся несоответствия. Различные люди используют разные аббревиатуры. Вы можете встретить различное написание одних и тех же слов. Некоторые люди просто не будут вводить данные, если они слишком сложные. Когда это происходит, вы не можете полагаться на данные (цирк или зоопарк?). Вы также не можете проводить поиск в данных. (Если я просмотрю все операции в Lower Volta, я пропущу операцию под названием Blackford, потому что в ее графе Location («Местоположение») стоит Lower Votla.) Если вы внимательно посмотрите, вы увидите, что я неправильно написал слово “sabatoge”[1]. Будет очень сложно найти все места, в которых это слово написано неправильно, и исправить его.

[1] Правильным написанием будет sabotage. – *Примеч. пер.*

Проблема с информацией об операциях

В этой базе данных существует еще одна проблема. Если по какой-либо причине агент Rahab будет удален из базы данных (возможно, она все это время была двойным агентом), информация об операции Raging Dandelion будет также удалена, потому что она хранится только в записи удаляемого агента. Данные об операции должны храниться отдельно от данных агентов.

Проблемы с полями списков

Поле специальности вносит собственные трудности в базу данных. Это поле может содержать более одной записи, потому что шпионы должны уметь делать множество вещей. (Моей любимой комбинацией является взрывчатые вещества и составление букетов.) Поля, содержащие списки, могут приводить к трудностям. Прежде всего, трудно вычислить размер поля, которое должно содержать несколько записей. Если ваш самый талантливый шпион имеет десять различных способностей, вам необходимо выделить место для хранения десяти способностей для каждой записи каждого шпиона. Проводить поиск по полям, содержащим списки, может быть затруднительно. Вы можете попытаться ввести несколько различных полей различных навыков (например, поля специализация 1, специализация 2, специализация 3), однако это не решает всей проблемы. Намного лучше будет иметь гибкую систему, которая может содержать любое число навыков. Плоская файловая система[1], которую вы видели в базе данных `badSpy`, не позволяет сделать этого.

Проектируем улучшенную структуру данных

База данных мастеров шпионажа на самом деле не так уж и сложна, однако база данных `badSpy` показала вам, что даже с простой базой могут возникать трудности. Эта база данных является очень важной, потому что она будет использоваться для спасения свободного мира, поэтому она заслуживает дополнительного внимания. К счастью, разработчики данных придумали несколько способов размышлений над структурой данных. Зачастую лучше отойти от компьютера и хорошо подумать о способе использования ваших данных, прежде чем начинать писать код.

---

[1] система организации файлов, при которой они не могут иметь одинаковых имен, даже если эти файлы хранятся в разных каталогах. – *Примеч. науч. ред.*

Определяем правила для правильного проектирования данных

Разработчики данных придумали несколько правил, позволяющих создавать корректные базы данных.

- Разбейте ваши данные на несколько таблиц.
- Ни одно из полей не может иметь несколько значений.
- Не дублируйте данные.
- Сделайте так, чтобы каждая таблица описывала только одну сущность.
- Для каждой таблицы создайте одиночное поле первичного ключа.

База данных, соответствующая всем этим правилам, избежит большей части проблем, выявленных в базе данных `badSpy`. К счастью, существует несколько широко известных процедур улучшения базы данных, чтобы она соответствовала всем этим правилам.

Нормализация данных

Программисты данных попытались избежать проблем, выявленных в базе данных `badSpy`, с помощью процесса, называемого *нормализацией данных*. Основной идеей нормализации является разбиение базы данных на последовательность таблиц. Если каждая их этих таблиц будет спроектирована корректно, база данных, скорее всего, не будет иметь проблем, описанных ранее в этой главе. О нормализации были написаны целые книги, однако весь процесс разбивается на три основных шага, называемого *нормальными формами*.

Первая нормальная форма: исключение повторений

Целью первой нормальной формы (Normal Form) (иногда используется аббревиатура 1NF) является исключение повторений в базе данных. Основным источником проблем в базе данных `badSpy` является поле специальности. Одним из решений этой проблемы будет разбиение базы данных на две таблицы: одной для агентов, а другой – для специальностей.

Проектировщики данных, похоже, играют на однострунном банджо. Решением почти любой проблемы проектирования данных является создание новой таблицы. Как вы увидите, для того – чтобы решить, что именно будет входить в эту новую таблицу, необходимо обладать определенным талантом.

Эти две таблицы выглядят аналогично таблицам 9.1 и 9.2.

Табл. 9.1. Таблица агентов в 1NF[a]

agentID	Name	Assignment	Description	Location
1	Rahab	Raging Dandelion	Plant Crabgrass	Sudan
2	Gold Elbow	Dancing Elephant	Infiltrate suspicious zoo	London
3	Falcon	Dancing Elephant	Infiltrate suspicious circus	London

a. agentID – идентификатор агента, name – имя, assignment – задание, description – описание, location – местоположение

Табл. 9.2. Таблица специальностей в 1NF[a]

SpecialityID	Name
1	Electronics (Электроника)
2	Counterintelligence (Контрразведка)
3	Sabotage (Саботаж)

a. SpecialityID – идентификатор специальности, Name – название

Заметьте, что я привел не все данные в этих таблицах, однако достаточно информации для того, чтобы вы поняли принцип их формирования. Кроме того, пока что нет хорошего способа объединения этих таблиц, вы рассмотрите его позднее в этой главе.

Вторая нормальная форма: исключение избыточности

Как только вы привели все ваши таблицы к первой нормальной форме, следующим шагом будет работа с вопросом избыточности. Избыточность зачастую возникает из-за того, что данные были введены более одного раза. Для исправления этого вам необходимо (вы угадали) создать новые таблицы. Таблица агентов может быть улучшена путем перемещения всех данных об операциях в отдельную таблицу. На рис. 9.3 показана специальная диаграмма, называемая диаграмма связей сущностей (Entity Relationship diagram), которая иллюстрирует взаимосвязи между этими таблицами.

Диаграмма связей сущностей (ER-диаграмма) используется для отображения взаимосвязей между элементами данных. В данной ситуации я хорошенько подумал о данных в базе данных шпионов. В результате я понял, что необходимо создать три различные таблицы. Отделяя данные операции от данных агентов, я избежал избы-

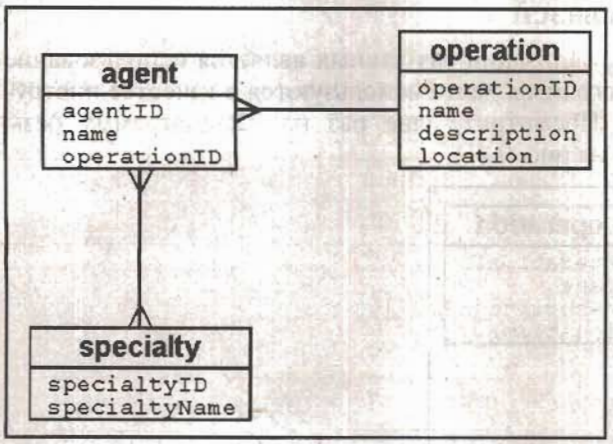


Рис. 9.3. Простая диаграмма связей сущностей для базы данных шпионов

точности, потому что пользователь будет вводить данные операции только один раз. Это позволит решить несколько проблем, присутствовавших в исходной базе данных. Это также исправляет ситуацию, когда данные операции теряются из-за того, что шпион стал двойным агентом. (Меня очень мучают такие измены.)

Третья нормальная форма: проверка функциональных зависимостей

Третья нормальная форма работает с элементами, содержащимися в каждой таблице. Чтобы таблица была в третьей нормальной форме, эта таблица должна иметь один первичный ключ и каждое поле таблицы должно зависеть только от него. Например, поле описания является описанием *операции*, а не *агента*, поэтому оно принадлежит таблице операций. В третьей фазе нормализации вы рассматриваете все части данных и убеждаетесь, что они непосредственно связаны с таблицей, в которой они находятся. Если же такой связи нет, вам необходимо либо переместить эти данные в соответствующую таблицу, либо создать для них новую таблицу.

Возвращаясь к реальности

Вы могли заметить, что моя база данных стала автоматически соответствовать третьей нормальной форме после приведения ко второй нормальной форме. Это типично для маленьких таблиц, однако для больших и сложных таблиц, используемых для описания настоящих предприятий, такой случай редок. Даже если вам кажется, что таблица уже находится в третьей нормальной форме, необходимо перебрать все поля и посмотреть, относятся ли они к таблице

Определяем типы взаимосвязей

Самым простым способом нормализации баз данных является использование их художественного вида. ER-диаграммы часто используются в качестве инструмента проектирования данных. Посмотрите еще раз на ER-диаграмму базы данных шпионов, изображенную на рис. 9.4.

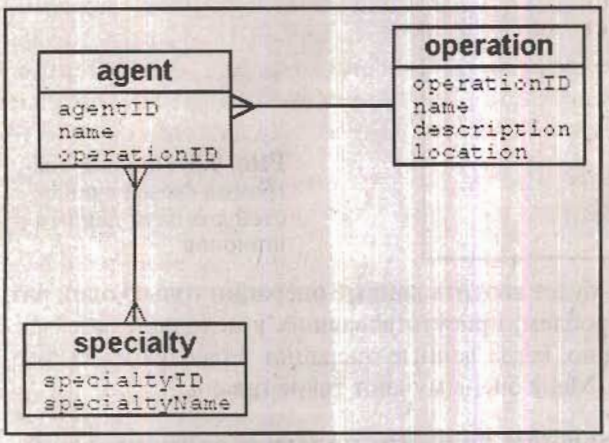


Рис. 9.4. ER-диаграмма для базы данных шпионов

Эта диаграмма иллюстрирует три таблицы базы данных шпионов (по крайне мере, на данный момент) и их взаимосвязи. Каждая таблица находится в прямоугольнике, а строки между ними представляют собой их взаимосвязи. Внимательно посмотрите на линии взаимосвязей. На их концах располагаются так называемые «птичьи лапки», обозначающие специальные характеристики взаимосвязей. Существует три основных вида взаимосвязей (по крайне мере, в этом упрощенном рассмотрении моделирования данных).

Распознаем взаимосвязи «один к одному»

Взаимосвязи «один к одному» возникают, когда каждому экземпляру сущности A соответствует только один экземпляр сущности B. Взаимосвязь «один к одному» описывается в виде простой линии между двумя сущностями, на концах которой ничего нет.

На самом деле такие взаимосвязи редки, потому что если две сущности так тесно связаны, их можно без последствий объединить в одну таблицу. На ER-диаграмме базы данных шпионов, изображенной на рис. 9.4, нет взаимосвязей «один к одному».

Описываем взаимосвязи «многие к одному»

Взаимосвязи вида «один ко многим» (и «многие к одному») возникают, когда одной сущности может соответствовать один или более экземпляров другой сущности. Например, в каждой операции может быть задействовано несколько шпионов, однако (в целях этого примера) каждый агент в отдельный момент времени может быть только на одном задании. Поэтому взаимосвязь агентов и операций принимается как «многие к одному», потому что шпион может участвовать только в одной операции, однако в операции может участвовать несколько агентов. В этом очень простом варианте ER нотации, я использую так называемые «птичьи лапки» для обозначения множественной стороны взаимоотношения.

На самом деле существует несколько различных видов взаимосвязей «один ко многим», каждый из которых используется в различных случаях и обозначается по-разному. Для этого обзора мы не будем их подразделять и будем использовать для них общий символ «птичьих лапок», однако как только вы начнете создавать более сложные базы данных, вам захочется более пристально изучить диаграммы данных. Для этого вам будет необходимо посмотреть соответствующие книги по нормализации данных и разработке программного обеспечения. Более того, нормализация данных – намного более широкая тема, которая лишь поверхностно рассмотрена в данной книге. Однажды вам захочется изучить эту тему более подробно.

Распознаем взаимосвязи «многие ко многим»

Последним типом взаимосвязей, показанным на ER-диаграмме базы данных шпионов, является отношение «многие ко многим». Этот тип отношения возникает, когда каждой сущности может соответствовать множество экземпляров другой сущности. Такой тип взаимосвязи существует между агентами и навыками, потому что один агент может иметь несколько способностей, а каждым отдельным навыком может может обладать несколько агентов. Отношение «многие ко многим» обычно изображается с символами «птичьих лапок» на обоих сторонах соединяющей линии.

Очень важно сгенерировать ER-диаграмму для ваших данных, включающую типы взаимосвязей, потому что для создания каждого типа взаимосвязи существует различные стратегии. Вы увидите использование этих стратегий, когда я буду создавать SQL для улучшенной версии базы данных шпионов.

Возвращаясь к реальности

Профессиональные программисты обычно используют дорогие программы для построения диаграмм данных, однако вам будет достаточно листка бумаги и ручки для изображения ER фигур. Я делаю свои лучшие разработки данных с ассистентом, который рисует на белой доске. Я люблю громко обсуждать ход проектирования и смотреть на результаты в большом формате. После того как у меня начало что-то получаться, я обычно использую редактор векторной графики для создания более формальной версии придуманной диаграммы. Такой тип инструмента очень полезен, потому что позволяет вам соединять элементы и уже содержит линии с «птичьими лапками» и позволяет вам передвигать элементы, не нарушая соединений между ними. Dia является замечательной программой с открытым исходным кодом для рисования любых видов диаграмм. Я использовал ее для создания всех ER-диаграмм, приведенных в этой главе

Создаем таблицу данных

Как только вы разработали структуру данных в соответствии с правилами нормализации, вы готовы к созданию таблиц данных в SQL. Таблицы необходимо создавать очень аккуратно, чтобы избежать проблем. Я предпочитаю создавать мои таблицы с помощью SQL скрипта, чтобы я мог легко переделать базу данных, если (хорошо, когда) мои программы испортят структуру данных. Кроме того, вражеские агенты всегда готовятся организовать диверсии, чтобы сорвать мои операции.

Я также добавляю множество тестовых данных в скрипт. На ранних этапах нет никакой нужды работать с настоящими данными, потому что во время разработки вы обязательно испортите структуру данных. Однако хорошей идеей является работа с данными, которые являются копией настоящих данных. Тестовые данные должны помочь выявить проблемы, которые могут возникнуть с реальными данными (например, что будет, если у человека нет отчества?). Полная версия скрипта для создания базы данных шпионов располагается на нашем сайте.

Настраиваем систему

Я начал свой SQL скрипт с комментариев, которые описывают базу данных и некоторые проектные решения, которые я сделал для создания базы данных.

```
########################################
# buildSpy.sql
```

```
# создает и  заполняет все базы данных для примера со шпионами
# использует mysql - должен легко адаптироваться к другим СУРБД
# автор Энди Харрис (Andy Harris), книга «"PHP/MySQL с нуля"»
#######################################

#######################################
# соглашения
#######################################
# первичный ключ = название таблицы . ID
# первичный ключ всегда является первым полем
# все первичные ключи автоматически нумеруются
# все названия полей создаются по следующей схеме: первый символ
первого
# логического слова с маленькой буквы, остальные логические
слова с
# большой
# только таблицы связей используют символ подчеркивания
# внешние ключи также выделены, хотя mySQL не требует, чтобы
каждая таблица, # используемая в качестве внешней ссылки,
содержала поле названия
#######################################

#######################################
#настройки
#######################################

use chapter9;
DROP TABLE IF EXISTS badSpy;
DROP TABLE IF EXISTS agent;
DROP TABLE IF EXISTS operation;
DROP TABLE IF EXISTS specialty;
DROP TABLE IF EXISTS agent_specialty;
DROP TABLE IF EXISTS spyFirst;
```

Заметьте, что я определил набор соглашений. Эти правила помогут упростить управление моей базой данных, когда она станет сложнее. Некоторые из этих правил пока могут не иметь для вас смысла (например, потому что я еще не рассказал, что такое внешний ключ), однако важным является то, что я четко определил правила, которые помогут мне позднее.

После этого код выбирает базу данных `chapter9` и удаляет все таблицы, если они уже существуют. Такое поведение позволяет убедиться, что я буду работать с корректной версией данных.

Создаем таблицу агентов

Нормализованная таблица агентов на самом деле очень проста. Она показана в табл. 9.3.

Табл. 9.3. Таблица агентов[a]

AgenID	name	operationID
1	Bond	1
2	Falcon	1
3	Cardinal	2
4	Blackford	2

a. AgentID – идентификатор агента, name – имя агента, operationID – идентификатор операции

Единственными данными, оставшимися в таблице агентов, являются имя агента и численное поле выполняемой им операции. Поле `operationID` используется в качестве «соединителя» таблиц агентов и операций.

Я добавил некоторые вещи для улучшения SQL кода создания таблицы агентов, чтобы быть уверенным, что она будет корректной.

```
CREATE TABLE agent (
  agentID int(11) NOT NULL AUTO_INCREMENT,
  name varchar(50) default NULL,
  operationID int(11) default NULL,
  PRIMARY KEY  (agentID),
  FOREIGN KEY (operationID) REFERENCES operation (operationID)
);
```

Вспомните, что первым полем таблицы обычно является первичный ключ. Первичные ключи должны быть уникальными и соответствовать каждой записи. Я также решил назвать каждый первичный ключ в соответствии со специальным соглашением. Названия первичных ключей всегда начинаются с названия таблицы и заканчиваются на «ID» (от англ. «идентификатор»). Я добавил это соглашение, потому что это сильно облегчит мою работу при написании программ для работы с данными. Модификатор `NOT NULL` позволяет удостовериться, что у всех записей этой таблице будет задано значение первичного ключа. Идентификатор `AUTO_INCREMENT` является специальным инструментом, позволяющим MySQL выбрать новое значения для этого поля, если это значение не задано. Это позволяет удостовериться, что все записи уникальны.

Не все базы данных используют `AUTO_INCREMENT` *одинаковым образом. Возможно, вам придется поискать другой способ автоматической генерации ключевых полей, если вы не используете MySQL.*

В конце выражения `CREATE TABLE` я добавил индикатор, который говорит, что `agentID` является первичным ключом таблицы агентов.

Создаем ссылку на таблицу операций

Внимательно посмотрите на поле `operationID`. Это поле содержит целое число, которое будет использовано для определения конкретной операции. Я также добавил индикатор, означающий, что `operationID` является ссылкой на внешний ключ таблицы операций. Поле `operationID` таблицы агентов будет содержать ссылку на первичный ключ таблицы операций. Такой тип поля также называется *внешним ключом*.

Некоторые СУБД требуют задания как первичного, так и внешних ключей. MySQL пока что не требует этого, однако такая идея является очень привлекательной по двум причинам. Во-первых, похоже, что будущие версии MySQL будут требовать таких определений, потому что они используются для улучшения надежности базы данных. Во-вторых, очень хорошо иметь возможность задания в коде того факта, что поле имеет специальное предназначение, даже если СУБД не делает ничего с информацией.

Вставляем значения в таблицу агентов

Так как первичный ключ определен с использованием модификатора `AUTO_INCREMENT`, в выражениях `INSERT` для таблицы агентов можно использовать один трюк.

```
INSERT INTO agent VALUES(
  null, 'Bond', 1
);
```

Первичный ключ инициализируется значением `null`. Это может показаться странным, потому что первичные ключи разработаны таким образом, чтобы никогда не содержать такого значения. Однако так как поле `agentID` установлено в `AUTO_INCREMENT`, значение `null` автоматически замещается неиспользованным целым числом. Этот хитрый ход позволяет убедиться, что каждое значение первичного ключа будет уникальным.

Создаем таблицу операций

Новая таблица операций содержит информацию, относящуюся к операции. (См. табл. 9.4 для описания некоторых операций.)

Табл. 9.4. Таблица операций

OperationID	name	description	location
1	Dancing Elephant	Infiltrate suspicious zoo	London
2	Enduring Angst	Make bad guys feel really guilty	Lower Volta
3	Furious Dandelion	Plant crabgrass in enemy lawns	East Java

Каждая операция получает собственную запись в таблице операций. Все данные, относящиеся к операции, хранятся в записи операции. Данные каждой операции хранятся только один раз. У такого способа хранения есть несколько плюсов.

- Данные для операции необходимо вводить только один раз, что сохраняет время ввода данных.
- Так как данные не повторяются, у вас не будет проблем с несогласованностью данных (как проблема цирк/зоопарк).
- Новая база данных будет занимать меньше места, потому что в ней нет повторяющихся данных.
- Операция не привязана к агенту, поэтому вы не сможете случайно удалить все ссылки на операцию, удалив агента, выполняющего ее (помните, что это возможно в исходной структуре данных).
- Если вам необходимо обновить данные операции, нет нужды перебирать всех агентов, чтобы выяснить, какие из них выполняют эту операцию (опять же, вам надо было делать это в исходной структуре данных).

Код SQL, использованный для создания таблицы операций, очень похож на код для создания таблицы агентов.

```
    CREATE TABLE operation (
  operationID int(11) NOT NULL AUTO_INCREMENT,
  name varchar(50) default NULL,
  description varchar(50) default NULL,
  location varchar(50) default NULL,
  PRIMARY KEY  ('OperationID')
);
```

```
INSERT INTO operation VALUES(
  null, 'Dancing Elephant',
  'Infiltrate suspicious zoo', 'London'
);
```

Как вы можете видеть, таблица операций соответствует правилам нормализации и очень похожа на таблицу агентов. Заметьте, что я очень аккуратно все называл. SQL (теоретически) нечувствителен к регистру, однако я обнаружил, что это не всегда так (особенно в MySQL, где версия для Windows не чувствительна к регистру, а версия для UNIX воспримет `operationID` и `OperationID` в качестве названий разных полей). Я решил, что все названия полей будут использовать специальную нотацию (точно так же, как вы называете переменные в PHP). Кроме этого, я назвал ключевое поле в соответствии с собственной формулой (название таблицы, после которого следует «ID»).

Используем связи для соединения таблиц

Единственным минусом разделения данных на таблицы является необходимость каким-нибудь образом воссоединить их, когда это потребуется. Пользователя совсем не заботит, что операции и агенты располагаются в различных таблицах, он хочет видеть данные, как если бы они располагались в одной таблице. Секретом для воссоединения таблиц является инструмент, называемый *внутренним объединением*. Посмотрите на следующее выражение SELECT в SQL.

```
SELECT agent.name AS agent, operation.name AS operation
FROM agent, operation
WHERE agent.operationID = operation.operationID
ORDER BY agent.name;
```

На первый взгляд, это выглядит как обычный запрос, однако есть отличия, потому что этот запрос объединяет данные двух различных таблиц. На рис. 9.5 показаны результаты выполнения этого запроса.

Табл. 9.5. Объединение двух таблиц

Agent	Operation
Blackford	Enduring Angst
Bond	Dancing Elephant
Cardinal	Enduring Angst
Falcon	Dancing Elephant
Rahab	Furious Dandelion

Создаем полезные соединения

Запрос SQL может получать данные более чем из одной таблицы. Чтобы сделать это, существует несколько простых правил.

Во-первых, вам, возможно, придется задать названия полей более формально. Заметьте, что выражение SELECT определяет agent.name, а не просто name. Это необходимо, потому что обе таблицы содержат поля, имеющие название name. Использование синтаксиса «таблица.поле» очень похоже на использование имени и фамилии человека. Такое использование не обязательно, если не может возникнуть путаницы, однако при большом количестве человек полное именование может помочь избежать путаницы.

Также заметьте, что используется команда AS. Таким образом, у столбца будет другое название и более привлекательный внешний вид.

Команда FROM до этого момента использовалась только с одной таблицей. В этом примере необходимо указать, что данные будут содержаться в двух таблицах.

Изучаем соединений без использования команды WHERE

Оператор WHERE помогает определить взаимосвязи между двумя таблицами. В качестве примера рассмотрим следующий запрос.

```
SELECT
  agent.name AS 'agent',
  agent.operationID as 'agent opID',
  operation.operationID as 'op opID',
  operation.name AS 'operation'
FROM agent, operation
ORDER BY agent.name;
```

Это запрос очень похож на запрос, приведенный выше, за исключением того, что содержит поле operationID каждой таблицы, что позволяет исключить использование оператора WHERE. Вы можете очень удивиться результатам.

Добавляем команду WHERE для создания корректных объединений

Если не использовать команду WHERE, возвращаются все возможные комбинации. Нас же интересовали только те записи, для которых поля operationID таблиц операций и агентов совпадали. Команда WHERE позволяет получить только эти значения, соединенные по ID операции.

Секретом является использование полей operationID в обеих таблицах. Вы уже знаете, что каждая таблица должна содержать первичный ключ. Поле первичного ключа используется для уникальной идентификации каждой записи

в базе данных. В таблице агентов первичным ключом является `agentID`. В таблице операций первичным ключом является `operationID`. (Вы могли заметить, что моя схема именования не слишком оригинальна, зато чрезвычайно полезна.) Мне удалось получить все данные, относящиеся к операции, из таблицы агентов, благодаря тому, что я заменил нужные мне поля значениями, соответствующими первичным ключам таблицы операций. Поле, представляющее первичный ключ другой таблицы, называется *внешним ключом*. Первичный и внешний ключи цементируют взаимосвязи между таблицами. См. табл. 9.6.

Табл. 9.6. Объединение таблиц агентов и операций без использование команды WHERE

agent	agent opID	op opID	Operation[a]
Blackford	1	1	Dancing Elephant
Blackford	1	2	Enduring Angst
Blackford	1	3	Furious Dandelion
Bond	1	1	Dancing Elephant
Bond	1	2	Enduring Angst
Bond	1	3	Furious Dandelion
Cardinal	2	2	Enduring Angst
Cardinal	2	3	Furious Dandelion
Cardinal	2	1	Dancing Elephant
Falcon	1	1	Dancing Elephant
Falcon	1	2	Enduring Angst
Falcon	1	3	Furious Dandelion
Rahab	3	1	Dancing Elephant
Rahab	3	2	Enduring Angst
Rahab	3	3	Furious Dandelion

a. agent - агент, agent opID – идентификатор операции в таблице агентов, op opID – идентификатор операции в таблице операций, Operation – операция

Добавляем условия в запрос объединения

Конечно, вы все еще можете использовать команду WHERE для ограничения отображаемых записей. А также использовать структуру AND для создания сложных условий. Например, этот код:

```
SELECT
  agent.name AS 'agent',
  operation.name AS operation
FROM agent, operation
WHERE agent.operationID = operation.operationID
  AND agent.name LIKE 'B%';
```

вернет кодовое имя и название операции каждого агента, чье кодовое имя начинается с «B».

Правда о внутренних объединениях

Вы должны знать, что синтаксис, который я использовал здесь, является удобным сокращением, поддерживаемым большинством СУБД. Формальный синтаксис внутренних объединений выглядит следующим образом.

```
SELECT agent.name, operation.name
FROM
  agent INNER JOIN operation
  ON agent.OperationID = operation.OperationID
ORDER BY agent.name;
```

Множество программистов предпочитают считать соединения частью команды WHERE и использовать синтаксис WHERE. Некоторые базы данных SQL (большинство предлагаемых корпорацией Microsoft) не позволяют использовать синтаксис WHERE для внутренних связей и требуют задания INNER JOINT как части команды FROM. Когда вы используете такой синтаксис INNER JOIN, команда ON определяет способ соединения таблиц

Создаем таблицы связей для отношения «многие ко многим»

После того как вы создали ER-диаграмму, вы можете создавать новые таблицы, обрабатывающие все отношения «один ко многим». Что делать с отношениями «многие ко многим», как, например, связь между агентами и их навыками, немного менее очевидно. Вспомните, что каждый агент может иметь несколько навыков, и несколько агентов могут использовать один и тот же навык. Лучшим способом для обработки таких случаев является создания специального типа таблицы.

Улучшаем ER-диаграмму

На рис. 9.5 показана новая версия ER-диаграммы, которая ликвидирует все отношения вида «многие ко многим».

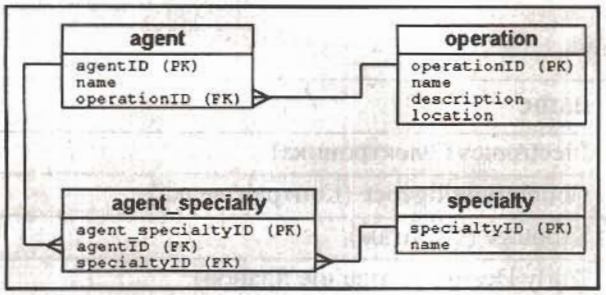


Рис. 9.5. Эта новая ER-диаграмма содержит специальные таблицы для обработки отношений «многие ко многим»

ER-диаграмма, показанная на рис. 9.5, имеет несколько улучшений в отличие от приводимой ранее версии. Во-первых, я добавил (PK) к концу каждого первичного ключа. Я также добавил (FK) в конец всех внешних ключей. Расположение строк диаграммы теперь имеет очень большое значение. Теперь я рисую линии только между ссылкой на внешний ключ и соответствующим первичным ключом другой таблицы. Другим важным улучшением является добавление таблицы `agent_specialty`. Эта таблица интересна тем, что она содержит только первичные и внешние ключи. Каждая запись этой таблицы представляет одну связь между таблицами агентов и специальности.

Большинство таблиц реляционной базы данных содержат сущности набора данных, однако таблицы связей содержат отношения между сущностями.

Все фактические данные, относящиеся к агенту или специальности, находятся в других таблицах. Такое расположение данных предоставляет огромную гибкость.

Создаем таблицу специальностей

Таблица специальностей на самом деле чрезвычайно проста, как это показано в табл. 9.7.

Табл. 9.7. Таблица специальностей

specialityID	name
0	Electronics (Электроника)
1	Counterintelligence (Контрразведка)
2	Sabotage (Саботаж)
3	Doily Design (Создание планов)
4	Explosives (Взрывчатые вещества)
5	Flower Arranging (Составление букетов)

Как вы можете видеть, в этой таблице нет ничего такого, что особым образом связывало бы ее с определенным агентом. Более того, вы не найдете ссылок на специальности в таблице агентов. Сложные взаимосвязи между этими таблицами обрабатываются при помощи новой таблицы `agent_specialty`. Такой специальный вид таблиц называется *таблицей связей*, потому что она используется для управления взаимосвязями других таблиц. В табл. 9.8 показан примерный набор данных, содержащихся в таблице `agent_specialty`.

Табл. 9.8. Таблица agent_specialty

agent_speciality_ID	agent ID	speciality ID
1	1	2
2	1	3
3	2	1
4	2	6
5	3	2
6	4	4
7	4	5

Интерпретируем таблицу agent_specialty с помощью запроса

Конечно же, таблица `agent_specialty` непосредственно не нужна пользователю, потому что она содержит только ссылки на внешние ключи. Вы можете сделать данные более читаемыми с помощью запроса SQL.

```
SELECT agent_specialtyID,
       agent.name AS 'agent',
       specialty.name AS 'specialty'
FROM agent_specialty,
     agent,
     specialty
WHERE agent.agentID = agent_specialty.agentID
  AND specialty.specialtyID = agent_specialty.specialtyID;
```

Этот запрос требует двух сравнений для объединения трех таблиц. Необходимо «подделать» взаимосвязи между таблицами `agent` и `agent_specialty` с помощью общих значений `agentID`. Это также необходимо для обеспечения безопасности связей между таблицами `speciality` и `agent_specialty` при помощи сравнения полей `specialtyID`. Результат выполнения такого запроса показывает, что корректные записи действительно были соединены, как показано в табл. 9.9.

Таблица связей нужна для соединения таблиц, которые имеют отношения «многие ко многим». Каждый раз, когда вы добавляете новую взаимосвязь между агентом и специальностью, вы добавляете новую запись в таблицу `agent_specialty`.

Табл. 9.9. Интерпретация запроса таблицы agent_specialty

agent_specialityID	agent	speciality
1	Bond	Sabotage
2	Bond	Doily Design
3	Falcon	Counterintelligence
5	Cardinal	Sabotage
6	Blackford	Explosives
7	Blackford	Flower Arranging

Создаем запросы, использующие таблицы связей

Если вы хотите знать о взаимосвязях между агентами и специальностями, вы можете получить эти данные из таблицы `agent_specialty`. Например, если вам необходимо узнать, какие агенты умеют делать букеты, вы можете использовать следующий запрос.

```
SELECT
  agent.name
FROM
  agent,
  specialty,
  agent_specialty
WHERE agent.agentID = agent_specialty.agentID
  AND agent_specialty.specialtyID = specialtiy.specialtyID
  AND specialty.name = 'Flower Arranging';
```

Этот запрос выглядит немного пугающим, однако на самом деле все не так плохо. Этот запрос получает данные из трех различных таблиц. Результату необходимо получить имя, содержащееся в таблице агентов. Я не хочу запоминать номер специальности, ассоциированный с «Flower Arranging», поэтому я позволю запросу найти его в таблице специальностей. Так как мне необходимо знать, какой агент ассоциирован с определенной специальностью, я буду использовать таблицу `agent_specialty` для связи с двумя оставшимися таблицами. Команда `WHERE` позволяет делать объединения. Фраза

```
agents.agentID = agent_specialty.agentID
cements the relationship between agents and agent_specialty.
Likewise,
agent_specialty.specialtyID = specialties.specialtyID
```

позволяет убедиться, что были объединены таблицы `speciality` и `agent_specialty`. Последняя часть команды `WHERE` является условной частью запроса, которая возвращает только те записи, специальности которых соответствует созданию букетов. (Знаете ли, создание букетов может быть смертельным искусством в руках умелого человека...)

Итоги

В этой главе вы перешли от программирования к пониманию данных, что является очень важным в современных приложениях. Вы узнали, как переделать плохо спроектированную таблицу в набор хорошо организованных таблиц, которые помогут избежать множества проблем. Вы узнали о трех стадиях нормализации. Вы научились создавать ER-диаграммы. Вы можете распознать три типа взаимоотношений между сущностями. Вы можете создавать нормализованные таблицы в SQL, содержащие указатели на первичные и внешние ключи. Вы можете соединять нормализованные таблица с помощью выражения `SQL INNER JOIN`. Вы знаете, как моделировать отношения «многие ко многим» путем построения таблиц связей. Цивилизованный мир стал безопаснее после ваших успехов.

Домашнее задание

1. Посмотрите, сможете ли вы найти ER-диаграммы для данных, с которыми работаете каждый день. Проверьте эти документы и посмотрите, поймете ли вы их.
2. Проанализируйте базу данных, которую вы регулярно используете. Определите, соответствует ли она требованиям, определенным в этой главе для хорошо организованной структуры данных. Если нет, объясните, что неправильно в структуре данных и как это можно исправить.
3. Сделайте диаграмму улучшенной структуры данных для базы данных, которую вы рассматривали в предыдущем вопросе. Создайте необходимые таблицы в SQL и заполните их тестовыми данными.
4. Разработайте базу данных для данных, используемых вами каждый день. (Будьте осторожны: большинство задач намного сложнее, чем это кажется на первый взгляд.) Создайте диаграмму данных, после чего создайте таблицы и заполните их тестовыми данными.

Глава 10

Создаем приложения, использующие трехзвенную модель данных

В начале этой книги вы увидели страницы HTML, которые по существу являются статичными документами. После этого вы узнали, как генерировать динамические страницы при помощи мощного языка PHP. В последних нескольких главах вы научились использовать системы управления базами данных, такие, как MySQL, для создания мощных структур данных. В этой последней главе мы соединим аспекты программирования PHP и данных и создадим полноценную систему управления данными для базы данных шпионов. Эту систему можно будет легко адаптировать к любому проекту, придуманному вами, включая приложения для электронной торговли. В частности, в этой главе вы узнаете, как:

- разработать средние и большие приложения данных;
- создать библиотеку функций для работы с данными, которые можно использовать заново;
- оптимизировать функции для использования в различных наборах данных;
- включать библиотечные файлы в ваши программы.

В этой главе практически не содержится нового PHP или MySQL кода. Целью этой главы является демонстрация того, как создать большой проект, затратив при этом минимум усилий.

Представляем программу Spy Master

Программа Spy Master – это набор PHP программ, позволяющих получить доступ к базе данных шпионов, созданной в главе 9 «Нормализация данных». Хотя база данных, созданная в той главе, и гибкая, и мощная, она не проста в использовании, если вы только не знаете SQL. Даже если ваши пользователи понимают SQL, вы не захотите предоставить им возможность непосредственного управления базой данных, потому что это может привести к непредсказуемым результатам. Вам необходимо создать что-то вроде приложения интерфейса для базы данных. В частности, в этой системе существует три уровня. Компьютер

клиента ответственен за взаимодействие с пользователем. Сервер базы данных (MySQL) управляет самими данными. Программа PHP, находящаяся между клиентом и базой данных, выступает в качестве интерпретатора. PHP является мостом между языком HTML клиента и языком SQL базы данных. Такой тип организации системы часто называют трехзвенной[1] *архитектурой*. По мере того как вы будете рассматривать программу Spy Master в этой главе, вы узнаете о преимуществах такого подхода.

Просмотр основного экрана

Мы начнем нашу работу, посмотрев на программу с точки зрения пользователя, как показано на рис. 10.1.

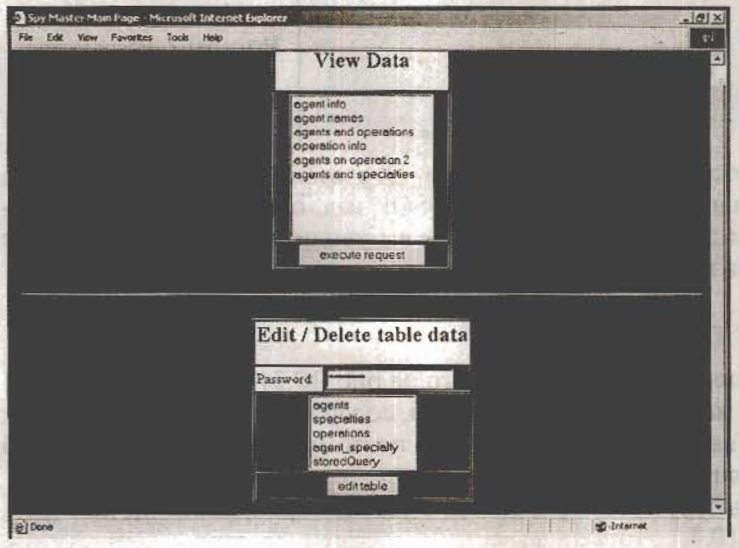


Рис. 10.1. Точка входа в базу данных Spy Master является очень простой и понятной

Основная страница разделена на две части. В первой части содержится несколько «заявок» на данные. Каждая из этих заявок является запросом.

Просмотр результатов выполнения запроса

Когда пользователь выбирает запрос и нажимает кнопку `Submit`, появляется экран, похожий на изображенный на рис. 10.2.

---

[1] Трехзвенная модель – это система клиент-сервер, в которой используется промежуточное звено – компьютер, обычно работающий как монитор обработки транзакций или брокер объектных запросов, что предоставляет еще одно место для выполнения приложений и обеспечивает работу большего числа клиентов, чем в двухзвенной модели – *Примеч. пер.*

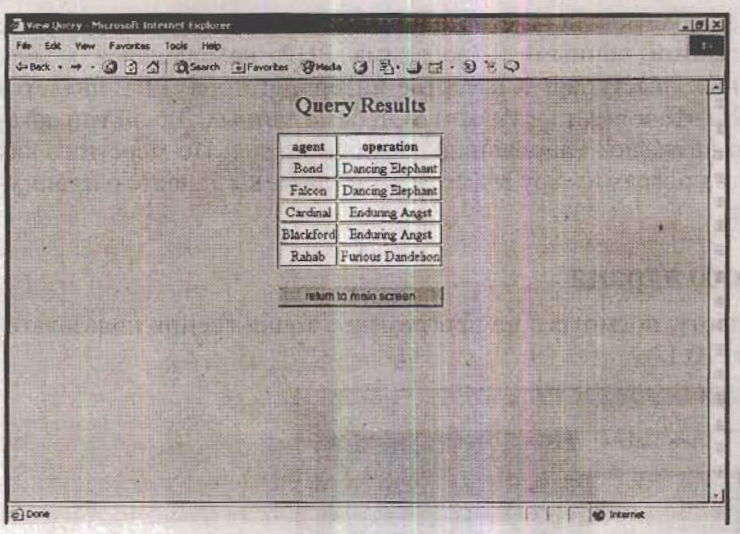


Рис. 10.2. Результаты выполнения запроса отображаются в таблице HTML

Все запросы являются заранее созданными. Это означает, что пользователь не может ошибиться, напечатав некорректный SQL код, однако также ограничивает возможности базы данных. К счастью, как вы увидите позднее, существует система для добавления новых запросов.

Просмотр данных таблицы

Вторая часть основного экрана (показанного на рис. 10.3) позволяет пользователю непосредственно манипулировать данными, содержащимися в таблице. Так как такая возможность является очень мощной (и поэтому опасной), доступ к этой части системы контролируется с помощью пароля.

Например, если я выберу таблицу агентов, я увижу экран, похожий на экран, изображенный на рис. 10.4.

На этой странице пользователь может увидеть все данные выбранной таблицы. Кроме этого, эта страница содержит ссылки для добавления, редактирования и удаления записей из таблицы.

Редактируем запись

Если пользователь захочет отредактировать запись, появится экран, похожий на экран, изображенный на рис. 10.5.

Страница «Edit Record» имеет несколько важных особенностей. Во-первых, пользователь не может непосредственно изменить первичный ключ. Если бы пользователь мог это сделать, это привело бы к полной дестабилизации базы дан-

Рис. 10.3. Из основного экрана вы можете также получить доступ к данным таблицы, имея соответствующий пароль

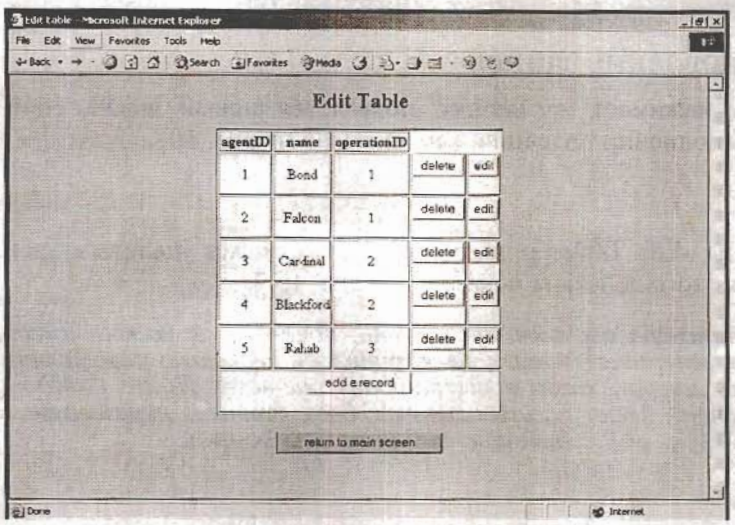


Рис. 10.4. Экран `editTable` отображает всю информацию, содержащуюся в таблице

ных. Также заметьте, как отображается поле `operationID`. Это поле является первичным ключом, содержащим целочисленное значение, однако пользователю будет очень сложно непосредственно работать с числовыми значениями. Вместо этого программа предоставляет «выпадающий» список операций. Когда пользователь выбирает элемент этого списка, следующей странице посылается соответствующий численный индекс.

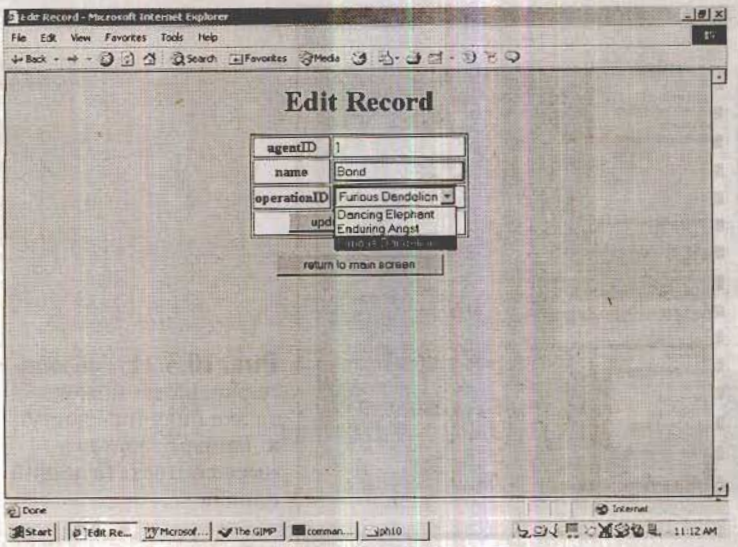


Рис. 10.5. Пользователь редактирует запись в таблице агентов

Подтверждаем обновление записи

Когда пользователь нажимает на кнопку, появляется новый экран, сообщающий об удачном выполнении операции, как показано на рис. 10.6.

Удаляем запись

С помощью страницы «Edit Table» пользователь также может удалять записи. Результаты выполнения этого действия показаны на рис. 10.7.

Из этого примера вы можете понять, почему так важно иметь скрипт для генерации данных. Мне пришлось несколько раз удалять и изменять записи, когда я тестировал систему. После каждого теста я очень легко восстанавливал базу данных, перезагружая файл `buildSpy.sql` *с помощью команды* `SQL SOURCE`.

Добавляем запись

Добавление записи в таблицу является многошаговой процедурой, очень похожей на редактирование записи. Первая страница (изображенная на рис. 10.8) позволяет вам ввести новые данные в соответствующие поля.

Так же, как экран «Edit Record», страница «Add Record» не позволяет пользователю непосредственно ввести первичный ключ. Эта страница также генерирует «выпадающие» списки `SELECT` для полей внешних ключей, как было с `operationID`.

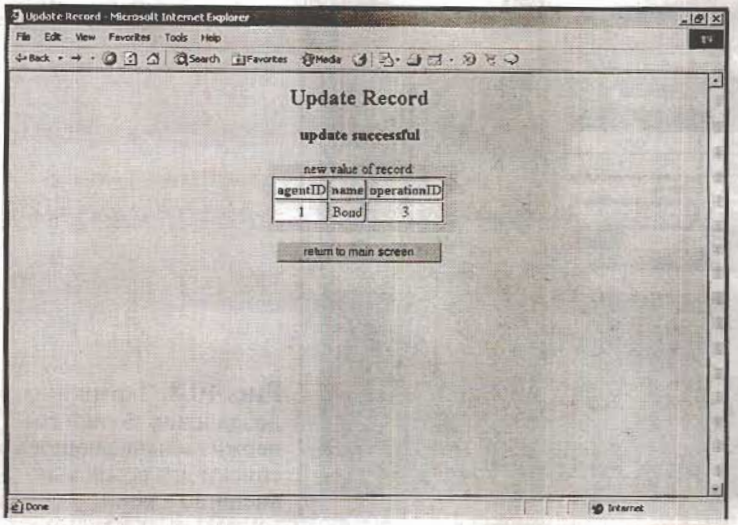


Рис. 10.6. Пользователь может видеть обновленную запись

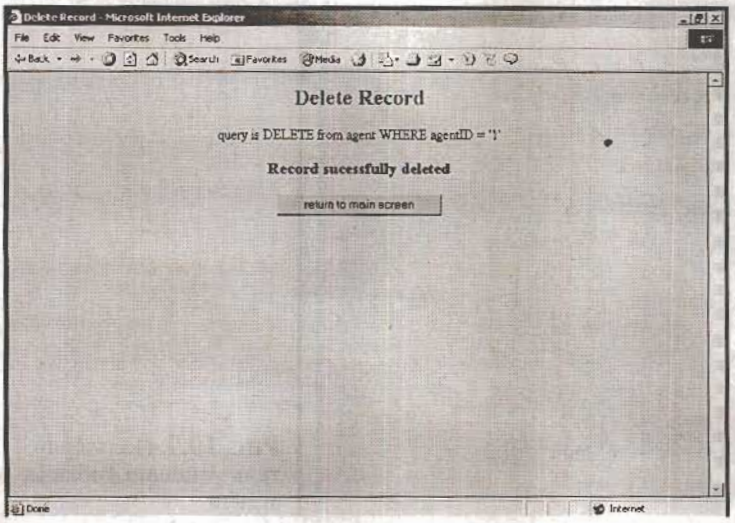


Рис. 10.7. Удалить запись очень просто

Обрабатываем добавление

Когда пользователь решает завершить добавление записи, появляется другая страница, подтверждающая добавление (или, конечно же, описывающая ошибку, если по какой-то причине невозможно добавить запись). Эта страница показана на рис. 10.9.

Рис. 10.8. Экран добавления записи содержит «выпадающие» списки для ссылок на внешние ключи

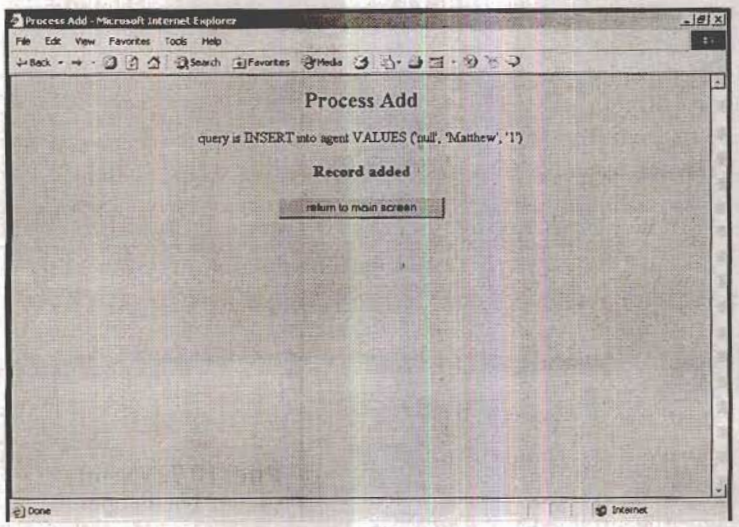


Рис. 10.9. Пользователь успешно добавил агента

Создаем модель системы SpyMaster

Если подумать о всех операциях системы «Spy Master», это может показаться устрашающим. Эта программа содержит множество функциональностей. Неосмотрительно начинать кодирование этой системы, не составив какого-нибудь стратегического плана.

Создаем диаграмму состояний

Существует множество способов решения сложных проблем программирования. В данном конкретном случае я решил сосредоточиться на движении потока данных через различные модули. На рис. 10.10 изображена стратегия, которую я использовал для создания этой программы.

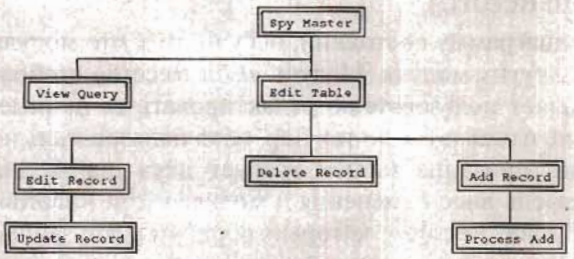


Рис. 10.10. Диаграмма состояний системы «Spy Master»

Иллюстрация, приведенная на рис. 10.10, иногда называется *диаграммой состояний*. Такой тип иллюстраций используется для обозначения решаемых проблем и модулей, которые могут помочь в решении этих проблем. Я начал создание этой диаграммы, думая о том, что должна уметь делать система управления данными. Каждое действие, придуманное мною, было вынесено в отдельный «модуль». Зачастую модуль представляет собой отдельный экран. Часто (однако не всегда) каждый модуль будет представлен в виде программы PHP.

Модуль «Просмотр запроса» (View Query)

Очевидно, что пользователи должны получать результаты выполнения запросов в базе данных. Это будет одной из наиболее часто выполняемых функций системы. Я решил, что модуль «View Query» должен уметь просматривать любой запрос, посланный ему, а также отображать результаты его выполнения.

Модуль «Редактирование таблицы» (Edit Table)

Другим основным назначением системы данных является определение данных, которое включает в себя добавление новых записей, удаление существующих и обновление информации. Такой вид деятельности может быть деструктивным, поэтому он должен контролироваться с помощью некоторой системы доступа. Все определения данных основаны на соответствующей структуре таблицы базы данных, поэтому важно разрешить три основных типа определения данных (редактирование, удаление и обновление) для каждой таблицы. Модуль «Edit Table» предоставляет интерфейс для реализации соответствующих функциональностей. Он отображает все текущие записи в таблице и позволяет пользо-

вателю редактировать или удалять любую из них. Он также содержит кнопку, позволяющую пользователю добавлять новую запись в текущую таблицу. Важно понимать, что «Edit Table» не вносит изменений в базу данных. Вместо этого он выступает в качестве шлюза для остальных модулей редактирования.

Модули «Редактирование записи» (Edit Record) и «Обновление записи» (Update Record)

Если вы еще раз посмотрите на диаграмму состояний, вы увидите, что модуль «Edit Table» подразделяется на три других модуля. Модуль «Edit Record» отображает одну запись таблицы и позволяет пользователю редактировать ее данные. Однако при этом сама база данных не изменяется до тех пор, пока пользователь не отправит изменения, поэтому редактирование записи требует двух шагов для обработки. После того как пользователь внес изменения в модуле «Edit Record», управление передается модулю «Update Record», который обрабатывает запрос и вносит изменения в базу данных.

Модули «Добавление записи» (Add Record) и «Обработка добавления» (Process Add)

Добавление записи схоже с редактированием в том, что требует двух шагов. Первый модуль («Add Record») генерирует форму, которая позволяет пользователю ввести информацию о новой записи. Как только пользователь определит данные записи, модуль «Process Add» создает соответствующий код SQL, необходимый для добавления новой записи в таблицу.

Модуль «Удаление записи» (Delete Record)

Удаление записи является простым процессом. Нет необходимости получать какие-либо данные от пользователя, поэтому для обработки запроса на удаления достаточно одного модуля.

Проектируем архитектуру системы

Диаграмма состояний очень полезна, потому что она позволяет вам увидеть работу всей системы в целом. Однако необходим дополнительный анализ, потому что простая диаграмма состояний не решает множества вопросов.

- Необходимо ли создавать модуль «Edit Table» для каждой таблицы?
- Если да, будут ли нам необходимы копии остальных модулей для редактирования?
- Существует ли возможность автоматизации процесса?
- Что произойдет, если изменится соответствующая структура данных?

- Что произойдет, если я захочу использовать аналогичную структуру в другой базе данных?
- Как я могу предусмотреть возможность добавления запросов в систему?

Заманчиво создать систему исключительно для управления базой данных шпионов. Преимуществом такой системы будет точной знание, как разрешать трудности, относящиеся к системе шпионов. Например, `operationID` является ссылкой на внешний ключ таблицы агентов, поэтому, где это только возможно, он должен выбираться с помощью «выпадающего» списка. Если вы создаете специализированный модуль для управления редактированием таблицы агентов, вы можете реализовать такую функциональность. Однако такой подход очень быстро станет громоздким, если имеется несколько таблиц. Намного лучшим решением является использование «умной» процедуры, которая может создать экран редактирования для любой таблицы базы данных. Еще лучшим подходом будет предусмотрение в вашей программе возможности автоматического распознавания полей внешних ключей и создания соответствующих элементов пользовательского интерфейса (команды HTML `SELECT`) в подходящих местах. На самом деле, вы могли бы создать целую библиотеку основных процедур, работающих с базой данных. Именно этот подход я и выбрал.

Создаем библиотеку функций

Хотя программа «Spy Master» и является самой большой в этой книге, вы обнаружите, что большая ее часть удивительно проста. Ядром системы является файл «spyLib.php». Этот файл не предназначен для выполнения в браузере пользователя. Вместо этого он содержит библиотеку функций, которые упрощают программирование любой базы данных. В этой библиотеке я использовал столько кода PHP, сколько смог. Все остальные программы PHP системы используют различные функции этой библиотеки. У такого подхода есть несколько преимуществ.

- Уменьшается общий размер кода, потому что исключается его повторение.
- Если мне необходимо улучшить модуль, я изменяю код только в библиотеке, а не в нескольких местах.
- Очень просто изменить код библиотеки, чтобы она могла работать с другой базой данных.
- Подробности реализации каждого модуля спрятаны в нем, поэтому при написании каждой страницы PHP я смогу уделить больше времени «общей картине».
- Процедуры можно использовать заново при работе с любой таблицей базы данных.
- Процедуры могут автоматически изменяться в соответствии с изменениями в структуре данных.

- Библиотеку можно без труда использовать заново в другом проекте.

На рис. 10.11 показана более подробная диаграмма состояний.

Посмотрев на сам код, вы увидите, что большинство программ PHP очень просты. Зачастую они просто собирают данные для библиотечных функций и передают им управление, после чего распечатывают результаты их выполнения..

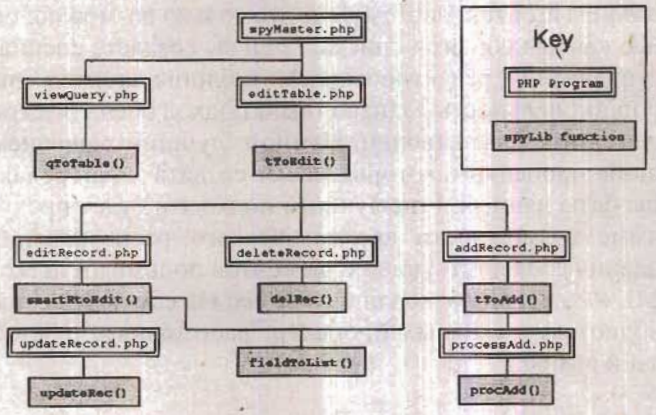


Рис. 10.11. Эта диаграмма состояний иллюстрирует взаимосвязи между программами PHP и функциями библиотеки `spyLib`

Создаем не входящий в библиотеку код

Я начну с описания частей этого проекта, не относящихся к библиотеке. Модуль библиотеки предназначен для использования в других программах PHP, поэтому разумно сначала посмотреть на них.

Подготавливаем базу данных

База данных для этого сегмента практически такая же, которая использовалась в главе 9 «Нормализация данных». Я добавил одну таблицу для хранения запросов. Все остальные таблицы совпадают с теми, что были в главе 9. SQL скрипт, необходимый для создания новой версии базы данных шпионов, располагается на нашем сайте и называется «buildSpy.sql». Заметьте, что эта версия базы данных немого отличается от виденной вами в главе 9, потому что в данных содержатся несколько запросов! Чтобы сделать программу достаточно безопасной, простые пользователи не могут делать запросы. Я также не хочу ограничивать пользователей небольшим количеством запросов, созданных мною при разработке системы. Одним из решений этой проблемы является хранение набора запросов в базе данных и разрешение некоторым пользователем вносить изменения в запросы. Я назвал эту новую таблицу

storedQuery. Этой таблицей можно управлять, как и другими таблицами системы, поэтому пользователь, имеющий подходящий пароль, может добавлять, редактировать и удалять запросы. Вот дополнительный код, используемый для создания таблицы storedQuery.

```
#####################################
# создать таблицу storedQuery
#####################################
CREATE TABLE storedQuery (
  storedQueryID int(11) NOT NULL AUTO_INCREMENT,
  description varchar(30),
  text varchar(255),
  PRIMARY KEY (storedQueryID)
);
INSERT INTO storedQuery VALUES (
  null,
  'agent info',
  'SELECT * FROM agent'
);
```

Таблица storedQuery имеет три поля. Поле description содержит краткое описание каждого запроса на английском. Поле text хранит SQL код запроса.

Корректный синтаксис SQL очень важен при сохранении запросов SQL в базе данных SQL, как я делаю это в данном случае. Особенно важно следить за одиночными и двойными кавычками. Чтобы использовать одиночные кавычки, требуемые в некоторых запросах, вам необходимо поставить перед ними символ обратной наклонной черты. Например, я хочу сохранить следующий запрос.

```
SELECT * FROM agent WHERE agent.name = 'Bond',
тогда в базе данных я сохраню следующий текст:
SELECT * FROM agent WHERE agent.name = \'Bond\'
```

Это необходимо для того, чтобы сохранить символы одиночных кавычек. В противном случае эти символы будут интерпретированы некорректно. Позже я покажу вам, как убрать символы обратной наклонной черты.

Исследуем программу spyMaster.php

Программа spyMaster.php является точкой входа в систему. Доступ ко всем частям системы осуществляется с этой страницы. Эта страница состоит из двух основных частей. Каждый сегмент инкапсулирует форму HTML, которая будет

отсылать запрос определенной программе PHP. Первый сегмент содержит небольшое количество кода PHP, который настраивает окно списка запроса.

Создаем форму запроса

```
<html>
<head>
<title>Spy Master Main Page</title>
<?
  include "spyLib.php";
?>
</head>
<body>
<form action = "viewQuery.php"
      method = "post">
<table border = 1
       width = 200>
<tr>
  <td><center><h2>View Data</h2></center></td>
</tr>
<tr>
  <td><center>
    <select name = "theQuery" size = 10>
<?
// получить запросы из таблицы storedQuery
$dbConn = connectToSpy();
$query = "SELECT * from storedQuery";
$result = mysql_query($query, $dbConn);
while($row = mysql_fetch_assoc($result)){
  $currentQuery = $row['text'];
  $theDescription = $row['description'];
  print <<<HERE
      <option value = "$currentQuery">$theDescription</option>
HERE;
  } // завершение цикла while
?>
</select>
  </center>
</tr>
<tr>
  <td><center>
    <input type = "submit"
       value = "execute request" >
  </center></td>
```

```
</tr>
</table>
</form>
```

Большая часть этого кода является обычным HTML. Код HTML создает форму, которая будет вызывать `viewQuery.php`, если пользователь нажмет кнопку `Submit`. Кроме этого, я использовал код PHP, который генерирует специальное окно ввода на основе записей таблицы `storedQuery`.

Включаем библиотеку spyLib

Прежде всего необходимо обратить внимание на выражение `include()`. Эта команда позволяет вам импортировать файл. PHP считает этот файл и интерпретирует его как HTML. Включаемый файл может содержать код HTML, CSS или PHP. Большая часть функциональности программы шпионов хранится в библиотечной программе `spyLib.php`. Все остальные программы PHP системы начинаются с включения `spyLib.php`. После этого возможно получение доступа ко всем функциям библиотеки, как если бы они были локально определенными. Как вы увидите, это предоставляет невероятную мощь и гибкость для программирования системы.

Соединяемся с базой данных шпионов

Полезность библиотеки `spyLib` становится стразу же заметной после подключения к базе данных шпионов. Вместо того чтобы волноваться, к какой базе данных я подключаюсь, я просто полагаюсь на функцию `connectToSpy()`, находящуюся в `spyLib()`. В текущем коде мне не надо волноваться о подробностях соединения с базой данных. Используя библиотеку, я могу один раз написать код соединения, после чего использовать заново эту функцию по мере надобности.

В использовании библиотеки при соединении с базой данных есть другое преимущество. Практически наверняка при использовании этого кода в другой системе вы будете использовать другой способ для соединения с сервером данных. Если код соединения с сервером выделен в функцию, то если вы хотите его обновить, вам необходимо сделать изменения только в одном месте. Это является намного более эффективным решением, чем поиск в дюжине программ мест, в которых происходит вызов функции `mysql_connect()`.

Заметьте, что функция `connectToSpy()` возвращает указатель на соединение, который я использую для других операций с базой данных.

Получаем запросы

Я решил закодировать несколько предварительно созданных запросов в таблицу. Я расскажу о причинах этого более подробно в разделе программы `viewQue-`

ry. Основная форма должна содержать список описаний запросов и позволять пользователю выбирать один из этих запросов. Я использую выражение SQL SELECT для извлечения всех данных из таблицы storedQuery. После этого я использую поля description и text для создания многострочного списка.

Создаем форму редактирования таблицы

Вторая часть программы spymMaster отображает все таблицы базы данных и позволяет пользователю выбрать таблицу для редактирования. Большая часть функциональности системы расположена здесь. Удивительно, но в этой части страницы совсем нет PHP кода. Форма HTML отправит пользователя в программу editTable.php.

```
    <hr>
<form action = "editTable.php"
      method = "post">

<table border = 1>
<tr>
  <td colspan = 2><center>
    <h2>Edit / Delete table data</h2>
  </center></td>
</tr>
<tr>
<td>Password:</td>
  <td>
    <input type = "password"
       name = "pwd"
       value = "absolute"><br>
  </td>
</tr>
<tr>
  <td colspan = 2><center>
    <select name = "tableName"
        size = 5>
      <option value = "agent">agents</option>
      <option value = "specialty">specialties</option>
      <option value = "operation">operations</option>
      <option value = "agent_specialty">agent_specialty</option>
      <option value = "storedQuery">storedQuery</option>
    </select>
  </center></td>
</tr>
<tr>
  <td colspan = 2><center>
```

```
    <input type = "submit"
       value = "edit table">
  </center></td>
</tr>
</table>
</form>
</body>
</html>
```

Чтобы упростить отладку, я предварительно заполнил поле пароля соответствующим паролем, чтобы мне не приходилось набирать его каждый раз. В производственной среде вам конечно же необходимо оставить это поле пустым, чтобы пользователь не мог попасть в систему, не введя пароля.

Создаем программу viewQuery.php

Когда пользователь выбирает запрос, управление передается программе `viewQuery.php`. Сама по себе эта программа выполняет удивительно мало действий.

```
<html>
<head>
<title>View Query</title>
</head>
<body>
<center>
<h2>Query Results</h2>
</center>
<?
include "spyLib.php";
$dbConn = connectToSpy();
// удалить знаки перехода…
$theQuery = str_replace("\'", "'", $theQuery);
print qToTable($theQuery);
print mainButton();
?>
</body>
</html>
```

После того как `viewQuery.php` соединилась с библиотекой, она использует ее функции для соединения с базой данных и распечатки желаемых результатов. Функция `qToTable()` выполняет большую часть работы. Она принимает передаваемый запрос и генерирует таблицу, содержащую кнопки добавления, удаления и редактирования.

Почему я храню запросы в базе данных?

Возможно, вам интересно, почему я решил применить такой подход к запросам. В конце концов, я бы мог позволить пользователю непосредственно вводить запрос или предоставить некоторого рода форму, позволяющую пользователю искать определенные значения. У каждого из этих подходов есть свои преимущества, однако они таят в себе проблемы. Очень опасно предоставлять непосредственный доступ к данным из веб-формы. Злонамеренные пользователи могут создать так называемых «троянских коней», которые будут шпионить за вашими данными, изменять или даже удалять информацию из базы данных. Иногда я создаю форму, содержащую достаточно информации для построения SQL запроса, после чего встраиваю этот запрос в функции серверной стороны (звучит как хорошее упражнение для выполнения после прочтения этой главы). В этом случае я сохраняю запросы в другой таблице. Люди, имеющие доступ администратора, могут добавлять новые запросы в базу данных, а обычные пользователи – нет. Я предварительно заполнил базу данных `storedQuery` несколькими полезными запросами, после чего предусмотрел место для добавления новых запросов, если ситуация потребует этого. У этой системы все еще есть недостатки (в основном заключающиеся в том, что обычные пользователи не могут создавать специализированные запросы), однако она намного более безопасна, чем системы, создающие запросы по данным пользователя

Функция `str_replace()` необходима потому, что запросы SQL содержат символы одиночных кавычек (‘). При сохранении запроса в виде записи VARCHAR одиночные кавычки, содержащиеся в запросе, приводят к ошибкам. Обычным решением этой проблемы является использование обратной наклонной черты, обозначающей, что кавычки не надо интерпретировать, а нужно воспринимать их в качестве части данных. Проблема заключается в том, что обратная наклонная черта так и остается в строке, когда я пытаюсь выполнить запрос. Функция `str_replace()` заменяет все вхождения “\”, на обычную одинарную кавычку (‘).

Заметьте, что функция `qToTable()` сама не печатает ничего на экране. Она просто создает сложную строку, содержащую код HTML. Программа `viewQuery.php` распечатывает этот код на экран.

Если вы используете библиотеку, лучше всего, если ее код не распечатывает непосредственно ничего на экране. Вместо этого он должен просто возвращать нужное значение вызвавшей его программе. Это позволит реализовать многократное использование данных. Например, если функция `qToTable()` печатала бы непосредственно на экран, вы не могли бы использовать ее для генерации файла. Так как библиотечный код возвращает значение, однако ничего с ним не делает, код, вызвавший библиотечную функцию, может использовать результаты ее выполнения как угодно.

Функция `mainButton()` создает простую форму HTML, которая направляет пользователя обратно на страницу `spyMaster.php`. Хотя этот код и относительно прост, он повторяется настолько часто, что его лучше сохранить в функции, чем копировать и вставлять на каждую страницу системы.

Просмотр программы editTable.php

`editTable.php` следует знакомому шаблону. Она содержит небольшое количество кода PHP, а большая часть работы перенаправляется библиотеке. Основным предназначением этого модуля является проверка пароля администратора. Если пользователь не имеет соответствующего пароля, дальнейший доступ в систему блокируется. Если пользователь имеет корректный пароль, очень мощная функция `tToEdit()` предоставляет доступ к функциям добавления, редактирования и удаления.

```
<html>
<head>
<title>Edit table</title>
</head>
<body>
<h2>Edit Table</h2>
<?
include "spyLib.php";
// проверить пароль
if ($pwd == $adminPassword){
  $dbConn = connectToSpy();
  print tToEdit("$tableName");
} else {
   print "<h3>You must have administrative access to proceed</
h3>\n";
} // завершение if
print mainButton();
?>
</body>
</html>
```

Значение `$pwd` берется из поля страницы `spyMaster.php`. Значение `$adminPassword` хранится в `spyLibrary.php`. (Паролем администратора по умолчанию является «absolute», однако вы можете изменить его на любое значение в `spyLib.php`.)

Просмотр программы editRecord.php

Программа `editRecord.php` вызывается из формы, сгенерированной программой `editTable.php`. (На самом деле форму генерирует `tToEdit()`, однако `tToEdit()` вызывается из `editTable.php`.) Эта программа ожидает передачи

переменных $tableName, $keyName и $keyVal. Эти переменные (автоматически предоставляемые tToEdit()) помогают editRecord создать запрос, который вернет выбранную пользователем запись. (Пока что вам придется довериться мне в том, как посылается походящая запись данных. Вы можете перейти к описанию tToEdit(), чтобы увидеть, как работает эта функция.)

```
<html>
<head>
<title>Edit Record</title>
</head>
<body>
<h1>Edit Record</h1>
<?
// ожидает $tableName, $keyName, $keyVal
include "spyLib.php";
$dbConn = connectToSpy();
$query = "SELECT * FROM $tableName WHERE $keyName = $keyVal";
print smartRToEdit($query);
print mainButton();
?>
</body>
</html>
```

Программа editRecord.php печатает результаты выполнения библиотечной функции smartRToEdit(). Эта функция принимает в качестве параметра запрос и печатает HTML код, который позволяет пользователю корректно обновить запись.

Просмотр программы updateRecord.php

Функция smartRToEdit() вызывает другую программу PHP, имеющую название updateRecord.php. Эта программа вызывает библиотечную функцию, которая непосредственно вносит изменения пользователя в базу данных.

```
<html>
<head>
<title>Update Record</title>
</head>
<body>
<h2>Update Record</h2>
<?
include "spyLib.php";
$dbConn = connectToSpy();
$fieldNames = "";
$fieldValues = "";
```

```
foreach ($_REQUEST as $fieldName => $value){
  if ($fieldName == "tableName"){
    $theTable = $value;
  } else {
    $fields[] = $fieldName;
    $values[] = $value;
  } // завершение if
} // завершение цикла foreach
print updateRec($theTable, $fields, $values);
print mainButton();
?>
</body>
</html>
```

Для функции `updateRec()` более удобно, если названия полей и значения посылаются в виде массивов, поэтому PHP код в `updateRecord.php` преобразует массив `$_REQUEST` в массив полей и отдельный массив значений. Эти два массива передаются функции `updateRec()`, которая их обрабатывает.

Просмотр программы deleteRecord.php

Программа `deleteRecord.php` работает в уже знакомом нам стиле. Она в основном служит в качестве «упаковки» для функции библиотеки `spyLib`. В данном случае программа просто отсылает название текущей таблицы, название ключевого поля и значения ключа текущей записи функции `delRec()`. Эта функция удалит запись и вернет сообщение об успешности выполнения операции.

```
<html>
<head>
<title>Delete Record</title>
</head>
<body>
<h2>Delete Record</h2>
<?
include "spyLib.php";
$dbConn = connectToSpy();
print delRec($tableName, $keyName, $keyVal);
print mainButton();
?>
</body>
</html>
```

Просмотр программы addRecord.php

Добавление записи на самом деле очень похоже на редактирование записи. Оно требует двух шагов. Программа `addRecord.php` вызывает функцию `tToAdd()`, которая создает форму, позволяющую пользователю добавлять данные в таблицу, которая выбрана в данный момент. В эту функцию необходимо передать только название таблицы, потому что значение ключа будет автоматически сгенерировано функцией `tToAdd()`.

```
<html>
<head>
<title>Add a Record</title>
</head>
<body>
<h2>Add Record</h2>
<?
include "spyLib.php";
$dbConn = connectToSpy();
print tToAdd($tableName);
print mainButton();
?>
</body>
</html>
```

Просмотр программы processAdd.php

Функция `tToAdd()`, вызываемая программой `processAdd.php`, на самом деле не добавляет запись. Вместо этого она помещает на экране форму HTML, которая позволяет пользователю ввести данные новой записи. Когда пользователь отправит эту форму, он перейдет к программе `processAdd.php`, которая вызывается библиотечную функцию `procAdd()`. Функция `procAdd()` генерирует соответствующий код SQL, необходимый для добавления новой записи в таблицу. Чтобы сделать это, `procAdd()` должна знать названия полей и значения. Эти параметры передаются функции в виде массивов, как и в `updateRecord.php`.

```
<html>
<head>
        <title>Process Add</title>
</head>
<body>
<h2>Process Add</h2>
<?
include "spyLib.php";
$dbConn = connectToSpy();
```

```
$fieldNames = "";
$fieldValues = "";
foreach ($_REQUEST as $fieldName => $value){
  if ($fieldName == "tableName"){
    $theTable = $value;
  } else {
    $fields[] = $fieldName;
    $values[] = $value;
  } // завершение if
} // завершение цикла foreach
print procAdd($theTable, $fields, $values);
print mainButton();
?>
</body>
</html>
```

Создаем библиотечный модуль spyLib

Хотя я и описал несколько PHP программ в этой главе, большая их часть очень проста. Основная работа выполняется кодом библиотеки `spyLib`. Наличие такой библиотеки, как `spyLib`, делает программирование данных простым, потому что вам не нужно знать подробностей реализации `spyLib`, чтобы использовать ее. Все, что вам нужно, – понимание предназначения функций библиотеки, знание их параметров и возвращаемых ими значений. Хотя в этой библиотеке и содержится приличное количество кода (порядка 500 строк), в нем практически нет ничего такого, чего бы вы раньше не видели. Лучше внимательно рассмотреть этот код, потому что он поможет вам узнать, как создавать собственные библиотеки. Вы также обнаружите, что нет лучшего способа понять библиотеку, чем посмотреть ее код.

Устанавливаем стиль CSS

Некоторые простейшие элементы могут быть чрезвычайно эффектны. Одним из примеров этой аксиомы является хранение CSS стилей в коде. Каждая программа системы будет работать, используя стиль, определенный в библиотеке. Это означает, что вы можете с легкостью изменять внешний вид всей системы, модифицировав только один блок `<style></style>`.

```
<style type = "text/css">
body{
  background-color: black;
  color: white;
  text-align:center
}
</style>
```

Помните, что когда вы подключаете файл, он интерпретируется как HTML, а не PHP. Это означает, что вы можете поместить любой код HTML во включаемый файл, и он будет автоматически вставлен в результат, как только обнаружится подключенная функция. Я воспользовался этим и включил блок CSS в библиотеку. Если вы хотите включить в библиотеку код PHP, то в этом файле вам будет необходимо выделить его тегами PHP (<? ?>).

Настраиваем глобальные переменные

Другим большим преимуществом библиотечного файла является возможность настройки и использования переменных, которые видимы из любой части системы. Так как каждая программа PHP системы использует библиотеку, все они смогут получить доступ к любой переменной, объявленной в основном разделе библиотечного файла. Конечно же, вам все еще придется использовать ключевое слово global для получения доступа к глобальной переменной из функции.

```
<?
//spyLib.php
//содержит вспомогательные функции для базы данных шпионов
//переменные
$userName = "";
$password = "";
$serverName = "localhost";
$dbName = "chapter10";
$dbConn = "";
$adminPassword = "absolute";
$mainProgram = "spyMaster.php";
```

Я сохранил несколько ключевых значаний данных в глобальных переменных. Переменные `$userName`, `$password` и `$serverName` используются для установки соединения с данными. Я сделал это, потому что я ожидаю, что люди будут использовать заново мою библиотеку для собственных баз данных. Им определенно придется изменить эту информацию, чтобы соединиться с собственной копией MySQL. Намного безопаснее изменить эти данные в глобальных переменных, чем в программном коде. Если вы создаете код для повторного использования, вы можете переместить все его изменяемые части в переменные, как я сделал это здесь.

Переменная `$adminPassword` будет содержать пароль, используемый для редактирования данных системы. Опять же, я хочу, чтобы любой, использующий заново эту библиотеку (включая меня), мог изменить это значение, не углубляясь в код.

Переменная `$mainProgram` содержит унифицированный указатель ресурсов «управляющей» программы системы. В системе шпионов я хочу предоставить доступ к `spyMaster.php` с каждого экрана. Функция `mainButton()` использует значение `$mainProgram` для создания ссылки на основную страницу в любом документе, созданном системой.

Соединяемся с базой данных

Функция `connectToSpy()` является основной в системе шпионов. Она использует переменные уровня системы для создания соединения с базой данных. Она возвращает сообщение об ошибке, если было невозможно установить соединение с базой данных. Функция `mysql_error()` печатает сообщение об ошибке SQL, если установка соединения окончилась неуспешно. Эта информация навряд ли будет полезна пользователю, однако она может помочь вам при отладке системы.

```
function connectToSpy(){
  //соединяется с базой данных шпионов
  global $serverName, $userName, $password;
  $dbConn = mysql_connect($serverName, $userName, $password);
  if (!$dbConn){
   print "<h3>problem connecting to database...</h3>\n";
  } // завершение if
  $select = mysql_select_db("chapter10");
  if (!$select){
    print mysql_error() . "<br>\n";
  } // завершение if
  return $dbConn;
} // завершение connectToSpy
```

Функция `connectToSpy()` возвращает соединение с базой данных, которое потом будет использована в запросах, передаваемых базе данных, во время функционирования системы.

Создаем список из запроса

В библиотеке `spyMaster` я создал несколько функций, которые не используются в финальной версии проекта. Функция `qToList()` является хорошим примером. Эта программа принимает запрос SQL и возвращает просто отформатированный сегмент HTML, описывающий данные запроса. Я обнаружил, что такой формат очень полезен при отладке, потому что при сложном форматировании разобраться в запросе невозможно.

```
function qToList($query){
  //по заданному запросу создает список данных
  global $dbConn;
```

```
  $output = "";
  $result = mysql_query($query, $dbConn);
  //print "dbConn is $dbConn<br>";
//print "result is $result<br>";
  while ($row = mysql_fetch_assoc($result)){
    foreach ($row as $col=>$val){
      $output .= "$col: $val<br>\n";
    } // завершение цикла foreach
    $output .= "<hr>\n" ;
  } // завершение цикла while
  return $output;
} // завершение qToList
```

Создаем HTML таблицу из запроса

Функция `qToTable()` немного мощнее, чем `qToList()`. Она получает любое корректное выражение `SQL SELECT` и создает по нему HTML таблицу. Код этой функции использует `mysql_fetch_field()` для определения названий полей по результату выполнения запроса. Она также просматривает все строки результата выполнения запроса и печатает строку HTML, соответствующую этой записи.

```
function qToTable($query){
  //по заданному запросу автоматически создает HTML таблицу
  global $dbConn;
  $output = "";
  $result = mysql_query($query, $dbConn);
  $output .= "<table border = 1>\n";
  // получить названия столбцов
  //получить названия полей
  $output .= "<tr>\n";
  while ($field = mysql_fetch_field($result)){
    $output .= "  <th>$field->name</th>\n";
  } // завершения цикла while
  $output .= "</tr>\n\n";
  //получить данные строк в виде ассоциативного массива
  while ($row = mysql_fetch_assoc($result)){
    $output .= "<tr>\n";
    //просмотреть каждое поле
    foreach ($row as $col=>$val){
      $output .= "  <td>$val</td>\n";
    } // завершение цикла foreach
    $output .= "</tr>\n\n";
  }// завершение цикла while
  $output .= "</table>\n";
```

```
  return $output;
} // завершение qToTable
```

Функция `qToTable()` вызывается программой `viewQuery.php`, однако она может быть использована в любой момент для преобразования запроса SQL в таблицу HTML (что происходит достаточно часто).

Создаем таблицу HTML для редактирования таблицы SQL

Если у пользователя есть подходящие права, то он может добавлять, редактировать или удалять записи любой таблицы базы данных. Хотя `qToTable()` и подходит для просмотра результатов выполнения любого запроса SQL, она не предоставляет необходимой функциональности. Функция `tToEdit()` основана на `qToTable()`, однако в нее внесено несколько изменений. Во-первых, `tToEdit()` принимает в качестве параметра не запрос, а название таблицы. Вы не можете редактировать объединенные запросы непосредственно, а только таблицы, поэтому это ограничение существенно `tToEdit()` создает запрос, возвращающий все записи заданной таблицы. В дополнение к печатанию данных таблицы, `tToEdit()` добавляет к каждой записи две формы. Одна форма содержит данные, необходимые программе `editRecord.php` для начала редактирования записи. Вторая форма, добавленная к каждой записи, посылает все необходимые для удаления записи данные и вызывает программу `deleteRecord.php`. Еще одна форма, расположенная снизу HTML таблицы, позволяет пользователю добавить запись в эту таблицу. Эта форма содержит информацию, необходимую для программы `addRecord.php`.

```
function tToEdit($tableName){
   //по заданному названию таблицы генерирует таблицу HTML,
содержащую
  //кнопки добавления, удаления и редактирования
  global $dbConn;
  $output = "";
$query = "SELECT * FROM $tableName";
  $result = mysql_query($query, $dbConn);
  $output .= "<table border = 1>\n";
  //получить названия колонок
  //получить названия полей
  $output .= "<tr>\n";
  while ($field = mysql_fetch_field($result)){
    $output .= "  <th>$field->name</th>\n";
  } // завершение цикла while
  //получить название индексного поля (полагая, что оно является
первым)
  $keyField = mysql_fetch_field($result, 0);
  $keyName = $keyField->name;
```

```
    //добавить пустой столбец для добавления, редактирования
и удаления
  $output .= "<th></th><th></th>\n";
  $output .= "</tr>\n\n";
  //получить данные строк в виде ассоциативного массива
  while ($row = mysql_fetch_assoc($result)){
    $output .= "<tr>\n";
    //просмотреть все поля
    foreach ($row as $col=>$val){
      $output .= "  <td>$val</td>\n";
    } // завершение цикла foreach
        //создать маленькие формы для добавления, удаления
и редактирования
    //delete = DELETE FROM <table> WHERE <key> = <keyval>
    $keyVal = $row["$keyName"];
    $output .= <<< HERE
  <td>
    <form action = "deleteRecord.php">
    <input type = "hidden"
           name = "tableName"
           value = "$tableName">
    <input type= "hidden"
           name = "keyName"
           value = "$keyName">
    <input type = "hidden"
           name = "keyVal"
           value = "$keyVal">
    <input type = "submit"
           value = "delete"></form>
  </td>
HERE;
      //обновление: пока не обновляем, однако настроим форму
редактирования
    $output .= <<< HERE
  <td>
    <form action = "editRecord.php"
          method = "post">
    <input type = "hidden"
           name = "tableName"
           value = "$tableName">
    <input type= "hidden"
           name = "keyName"
           value = "$keyName">
    <input type = "hidden"
```

```
            name = "keyVal"
            value = "$keyVal">
<input type = "submit"
            value = "edit"></form>
  </td>
HERE;
    $output .= "</tr>\n\n";

  }// завершение цикла while
    //add = INSERT INTO <table> {values}
    //настроить форму для вставки, посылающую название таблицы
    $keyVal = $row["$keyName"];
    $output .= <<< HERE
  <td colspan = "5">
    <center>
    <form action = "addRecord.php">
    <input type = "hidden"
           name = "tableName"
           value = "$tableName">
    <input type = "submit"
           value = "add a record"></form>
    </center>
  </td>
HERE;
  $output .= "</table>\n";
  return $output;
} // завершение tToEdit
```

Посмотрите внимательно на формы для редактирования и удаления записей. Эти формы содержат скрытые поля для названия таблицы, названия ключевого поля и номера записи. Эта информация будет использована в последующих вызовах функций, создающих запрос для записи, соответствующей заданной строке таблицы.

Создаем основную форму для редактирования записи

Таблица, созданная в `tToEdit()`, вызывает программу `editRecord.php`. Эта программа принимает в качестве параметра запрос на одну запись. Она печатает HTML таблицу, основываясь на результатах выполнения этого запроса. Результат выполнения `rToEdit()` показан на рис. 10.12.

Функция `rToEdit()` создает очень простую HTML таблицу. Каждое поле имеет соответствующее текстовое окно. Преимуществом такого подхода является то, что он работает с любой таблицей. Однако использовать такую форму доста-

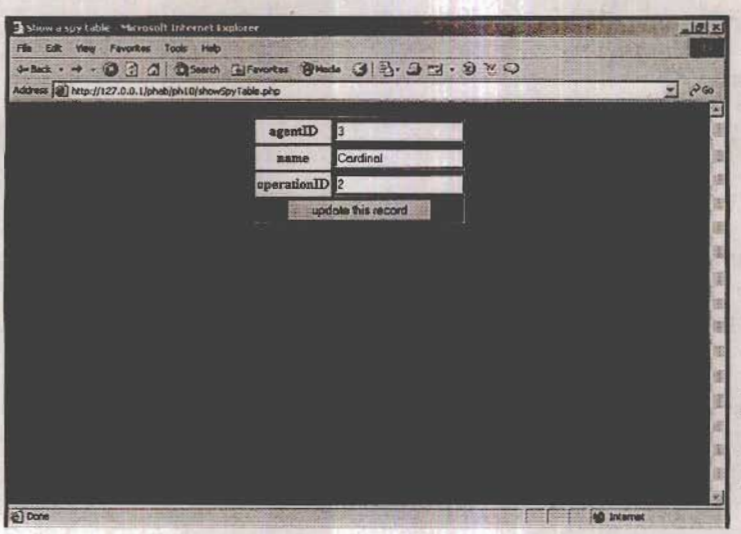


Рис. 10.12. Функция `rToEdit()` является простой, однако результат ее работы может быть опасен

точно рискованно. Во-первых, пользователь не должен иметь возможность изменять первичный ключ, потому что это приведет к редактированию другой записи, что может иметь ужасающие последствия. Во-вторых, поле `operationID` является ссылкой на внешний ключ. В этом поле могут содержаться только целочисленные значения, соответствующие записям в таблице операций. Пользователь никак не сможет определить, какой номер соответствует какой операции. Хуже того, он может ввести в это поле любое число (или, например, любой текст). Результаты этого непредсказуемы, однако практически наверняка они будут плачевны. Я исправлю эти недостатки в функции `smartRToEdit()`, которую рассмотрю позже, однако начну с рассмотрения этой, более простой, функции, потому что `smartRToEdit()` основывается на `rToEdit()`.

```
function rToEdit ($query){
   //по заданному запросу на одну запись создает форму для ее
редактирования
  //работает с любой таблицей, однако позволяет непосредственное
  //редактирование ключей
  //если возможно, используйте вместо нее smartRToEdit
  global $dbConn;
  $output = "";
  $result = mysql_query($query, $dbConn);
  $row = mysql_fetch_assoc($result);
  //получить название таблицы из объекта поля
  $fieldObj = mysql_fetch_field($result, 0);
  $tableName = $fieldObj->table;
```

```
  $output .= <<< HERE
<form action = "updateRecord.php"
      method = "post">
<input type = "hidden"
       name = "tableName"
       value = "$tableName">
<table border = 1>
HERE;
  foreach ($row as $col=>$val){
    $output .= <<<HERE
  <tr>
    <th>$col</th>
    <td>
      <input type = "text"
             name = "$col"
             value = "$val">
    </td>
  </tr>
HERE;
  } // завершение цикла foreach
  $output .= <<< HERE
  <tr>
    <td colspan = 2>
      <center>
      <input type = "submit"
             value = "update this record">
      </center>
    </td>
  </tr>
</table>
HERE;
  return $output;
} // завершение rToEdit
```

Создаем более умную форму редактирования

Функция `smartRToEdit()` построена на основе `rToEdit()`, однако исправляет некоторые ее недостатки. Посмотрите на приведенный ниже код, после чего я объясню, почему он лучше.

```
function smartRToEdit ($query){
   //по заданному запросу на одну запись создает форму для ее
редактирования
```

```
    //Не позволяет пользователю редактировать первое поле
(первичный ключ)
  //генерирует «выпадающий» список для внешних ключей
  //НАМНОГО безопаснее, чем обычная функция rToEdit
  // --ограничения на структуру таблицы --
   //внешние ключи ДОЛЖНЫ называться tableID, где 'table' -
название таблицы
//  (потому что mySQL не распознает указания на внешние ключи)
  // Я также ожидаю, что поле 'name' любой таблицы используется
в качестве
  //внешнего ключа (по тем же причинам)
  global $dbConn;
  $output = "";
  $result = mysql_query($query, $dbConn);
  $row = mysql_fetch_assoc($result);
    //получить название таблицы из объекта поля
  $fieldObj = mysql_fetch_field($result, 0);
  $tableName = $fieldObj->table;
  $output .= <<< HERE
<form action = "updateRecord.php"
      method = "post">
<input type = "hidden"
       name = "tableName"
       value = "$tableName">
<table border = 1>
HERE;
  $fieldNum = 0;
  foreach ($row as $col=>$val){
    if ($fieldNum == 0){
      //это первичный ключ. Не создавать текстовое окно
      //однако вместо этого сохранить значение в скрытом поле
       //пользователь не должен иметь возможности редактировать
первичные
      //ключи
      $output .= <<<HERE
  <tr>
    <th>$col</th>
    <td>$val
      <input type = "hidden"
             name = "$col"
             value = "$val">
    </td>
  </tr>
HERE;
```

```
    } else if (preg_match("/(.*)ID$/", $col, $match)) {
      //это ссылка на внешний ключ
      //получить название таблицы(match[1])
      //создать список, основываясь на названии таблицы и назва-
нии ее полей
      $valList = fieldToList($match[1],$col, $fieldNum, "name");

      $output .= <<<HERE
  <tr>
    <th>$col</th>
    <td>$valList</td>
  </tr>

HERE;

    } else {
      $output .= <<<HERE
  <tr>
    <th>$col</th>
    <td>
      <input type = "text"
             name = "$col"
             value = "$val">
    </td>
</tr>
HERE;
    } // завершение if
    $fieldNum++;
  } // завершение foreach
  $output .= <<< HERE
  <tr>
    <td colspan = 2>
      <center>
      <input type = "submit"
             value = "update this record">
      </center>
    </td>
  </tr>
</table>
</form>
HERE;
  return $output;
} // завершение smartRToEdit
```

Возможность проверки каждого поля записи и предположение о его типе и делают эту функцию «умной». На рис. 10.13 показаны результаты выполнения программы `smartRToEdit()`, поэтому вы можете сравнить их с «не такой умной» функцией на рис. 10.12.

Определяем тип поля

Для этой функции предусмотрено три различных типа полей в записи, которые должны обрабатываться по-разному.

Сначала идет первичный ключ. Если поле является первичным ключом, его значение должно быть передано следующей программе и пользователь не должен иметь возможности редактировать его.

Если поле является ссылкой на внешний ключ другой таблицы, пользователь должен иметь возможность только косвенно редактировать его значение. Лучшим решением для этого является использование «выпадающего» списка, отображающего понятные пользователю значения. Каждое из этих значений соответствует ключу во вторичной записи. Например, на рис. 10.13 изображен список для поля `operationID`. Поле `operationID` является ссылкой на внешний ключ таблицы агентов. Обычная функция `rToEdit()` позволяла пользователю печатать любой индекс в текстовом окне, не указывая при этом, какие данные соответствуют этому индексу. Эта версия функции создает «выпадающий» список, содержащий названия операций. Значения ключа, ассоциированное с этими названиями, хранится в значении атрибута каждого элемента (подробнее об этом будет рассказано в функции `fieldToList()`). Пользователю не обязательно знать все о ссылках на внешние ключи или о реляционных структурах. Он или она просто выбирает операцию из списка. Этот список динамически генерируется каждый раз, когда пользователь добавляет запись, поэтому он всегда содержит все операции агентсва.

Последней является возможность того, что поле не является ни первичным, ни вторичным ключом. В этом случае я будут печатать простое текстовое окно, чтобы пользователь мог ввести значение этого поля. Во всех случаях результат будет отражать текущее значение поля.

Работаем с первичным ключом

Значение первичного ключа намного более важно для программы, чем для пользователя. Я решил отобразить его, однако не разрешать редактировать. Первичные ключи не должны быть редактируемыми. Они могут изменяться только посредством добавления и удаления записей.

Я решил использовать некоторые соглашения для определения того, является ли поле первичным ключом или нет. Я предположил, что первое поле записи

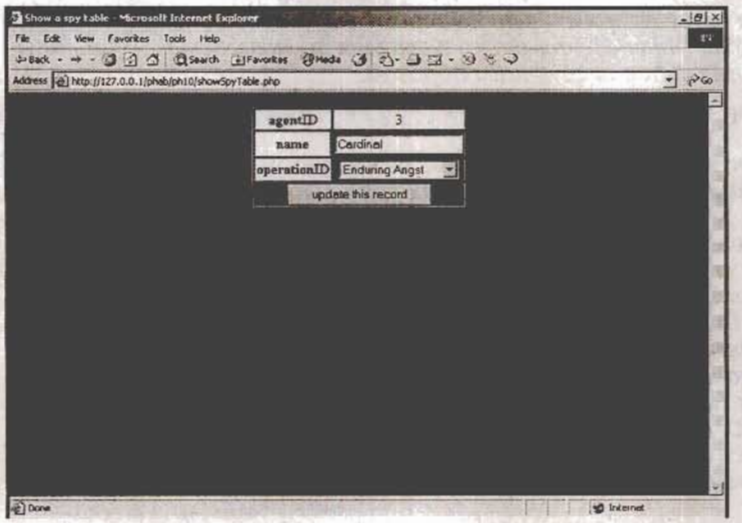


Рис. 10.13. Более умная функция не позволяет пользователю редактировать первичный ключ и предоставляет «выпадающий» список для всех ссылок на внешние ключи

(поле номер 0) является первичным ключом. Это очень распространенное соглашение, однако оно не универсальное. Так как в данной ситуации проектированием данных занимался я, я могу быть уверен, что поле номер 0 любой таблицы является первичным ключом. Для этого поля я просто распечатываю его название и значение в обычной строке таблицы HTML. Я добавил значение ключа в скрытое поле, чтобы следующая программа могла получить к нему доступ.

Распознаем внешние ключи

К сожалению, не существует способа (по крайне мере в MySQL) определить, является ли поле ссылкой на внешний ключ. Мне придется полагаться на соглашение в названии полей, чтобы убедиться, что моя программа распознает поле в качестве ссылки на внешний ключ. Я решил, что все поля внешних ключей моей базы данных будут содержать в своем названии название внешней таблицы, после которого следует значение ID. Например, ссылка на внешний ключ таблицы операций в моей базе данных всегда будет называться `operationID`. Это очень изящное соглашение, потому что оно позволяет легко запомнить названия полей. В `smartRToEdit()` такое соглашение очень важно, потому что является единственным способом определить, является ли поле ссылкой на внешний ключ. Я использовал операторы `else if` для проверки названий всех полей, которые не являются первичными ключами (я определял это с помощью оператора `if`). Функция `preg_match()` позволяет мне использовать регулярные выражения для определения названия поля.

Подробнее о регулярных выражениях: выражение, которое я использовал для определения того, является ли поле ссылкой на внешний ключ, выглядит следующим образом.

```
} else if (preg_match("/(.*)ID$/", $col, $match)) {
```

Оно использует простое, но мощное регулярное выражение : /(.)ID$/ . Это выражение ищет строку, которая оканчивается на ID (вспомните, что $ означает окончание строки). .* означает любое количество символов. Скобки вокруг .* говорят PHP сохранить все символы перед ID в специальном массиве, называемом* `$match`. *Так как в этом выражении используется только один шаблон, все символы, стоящие перед ID, будут содержать название таблицы. Поэтому это регулярное выражение принимает название поля и определяет, оканчивается ли оно на ID. Если да, то первая часть названия поля (все, за исключением ID) сохраняется в* `$match[1]`. *Если* `$col` *содержит* `operationID`, *эта строка вернет* `TRUE` *(потому что* `operationID` *заканчивается на ID) и название таблицы (*`operation`*) сохранится в переменной* `$match[1]`.

Создаем окно списка внешних ключей

Если поле является ссылкой на внешний ключ, необходимо создать окно списка, содержащее значения, которые понятны пользователю. Так как мне потребуется эта возможность в нескольких местах (а `smartRToEdit()` уже является достаточно сложной), я создам новую функцию – `fieldToList()`. Эта функция (подробнее описанная далее в этой главе) создает «выпадающий» список HTML на основании названия поля и таблицы. Вместо того чтобы рассказывать о функции `fieldToList()`, я просто показал, какие она требует параметры и результаты ее выполнения.

Работаем с обычными полями

Любое поле, которое не является первичным или внешним ключом, обрабатывается оператором `else`, который печатает текстовое окно стиля `rToEdit()` для пользовательского ввода. Это будет применяться для всех полей, в которые пользователь может вводить данные, однако это не поможет обнаружить некоторые ошибки, такие, как строковые данные, помещенные в численное поле или размер данных, превышающий длину поля. Это было бы хорошим улучшением кода. Если разработчик не назвал ссылки на внешние ключи в соответствии с моими соглашениями, эти поля можно будет редактировать с помощью текстовых окон, однако ошибки, которые могут возникнуть в `rToEdit()`, все еще являются проблемой.

Обновляем запись

В результате выполнения rToEdit() или smartRToEdit() получится HTML форма, содержащая название таблицы и набор названий полей и значений. updateRecord.php принимает эти значения и преобразует их в массивы, прежде чем вызвать функцию updateRec(). Намного проще работать с полями и значениями, если они представлены в виде массивов, чем с тем аморфным контекстом, который они представляют после smartRToEdit() или rToEdit().

```
function updateRec($tableName, $fields, $vals){
  //ожидает названия записи, массивов полей и значений
  //обновляет базу данных новыми значениями
  global $dbConn;
  $output = "";
  $keyName = $fields[0];
  $keyVal = $vals[0];
  $query = "";
  $query .= "UPDATE $tableName SET \n";
  for ($i = 1; $i < count($fields); $i++){
    $query .= $fields[$i];
    $query .= " = '";
    $query .= $vals[$i];
    $query .= "',\n";
} // завершения цикла for
  //убрать последнюю запятую из результата
  $query = substr($query, 0, strlen($query) - 2);
  $query .= "\nWHERE $keyName = '$keyVal'";
  $result = mysql_query($query, $dbConn);
  if ($result){
     $query = "SELECT * FROM $tableName WHERE $keyName = '$key-
Val'";
    $output .= "<h3>update successful</h3>\n";
    $output .= "new value of record:<br>";
    $output .= qToTable($query);
  } else {
     $output .= "<h3>there was a problem...</h3><pre>$query</
pre>\n";
  } // завершение if
  return $output;
} // завершение updateRec
```

Основным назначением updateRec() является создание SQL выражения UPDATE на основе переданных ей параметров. Эта функция принимает в качестве параметров название таблицы, массив, содержащий названия полей, и еще один

массив, содержащий значения полей. Выражение UPDATE является в основном списком названий и значений полей, которые могут быть легко получены с помощью цикла for, путем просмотра массивов $fields и $vals.

После создания запроса он отправляется базе данных. Успешность или неуспешность выполнения сообщается пользователю.

Удаляем запись

По сравнению с добавлением и обновлением удалить запись достаточно просто. Требуется только название таблицы, название ключевого поля и его значение. Функция deleteRec() принимает эти значения в качестве параметров и использует их для создания выражения SQL DELETE. Как обычно, успешное или неуспешное выполнение операции возвращается как часть результирующей строки.

```
function delRec ($table, $keyName, $keyVal){
  //удаляет запись $keyVal из $table
  global $dbConn;
  $output = "";
  $query = "DELETE from $table WHERE $keyName = '$keyVal'";
  print "query is $query<br>\n";
  $result = mysql_query($query, $dbConn);
  if ($result){
    $output = "<h3>Record successfully deleted</h3>\n";
  } else {
    $output = "<h3>Error deleting record</h3>\n";
  } //завершение if
  return $output;
} // завершение delRec
```

Добавляем запись

Добавление новой записи очень похоже на редактирование записи. Добавление состоит из двух шагов. Первый экран создает страницу для добавления записи, которая очень похожа на экран для редактирования записи. Я использовал методы из функции smartRToEdit(), чтобы убедиться, что ссылки на основной и внешние ключи редактируются корректно.

```
function tToAdd($tableName){
    //по заданному имени таблицы генерирует форму HTML,
позволяющую добавить
   //запись в таблицу. Работает аналогично smartRToEdit при
распознавании
  //внешних ключей
  global $dbConn;
  $output = "";
```

```
  //выполнить запрос, чтобы получить названия полей
  $query = "SELECT * FROM $tableName";
  $result = mysql_query($query, $dbConn);
  $output .= <<<HERE
  <form action = "processAdd.php"
        method = "post">
  <table border = "1">
    <tr>
      <th>Field</th>
<th>Value</th>
    </tr>
HERE;
  $fieldNum = 0;
  while ($theField = mysql_fetch_field($result)){
    $fieldName = $theField->name;
    if ($fieldNum == 0){
      //это поле первичного ключа. Оно будет autoNumber
      $output .= <<<HERE
      <tr>
        <td>$fieldName</td>
        <td>AUTONUMBER
          <input type = "hidden"
                 name = "$fieldName"
                 value = "null">
        </td>
      </tr>
HERE;
    } else if (preg_match("/(.*)ID$/", $fieldName, $match)) {
      //это ссылка на внешний ключ. Используйте fieldToList для
получения
      //объекта выбора для этого поля
      $valList = fieldToList($match[1],$fieldName, 0, "name");
      $output .= <<<HERE
      <tr>
        <td>$fieldName</td>
        <td>$valList</td>
      </tr>
HERE;
    } else {
    //это обычное поле. Напечатать текстовое окно
    $output .= <<<HERE
      <tr>
        <td>$fieldName</td>
        <td><input type = "text"
```

```
                    name = "$fieldName"
                    value = "">
        </td>
      </tr>
HERE;
    } // завершение if
    $fieldNum++;
  } // завершение цикла while
  $output .= <<<HERE
    <tr>
      <td colspan = 2>
        <input type = "hidden"
               name = "tableName"
               value = "$tableName">
        <input type = "submit"
               value = "add record">
      </td>
    </tr>
  </table>
  </form>
HERE;
  return $output;

} // завершение tToAdd
```

Выражение INSERT, создаваемое этой функцией, будет использовать NULL в качестве значения первичного ключа, потому что все таблицы системы установлены в AUTO_INCREMENT. Я использовал те же самые приемы с регулярными выражениями, что и в smartRToEdit(), для распознавания ссылок на внешние ключи. Если они существуют, я создаю «выпадающий» список с помощью fieldToList(), чтобы отобразить все возможные значения для этого поля, и посылаю подходящий ключ. Любое поле, не распознанное как первичный или внешний ключ, будет содержать обычное текстовое окно.

Обрабатываем добавленную запись

Функция tToAdd() посылает результаты своей работы в программу processAdd.php, которая распознает данные совсем как updateRecord.php. Названия и значения полей преобразуются в массивы, которые передаются в функцию procAdd().

```
function procAdd($tableName, $fields, $vals){
  //генерирует запрос INSERT и применяет его к базе данных
  global $dbConn;
```

```
  $output = "";
  $query = "INSERT into $tableName VALUES (";
  foreach ($vals as $theValue){
    $query .= "'$theValue', ";
  } // end foreach
  //удалить конечный пробел и запятую
  $query = substr($query, 0, strlen($query) - 2);
  $query .= ")";
  $output = "query is $query<br>\n";
  $result = mysql_query($query, $dbConn);
  if ($result){
    $output .= "<h3>Record added</h3>\n";
  } else {
    $output .= "<h3>There was an error</h3>\n";
  } // завершение if
  return $output;
} // завершение procAdd
```

Основной задачей `procAdd()` является создание выражения SQL `INSERT` на основе результатов выполнения `tToAdd()`. Этот запрос передается базе данных, а результат выполнения операции сообщается пользователю.

Создаем окно списка из поля

Обе функции – `smartRToEdit()` и `tToAdd()` – требуют «выпадающих» списков HTML, удовлетворяющих специальному шаблону. В обоих случаях мне необходимо создать список, который позволяет пользователю выбрать значение ключа в зависимости от другого поля записи. Этот список должен быть настроен таким образом, чтобы можно было выбрать любое из его значений. Функция `fieldToList()` принимает четыре параметра и использует их для создания как раз такого списка.

```
function fieldToList($tableName, $keyName, $keyVal, $fieldName){
  //по заданной таблице и полю генерирует структуру выбора HTML
  //с именем $keyName. Значениями будут значения ключевого поля
таблицы
  //однако текст будет использован из переменной $fieldName.
  //keyVal определяет текущий выбранный элемент

  global $dbConn;
  $output = "";
  $query = "SELECT $keyName, $fieldName FROM $tableName";
  $result = mysql_query($query, $dbConn);
  $output .= "<select name = $keyName>\n";
  $recNum = 1;
  while ($row = mysql_fetch_assoc($result)){
```

```
    $theIndex = $row["$keyName"];
    $theValue = $row["$fieldName"];
    $output .= <<<HERE
    right now, theIndex is $theIndex and keyVal is $keyVal
  <option value = "$theIndex"
HERE;

    //сделать текущий элемент выбранным
    if ($theIndex == $keyVal){
      $output .= " selected";
    } // завершение if
$output .= ">$theValue</option>\n";
    $recNum++;
  } // завершение цикла while
  $output .= "</select>\n";
  return $output;
} // завершение fieldToList
```

Функция `fieldToList()` начинает работу с генерации запроса, который вернет все записи во внешней таблице. Я создаю объект `HTML SELECT` в зависимости от результатов выполнения запроса. Когда я просматриваю все записи, я проверяю, соответствует ли текущая запись параметру `$keyVal`. Если да, то этот элемент выбирается в HTML.

Создание кнопки, возвращающей на основную страницу

Чтобы упростить навигацию, в конце каждой программы PHP я добавил кнопку, которая возвращает пользователя на основную страницу программы. Программа `mainButton()` создает очень простую форму, которая вызывает программу, задаваемую переменной `$mainProgram`, определенной в начале библиотеки.

```
function mainButton(){
  // создает кнопку для возвращения в основную программу
  global $mainProgram;
  $output .= <<<HERE
<form action = "$mainProgram"
      method = "get">
<input type = "submit"
       value = "return to main screen">
</form>
HERE;
  return $output;
} // завершение mainButton
```

Итоги

Подробности системы `Spy Master` могут запутывать, однако в результате получилась гибкая библиотека, которую можно легко обновить и изменить. Эта система может принимать изменения соответствующей базы данных и может быть адаптирована к совершенно другому набору данных, требуя при этом затраты минимума усилий. Хотя вы и не научились новому синтаксису PHP в этой главе, вы увидели пример написания кода для повторного использования. Вы узнали, как использовать подключаемые файлы для упрощения процесса программирования сложных систем. Вы увидели, как создать библиотечный файл, содержащий вспомогательные процедуры. Вы узнали, как создавать код, который можно адаптировать к различным наборам данных. Вы научились создавать код, который предотвращает появление некоторых пользовательских ошибок путем ограничения выбора только корректными значениями. Вы узнали, как создавать программы, которые помогают связать реляционные структуры данных. Понятия, изученные вами в этой главе, образуют основу всего веб-программирования, связанного с данными, что в свою очередь является основной систем электронной коммерции и системы управления содержимым.

Домашнее задание

1. Добавьте модуль, который позволит пользователю создавать интерактивные запросы. Начните со страницы, которая позволяет пользователю печатать имя агента, по которому возвращается вся информация о нем.
2. После завершения создания основной функциональности программы «поиск агентов», добавьте флажки, позволяющие отображать определенные характеристики агента (задание и навыки).
3. Создайте программы, которые позволят искать по другим характеристикам данных, включая навыки и операции.
4. Изменить базу данных шпионов так, чтобы она поддерживала другой набор данных.

Предметный указатель

Содержание

категория › программирование

ПРИГЛАШАЕМ
АВТОРОВ КНИГ ПО КОМПЬЮТЕРНОЙ ТЕМАТИКЕ, ПЕРЕВОДЧИКОВ И НАУЧНЫХ РЕДАКТОРОВ

ВОЗМОЖНО, ВАС ЗАИНТЕРЕСУЮТ ДРУГИЕ НАШИ ИЗДАНИЯ

Г. Бекет, М. Куннумпурат, Ш. Роди, А. Тост

Java: основы Web-служб

ISBN 5-93378-092-8

С. Силва

Администрирование веб-серверов

ISBN 5-9579-0040-0

Б. Хохгуртль

C# и Java: межплатформенные Web-службы

ISBN 5-9579-0015-X

Д. Стотлемайер

Тестирование Web-приложений

ISBN 5-93378-064-2

К. Уодтке

Информационная архитектура: чертежи для сайта

ISBN 5-93378-081-2

А. Гуруге

Корпоративные порталы на основе XML и Web-служб

ISBN 5-9579-0009-5

По вопросу приобретения книг обращайтесь в издательство по тел.: 333-82-11, с 11^{00} до 17^{00}. Наш адрес: ул. Профсоюзная, д. 84/32, под. 6, эт. 11

ISBN 5-9579-0046-X

КУДИЦ-ОБРАЗ

Тел./факс: (095) 333-82-11, 333-65-67

E-mail: ok@kudits.ru; http://books.kudits.ru

121354, Москва, а/я 18, "КУДИЦ-ОБРАЗ"