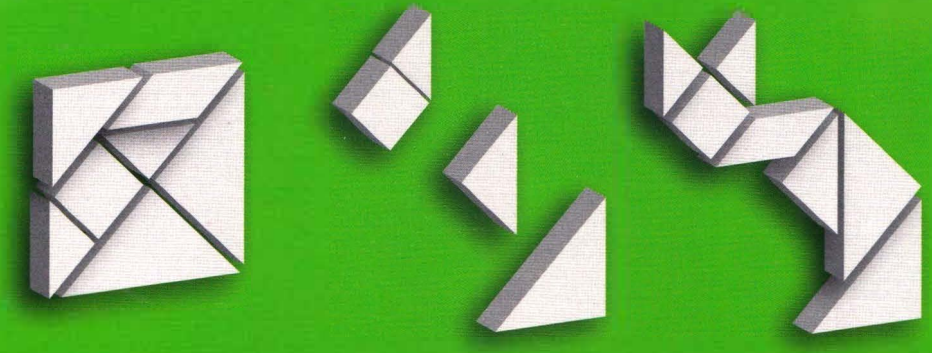
Владимир Дронов

PHP и MySQL

25 уроков для начинающих



+ Материалы на www.bhv.ru

bhv®

Владимир Дронов

PHP и MySQL

25 уроков для начинающих

Санкт-Петербург
«БХВ-Петербург»
2021

УДК 004.43+004.738.5
ББК 32.973.26-018.1
Д75

Дронов В. А.

Д75 PHP и MySQL. 25 уроков для начинающих. — СПб.: БХВ-Петербург, 2021. — 432 с.: ил. — (Для начинающих)

ISBN 978-5-9775-6651-3

В книге 25 иллюстрированных уроков и более 30 практических упражнений. В доступной и наглядной форме, на сквозном примере рассказано о программировании динамических веб-сайтов на языке PHP с применением СУБД MySQL и MariaDB. Описывается программное генерирование веб-страниц, получение данных от пользователей и проверка их на корректность, работа с файлами, программное рисование графики, обработка cookie и сессий, отправка электронной почты. Рассмотрена архитектура «модель-шаблон-контроллер» и структурирование кода для дальнейшего сопровождения сайта. Рассказано о мерах защиты сайта: разграничение доступа, перевод на безопасный протокол HTTPS, защита от атак, шифрование данных. Дан краткий курс программирования веб-служб REST.

Сквозной пример разработки веб-сайта фотогалереи и PHP-фреймворка поможет при построении собственных сайтов. Электронное приложение-архив на сайте издательства содержит коды всех примеров.

Для начинающих веб-разработчиков

УДК 004.43+004.738.5
ББК 32.973.26-018.1

Группа подготовки издания:

Руководитель проекта	*Евгений Рыбаков*
Зав. редакцией	*Екатерина Сависте*
Компьютерная верстка	*Ольги Сергиенко*
Дизайн обложки	*Карины Соловьевой*

Подписано в печать 07.07.20.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 34,83.
Тираж 1000 экз. Заказ № 5967.
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Отпечатано в ОАО «Можайский полиграфический комбинат».
143200, Россия, г. Можайск, ул. Мира, 93.
www.oaompk.ru, тел.: (495) 745-84-28, (49638) 20-685

ISBN 978-5-9775-6651-3

Оглавление

Введение

Когда-то веб-сайты представляли собой наборы обычных веб-страниц, сохраненных в текстовых файлах с расширением htm или html, и назывались *статическими*. Веб-сервер, получив запрос на какую-либо страницу, просто загружал хранящий ее файл и отправлял клиенту.

Все было просто, понятно, и все были довольны. Так продолжалось до тех пор, пока сайты не стали большими и сложными.

Динамические веб-сайты

Предположим, мы делаем сайт-фотогалерею. Пока число картинок в ней не превышает полусотни, мы можем создать отдельную страницу для каждой картинки. Но что, если их тысяча? Создавать тысячу страниц? Сколько времени и сил это отнимет?

Затраты и того, и другого можно радикально сократить, сделав следующее:

- занеся сведения об опубликованных изображениях в базу данных;
- написав программу, извлекающую из базы данных сведения о запрошенном изображении и генерирующую на их основе обычную веб-страницу, которая будет отправлена клиенту.

Тогда вместо того, чтобы писать тысячу веб-страниц — по одной на каждое из опубликованных изображений, — нам понадобится сделать лишь одну программу.

Такие программы, работающие под управлением веб-сервера в составе сайтов, называются *серверными веб-приложениями*, а сайты, включающие их в свой состав, — *динамическими*.

Динамические сайты имеют ряд преимуществ перед статическими:

- значительно уменьшается трудоемкость разработки;
- одни и те же данные, хранящиеся в базе, могут быть представлены по-разному (например, сведения об изображениях — в виде их полного перечня, перечня изображений, опубликованных определенным человеком, перечня, отфильтрованного по заданному слову, и др.);
- можно без особых затрат создавать сайты, чье содержание будет пополняться самими посетителями;
- почтовые веб-службы, интернет-магазины, поисковые системы и прочие подобные сервисы возможно реализовать лишь в виде динамических веб-сайтов.

Из недостатков можно отметить лишь необходимость изучения дополнительного ПО: СУБД (системы управления базами данных) и программной платформы, на которой создаются серверные веб-приложения.

Например, СУБД MySQL и программной платформы PHP.

Почему MySQL и PHP?

- MySQL и PHP — одни из безусловных лидеров на рынке веб-программирования.
- Они предлагают все необходимые функциональные возможности для создания баз данных и написания серверных приложений.
- Они существуют уже более 20 лет, активно развиваются, отлично поддерживаются и будут существовать очень долго.
- Они довольно просты в изучении.
- Они прекрасно работают совместно.
- Они совершенно бесплатны.

Чему научит эта книга?

- Программировать динамические веб-сайты на PHP с применением MySQL.

  «Связка» PHP и MySQL — один из лидеров на рынке веб-программирования. Так что время, потраченное на изучение этих программных продуктов, не будет потеряно зря, а приобретенные знания останутся актуальными еще очень долго.

- Следовать новейшим тенденциям.

  Умение писать веб-службы REST[1] в настоящее время весьма востребовано. А навыки в написании максимально безопасных веб-сайтов и веб-служб, работающих по защищенному протоколу HTTPS, шифрующих данные пользователей и устойчивых к сетевым атакам, — тем более!

- Применять знания на практике.

  Описанию каждого из программных инструментов сопутствует практический пример. Кроме того, на протяжении всей книги описывается разработка полнофункционального сайта фотогалереи.

> Файлы тестового сайта находятся в папке 24\24.4\ex, а файл photogallery.sql с дампом базы данных `photogallery` — в папке 22\22.2\ex сопровождающего книгу электронного архива (см. *приложение 3*).

[1] REST (REpresentational State Transfer) — стиль архитектуры программного обеспечения для распределенных систем, который, как правило, используется для построения веб-служб.

- Использовать PHP на полную мощь.

  Генераторы, регулярные выражения, новые объектные инструменты для работы с базами данных, исключения, средства для валидации данных, функции сильного шифрования и защиты информации — все это позволит вам достичь поставленных целей, написав меньше кода. Пишите меньше — делайте больше!

- Программировать с умом.

  Применив архитектуру «модель-шаблон-контроллер» и разделив программный код на отдельные программные модули, вы существенно упростите дальнейшую разработку и сопровождение готового сайта.

- Писать повторно используемый код.

  Код, выполняющий типовые задачи: работу с базами данных, вывод страниц, проверку корректности данных, полученных от пользователя, и др. — вы сделаете универсальным, оформив в виде функций и классов и составив из них свой собственный фреймворк.

  > *Фреймворк* — программная библиотека, образующая своего рода каркас, на основе которого строится решение (например, веб-сайт). Реализует всю необходимую базовую функциональность — конечному программисту остается лишь добавить программные модули, создающие специфические для решения функции. Позволяет значительно упростить разработку.

  Его вы сможете применить при программировании следующих сайтов. А при необходимости без проблем «пересядете» на любой существующий на рынке готовый PHP-фреймворк: Yii, Laravel, Symfony, CodeIgniter, CakePHP и др. Что, безусловно, пойдет вам как программисту на пользу.

Что вам понадобится?

- *Веб-обозреватель*, разумеется. Автор использовал актуальные на момент написания книги версии Google Chrome и Mozilla Firefox (остальные веб-обозреватели либо аналогичны Chrome, либо малопопулярны).
- *Текстовый редактор* — можно использовать Блокнот, но удобнее работать в каком-либо из чисто «программистских» редакторов: Visual Studio Code (**https://code.visualstudio.com/**) — применялся автором, Atom (**https://atom.io/**), Notepad++ (**https://notepad-plus-plus.org/**), Sublime Text (**https://www.sublimetext.com/**), Brackets (**http://brackets.io/**) и т. п.
- *Пакет веб-хостинга XAMPP* (**https://www.apachefriends.org/ru/index.html**). Автор применял версии 7.3.10 (64-разрядную редакцию), включающую веб-сервер Apache HTTP Server 2.4.41, PHP 7.3.10, СУБД MariaDB 10.4.8, полностью совместимую с MySQL 5.7, и phpMyAdmin 4.9.1. Описание пакета веб-хостинга XAMPP и инструкция по его установке на компьютер приведены в *приложении 2*.

Еще от вас требуется:

- владеть языками HTML, CSS и JavaScript, на которых пишутся сами веб-страницы, оформление для них и клиентские веб-сценарии;
- иметь минимальные навыки веб-верстки;
- знать в общих чертах, как работает веб-сервер.

Типографские соглашения

В книге будут часто приводиться различные языковые конструкции, применяемые при написании программного кода. Для наглядности в них использованы следующие типографские соглашения:

- программный код набран моноширинным шрифтом:

```
for ($i = 0; $i < $arr_cnt; $i++) {
    echo '<p>';
    echo $arr[$i];
    echo '</p> ';
}
```

- в угловые скобки <> заключаются наименования различных параметров и фрагментов кода, которые дополнительно выделяются *курсивом*. В реальный код, разумеется, должны быть подставлены реальные значения. Например:

```
return <возвращаемое значение>;
```

  Сюда вместо подстроки *возвращаемое значение* должно быть подставлено реальное возвращаемое значение;

- в квадратные скобки [] заключаются необязательные параметры и фрагменты кода. Например:

```
getdate([<временна́я отметка>])
```

  Здесь *временна́я отметка* может указываться, а может и не указываться;

- слишком длинные, не помещающиеся на одной строке книги фрагменты, автор разрывал на несколько строк и в местах разрывов ставил знаки ↳. Например:

```
echo "<p>{$ins} дюйм{$ins_ending} = {$cents} ↳
сантиметр{$cents_ending}.</p>";
```

  Приведенный код здесь разбит на две строки, но должен быть набран в одну. Символ ↳ при этом нужно удалить;

- троеточием . . . помечены фрагменты кода, пропущенные ради сокращения объема книги:

```
if (isset($_POST['inches'])) {
    $ins = (double)$_POST['inches'];
    . . .
}
```

Здесь весь код между второй и последней строками пропущен.

Обычно такое можно встретить в дополненных и исправленных фрагментах кода — приведены лишь собственно исправленные строки, а оставшиеся неизмененными пропущены. Также троеточие используется, чтобы показать, в какое место должен быть вставлен вновь написанный код: в начало исходного фрагмента, в его конец или в середину — между уже присутствующими в нем строками.

Внимание!

Все приведенные здесь типографские соглашения относятся только к примерам написания конструкций языков программирования.

ЧАСТЬ I

Язык PHP

- *Урок 1*. Основы PHP
- *Урок 2*. Работа с данными разных типов
- *Урок 3*. Хранение данных: переменные, ссылки и константы
- *Урок 4*. Массивы
- *Урок 5*. Управляющие конструкции
- *Урок 6*. Функции
- *Урок 7*. Объектное программирование: классы и объекты
- *Урок 8*. Объектное программирование: трейты, интерфейсы и пространства имен
- *Урок 9*. Архитектура «модель-шаблон-контроллер»
- *Урок 10*. Генераторы и итераторы
- *Урок 11*. Регулярные выражения
- *Урок 12*. Обработка ошибок. Исключения

Урок 1
Основы PHP

Бóльшая часть современных веб-сайтов имеет в своей основе серверные веб-приложения.

> *Серверное веб-приложение* — программа, работающая совместно с веб-сервером, запускаемая по запросу от веб-обозревателя и выдающая в качестве результата обычную веб-страницу.

Для разработки серверных приложений применяются самые разные платформы: Python, Ruby, JavaScript, .NET, Java и др. Но наиболее популярной является PHP.

> *PHP* (акроним от PHP: Hypertext Preprocessor, PHP: препроцессор гипертекста) — программная платформа для разработки серверных веб-приложений на одноименном языке программирования.

PHP-приложения исполняются под управлением *исполняющей среды* (или *интерпретатора*) — вспомогательной программы, которая считывает исходный текст приложения (программисты называют его *программным кодом*) и переводит его в команды, «понятные» центральному процессору компьютера.

> ***Замечание по поводу веб-сервера и исполняющей среды PHP***
>
> Чтобы эти две программы работали совместно, их необходимо соответственно настроить. Веб-сервер Apache HTTP Server и исполняющая среда PHP, входящие в состав пакета хостинга XAMPP (см. *приложение 2*), уже настроены нужным образом.

1.1. Экспресс-курс PHP-программирования

Язык PHP очень прост (что и неудивительно — ведь все языки программирования создавались с целью максимально облегчить жизнь разработчика). Мы изучим его основы на примерах.

1.1.1. Программа. Выражения

Любая *программа* (в том числе и серверное веб-приложение) — это последовательность инструкций, которые компьютер должен выполнить для достижения нужного программисту результата.

> *Выражение* — одна из инструкций, составляющих программу. Описывает одно законченное действие: вычисление какой-либо величины, вывод ее на экран, проверку существования файла, получение записи из базы данных и т. п.

Выражение PHP *обязательно* должно заканчиваться знаком точки с запятой (`;`).

Выражения выполняются последовательно, одно за другим, в порядке от начала программы к ее концу.

Выражения могут быть как простыми, так и сложными, состоящими из более простых выражений.

1.1.2. Числа, переменные, операторы, функции, строки, типы данных и вывод результатов

Любая программа манипулирует какими-либо значениями — например, числами: как целыми, так и имеющими дробную часть (*вещественными*). Числа записываются по обычным правилам, только в вещественных числах разделителем целой и дробной частей служит точка (`.`). Примеры: `100`, `2.5`, `15`, `-3`.

Значение можно сохранить для дальнейшего использования в переменной.

> *Переменная* — ячейка памяти, рассчитанная на временное хранение одного значения. Должна иметь уникальное имя, по которому и выполняется обращение к переменной.

В PHP имя переменной должно предваряться знаком доллара (`$`).

Присвоим переменной с именем `$value1` числовое значение 100, используя оператор присваивания = (знак «равно»):

```
$value1 = 100;
```

> *Присваивание* — занесение какого-либо значения в переменную. При этом значение, ранее хранившееся в переменной, теряется. Если переменная еще не существует, она создается автоматически.

> *Оператор* — выполняет над заданными значениями (операндами) какое-либо элементарное действие (например, оператор присваивания заносит значение в переменную, а оператор сложения вычисляет сумму двух операндов).

Создадим переменную с именем `$value2`, присвоив ей числовое значение 2,5:

```
$value2 = 2.5;
```

Арифметика в стиле PHP — оператор сложения + (знак «плюс»):

```
$value3 = 45 + 15;
```

Результат, возвращенный оператором сложения «плюс»: (45 + 15 = 60), поступит оператору присваивания =, который занесет его в переменную `$value3`.

> *Возврат результата* — так в PHP называется выдача оператором результата вычислений. Результат возвращают не все операторы.

Оператор сложения имеет больший приоритет, нежели оператор присваивания. Поэтому в написанном нами выражении сначала будет выполнено сложение чисел 45 и 15, а уже потом полученная сумма присвоится переменной `$value3`.

Приоритет оператора — задает очередность выполнения операторов: операторы с большим приоритетом выполняются раньше операторов с меньшим приоритетом.

Примечание

Таблица приоритетов операторов приведена в *приложении 1*.

Вот еще один пример выражения, в котором используются операторы с разным приоритетом (знак звездочки * здесь — это оператор умножения):

```
$value4 = 14 + 5 * 7;
```

Поскольку приоритет оператора умножения * выше, чем оператора сложения +, сначала будет выполнено умножение 5 на 7, а уже потом к получившемуся произведению прибавится 14. В результате получится 49.

Порядок выполнения операторов можно менять с помощью круглых скобок (). Оператор в круглых скобках будет выполнен раньше, независимо от его приоритета. Пример:

```
$value5 = (14 + 5) * 7;
```

Здесь сначала выполнится сложение 14 и 5, после чего полученная сумма умножится на 7, дав в результате 133.

В арифметических вычислениях можно использовать значения, ранее сохраненные в переменных. Например, умножим значение переменной $value1 на значение переменной $value2, вычтем из суммы значение из переменной $value3, а разность присвоим переменной $result1:

```
$result1 = $value1 * $value2 - $value3;
```

Здесь мы использовали оператор вычитания - (знак «минус»). В переменной $result1 окажется число: (100 × 2,5 – 60) = 190.

Обращение к переменной выполняется простым указанием имени этой переменной.

Если обращение к переменной записано левее оператора присваивания, переменной будет присвоено новое значение, а если справа — из переменной будет извлечено значение.

Испробуем оператор деления / (прямой слеш):

```
$result2 = 7 / 3;
```

В переменной $result2 окажется вещественное число 2,3333333333333[1]. Слишком длинное, на наш взгляд... Давайте сократим его до второго знака после запятой, для чего вызовем функцию round.

[1] PHP при расчетах с вещественными числами манипулирует не более чем 13-ю знаками после запятой.

> *Функция* — выполняет над переданными ей значениями (*параметрами*) какое-либо сложное действие: округление заданного числа, проверку существования файла с указанным путем, подключение к базе данных и т. п. Многие функции возвращают результат.

> *Вызов функции* — запуск функции на выполнение с передачей ей необходимых параметров.

```
$result2 = round($result2, 2);
```

Чтобы вызвать функцию `round`, мы написали ее имя, поставили круглые скобки и в них, через запятую, указали нужные параметры: значение, требующее округления (оно будет взято из переменной `$result2`), и количество знаков после запятой, до которых первое число будет округлено. Возвращенный функцией результат (2,33) мы присвоили той же переменной `$result2`, чтобы сэкономить память компьютера.

Помимо чисел, мы можем манипулировать строками.

> *Строка* — последовательность произвольных символов, взятая в одинарные или двойные кавычки.

```
$str1 = 'Программная платформа';
```

Оператор «точка» (`.`) выполняет объединение (*конкатенацию*) строк:

```
$str2 = $str1 . ' PHP!';
```

Переменной `$str2` будет присвоена результирующая строка `'Программная платформа PHP!'`.

Как видим, PHP может работать с данными целочисленного, вещественного и строкового типов.

> *Тип данных* — определяет природу обрабатываемого программой значения и набор операций, которые можно над ним выполнить.

Вычислив нужный результат в виде числа или строки, его следует вывести.

> *Вывод* в PHP — вставка указанного значения (обычно результата вычислений) в заданное место HTML-кода генерируемой веб-приложением страницы.

Вывод выполняется языковой конструкцией `echo`, после которой, через пробел, указывается выводимое значение:

```
echo $result1;                    // Выведет: 190
echo $str2;                       // Выведет: Программная платформа PHP!
```

Можно указать несколько значений через запятую — тогда они будут выведены вплотную друг к другу:

```
echo '<strong>', $str2, '</strong>';
                  // Выведет: <strong>Программная платформа PHP!</strong>
$s1 = 'PHP';
```

```
$s2 = 'MySQL';
echo $s1, $s2;                          // Выведет: PHPMySQL
// Поскольку значения выводятся вплотную, их следует явно разделять
// строкой, содержащей пробел
echo $s1, ' ', $s2;                     // Выведет: PHP MySQL
```

Полезно знать...

- При написании выражений в PHP используется тот же синтаксис, что и при записи обычных алгебраических формул. Говорят, что PHP использует *алгебраическую нотацию*.
- Целые и вещественные числа в PHP, хоть и относятся к разным типам, но их природа одинакова, и набор выполняемых над ними операций также схож.
- Числа могут быть записаны в виде строк: `'20'`, `'1234.567'`, `'-10'`. Над ними можно выполнить арифметические действия — в этом случае они будут автоматически преобразованы в настоящие числа (более подробно о преобразовании типов рассказывается на *уроке 2*).
- Строки, взятые в одинарные и двойные кавычки, обрабатываются по-разному (как именно, будет рассказано на *уроке 2*).
- Языковая конструкция `echo`, выводящая данные, не относится ни к операторам, ни к функциям, а занимает некое промежуточное положение.

1.1.3. Написание PHP-приложений. Наше первое серверное веб-приложение

Код программы на языке PHP записывается в обычном текстовом файле, который должен иметь расширение php.

Какой текстовый редактор выбрать?

Можно использовать Блокнот, стандартно поставляемый в составе Windows, но удобнее применять какой-либо редактор, специально предназначенный для программистов, — например: Visual Studio Code (**https://code.visualstudio.com/**) или Atom (**https://atom.io/**).

Файлы с PHP-программами рекомендуется сохранять в кодировке UTF-8.

Замечание для пользователей Блокнота

При сохранении PHP-программы, набранной в Блокноте, в раскрывающемся списке **Тип файла** диалогового окна сохранения файла следует выбрать пункт **Все файлы (*.*)** — иначе Блокнот добавит к заданному имени файла расширение *txt*. В раскрывающемся списке **Кодировка** нужно указать кодировку файла — **UTF-8**.

Файл с PHP-приложением может содержать как один лишь PHP-код (этот случай мы рассмотрим позже), так и смесь HTML- и PHP-кода (такие веб-приложения называются *серверными веб-страницами*).

Каждый фрагмент PHP-кода, помещенный в HTML-код, должен быть заключен в парный тег `<?php . . . ?>`.

У последнего во фрагменте PHP-кода выражения конечный символ ; можно не ставить.

Весь HTML-код, присутствующий в таком файле, переносится в результирующую веб-страницу без изменений.

Упражнение 1.1

Напишем на PHP наше первое серверное веб-приложение, переводящее величину 20 дюймов в сантиметры и выдающее результат в виде веб-страницы. Это будет серверная страница (смесь HTML- и PHP-кода) — их писать проще.

> ***Как перевести дюймы в сантиметры***
>
> 1 дюйм равен 2,54 см. Следовательно, для перевода достаточно умножить величину в дюймах на 2,54.

1. В папке 1\!sources сопровождающего книгу электронного архива (см. *приложение 3*) найдем файл convertor.php и скопируем его в корневую папку веб-сервера из комплекта пакета XAMPP (при установке с параметрами по умолчанию это папка c:\xampp\htdocs).

   > ***Примечание***
   >
   > О работе с пакетом XAMPP рассказывается в *приложении 2*.

2. Откроем копию файла convertor.php в текстовом редакторе.

   Изначально он хранит «заготовку» для написания приложения, содержащую лишь HTML-код (листинг 1.1).

Листинг 1.1. Начальный код серверного веб-приложения convertor.php

```
<!doctype html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Конвертор</title>
    </head>
    <body>
        <h1>Преобразование из дюймов в сантиметры</h1>
        <p> дюймов = сантиметров.</p>
    </body>
</html>
```

3. Сохраним изначальное значение 20 дюймов в переменной с именем `$ins`, переведем его в сантиметры, результат присвоим переменной `$cents` и округлим до целых.

   Вставим под HTML-кодом, создающим заголовок, фрагмент PHP-кода, который выполнит все эти действия (здесь и далее добавления и правки в уже написанном коде выделены полужирным шрифтом):

```
<h1>Преобразование из дюймов в сантиметры</h1>
<?php
    $ins = 20;
    $cents = $ins * 2.54;
    $cents = round($cents);
?>
<p> дюймов = сантиметров.</p>
```

4. В HTML-код, создающий абзац, сразу после открывающего тега `<p>`, вставим фрагмент PHP-кода, выполняющий вывод величины в дюймах (переменная `$ins`):

```
<p><?php echo $ins ?> дюймов = сантиметров.</p>
```

5. В том же абзаце после знака равенства поставим пробел и впишем PHP-фрагмент, выводящий переведенную в сантиметры величину из переменной `$cents`:

```
<p><?php echo $ins ?> дюймов = <?php echo $cents ?> сантиметров.</p>
```

6. Сохраним файл convertor.php.

7. Запустим веб-сервер, входящий в пакет XAMPP (как это сделать, описано в *приложении 2*).

8. Откроем любой веб-обозреватель (автор использовал Google Chrome) и в нем перейдем по интернет-адресу: **http://localhost/convertor.php**.

Веб-обозреватель выведет результат выполнения нашего веб-приложения — страницу, показанную на рис. 1.1.

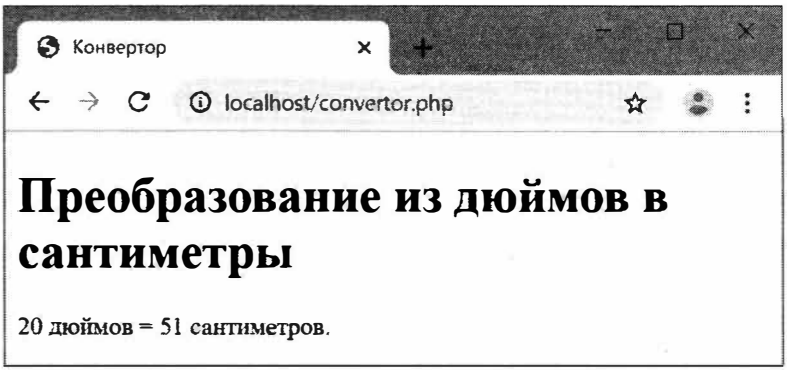


Рис. 1.1. Веб-страница, сгенерированная серверным веб-приложением convertor.php

Веб-приложение «говорит» не по-русски?

Не будем пока обращать внимание на несоблюдение правил согласования имен числительных в русском языке (мы исправим это на *уроке 5*).

Внимание!

В дальнейшем перед тем, как проверить веб-приложение в работе, не забываем:

- сохранить все файлы, хранящие код приложения;
- скопировать их в корневую папку веб-сервера из состава пакета XAMPP (по умолчанию: *c:\xampp\htdocs*), затерев имеющиеся там старые копии;
- запустить входящий в комплект XAMPP веб-сервер, если он еще не запущен;
- если приложение использует базу данных, запустить комплектный MySQL.

Напомним, что о работе с пакетом XAMPP рассказывается в *приложении 2*.

В листинге 1.2 показан HTML-код страницы, сгенерированной нашим первым веб-приложением.

Листинг 1.2. HTML-код результирующей страницы, сгенерированной веб-приложением convertor.php

```
<!doctype html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Конвертор</title>
    </head>
    <body>
        <h1>Преобразование из дюймов в сантиметры</h1>
        <p>20 дюймов = 51 сантиметров.</p>
    </body>
</html>
```

Оба значения, вычисленные приложением, выделены в листинге рамками. Что любопытно — в коде страницы нет никаких указаний на то, что она была создана программно.

Полезно знать...

Загрузив файл с запущенным PHP-приложением, исполняющая среда предварительно компилирует его, сохраняя результат в оперативной памяти. При последующем запуске того же приложения исполняется уже откомпилированный код.

> *Компиляция* — перевод программного кода перед выполнением в компактное внутреннее представление с целью повысить быстродействие программы.

Если PHP-файл с кодом был изменен, он будет перекомпилирован.

Сделайте сами

Доработайте приложение convertor.php таким образом, чтобы оно также переводило в сантиметры величину 27 дюймов (рис. 1.2). Подсказка: продублируйте и соответственно исправьте фрагмент PHP-кода, выполняющий вычисления, и HTML-код, выводящий абзац с результатами.

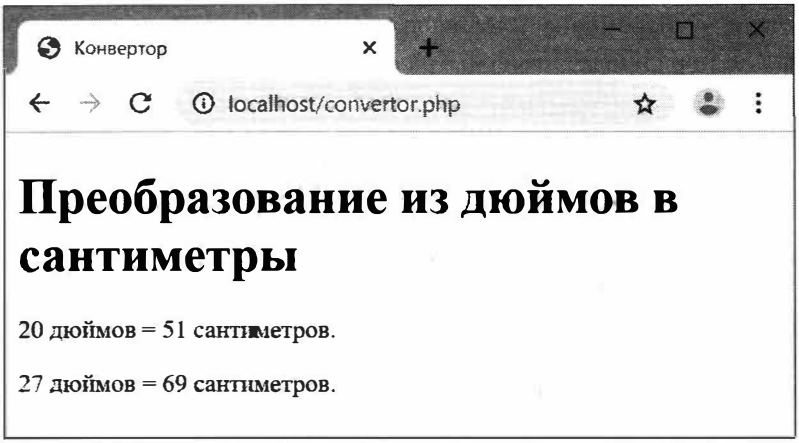


Рис. 1.2. Доработанное приложение convertor.php, преобразующее в сантиметры две величины

1.1.4. Массивы, циклы, условия и логические величины

Переменная может хранить лишь одно значение. Если нужно сохранить несколько значений, удобно создать массив.

> *Массив* — набор значений разных типов, хранящихся в одной переменной.

> *Элемент массива* — отдельное значение, хранящееся в массиве.

Массив создается простым указанием его элементов в квадратных скобках через запятую. После создания его следует присвоить какой-либо переменной:

```
$arr = ['HTML', 'CSS', 'JavaScript', 'PHP', 'MySQL'];
```

Извлечь значение любого элемента можно, записав имя переменной, хранящей массив, и поставив в квадратных скобках индекс этого элемента. Например, так можно извлечь второй элемент массива:

```
echo $arr[1];                                    // CSS
```

> *Индекс* — порядковый номер элемента массива. Первый элемент имеет индекс 0, второй — 1, а последний — индекс, равный размеру массива за вычетом 1.

> *Размер массива* — количество элементов, хранящихся в массиве.

Вычислить размер массива можно, вызвав функцию `count` и передав ей в качестве единственного параметра нужный массив:

```
$arr_cnt = count($arr);
```

Выведем все элементы массива `$arr` в виде набора абзацев (тегов `<p>`). Для этого сначала присвоим переменной-счетчику `$i` индекс первого элемента:

```
$i = 0;
```

> *Счетчик* — переменная, хранящая последовательно увеличивающиеся значения — порядковые номера. Традиционно имеет имя `$i`.

Выведем первый элемент массива:

```
echo '<p>', $arr[$i], '</p>';                    // <p>HTML</p>
```

Увеличим значение счетчика на 1, применив оператор инкремента ++ (два «плюса»):

```
$i++;
```

> *Инкремент* — увеличение значения переменной на единицу. Выполняется быстрее обычного сложения.

```
echo '<p>', $arr[$i], '</p> ';                   // <p>CSS</p>
```

Не достигнут ли последний элемент массива? Проверить это можно, убедившись, что индекс текущего элемента массива (хранящийся в счетчике `$i`) меньше размера массива (переменная `$arr_cnt`). Используем оператор «меньше» (<) и сохраним возвращенный им результат в переменной `$flag`:

```
$flag = $i < $arr_cnt;                           // TRUE
```

Оператор «меньше» возвращает результат сравнения в виде значения логического типа. В нашем случае он вернул `TRUE`, указав, что индекс текущего элемента меньше размера массива, и конец массива не достигнут.

> *Логическая величина* может иметь только два значения: `TRUE` («истина», сравнение выполнилось) и `FALSE` («ложь», сравнение не выполнилось).

> *Условие* — выражение, обычно сравнивающее два значения и выдающее результат в виде величины логического типа.

Что-то не хочется нам набирать еще трижды выражения для вывода очередного элемента массива на экран, инкремента счетчика и условия, проверяющего, не достигнут ли конец массива... Давайте запишем код компактнее — применив цикл.

> *Цикл* — набор выражений, исполняющийся многократно, пока остается истинным какое-либо условие.

Цикл наиболее универсального вида создается языковой конструкцией `for` и носит название *цикла* `for` (рис. 1.3).

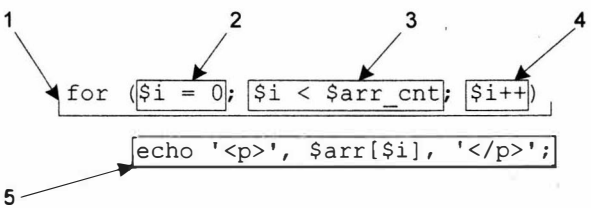


Рис. 1.3. Цикл `for`

Запись этого цикла на языке PHP включает в себя:

- *заголовок цикла* (**1**) — содержит конструкцию `for` и круглые скобки, в которых записаны три выражения, разделенные знаками точки с запятой (`;`) и управляющие выполнением цикла;
- *установку* (**2**) — выполнится *перед первой итерацией* цикла и задаст начальное значение переменной-счетчика;

  > *Итерация* (или *проход*) — одно исполнение цикла.

- *условие* (**3**) — выполнится *перед каждой итерацией* и выяснит, продолжать выполнение цикла или прервать его. Если в результате вычисления условия получится `TRUE`, очередная итерация будет выполнена, если `FALSE` — цикл завершится. У нас условие проверяет, не достигнут ли еще конец массива;
- *приращение* (**4**) — выполнится *после каждой итерации* и изменит (обычно инкрементирует) значение счетчика;
- *тело цикла* (**5**) — выражение, которое должно исполняться в цикле, его «полезная нагрузка».

```
// Результат выполнения цикла:
// <p>HTML</p> <p>CSS</p> <p>JavaScript</p> <p>PHP</p> <p>MySQL</p>
```

Если тело цикла состоит из нескольких выражений или содержит фрагмент HTML-кода, то оно должно быть оформлено в виде блока.

> *Блок* (или *блочное выражение*) включает в себя произвольное количество единичных выражений.

Блок создается фигурными скобками `{` и `}`, в которые помещаются выражения, входящие в его состав.

Примеры циклов, записанных с использованием блоков:

```
// Цикл, тело которого состоит из трех PHP-выражений
for ($i = 0; $i < $arr_cnt; $i++) {
    echo '<p>';
    echo $arr[$i];
    echo '</p> ';
}

<?php // Цикл, тело которого включает фрагмент HTML-кода
      for ($i = 0; $i < $arr_cnt; $i++) { ?>
          <p><?php echo $arr[$i] ?></p>
<?php } ?>
```

Упражнение 1.2

Напишем приложение convertor2.php, которое преобразует в сантиметры величины 20, 24, 27, 32 и 45 дюймов. Для хранения исходных величин применим массив, а для их преобразования и вывода результатов — цикл.

1. В папке 1\!sources сопровождающего книгу электронного архива (см. *приложение 3*) найдем знакомый нам файл convertor.php, скопируем его в корневую папку веб-сервера и переименуем в convertor2.php.
2. Откроем файл convertor2.php в текстовом редакторе.
3. Массив со значениями в дюймах мы присвоим переменной `$aIns`. Сразу же вычислим и сохраним в переменной `$cnt` размер этого массива — он понадобится нам для выполнения цикла.

   Вставим под HTML-кодом, формирующим заголовок, фрагмент PHP-кода, который выполнит эти действия (здесь и далее добавления и правки в уже написанном коде выделены полужирным шрифтом):

```
<h1>Преобразование из дюймов в сантиметры</h1>
<?php
    $aIns = [20, 24, 27, 32, 45];
    $cnt = count($aIns);
?>
<p> дюймов = сантиметров.</p>
```

4. Теперь все готово для написания цикла, который переберет элементы массива `$aIns`, переведет каждый из них в сантиметры и для каждого выведет на страницу абзац с результатом перевода. Поскольку тело цикла будет состоять из нескольких выражений и включать фрагмент HTML-кода, мы оформим его в виде блока.

   Добавим в тот же фрагмент заголовок цикла и открывающую фигурную скобку:

```
<?php
    $aIns = [20, 24, 27, 32, 45];
    $cnt = count($aIns);
    for ($i = 0; $i < $cnt; $i++) {
?>
```

   Непосредственно под HTML-кодом, создающим абзац, вставим PHP-фрагмент с закрывающей скобкой:

```
<p> дюймов = сантиметров.</p>
<?php } ?>
```

   Таким образом мы сделали этот абзац частью тела цикла.

5. После открывающей фигурной скобки в предыдущем PHP-фрагменте запишем код, который станет частью тела цикла, извлечет из массива значение очередного элемента и преобразует его в сантиметры:

```
<?php
    . . .
    for ($i = 0; $i < $cnt; $i++) {
        $ins = $aIns[$i];
        $cents = $ins * 2.54;
        $cents = round($cents);
?>
```

6. В HTML-код, выводящий абзац, впишем два PHP-фрагмента, которые выполнят вывод результатов:

```
<p><?php echo $ins ?> дюймов = <?php echo $cents ?> сантиметров.</p>
```

Проверим написанное веб-приложение в работе, выполнив переход по интернет-адресу: **http://localhost/convertor2.php**. Результат показан на рис. 1.4.

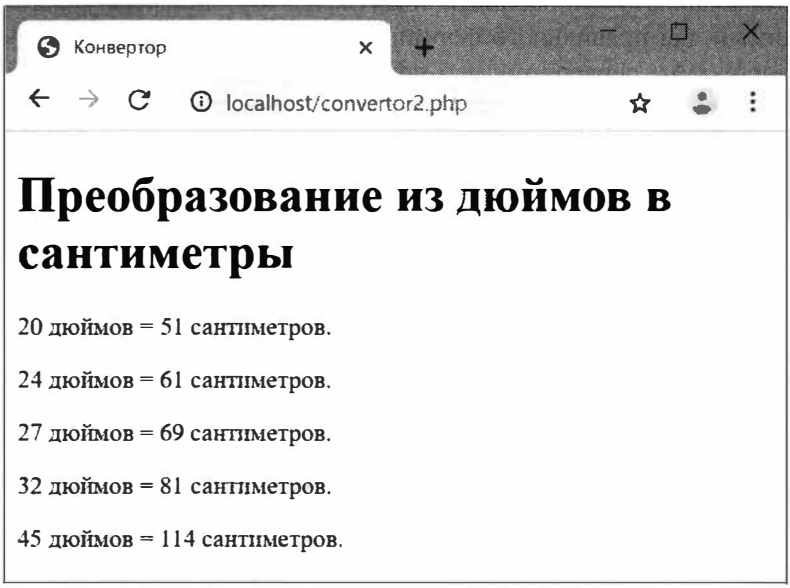


Рис. 1.4. Результат работы веб-приложения convertor2.php

Полезно знать...

Мы можем записать тело цикла компактнее:

```
<?php
    . . .
    for ($i = 0; $i < $cnt; $i++) {
?>
    <p><?php echo $aIns[$i] ?> дюймов =
    <?php echo round($aIns[$i] * 2.54) ?> сантиметров.</p>
<?php } ?>
```

Здесь все необходимые вычисления выполняются непосредственно в конструкциях echo, и их результаты сразу выводятся на страницу. Так мы сэкономим оперативную память и повысим быстродействие (поскольку не создаем переменные и не тратим время на обращение к ним).

Сделайте сами

Доработайте приложение convertor2.php, чтобы оно также переводило в сантиметры величины 19 и 80 дюймов.

1.1.5. Получение данных от пользователя и условные выражения

Для ввода данных на страницах предусматриваются веб-формы с элементами управления. В теге, создающем каждый элемент управления, записывается наименование этого элемента.

> *Наименование* — уникальный идентификатор элемента управления, необходимый для правильного формирования данных, которые будут отосланы из веб-формы серверному веб-приложению. Задается атрибутом тега `name`, поддерживаемым всеми тегами, что создают элементы управления: `<input>`, `<textarea>` и `<select>`.

Рассмотрим пример веб-формы с полями для ввода имени и пароля пользователя. У первого поля указано наименование `username`, у второго — `userpassword`. Данные отправляются методом `POST` — он указан в атрибуте `method` тега `<form>`:

```
<form action="processdata.php" method="post">
    Имя: <input type="text" name="username" value="user">
    Пароль: <input type="password" name="userpassword" value="123456">
    <input type="submit" value="Отправить">
</form>
```

Данные из этой формы в PHP-приложении можно получить из переменной `$_POST`, созданной самой исполняющей средой и хранящей ассоциативный массив.

> *Ассоциативный массив* — массив, элементы которого доступны не по целочисленным, а по произвольно заданным строковым индексам (*ключам*).

Элементы этого массива хранят значения, занесенные в элементы управления, и доступны через ключи, совпадающие с наименованиями элементов управления. Например, так можно получить значение, занесенное в поле ввода с наименованием `username`:

```
$name = $_POST['username'];
```

Если пользователь ничего не ввел в это поле, элемент `username` в ассоциативном массиве `$_POST` будет отсутствовать. Проверить, существует ли элемент в массиве, можно с помощью функции `isset`. Она возвращает `TRUE`, если переданный ей в качестве параметра элемент массива существует, и `FALSE` в противном случае:

```
$flag = isset($_POST['username']);
```

Если элемент `username` в массиве `$_POST` существует, нужно выполнить фрагмент кода, извлекающий значение этого элемента и производящий вход на сайт. В противном случае следует выполнить другой фрагмент кода — например, выводящий соответствующее сообщение. Для этого применяется условное выражение.

> *Условное выражение* включает в себя условие (аналогичное тому, что применялось в цикле из *разд. 1.1.4*) и два выражения. Если условие возвращает `TRUE`, выполняется первое выражение, если возвращается `FALSE` — второе.

Пример условного выражения можно увидеть на рис. 1.5.

Здесь:

- *условие* (1) — записывается в скобках после языковой конструкции `if`;
- *выражение if* (2) (также можно использовать блок, как это сделали мы) — записывается после скобок с условием и выполняется, если условие вычислит `TRUE` (у нас оно извлекает из массива `$_POST` имя и пароль);
- *выражение else* (3) (или блок) — записывается после языковой конструкции `else` и выполняется, если условие вернет `FALSE` (у нас выводит абзац с сообщением).

Рис. 1.5. Условное выражение

Языковая конструкция `else` вместе с *выражением else* может отсутствовать — тогда в случае ложности *условия* ничего не произойдет.

Упражнение 1.3

Напишем более полезное серверное приложение convertor3.php, которое переводит в сантиметры «дюймовое» значение, введенное пользователем в веб-форму.

1. В папке 1\!sources сопровождающего книгу электронного архива (см. *приложение 3*) найдем файл convertor3.php и скопируем его в корневую папку веб-сервера.
2. Откроем копию файла convertor3.php в текстовом редакторе и посмотрим на хранящийся там код (листинг. 1.3).

Листинг 1.3. Начальный код серверного веб-приложения convertor.php

```
<!doctype html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Конвертор</title>
    </head>
    <body>
        <h1>Преобразование из дюймов в сантиметры</h1>
```

```
        <form action="convertor3.php" method="post">
            <p>Величина в дюймах:
            <input type="number" name="inches" size="10"></p>
            <p><input type="submit" value="Преобразовать"></p>
        </form>
    </body>
</html>
```

Запомним наименование, указанное у поля для ввода величины в дюймах, — `inches` (выделено рамкой).

Обратим внимание, что это приложение отправляет введенные в форму данные самому себе — в атрибуте `action` тега `<form>` указано имя файла `convertor3.php`. Это обычная практика PHP-программирования, позволяющая сократить количество веб-приложений, составляющих сайт.

3. Под закрывающим тегом формы `</form>` вставим фрагмент PHP-кода, который переведет заданную величину в сантиметры и выведет абзац с результатом (добавленный код выделен полужирным шрифтом):

```
</form>
<?php
    if (isset($_POST['inches'])) {
        $ins = $_POST['inches'];
        $cents = round($ins * 2.54);
        echo '<p>', $ins, ' дюймов = ', $cents, ' сантиметров.</p>';
    }
?>
```

4. В веб-обозревателе перейдем по интернет-адресу: **http://localhost/convertor3.php** и увидим страницу, показанную на рис. 1.6.

При первом обращении к приложению convertor3.php никаких данных методом `POST` ему передано не будет. В ассоциативном массиве `$_POST` не будет элемента

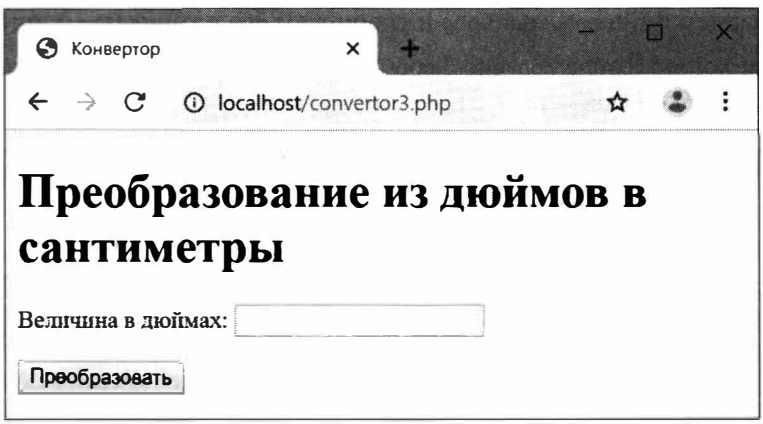


Рис. 1.6. Веб-приложение convertor3.php после первого обращения к нему

с ключом inches. Следовательно, код, выполняющий преобразование из дюймов в сантиметры, не исполнится, и на странице мы увидим лишь форму для ввода данных (как и показано на рис. 1.6).

5. Занесем в поле ввода **Величина в дюймах** число 85 и нажмем кнопку **Преобразовать**. По нажатию кнопки веб-форма отправит результат тому же приложению convertor3.php, что вызовет повторное обращение к нему — уже с отправкой данных методом POST. В ассоциативном массиве $_POST появится элемент inches, код, выполняющий преобразование и выводящий результат, успешно выполнится, и под формой появится абзац с результатом (рис. 1.7).

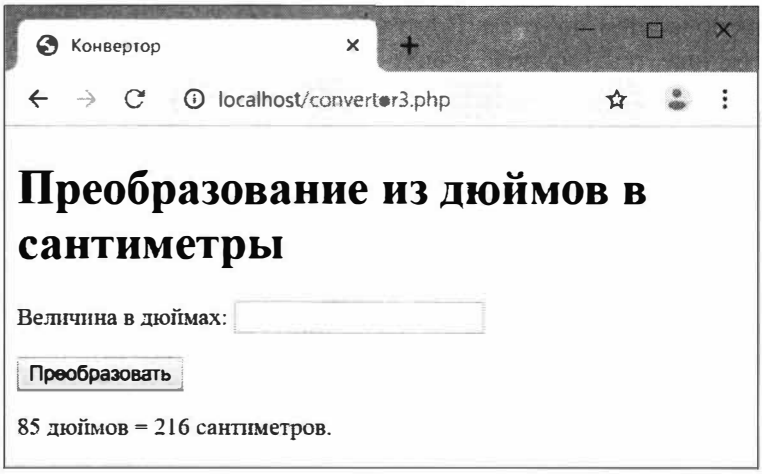


Рис. 1.7. Веб-приложение convertor3.php вывело результат

Полезно знать...

Можно отправлять данные и методом GET. Для этого в атрибуте method тега <form> нужно указать значение "get", а за отправленными формой данными обращаться к ассоциативному массиву из переменной $_GET.

Сделайте сами

Перепишите приложение convertor3.php так, чтобы форма пересылала данные методом GET.

1.1.6. Программные модули и включения

Простейшие веб-приложения, написанные на PHP, хранятся в одном файле. Но код более сложных приложений удобнее разнести по разным файлам — модулям.

> *Программный модуль* — файл, в котором хранится часть кода приложения.

В разных модулях могут храниться фрагменты кода, разрабатываемые разными программистами. Кроме того, в отдельный модуль может быть вынесен код, создающий переменные, константы, функции и классы (о них мы поговорим на

следующих уроках), которые используются в других модулях или даже в других сайтах.

> *Библиотека* — модуль, хранящий код, который используется при разработке разных сайтов, в том числе другими программистами.

> *Включение* — присоединение одного модуля к другому, после чего все переменные, константы, функции и классы, созданные во включаемом модуле, становятся доступными во включающем.

Включение выполняется языковой конструкцией `require`, которая записывается во включающем модуле и после которой, через пробел, ставится строка с путем к файлу с включаемым модулем. В путях в качестве разделителей можно использовать как обратные, так и прямые слеши.

```
// Включаем модуль module.php из папки modules
require 'modules\module.php';
```

Если записать включение одного и того же модуля несколько раз:

```
// Модуль modules\module.php будет включен трижды
require 'modules\module.php';
require 'modules/module.php';
require 'modules/module.php';
// В путях можно использовать как обратные, так и прямые слеши
// (автор предпочитает обратные)
```

включение столько же раз и будет выполнено. Часто это неприемлемо (например, если во включаемом модуле какой-либо переменной присваивается начальное значение, которое далее изменяется во включающем модуле, то повторное включение модуля вновь присвоит переменной начальное значение, что нарушит работу программы). Тогда удобнее применить аналогичную языковую конструкцию `require_once`, которая включает каждый модуль один раз и игнорирует последующие включения:

```
// Включаем модуль modules\module.php
require_once 'modules\module.php';
// Последующие включения того же модуля игнорируются
require_once 'modules\module.php';
require_once 'modules\module.php';
```

> Если модуль содержит только PHP-код, закрывающий тег `?>` в его конце ставить необязательно.

Упражнение 1.4

Напишем веб-приложение фотогалереи. Фрагмент кода, задающий массив с интернет-адресами графических файлов для вывода, будет вынесен в модуль data.php, а код, собственно выводящий страницу с картинками, — в модуль index.php.

1. В папке 1\!sources сопровождающего книгу электронного архива (см. *приложение 3*) найдем папку images, хранящую картинки, файл таблицы стилей styles.css,

файлы data.php (с данными об изображениях) и index.php (с заготовкой для написания приложения). Скопируем все это в корневую папку веб-сервера.

2. В корневой папке создадим папку modules, в которой будут храниться файлы модулей, и переместим в нее из корневой папки файл data.php.

3. Откроем файл data.php в текстовом редакторе и посмотрим на хранящийся в нем код (листинг 1.4).

Листинг 1.4. PHP-код модуля modules\data.php

```
<?php
$arr_images = ['200804282020.jpg', '200807192032.jpg',
    '200901092053.jpg', '201410131614.jpg', '201506152037.jpg',
    '201709180821.jpg', '201709252026.jpg', '201710242002.jpg',
    '201802131844.jpg'];
```

Этот код создает массив `$arr_images`, что содержит изображения, входящие в состав фотогалереи. Каждое изображение представляется строкой с именем хранящего его файла.

Поскольку файл хранит только PHP-код, закрывающий тег `?>` не поставлен.

4. Откроем копию файла index.php в текстовом редакторе, найдем парный тег `<section>` с якорем `gallery` и вставим в него PHP-код, выводящий изображения (выделен полужирным шрифтом):

```
<section id="gallery">
    <?php
        require_once 'modules\data.php';
        $cnt = count($arr_images);
        for ($i = 0; $i < $cnt; $i++) {
    ?>
    <img src="/images/<?php echo $arr_images[$i] ?>">
    <?php } ?>
</section>
```

Сначала выполняем включение модуля `modules\data.php`, чтобы получить доступ к массиву `$arr_images`, содержащего изображения из фотогалереи. Далее перебираем элементы этого массива и для каждого формируем тег `<img>`, выводящий соответствующее изображение.

Чтобы проверить готовое приложение в работе, выполним переход по интернет-адресу: **http://localhost/**. Что ж, получилось не так уж и плохо (рис. 1.8).

Мы только что написали полноценный (хотя и простенький) сайт, единственная страница которого генерируется серверным веб-приложением. Полноценные сайты включают в свой состав десятки таких приложений.

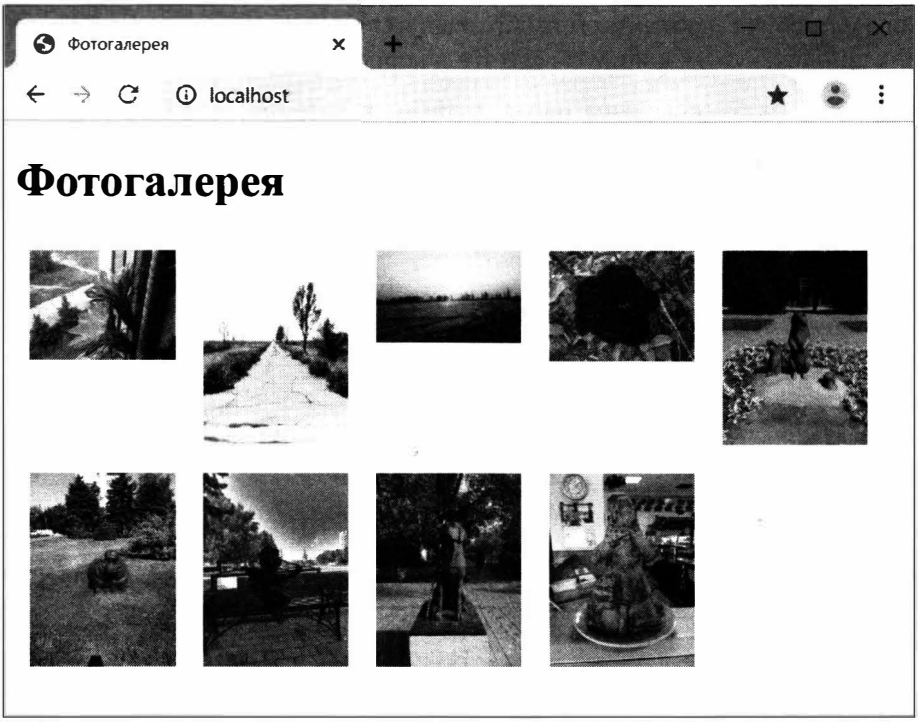


Рис. 1.8. Веб-приложение фотогалереи

1.2. Ошибки в PHP-коде

> Встретив в программном коде ошибку, исполняющая среда PHP вставляет подробное сообщение о ней прямо в HTML-код генерируемой страницы.

Чаще всего такое сообщение выводится на экран, но в ряде случаев (например, если оно записалось в интернет-адресе гиперссылки или тега `<img>`), чтобы его увидеть, придется просмотреть HTML-код страницы. В Google Chrome для этого достаточно щелкнуть на странице правой кнопкой мыши и выбрать в появившемся контекстном меню пункт **Просмотр кода страницы**.

Вот пример сообщения об ошибке:

```
// В этом выражении допущена ошибка — обращение к несуществующей
// функции cont (вместо count)
$cnt = cont($arr_images);

// На сгенерированной странице будет выведено сообщение об ошибке:

// Fatal error:
// Uncaught Error: Call to undefined function cont() in
// C:\xampp\htdocs\index.php:13 Stack trace: #0 {main} thrown in
// C:\xampp\htdocs\index.php on line 13
```

Оно достаточно исчерпывающее: включает полный путь PHP-файла (`C:\xampp\htdocs\index.php`) и номер строки кода (13), в которых встретилась ошибка (выделены рамками). А строка `Fatal error` в самом начале (выделена двойной рамкой) сообщает, что возникла фатальная ошибка.

> *Фатальная ошибка* — серьезная ошибка, вызывающая аварийное прерывание работы PHP-программы. Обозначается строкой **Fatal error** в самом начале сообщения.

> *Нефатальная ошибка* — не приводит к аварийному останову программы. Исполняющая среда пытается исправить ее по своему усмотрению, но приложение после этого практически всегда работает не так, как нужно. Обозначается строкой **Warning** в начале сообщения.

Пример сообщения о нефатальной ошибке (двойной рамкой в коде выделено слово `Warning`, а одинарными — путь PHP-файла и номер строки кода с ошибкой):

```
// Допущена ошибка – в начале имени переменной i не поставлен знак $
echo $arr_images[i];

// Будет выведено сообщение об ошибке:

// Warning:
// Use of undefined constant i - assumed 'i' (this will throw an Error
// in a future version of PHP) in
// C:\xampp\htdocs\index.php on line 16
```

1.3. Комментарии

> *Комментарий* — фрагмент программного кода, не обрабатываемый исполняющей средой. Служит для размещения в программном коде всевозможных заметок, пояснений, напоминаний и др.

PHP поддерживает два вида комментариев:

- однострочный — начинается с последовательности символов `//` (два слеша):

```
// Оператор умножения обозначается звездочкой (*)
$val = 25 * 60;
```

- многострочный — начинается с последовательности символов `/*` и заканчивается последовательностью `*/`:

```
/*
    Вызывая функцию round, вторым параметром можно указать номер знака
    после запятой, до которого нужно выполнить округление.
    Если второй параметр не указан, выполняется округление до целых.
*/
$result2 = round($result2, 2);
```

1.4. Способы набора PHP-кода

Каждое выражение PHP обычно записывается на отдельной строке:

```
$ins = 20;
$cents = $ins * 2.54;
$cents = round($cents);
```

Короткие выражения можно записать в одну строку:

```
$ins = 20; $cents = $ins * 2.54; $cents = round($cents);
```

Между отдельными значениями, именами переменных, операторами, функциями и прочими языковыми конструкциями обычно ставятся необязательные пробелы — так код лучше читается:

```
$cents = round(20 * 2.54);
```

Но их можно и не ставить — ради компактности:

```
$cents=round(20*2.54);
```

Так зачастую удается значительно уменьшить объем кода.

Вложенность одних выражений в другие, более сложные (такие, как условные выражения и циклы), отмечается отступами слева (обычно в 4 пробела):

```
if (isset($_POST['inches'])) {
    $ins = $_POST['inches'];
    $cents = round($ins * 2.54);
    echo '<p>', $ins, ' дюймов = ', $cents, ' сантиметров.</p>';
}

for ($i = 0; $i < $arr_cnt; $i++)
    echo '<p>', $arr[$i], '</p>';
```

Слишком длинные выражения можно переносить по строкам, разрывая строки в промежутках между языковыми конструкциями. Строки, на которых находятся оставшиеся части выражения, часто выделяются увеличенными отступами слева:

```
echo '<p class="convertion">100 дюймов = ', rount(100 * 2,54),
    ' сантиметров.</p>';
```

Полезно знать...

- PHP относится к так называемым *C-подобным* языкам программирования, чей синтаксис схож с синтаксисом языка C.
- Другие C-подобные языки программирования: C++, C#, Java, JavaScript, TypeScript.

1.5. Консоль PHP

> *Консоль PHP* — интерактивная исполняющая среда, поставляемая в составе PHP. В консоли можно набирать любые выражения и просматривать результаты их выполнения.

Для вывода консоли PHP следует запустить файл **php.exe**, хранящийся в папке *<путь установки XAMPP>*\php. Откроется окно консоли, похожее на окно командной строки Windows (рис. 1.9).

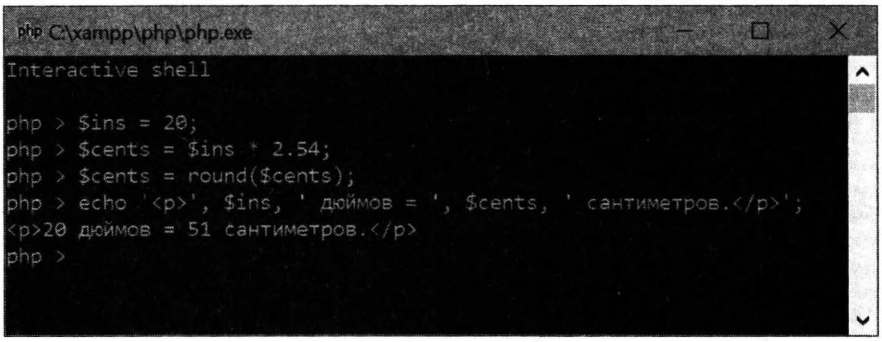


Рис. 1.9. Окно консоли PHP

Вот пример выражений, которые можно набрать в консоли (префиксом `php >` обозначен набираемый код, выведенный результат показан без этого префикса):

```
php > $ins = 20;
php > $cents = $ins * 2.54;
php > $cents = round($cents);
php > echo '<p>', $ins, ' дюймов = ', $cents, ' сантиметров.</p>';
<p>20 дюймов = 51 сантиметров.</p>
```

Чтобы закрыть консоль, нужно либо набрать в ней команду `exit` и нажать клавишу <Enter>, либо щелкнуть кнопку закрытия окна.

Урок 2
Работа с данными разных типов

PHP предусматривает разнообразные средства для обработки данных: арифметических действий над числами, поиска подстрок в строках, сравнения двух значений с целью получить результат в виде логической величины и др.

2.1. Числа: целые и вещественные

В PHP целые и вещественные числа относятся к разным типам данных. Однако набор операций, выполняемых над ними, одинаков.

2.1.1. Запись чисел

- Целые числа записываются как есть: `20`, `1000`, `58945`.
- В вещественных числах для разделения целой и дробной частей применяется точка (не запятая!): `5.34`, `1234.567`, `0.891`.
- Перед отрицательными числами ставится знак - (дефис): `-10`, `-1234.765`, `-0.534`.
- Экспоненциальная форма числа <*мантисса*> × $10^{<порядок>}$ записывается в формате: <*мантисса*>e<*порядок*>, буква «e» может быть набрана в любом регистре. Примеры: `1e3` (1000), `2.471E8` (247100000), `1.3e-4` (0,00013).
- Целые (но не вещественные) числа можно записывать в других системах счисления:
  - шестнадцатеричной — предварив число префиксом `0x` или `0X`: `0xf` (15), `0X4a5` (1189), `0xe5bc9` (941001);
  - восьмеричной — предварив число нулем (`0`): `010` (8), `07456` (3886), `-0404` (-260);
  - двоичной — предварив число префиксом `0b` или `0B`: `0b100` (4), `0b010110` (22), `0B11010110` (214).

2.1.2. Действия над числами

Арифметические действия

- Сложение — оператор + (плюс): <*слагаемое 1*> + <*слагаемое 2*>:

```
echo 45 + 15;                                    // 60
```

- вычитание — оператор - (минус): <уменьшаемое> - <вычитаемое>;

```
echo 15 - 45;                                    // -30
```

- умножение — оператор * (звездочка): <множитель 1> * <множитель 2>:

```
echo 15 * 40;                                    // 600
```

- деление — оператор / (прямой слеш): <делимое> / <делитель>:

```
echo 400 / 8;                                    // 50
echo 25 / 2;                                     // 12.5
```

- деление нацело — функция intdiv(<делимое>, <делитель>):

```
echo intdiv(25, 2);                              // 12
```

- остаток от деления — оператор % (процент): <делимое> % <делитель>:

```
echo 25 % 2;                                     // 1
echo 25 % 7;                                     // 4
```

- дробный остаток от деления — функция fmod(<делимое>, <делитель>):

```
echo fmod(25, 7);                                // 4
echo fmod(6.2, 1.5);                             // 0.2
```

- возведение в степень — оператор ** (две звездочки): <основание> ** <показатель>:

```
echo 8 ** 3;                                     // 512
echo 1.234 ** 1.7;                               // 1.4296698926485
```

  Для этой же цели можно использовать функцию pow(<основание>, <показатель>), оставшуюся от более старых версий PHP:

```
echo pow(8, 3);                                  // 512
```

- изменение знака числа — оператор - (минус) в формате: -<значение>:

```
$a = 10; echo -$a;                               // -10
```

- инкремент значения *переменной* — оператор ++ (два плюса). Может быть записан в двух форматах:
  - ++<переменная> — сначала инкрементирует значение, хранящееся в *переменной*, потом возвращает инкрементированное значение в качестве результата:

```
$i = 0; echo ++$i, ', ', $i;                     // 1, 1
```

  - <переменная>++ — сначала возвращает изначальное значение переменной, а потом инкрементирует его:

```
$i = 0; echo $i++, ', ', $i;                     // 0, 1
```

- декремент значения *переменной* — оператор -- (два минуса).

> *Декремент* — уменьшение значения переменной на единицу. Выполняется быстрее обычного вычитания.

Этот оператор может быть записан в двух форматах:

- `--<переменная>` — сначала декрементирует значение, хранящееся в *переменной*, потом возвращает его в качестве результата:

```
$i = 5; echo --$i, ', ', $i;                    // 4, 4
```

- `<переменная>--` — сначала возвращает изначальное значение переменной, а потом декрементирует его:

```
$i = 5; echo $i--, ', ', $i;                    // 5, 4
```

♦ абсолютное значение (модуль) *числа* — функция `abs(<число>)`:

```
echo abs(100);                                  // 100
echo abs(-100);                                 // 100
```

Алгебраические действия

♦ Квадратный корень от *числа* — функция `sqrt(<число>)`:

```
echo sqrt(2);                                   // 1.4142135623731
echo sqrt(25);                                  // 5
```

♦ $e^{<число>}$ — функция `exp(<число>)`:

```
echo exp(1);                                    // 2.718281828459
echo exp(3.24);                                 // 25.533721747352
```

♦ log *<число>* — функция `log10(<число>)`:

```
echo log10(10);                                 // 1
echo log10(20);                                 // 1.301029995664
```

♦ ln *<число>* — функция `log(<число>)`:

```
echo log(2);                                    // 0.69314718055995
echo log(8.5);                                  // 2.1400661634963
```

♦ округление *числа* до указанного *количества знаков после запятой* согласно заданному *режиму* — функция `round`. Формат вызова:

```
round(<число>[, <количество знаков после запятой>[, <режим>]])
```

Если *количество знаков после запятой* не указано, выполняется округление до целых. Примеры:

```
echo round(1.5738, 2);                          // 1.57
echo round(1.5738);                             // 2
```

Режим может быть задан в виде одной из следующих констант:

> *Константа* — переменная, чье значение после создания изменить нельзя.

- `PHP_ROUND_HALF_UP` — округляет *число* до заданного *количества знаков* в бóльшую сторону, если следующий знак находится посередине (поведение функции по умолчанию, если *режим* не указан):

```
echo round(1.5);                                    // 2
echo round(2.5);                                    // 3
```

- PHP_ROUND_HALF_DOWN — округляет *число* до заданного *количества знаков* в меньшую сторону, если следующий знак находится посередине:

```
echo round(1.5, 0, PHP_ROUND_HALF_DOWN);      // 1
echo round(2.5, 0, PHP_ROUND_HALF_DOWN);      // 2
```

- PHP_ROUND_HALF_EVEN — округляет *число* до заданного *количества знаков* в сторону ближайшего четного знака:

```
echo round(1.5, 0, PHP_ROUND_HALF_EVEN);      // 2
echo round(2.5, 0, PHP_ROUND_HALF_EVEN);      // 2
```

- PHP_ROUND_HALF_ODD — округляет *число* до заданного *количества знаков* в сторону ближайшего нечетного знака:

```
echo round(1.5, 0, PHP_ROUND_HALF_ODD);       // 1
echo round(2.5, 0, PHP_ROUND_HALF_ODD);       // 3
```

♦ округление *числа* до целых в меньшую сторону — функция floor(<*число*>):

```
echo floor(1.3);                                    // 1
echo floor(1.8);                                    // 1
```

♦ округление *числа* до целых в бóльшую сторону — функция ceil(<*число*>):

```
echo ceil(1.3);                                     // 2
echo ceil(1.8);                                     // 2
```

Тригонометрические функции

♦ Синус *угла* — функция sin(<*угол*>). Угол указывается в радианах:

```
echo sin(1.05);                                     // 0.86742322559402
```

♦ косинус *угла* — функция cos(<*угол*>). Угол указывается в радианах:

```
echo cos(1.05);                                     // 0.49757104789173
```

♦ тангенс *угла* — функция tan(<*угол*>). Угол указывается в радианах:

```
echo tan(1.05);                                     // 1.7433153099832
```

♦ арксинус от числа — функция asin(<*число*>). Результат возвращается в радианах:

```
echo asin(0.86742322559402);                        // 1.05
```

♦ арккосинус от числа — функция acos(<*число*>). Результат возвращается в радианах:

```
echo acos(0.49757104789173);                        // 1.05
```

♦ арктангенс от числа — функция atan(<*число*>). Результат возвращается в радианах:

```
echo atan(1.7433153099832);                         // 1.05
```

- перевод *угла* из градусов в радианы — функция `deg2rad(<угол в градусах>)`:

```
echo deg2rad(60);                                    // 1.0471975511966
```

- перевод *угла* из радианов в градусы — функция `rad2deg(<угол в радианах>)`:

```
echo rad2deg(1.0471975511966);                       // 60
```

Побитовые операции

Побитовые операции выполняются над отдельными битами целых чисел.

- Побитовое И (устанавливаются только те биты, что установлены в обоих *операндах*) — оператор `&` (амперсанд): `<операнд 1> & <операнд 2>`:

```
echo 0b1010 & 0b1001;                                // 0b1000
```

- побитовое ИЛИ (устанавливаются только те биты, что установлены, по крайней мере, в одном из *операндов*) — оператор `|` (вертикальная черта): `<операнд 1> | <операнд 2>`:

```
echo 0b1010 | 0b1001;                                // 0b1011
```

- побитовое Исключающее ИЛИ (устанавливаются только те биты, что установлены лишь в одном из *операндов*) — оператор `^` («крышка»): `<операнд 1> ^ <операнд 2>`:

```
echo 0b1010 ^ 0b1001;                                // 0b0011
```

- побитовое НЕ (устанавливаются только те биты, что не установлены в *операнде*) — оператор `~` (тильда): `~<операнд>`:

```
echo ~0b1010;
   // 11111111111111111111111111111111111111111111111111111110101
```

- сдвиг *числа* влево на заданное *количество разрядов* — оператор `<<` (два знака «меньше»): `<число> << <количество разрядов>`:

```
echo 0b1000 << 3;                                    // 0b1000000
```

- сдвиг *числа* вправо на заданное *количество разрядов* — оператор `>>` (два знака «больше»): `<число> >> <количество разрядов>`:

```
echo 0b1000 >> 2;                                    // 0b10
```

Побитовые операторы находят ограниченное применение, и, тем не менее, иногда без них не обойтись. Один из случаев их использования мы рассмотрим на *уроке 17*.

Прочие операции

- Значение числа π — константа `M_PI`:

```
echo M_PI;                                           // 3.1415926535898
```

Также можно вызвать не принимающую параметров функцию `pi`:

```
echo pi();                                           // 3.1415926535898
```

- значение числа Эйлера e — константа `M_E`:

```
echo M_E;                                    // 2.718281828459
```

- минимальное из указанных чисел — функция `min`. Может быть вызвана в двух форматах:
  - `min(<число 1>, <число 2>, <число 3> . . . <число n>)`:

```
echo min(9.89, 10, 4.8, -100);               // -100
```

  - `min(<массив с числами>)`:

```
echo min([-190, 3, 82.46, 11]);              // -190
```

- максимальное из указанных чисел — функция `max`. Может быть вызвана в двух форматах:
  - `max(<число 1>, <число 2>, <число 3> . . . <число n>)`:

```
echo max(9.89, 10, 4.8, -100);               // 10
```

  - `max(<массив с числами>)`:

```
echo max([-190, 3, 82.46, 11]);              // 82.46
```

- длина гипотенузы прямоугольного треугольника — функция `hypot(<длина катета 1>, <длина катета 2>)`:

```
echo hypot(2.3, 5.8);                        // 6.2393909959226
```

- получение целого псевдослучайного числа — функция `rand`. При каждом вызове возвращает очередное число. Может быть вызвана в двух форматах:
  - `rand()` (без параметров) — возвращает псевдослучайное число в диапазоне от 0 до максимального значения, зависящего от платформы:

```
echo rand();                                 // 1135038663
echo rand();                                 // 723031401
```

  - `rand(<минимальное значение>, <максимальное значение>)` — возвращает псевдослучайное число в диапазоне между указанными *минимальным* и *максимальным значениями*:

```
echo rand(1, 10);                            // 3
echo rand(1, 10);                            // 6
```

- максимальное псевдослучайное число, возвращаемое вызванной без параметров функцией `rand`, — функция `getrandmax`, не принимающая параметров:

```
echo getrandmax();                           // 2147483647
```

2.2. Строки

> *Строка* — это последовательность произвольных символов. Строки могут содержать, например, фрагменты слов, целые слова, предложения и фрагменты HTML-кода.

2.2.1. Запись строк

PHP поддерживает четыре способа записать строку:

- заключение в одинарные кавычки: `'PHP'`, `'Привет, мир!'`, `'<h2>2.2. Строки</h2>'`, `''` (*пустая строка*, не содержащая ни единого символа).

  Чтобы вставить в такую строку символ одинарной кавычки, нужно использовать литерал `\'`:

```
echo 'Шеймус O\'Брайен';                    // Шеймус O'Брайен
```

> *Литерал* — символ или последовательность символов, имеющая особое значение (например, обозначающее символ, который нельзя вставить в строку напрямую).

- заключение в двойные кавычки: `"HTML и CSS"`, `"20 дюймов"`, `"<p>В галерее пока нет изображений</p>"`, `""` (пустая строка).

  В строках, взятых в двойные кавычки, можно использовать больший набор литералов (табл. 2.1).

***Таблица 2.1.** Литералы, которые можно использовать в строках, взятых в двойные кавычки, и heredoc-строках (см. далее)*

Литерал	Описание	Литерал	Описание
`\r`	Возврат каретки	`\n`	Перевод строки
`\\`	Обратный слеш	`\$`	Знак доллара
`\"`	Двойная кавычка	`\u{<код>}`	Символ с заданным Unicode-*кодом*[1]

Примеры:

```
echo "Первая строка\r\nВторая строка";      // Первая строка
                                            // Вторая строка
echo "^ - знак \"крышка\"";                 // ^ - знак "крышка"
echo "Feldschl\u{f6}sschen";                // Feldschlösschen
```

Помимо этого, в таких строках выполняется обработка переменных (о которой мы узнаем далее);

- применив синтаксис *heredoc*, записываемый в формате:

```
<<<<литерал-ограничитель>
<содержимое строки>
<литерал-ограничитель>;
```

> *Литерал-ограничитель* — последовательность символов, помечающая начало и конец строки, записанной синтаксисом heredoc или nowdoc

[1] Указывается в шестнадцатеричной системе счисления.

(см. далее). Должна состоять из букв, цифр, знаков подчеркивания и не должна начинаться с цифры. Обычно в качестве литерала-ограничителя применяется последовательность символов EOD.

В heredoc-строках все разрывы строк сохраняются. Пример:

```
$str = <<<EOD
Первая строка
Вторая строка
EOD;
echo $str;                                    // Первая строка
                                              // Вторая строка
```

Кроме того, в heredoc-строках поддерживаются литералы из табл. 2.1 и выполняется обработка переменных;

- применив синтаксис *nowdoc*, записываемый в формате:

```
<<<'<литерал-ограничитель>'
<содержимое строки>
<литерал-ограничитель>;
```

В отличие от heredoc-строки, первый *литерал-ограничитель* здесь берется в одинарные кавычки.

В nowdoc-строках также сохраняются все разрывы строк. Пример:

```
$str = <<<'EOD'
Первая строка
Вторая строка
EOD;
echo $str;                                    // Первая строка
                                              // Вторая строка
```

Однако не поддерживаются ни литералы из табл. 2.1, ни литерал \'. Обработка переменных также не выполняется.

Heredoc- и nowdoc-строки удобно использовать для формирования больших фрагментов текста или HTML-кода, в которых желательно сохранить все разрывы строк.

2.2.2. Обработка переменных в строках

Если в строке записать имя переменной, заключенное в фигурные скобки, вместо него будет подставлено значение этой переменной.

Это условие выполняется только в строках, взятых в двойные кавычки, и heredoc-строках. Пример:

```
$ins = 20;
$cents = $ins * 2.54;
echo "{$ins} дюймов = {$cents} см.";          // 20 дюймов = 50.8 см.
```

Можно обойтись и без фигурных скобок

Если имя переменной ограничено символами, которые нельзя использовать в именах переменных, — например, пробелами или знаками препинания (подробно об именовании переменных будет рассказано на *уроке 3*), фигурные скобки вокруг него можно не ставить. В следующем примере мы обошлись без фигурных скобок, так как имена обеих переменных ограничены пробелами:

```
echo "$ins дюймов = $cents см.";          // 20 дюймов = 50.8 см.
```

2.2.3. Действия над строками

Первый символ строки имеет номер 0.

Если номер символа *положительный*, он отсчитывается от начала строки — так, второй символ имеет номер 1, третий — 2 и т. д.

Если номер символа *отрицательный*, он отсчитывается от конца строки — например, первый с конца строки символ имеет номер –1, второй — –2 и т. д.

Конкатенация и вычисление длин строк

- Конкатенация (объединение) строк — оператор . (точка): *<строка 1>* . *<строка 2>*:

```
echo 'PHP' . '7';                          // PHP7
echo 'Привет' . ', ' . 'мир!';             // Привет, мир!
```

- длина *строки* — функция `mb_strlen(<строка>)`:

```
echo mb_strlen('Тригонометрия');           // 13
```

Поиск и извлечение подстрок в строках

- Поиск заданной *подстроки* в *строке* с учетом регистра символов с начала строки — функция `mb_strpos`:

```
mb_strpos(<строка>, <подстрока>[,
    <номер символа, с которого начнется поиск>])
```

Если *номер символа, с которого начнется поиск*, не указан, поиск выполняется с первого символа (номер 0).

Метод возвращает номер символа в *строке*, в котором находится первое вхождение найденной *подстроки*. Если *подстрока* не была найдена, возвращается `FALSE`.

Примеры:

```
echo mb_strpos('Тригонометрия', 'и');      // 2
echo mb_strpos('Тригонометрия', 'и', 3);   // 11
echo mb_strpos('Тригонометрия', 'Ф');      // FALSE
echo mb_strpos('Тригонометрия', 'Т');      // 0
echo mb_strpos('Тригонометрия', 'Т', 3);   // FALSE
```

- поиск заданной *подстроки* в *строке* с учетом регистра символов с конца строки — функция `mb_strrpos`, имеющая тот же формат вызова и возвращающая такой же результат:

```
echo mb_strrpos('Тригонометрия', 'и');          // 11
echo mb_strrpos('Тригонометрия', 'и', -4);      // 2
echo mb_strrpos('Тригонометрия', 'Т');          // 0
```

- поиск заданной *подстроки* в *строке* без учета регистра символов с начала строки — функция `mb_stripos`, имеющая тот же формат вызова и возвращающая такой же результат:

```
echo mb_stripos('Тригонометрия', 'Т');          // 0
echo mb_stripos('Тригонометрия', 'Т', 3);       // 9
```

- поиск заданной *подстроки* в *строке* без учета регистра символов с конца строки — функция `mb_strripos`, имеющая тот же формат вызова и возвращающая такой же результат:

```
echo mb_strripos('Тригонометрия', 'Т');         // 9
echo mb_strripos('Тригонометрия', 'Т', -5);     // 0
```

- подсчет количества вхождений *подстроки* в *строку* — функция `mb_substr_count(<строка>, <подстрока>)`:

```
echo mb_substr_count('Тригонометрия', 'и');     // 2
```

- извлечение *подстроки* из *строки* — функция `mb_substr`:

```
mb_substr(<строка>, <номер начального символа подстроки>[,
    <количество символов в подстроке>])
```

Если *количество символов в подстроке* не указано, возвращается подстрока, содержащая все символы до конца *строки*. Примеры:

```
echo mb_substr('Тригонометрия', 0, 3);          // Три
echo mb_substr('Тригонометрия', 7, 4);          // метр
echo mb_substr('Тригонометрия', 7);             // метрия
```

Преобразование строк

- Преобразование символов *строки* к нижнему регистру — функция `mb_strtolower(<строка>)`:

```
echo mb_strtolower('ТриГоНОмеТРия');            // тригонометрия
```

- преобразование символов *строки* к верхнему регистру — функция `mb_strtoupper(<строка>)`:

```
echo mb_strtoupper('ТриГоНОмеТРия');            // ТРИГОНОМЕТРИЯ
```

- удаление начальных и конечных пробелов из *строки* — функция `trim(<строка>)`:

```
echo trim('   Тригонометрия   ');               // Тригонометрия
```

- объединение элементов *массива* в строку — функция `implode`. Поддерживаются два формата вызова:

  - `implode(<символ-разделитель>, <массив>)`

    Здесь *символ-разделитель* отделяет в строке один элемент массива от другого.

- `implode(<массив>)`

  Здесь элементы массива будут располагаться вплотную друг к другу.

Примеры:

```
$arr = [1, 2, '345', '!!!'];
echo implode(' ', $arr);                    // 1 2 345 !!!
echo implode(', ', $arr);                   // 1, 2, 345, !!!
echo implode($arr);                         // 12345!!!
```

♦ разделение *строки* на подстроки по указанному *символу-разделителю* и формирование из них массива — функция `explode`:

```
explode(<символ-разделитель>, <строка>[,
        <максимальное количество элементов в возвращаемом массиве>])
```

Если *максимальное количество элементов* не указано, массив включит все полученные в результате разбиения подстроки:

```
$arr1 = explode(' ', '1 2 345 !!!');
// В переменной $arr1 окажется массив с элементами '1', '2', '345' и
// '!!!'
```

Если *максимальное количество элементов* указано, последний элемент массива будет хранить оставшуюся часть строки:

```
$arr2 = explode(' ', '1 2 345 !!!', 3);
// В переменной $arr2 окажется массив с элементами '1', '2' и
// '345 !!!'
```

Обработка строк на русском языке

Все приведенные здесь функции успешно обрабатывают строки, записанные в кодировке Unicode, в том числе включающие символы кириллицы.

«Быстрая» обработка строк в кодировке ASCII

> Для обработки строк, содержащих лишь символы из кодировки ASCII (буквы латиницы, цифры, основные знаки препинания), рекомендуется применять операторы и функции, приведенные в табл. 2.2, поскольку они выполняются быстрее описанных ранее.

♦ «Быстрые» функции, аналогичные рассмотренным ранее (табл. 2.2).

***Таблица 2.2.** Функции с поддержкой Unicode и их «быстрые» аналоги*

Функция с поддержкой Unicode	«Быстрый» аналог	Функция с поддержкой Unicode	«Быстрый» аналог
mb_strlen	strlen	mb_strpos	strpos
mb_strrpos	strrpos	mb_stripos	stripos
mb_strripos	strripos	mb_substr_count	substr_count
mb_substr	substr	mb_strtolower	strtolower
mb_strtoupper	strtoupper		

Примеры:

```
echo strlen('PHP programming');                    // 15
echo strpos('PHP programming', 'gramm');           // 7
echo substr('PHP programming', 4, 6);              // progra
```

- оператор получения символа с указанным номером — `[]` (квадратные скобки), записанный в формате `<строка>[<номер символа>]`:

```
echo 'PHP programming'[2];                         // P
```

Отрицательные номера отсчитываются от конца строки: номер –1 имеет последний символ, номер –2 — предпоследний и т. д.:

```
echo 'PHP programming'[-1];                         // g
echo 'PHP programming'[-3];                         // i
```

- функцию замены в *строке подстроки 1* на *подстроку 2* — `str_replace(<подстрока 1>, <подстрока 2>, <строка>)`:

```
echo str_replace('PHP', 'CSS', 'HTML & PHP');      // HTML & CSS
```

> ***Внимание!***
>
> Не используйте эти операторы и функции для обработки строк в Unicode — вы рискуете получить странный результат.

Полезно знать...

PHP поддерживает мощное средство для выполнения поиска подстрок в строке, замены их другими подстроками и разбиения строки на фрагменты — *регулярные выражения*. Они будут рассмотрены на *уроке 11*.

2.3. Логические величины и написание условий

Логическая величина может принимать только два значения: `TRUE` («истина») и `FALSE` («ложь»). Такие величины возвращаются в качестве результата условиями и используются в условных выражениях и циклах.

> *Условие* — выражение, сравнивающее два значения и возвращающее результат в виде логической величины. При написании условий применяются операторы сравнения и логические операторы.

2.3.1. Операторы сравнения

Оператор сравнения (табл. 2.3) сравнивает значения двух операндов и возвращает в качестве результата логическую величину: `TRUE`, если сравнение выполняется, или `FALSE` — в противном случае.

Таблица 2.3. *Операторы сравнения PHP*

Оператор	Описание	Примеры
<оп.1> == <оп.2>	*Оп.1* равен *оп.2*?	`echo 2 == 2;` // TRUE `echo 2 == 3;` // FALSE `echo 'A' == 'A';` // TRUE `echo 'A' == 'B';` // FALSE
<оп.1> != <оп.2> или <оп.1> <> <оп.2>	*Оп.1* не равен *оп.2*?	`echo 2 != 2;` // FALSE `echo 2 != 3;` // TRUE `echo 'A' != 'A';` // FALSE `echo 'A' != 'B';` // TRUE
<оп.1> < <оп.2>	*Оп.1* меньше *оп.2*?	`echo 2 < 232;` // TRUE `echo 232 < 232;` // FALSE
<оп.1> <= <оп.2>	*Оп.1* меньше или равен *оп.2*?	`echo 2 <= 232;` // TRUE `echo 232 <= 232;` // TRUE `echo 233 <= 232;` // FALSE
<оп.1> > <оп.2>	*Оп.1* больше *оп.2*?	`echo 2 > 232;` // FALSE `echo 233 > 232;` // TRUE
<оп.1> >= <оп.2>	*Оп.1* больше или равен *оп.2*?	`echo 2 >= 232;` // FALSE `echo 232 >= 232;` // TRUE `echo 233 >= 232;` // TRUE
<оп.1> === <оп.2>	*Оп.1* строго равен *оп.2*?	`echo 2 === 2;` // TRUE `echo 2 === '2';` // FALSE
<оп.1> !== <оп.2>	*Оп.1* строго не равен *оп.2*?	`echo 2 === 2;` // FALSE `echo 2 === '2';` // TRUE

Примечание

Операторы строгого сравнения === и !== более подробно будут описаны чуть далее (см. *разд. 2.6.3*).

Оператор <оп.1> <=> <оп.2> («космический корабль») возвращает целочисленный результат:

- –1 — если *оп.1* меньше *оп.2*;
- 0 — если *оп.1* равен *оп.2*;
- 1 — если *оп.1* больше *оп.2*.

Пример:

```
echo 2 <=> 3;                                    // -1
```

2.3.2. Логические операторы

Логический оператор (табл. 2.4) принимает в качестве операндов логические величины и возвращает также логическую величину.

Таблица 2.4. *Логические операторы PHP*

Оператор	Описание	Примеры

<оп.1> **&&** *<оп.2>* или *<оп.1>* **and** *<оп.2>*	**Логическое И**: И *оп.1*, И *оп.2* должны быть TRUE	`echo 2 < 3 && 0 > -1000; // TRUE`
		`echo 2 >= 3 && 0 > -1000; // FALSE`

<оп.1> **||** *<оп.2>* или *<оп.1>* **or** *<оп.2>*	**Логическое ИЛИ**: ИЛИ *оп.1*, ИЛИ *оп.2*, ИЛИ оба должны быть TRUE	`echo 2 < 3 || 0 > -1000; // TRUE`
		`echo 2 < 3 || 0 == -1000; // TRUE`
		`echo 2 >= 3 || 0 == -1000; // FALSE`

<оп.1> **xor** *<оп.2>*	**Логическое Исключающее ИЛИ**: ИЛИ *оп.1*, ИЛИ *оп.2*, НО не оба вместе должны быть TRUE	`echo 2 < 3 xor 0 > -1000; // FALSE`
		`echo 2 < 3 xor 0 == -1000; // TRUE`
		`echo 2 >= 3 xor 0 == -1000; // FALSE`

!<операнд>	**Логическое НЕ**: *операнд* НЕ должен быть TRUE	`echo !(2 < 3); // FALSE`
		`echo !(2 > 3); // TRUE`

Примечание

У оператора **!** приоритет выше, чем у операторов сравнения, поэтому последние следует брать в круглые скобки.

Операторы **&&** и **and**, а также **||** и **or** имеют разный приоритет (см. *приложение 1*).

2.4. Временны́е отметки

> *Временна́я отметка* — комбинация даты и времени наступления какого-либо события.

Полезно знать...

Временна́я отметка в PHP хранится в виде целого числа, которое обозначает количество секунд, прошедших с полуночи 1 января 1970 года («начала эры UNIX»). На страницу она также будет выведена в виде целого числа (как вывести его в виде даты и времени, мы узнаем на *уроке 16*).

2.4.1. Создание временны́х отметок

- Временна́я отметка, обозначающая текущие дату и время, — функция `time`, не принимающая параметров:

```
echo time();                                    // 1564664230
```

- временна́я отметка, созданная на основе указанной *строки с датой и временем*, — функция `strtotime(<строка с датой и временем>)`. *Строка* может быть записана в формате:

```
<год>-<номер месяца>-<число>[ <часы>:<минуты>:<секунды>]
```

Пример:

```
echo strtotime('2019-08-01 17:02:48');          // 1564671768
```

Если *часы*, *минуты* и *секунды* отсутствуют, в качестве времени принимается полночь. Пример:

```
echo strtotime('2019-08-01');                   // 1564610400
```

Если функция `strtotime` не сможет раскодировать полученную *строку с датой и временем*, она вернет значение `FALSE`.

2.4.2. Получение компонентов даты и времени из временно́й отметки

Функция `getdate([<временна́я отметка>])` возвращает ассоциативный массив, чьи элементы содержат различные компоненты указанной *временно́й отметки*. Ключи этих элементов приведены далее:

- `'year'` — год из четырех цифр;
- `'mon'` — порядковый номер месяца от 1 до 12;
- `'mday'` — число;
- `'hours'` — количество часов;
- `'minutes'` — количество минут;
- `'seconds'` — количество секунд;
- `'wday'` — порядковый номер дня недели от 0 (воскресенье) до 6 (суббота).

Пример:

```
$date = time();
$arr_date = getdate($date);
echo 'Сегодня '. $arr_date['mday'], '.', $arr_date['mon'], '.',
    $arr_date['year'];                          // Сегодня 1.8.2019
```

Если вызвать функцию `getdate` без параметра, она вернет массив с компонентами временно́й отметки, хранящей текущие дату и время.

2.5. Значение *NULL*

> `NULL` — значение, обозначающее отсутствие какого-либо «значимого» значения и относящееся к отдельному типу данных.

Значение `NULL` можно получить, обратившись к несуществующей (еще не созданной или уже удаленной) переменной.

2.6. Типы данных

PHP поддерживает следующие типы данных:

- *целочисленный* — целые числа;
- *вещественный* — вещественные числа;
- *строковый* — строки;
- *логический* — логические величины;
- *NULL*;
- *массив* (любых типов);
- *объект* — будет описан на *уроках 7* и *8*;
- *ресурс* — ссылка на внешние по отношению к PHP сущности (например, указатель на открытый файл и т. п.).

2.6.1. Определение типа значения

Функция `gettype(<значение>)` возвращает название типа данных, к которому относится указанное *значение*, в виде строки:

- `'integer'` — целочисленный;
- `'double'` — вещественный;
- `'string'` — строковый;
- `'boolean'` — логический;
- `'NULL'` — NULL;
- `'array'` — массив;
- `'object'` — объект;
- `'resource'` — ресурс;
- `'unknown type'` — тип определить невозможно.

Примеры:

```
echo gettype(123);                          // integer
echo gettype(123.456);                      // double
echo gettype('PHP & MySQL');                // string
echo gettype(['PHP', 'MySQL']);             // array
```

Также поддерживается ряд функций, определяющих, относится ли переданное им значение к какому-либо типу, и возвращающих `TRUE`, если это так, и `FALSE` — в противном случае. Все эти функции приведены в табл. 2.5.

Таблица 2.5. *Функции для быстрого определения типа значения*

Функция	Описание	Примеры

`is_int(<п.>)`
`is_integer(<п.>)`
`is_long(<п.>)`
	П. — целое число?	`echo is_int(1);` `// TRUE`
		`echo is_int(1.23);` `// FALSE`

`is_float(<п.>)`
`is_double(<п.>)`
`is_real(<п.>)`
	П. — вещественное число?	`echo is_float(1.23);` `// TRUE`
		`echo is_float(1);` `// FALSE`

`is_string(<п.>)`	*П.* — строка?	`echo is_string('PHP');` `// TRUE`
		`echo is_string('1.23');` `// TRUE`
		`echo is_string(1.23);` `// FALSE`

`is_numeric(<п.>)`	*П.* — число или строка, содержащая число?	`echo is_numeric(1);` `// TRUE`
		`echo is_numeric(1.23);` `// TRUE`
		`echo is_numeric('1.23');` `// TRUE`
		`echo is_numeric('PHP');` `// FALSE`

`is_bool(<п.>)`	*П.* — логическая величина?	`echo is_bool(1 > 2);` `// TRUE` `echo is_bool(1 - 3);` `// FALSE`

`is_scalar(<п.>)`	*П.* — число, строка или логическая величина?	`echo is_scalar(1);` `// TRUE`
		`echo is_scalar(1.23);` `// TRUE`
		`echo is_scalar('PHP');` `// TRUE`
		`echo is_scalar([1, 2]);` `// FALSE`

`is_null(<п.>)`	*П.* — NULL?	`echo is_null(null);` `// TRUE` `echo is_null(0);` `// FALSE`
`is_array(<п.>)`	*П.* — массив?	`echo is_array([1, 2, 3]);` `// TRUE` `echo is_array(123);` `// FALSE`
`is_object(<п.>)`	*П.* — объект?	
`is_resource(<п.>)`	*П.* — ресурс?	

Примечание

Если в графе **Функция** указаны несколько функций, можно использовать любую из них.

2.6.2. Преобразование типов

Преобразование типов — перевод значения из одного типа в другой (например, из целочисленного в строковый).

Неявное преобразование типов

Неявное преобразование типов производится самой исполняющей средой PHP в случае использования в выражении значения неподходящего типа.

Преобразование в число

Если в выражении требуется целое или вещественное число, а подставлено значение другого типа, преобразование выполняется согласно следующим правилам:

- строки, содержащие числа или начинающиеся с числа, — преобразуются:
  - если содержат символ точки, букву «e» или «E» — в число с плавающей точкой;
  - в противном случае — в целое число;
- строки, не содержащие чисел в начале, — в 0 с выдачей сообщения о нефатальной ошибке;
- TRUE — в 1;
- FALSE и NULL — в 0.

Примеры:

```
echo 1 + '1';                                    // 2
echo 1 + '1.5';                                  // 2.5
echo 1 + '1.5e3';                                // 1501
echo 1 + '1.5abcd';                              // 2.5
echo 1 + 'abcd1.5';                              // 1
```

Вещественное число при преобразовании в целое округляется в сторону нуля.

Преобразование в строку

- Целые и вещественные числа — преобразуются как есть.
- Логическое значение TRUE — преобразуется в строку '1'.
- Значения FALSE и NULL — в пустую строку ''.

> ***Не выводите логические величины как есть***
>
> Если выполнить вывод логической величины, указав ее в конструкции echo: echo TRUE; или echo FALSE; — то в первом случае будет выведена цифра «1», а во втором — вообще ничего. Это справедливо и для консоли PHP (см. *разд. 1.5*).

Пример:

```
$str = 'Число ' . 123 . ' и логическая величина ' . TRUE;
echo $str;                                  // Число 123 и логическая величина 1
```

Преобразование в логическую величину

В качестве условий в циклах и условных выражениях можно применять выражения, которые возвращают результат типа, отличного от логического.

В этом случае в значение FALSE будут преобразованы:

- числа — целое 0 и вещественное 0.0;
- строки — пустая ('') и '0';

- пустой массив `[]`;
- `NULL`.

Все остальные значения будут преобразованы в `TRUE`.

Пример:

```
// Условие в этом условном выражении...
if ($value != 0)
    . . .
// ...можно записать короче, зная, что ненулевое число неявно преобразуется
// в TRUE:
if ($value)
    . . .
```

Явное преобразование типов

> *Явное* преобразование типов выполняется записью соответствующей команды.

Команда для явного преобразования типа записывается в формате:

`(<обозначение целевого типа>)<преобразуемое значение>`

Вот поддерживаемые *обозначения типов*:

- `int`, `integer` — целочисленный;
- `double`, `float`, `real` — вещественный;

> ***Преобразование в число всегда выполняется «тихо»***
> Даже если строка не содержит в начале числа, сообщение о нефатальной ошибке не выдается.

- `string` — строковый;
- `bool`, `boolean` — логический.

Примеры:

```
$a = '123';
echo gettype($a);                          // string
echo gettype((integer)$a);                 // integer
echo (boolean)2;                           // TRUE
echo (integer)'PHP & MySQL';               // 0
```

Для преобразования в число также можно использовать оператор + (плюс) в формате: `+<значение>`:

```
$a = '123';
echo gettype($a);                          // string
echo gettype(+$a);                         // integer
```

Явное преобразование применяется лишь в тех случаях, когда неявное преобразование может выполниться неправильно.

2.6.3. Строгое сравнение

Все операторы сравнения, приведенные в табл. 2.3, в случае несовпадения типов операндов преобразуют их к одному типу. Примеры:

```
echo 10 == 10;                                          // TRUE
// Второй, строковый, операнд перед сравнением будет преобразован в число
echo 10 == '10';                                        // TRUE
echo 10 > '9';                                          // TRUE
echo 10 > '11';                                         // FALSE
```

Исключением являются *операторы строгого сравнения*: `===` (строго равно) и `!==` (строго не равно).

> *Строгое сравнение* выполняется без преобразования типов. Если типы операндов не совпадают, операнды считаются не равными друг другу.

Примеры:

```
echo 10 === 10;                                         // TRUE
echo 10 === 11;                                         // FALSE
echo 10 !== 11;                                         // TRUE
echo 10 !== 10;                                         // FALSE
// Операнды с несовпадающими типами — не равны
echo 10 === '10';                                       // FALSE
echo 10 !== '10';                                       // TRUE
```

Где может пригодиться строгое сравнение?

Ряд функций PHP в разных случаях могут возвращать значения `0` и `FALSE`. При использовании операторов обычного сравнения значение `0` будет неявно преобразовано в `FALSE`, что приведет к неверному результату. Вот пример с использованием функции `mb_strpos`, выполняющей поиск подстроки в строке:

```
// Ищем букву «Т» в строке «Тригонометрия»
$result = mb_strpos('Тригонометрия', 'Т');
// Эта буква находится на позиции 0 (самый первый символ).
echo $result;                                           // 0
// Если поиск не увенчался успехом, функция mb_strpos возвращает FALSE.
// Проверяем, увенчался ли поиск успехом, для чего удостоверяемся,
// что возвращенный результат не равен FALSE.
// Полученный результат неверен.
echo $result != FALSE;                                  // FALSE
// Возвращенное значение 0 перед сравнением было преобразовано в FALSE,
// и сравнение не выполнилось, так что далее программа будет «считать»,
// что поиск подстроки прошел неуспешно.
// Исправить эту ошибку можно, применив оператор «строго не равно» !==.
echo $result !== FALSE;                                 // TRUE
```

Могут встретиться и другие ситуации, когда без операторов строгого сравнения не обойтись. Мы рассмотрим их далее.

2.7. Вычисление выражений, записанных в строках

Функция `eval(<строка с выражением>)` вычисляет выражение, записанное в переданной ей строке:

```
$str = '$a = sqrt(3 ** 2 + 4 ** 2);'; eval($str); echo $a;      // 5
```

Если в строке с выражением присутствует языковая конструкция `return <значение>`, то указанное *значение* будет возвращено функцией `eval` в качестве результата:

```
$str = 'return sqrt(3 ** 2 + 4 ** 2);'; echo eval($str);       // 5
```

2.8. Упражнение. Доработка веб-приложения convertor3.php

Доработаем написанное на *уроке 1* веб-приложение convertor3.php, чтобы оно, во-первых, могло преобразовывать вещественные числа, а, во-вторых, при вводе отрицательного числа выводило соответствующее предупреждение.

Файл convertor3.php можно найти в папке 1\1.3\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Поле ввода числа (создаваемое указанием в теге `<input>` атрибута `type` со значением `"number"`) позволяет вводить лишь целые числа. Чтобы дать пользователю возможность вводить вещественные числа, это поле следует превратить в обычное текстовое, дав атрибуту `type` тега `<input>` значение `"text"`.

   Откроем файл convertor3.php в текстовом редакторе и выполним нужные правки (здесь и далее вставки и исправления в написанном ранее коде выделены полужирным шрифтом, а удаленные фрагменты кода зачеркнуты):

   ```
   <input type="~~number~~text" name="inches" size="10">
   ```

2. Все введенные посетителем данные хранятся в массиве `$_POST` в виде строк. Чтобы далее при сравнивании величины в дюймах с нулем не случилось никаких эксцессов, явно преобразуем эту величину в вещественное число.

   В выражение, получающее из массива `$_POST` элемент `inches` (величину в дюймах), добавим операцию явного приведения к вещественному типу:

   ```
   if (isset($_POST['inches'])) {
       $ins = (double)$_POST['inches'];
       . . .
   }
   ```

3. Убедимся, что величина в дюймах больше нуля, и, если это так, преобразуем ее и выведем результат. В противном случае выведем на страницу абзац с предупреждением.

Внесем в код следующие исправления:

```
if (isset($_POST['inches'])) {
    $ins = (double)$_POST['inches'];
    if ($ins > 0) {
        $cents = round($ins * 2.54);
        echo '<p>', $ins, ' дюймов = ', $cents, ' сантиметров.</p>';
    } else
        echo '<p>Величина в дюймах должна быть больше нуля.</p>';
}
```

Проверим обновленное приложение в действии. Сначала введем какое-либо вещественное число в формате PHP (применив для разделения целой и дробной частей точку) и преобразуем его в сантиметры. После чего попробуем проделать то же самое с любым отрицательным числом.

Сделайте сами

Сейчас приложение convertor3.php обрабатывает только вещественные числа в формате PHP — с разделителем-точкой. Доработайте приложение так, чтобы поддерживалась и более привычная запятая. Подсказка: используйте функцию `str_replace`.

Урок 3
Хранение данных: переменные, ссылки и константы

Для оперативного хранения значений в PHP используются переменные. Помимо этого поддерживается создание ссылок на переменные и константы.

3.1. Переменные и работа с ними

> *Переменная* — это ячейка памяти, имеющая уникальное имя и способная хранить одно значение. Обращение к переменной для присваивания или извлечения значения выполняется указанием имени этой переменной.

Помимо переменных, создаваемых программистом, ряд переменных формируется самой исполняющей средой PHP. Они хранят всевозможные служебные данные (например, переменные `$_GET` и `$_POST` хранят данные, переданные приложению веб-формами).

3.1.1. Имена переменных

> *Имя переменной* должно состоять из латинских букв, цифр и знаков подчеркивания, причем не должно начинаться с цифры. В самом начале имени переменной обязательно ставится знак доллара — `$` (табл. 3.1).

***Таблица 3.1.** Имена переменных: правильные и неправильные*

Правильные имена переменных	Неправильные имена переменных
`$b`, `$value`, `$result`, `$picture_row`	`$ы`, `$значение 2`, `$123_value`, `123_value`

> Имена переменных в PHP *зависят* от регистра символов. Например, переменные `$value`, `$VALUE` и `$vAlUe` — разные.

Что случится, если в имени переменной не указать знак $?

- Встретив имя без знака доллара, PHP посчитает, что это константа (рассказ о них пойдет позже), и выполнит попытку обратиться к ней.
- Если константы с таким именем нет, PHP решит, что программист пытался написать строковую величину, но забыл взять ее в кавычки, и сделает это за программиста самостоятельно. При этом на страницу будет выведено сообщение о нефатальной ошибке, и приложение, скорее всего, станет работать неправильно.

Пример:

```
platform = 'PHP';
// Ошибка: имя переменной $platform записано без начального знака $
$str = platform . '7';
// Здесь будет выведено сообщение об ошибке
echo $str;                                          // platform7
```

3.1.2. Область видимости переменной

> *Область видимости* определяет, в каком месте программного кода доступна переменная.

Будучи созданной, переменная доступна во всем последующем программном коде, в том числе во всех включаемых программных модулях, равно как и в модуле, выполнившем включение текущего модуля.

Примеры:

```
// Создаем переменную
$a = 123456;
// И можем использовать ее в последующем коде
$result = $a / 2;

// Пример с включаемыми модулями
// Модуль module2.php
$str1 = $str1 . ' JavaScript';
$str2 = 'PHP MySQL';

// Модуль module1.php
$str1 = 'HTML CSS';
require 'module2.php';

// Модуль main.php
require 'module1.php';
$str = $str1 . ' ' . $str2;
echo $str;                                  // HTML CSS JavaScript PHP MySQL
```

Единственным исключением являются функции, написанные программистом, — переменные, созданные «снаружи», внутри них недоступны. Подробнее о функциях и локальных переменных будет рассказано на *уроке 6*.

> *Суперглобальная переменная* — доступна в любом месте программного кода, включая написанные программистом функции. Может быть создана только исполняющей средой PHP.

К числу суперглобальных относятся переменные `$_POST`, `$_GET` и некоторые другие, с которыми мы познакомимся позже.

Как только веб-приложение завершает работу, все созданные в нем переменные автоматически уничтожаются, и хранящиеся в них значения теряются.

3.1.3. Присваивание

> *Присваивание* — это занесение в переменную нового значения. При этом значение, ранее хранившееся в переменной, теряется.

Простое присваивание

> *Простое присваивание* лишь заносит в переменную новое значение, не выполняя никаких дополнительных действий.

Оператор простого присваивания — знак «равно» (=). Его синтаксис: *<переменная> = <значение>*.

Примеры:

```
$a = 2;
$str = 'PHP и MySQL — это сила!';
$s = sqrt(3 ** 4 + 5 ** 2);
```

Оператор = в качестве результата возвращает значение, присвоенное переменной. Благодаря этому можно писать выражения наподобие:

```
$a = $b = 123;
```

Здесь сначала переменной `$b` присвоится значение `123`, а потом результат этой операции — то же число `123` — будет присвоено переменной `$a`.

> Операторы присваивания выполняются в направлении *справа налево* (в то время как почти все остальные операторы с одинаковым приоритетом выполняются слева направо).

Комбинированное присваивание

> При *комбинированном присваивании* из указанной переменной извлекается значение, над ним и заданным операндом выполняется какое-либо действие, после чего результат присваивается той же переменной.

Все операторы комбинированного присваивания, поддерживаемые PHP, и их эквиваленты приведены в табл. 3.2.

Таблица 3.2. *Операторы комбинированного присваивания*

Оператор	Эквивалент	Оператор	Эквивалент
`a += b`	`a = a + b`	`a -= b`	`a = a - b`
`a *= b`	`a = a * b`	`a /= b`	`a = a / b`
`a %= b`	`a = a % b`	`a **= b`	`a = a ** b`
`a .= b`	`a = a . b`	`a &= b`	`a = a & b`
`a |= b`	`a = a | b`	`a ^= b`	`a = a ^ b`
`a <<= b`	`a = a << b`	`a >>= b`	`a = a >> b`

Пример:

```
$str1 .= ' JavaScript';
// Эквивалент: $str1 = $str1 . ' JavaScript';
```

Операторы комбинированного присваивания выполняются быстрее эквивалентных им выражений.

Копирование значения при присваивании. Значащие типы данных

> При присваивании переменной значения, хранящегося в другой переменной, ей присваивается *копия* этого значения.

Пример:

```
// Создаем переменную $a и присваиваем ее значение переменной $b
$a = 2;
$b = $a;
// Присваиваем другое значение переменной $a
$a = 333;
// Проверяем: значение переменной $b не изменилось
echo $b;                                                    // 2
```

> *Значащий тип данных* — тип, чьи значения при присваивании другим переменным копируются. К значащим относятся целочисленный, вещественный, строковый, логический типы и массив.

Полезно знать...

Значения, относящиеся к значащим типам данных, хранятся непосредственно в переменных.

3.1.4. Переменные переменных

> *Переменная переменной* — переменная, хранящая имя другой переменной в виде строки и используемая для доступа к этой переменной.

Имя переменной, хранящееся в переменной переменной, не должно включать начальный символ `$`.

Обращение к переменной переменной выполняется указанием в начале ее имени двойного знака доллара (`$$`).

Пример:

```
// Создаем переменную $var и присваиваем ей строку 'inches',
// которая станет именем создаваемой позже переменной
$var = 'inches';
// Используем $var как переменную переменной для создания переменной
// $inches
$$var = 27;
```

```
echo $inches;                                          // 27
echo $$var, ' дюймов = ', $inches * 2.54, ' сантиметров';
                                        // 27 дюймов = 68.58 сантиметров
```

Переменная переменной может пригодиться, если нужно обращаться к разным переменным в зависимости от выполнения или невыполнения какого-либо условия.

3.1.5. Проверка и удаление переменных

- Проверка, существует ли *переменная*, — функция `isset(<переменная>)`. В качестве результата возвращает `TRUE`, если *переменная* существует и хранит значение, отличное от `NULL`, в противном случае возвращается `FALSE`:

```
$var1 = 100; $var2 = NULL;
echo isset($var1);                                     // TRUE
echo isset($var2);                                     // FALSE
// Переменная $var3 не существует
echo isset($var3);                                     // FALSE
```

- Проверка, существуют ли все указанные *переменные*, — функция `isset` в формате:

```
isset(<переменная 1>, <переменная 2> . . . <переменная n>)
```

Возвращает `TRUE`, если все указанные *переменные* существуют и хранят значения, отличные от `NULL`, иначе возвращает `FALSE`:

```
echo isset($var1, $var2);                              // FALSE
```

- Проверка, хранит ли *переменная* пустое значение, — функция `empty(<переменная>)`. Возвращает `TRUE`, если *переменная* существует и хранит пустое значение, и `FALSE` — в противном случае. Пустыми значениями считаются пустая строка `''`, строка `'0'`, целое число `0`, вещественное число `0.0`, значения `FALSE`, `NULL` и пустой массив `[]`:

```
$var1 = 100; $var2 = 0; $var3 = '';
echo empty($var1);                                     // FALSE
echo empty($var2);                                     // TRUE
echo empty($var3);                                     // TRUE
// Переменная $var4 не существует
echo empty($var4);                                     // TRUE
```

- Удаление указанных *переменных* — функция `unset`:

```
unset(<переменная 1>, <переменная 2> . . . <переменная n>)
```

Результата функция не возвращает. Пример:

```
$var = 123;
echo isset($var);                                      // TRUE
unset($var);
// Переменная $var больше не существует
echo isset($var);                                      // FALSE
```

3.2. Ссылки

> *Ссылка* в PHP — указатель на одну переменную, сохраненный в другой переменной, или псевдоним первой переменной.

Чтобы получить ссылку на первую переменную для присваивания другой переменной, перед именем первой переменной следует поставить знак «апостроф» — `&`.

Пример:

```
$a = 10;
$b = &$a;
// Теперь переменная $b ссылается на значение переменной $a
// Меняем значение переменной $a, воспользовавшись ссылкой $b
$b = 20;
echo $a;                                                    // 20
```

Чаще всего ссылки применяются в функциях для передачи им параметров, значения которых должны изменяться внутри этих функций (о функциях будет рассказано на *уроке 6*).

3.3. Константы

> *Константа* — это переменная, чье значение после присваивания не может быть изменено.

Константы обычно задействуются, чтобы сохранить под легко запоминающимися именами какие-либо значения, которые не должны изменяться в процессе работы приложения.

Константа создается с помощью языковой конструкции `const`:

```
const <имя константы> = <значение константы>;
```

Имя константы должно удовлетворять тем же требованиям, что и имя переменной, только перед ним *не ставится* знак `$`.

Имена констант зависят от регистра символов. Традиционно их набирают буквами в верхнем регистре.

Значением константы может быть постоянная величина любого типа — в том числе массив или выражение, содержащие только постоянные величины.

Обращение к константе выполняется так же, как и обращение к переменной, — указанием имени.

Если попытаться присвоить константе новое значение, исполняющая среда PHP выдаст ошибку.

Примеры:

```
const PLATFORM_NAME = 'PHP';
const PLATFORM_VERSION = 7;
```

```
// Обращаемся к константам
echo 'Платформа ', PLATFORM_NAME, ' версии ', PLATFORM_VERSION;
                                                    // Платформа PHP версии 7
// Значением константы может быть и выражение, содержащее только
// постоянные величины
const PLATFORM = PLATFORM_NAME . PLATFORM_VERSION;
// Константе нельзя присвоить значение, вычисляемое в ходе работы
// приложения!
const qa = sqrt(200);                                  // Ошибка!
```

Другой способ создать константу, оставшийся от старых версий PHP, — использовать функцию `define(<имя константы>, <значение константы>)`, где *имя константы* задается в виде строки:

```
define('PLATFORM_NAME', 'PHP');
define('PLATFORM_VERSION', 7);
```

Помимо констант, создаваемых самим программистом, ряд констант среда PHP создает сама (например, константу `M_PI`, хранящую число π).

Полезно знать...

Константы вычисляются в момент компиляции программного модуля перед его первым исполнением. Именно поэтому константам можно присваивать только постоянные величины — их исполняющая среда сможет вычислить сразу же, еще до запуска приложения.

Урок 4
Массивы

Массив — это набор значений разных типов (элементов), хранящийся в одной переменной. Массив также относится к значащим типам данных (см. *разд. 3.1.3*). PHP поддерживает три разновидности массивов.

4.1. Индексированные массивы

Индексированным называется массив, элементы которого последовательно пронумерованы.

Доступ к элементу такого массива выполняется по порядковому номеру (индексу) элемента. Нумерация элементов в индексированном массиве начинается с 0.

Индексированные массивы знакомы нам по *разд. 1.1.4*. Они прекрасно подходят для хранения упорядоченных значений, которые часто приходится перебирать в цикле.

4.1.1. Создание массивов

Индексированный массив создается указанием входящих в него элементов в квадратных скобках через запятую:

```
$arr = ['HTML', 'CSS', 'JavaScript', 'PHP', 'MySQL'];
```

В таком случае первый элемент получит индекс `0`, второй — `1`, третий — `2`, а последний — равный количеству элементов в массиве (его размеру) за вычетом 1:

```
echo $arr;
// Элементы выведенного массива приведены в квадратных скобках
// в формате <индекс элемента> => <значение элемента>.
// Такой формат будет применяться и в дальнейшем[1].
// [0 => 'HTML', 1 => 'CSS', 2 => 'JavaScript', 3 => 'PHP', 4 => 'MySQL']
```

У элементов массива можно напрямую указать индексы, записав их в формате *<индекс элемента> => <значение элемента>*:

```
$arr2 = [1 => 'HTML', 2 => 'CSS', 3 => 'JavaScript', 10 => 'PHP',
    11 => 'MySQL'];
```

[1] На самом деле при попытке вывести массив посредством конструкции `echo` будет выведена лишь строка `'array'` и предупреждение о нефатальной ошибке. Автор использовал такое представление выведенного массива лишь для наглядности.

```
echo $arr2;
// [1 => 'HTML', 2 => 'CSS', 3 => 'JavaScript', 10 => 'PHP',
    11 => 'MySQL']
```

Индексы можно задавать не у всех элементов. Тогда «безындексные» элементы, следующие за элементом с явно указанным индексом, получат индексы, последовательно увеличивающиеся на единицу:

```
$arr3 = [1 => 'HTML', 'CSS', 'JavaScript', 5 => 'PHP', 'MySQL'];
echo $arr3;
// [1 => 'HTML', 2 => 'CSS', 3 => 'JavaScript', 5 => 'PHP', 6 => 'MySQL']
```

Для создания массива также можно использовать оставшуюся от предыдущих версий PHP функцию `array`, вызываемую в формате:

```
array(<элемент 1>, <элемент 2>, <элемент 3> . . . <элемент n>)
```

Примеры:

```
$arr = array('HTML', 'CSS', 'JavaScript', 'PHP', 'MySQL');
$arr3 = array(1 => 'HTML', 'CSS', 'JavaScript', 5 => 'PHP', 'MySQL');
```

4.1.2. Работа с элементами массивов

Обращение к элементу массива выполняется указанием после переменной, в которой хранится массив, квадратных скобок (`[]`) с индексом нужного элемента.

```
// Извлекаем и выводим значения элементов массивов
echo $arr[0];                                          // HTML
echo $arr2[2];                                         // CSS
echo $arr3[5];                                         // PHP
// Задаем для элемента массива новое значение
$arr2[11] = 'MariaDB';
echo $arr2;
// [1 => 'HTML', 2 => 'CSS', 3 => 'JavaScript', 10 => 'PHP',
    11 => 'MariaDB']
```

В качестве индекса элемента задается целое число

Если задан нецелочисленный индекс, он будет преобразован в целое число по правилам, описанным в *разд. «Неявное преобразование типов» урока 2*:

```
echo $arr[0.7];                                        // HTML
echo $arr3['3'];                                       // JavaScript
```

Единственное исключение: строки, содержащие буквы после цифр, не подвергнутся преобразованию в целые числа. Такие строки будут рассматриваться как ключи элементов ассоциативного массива, и, в случае отсутствия элементов с такими ключами, мы получим ошибку исполнения:

```
echo $arr3['3elem'];          // Ошибка: элемента с ключом 3elem нет
```

Чтобы создать в массиве новый элемент, достаточно указать у него еще не «занятый» индекс и присвоить нужное значение:

```
// Создаем в массиве $arr новый элемент с индексом 5
$arr[5] = 'PostgreSQL';
echo $arr;
// [0 => 'HTML', 1 => 'CSS', 2 => 'JavaScript', 3 => 'PHP', 4 => 'MySQL',
    5 => 'PostgreSQL']
```

Если не указать индекс в квадратных скобках, новый элемент получит следующий «свободный» индекс:

```
$arr[] = 'MSSQL';
$arr[] = 'SQLite';
echo $arr;
// [0 => 'HTML', 1 => 'CSS', 2 => 'JavaScript', 3 => 'PHP', 4 => 'MySQL',
    5 => 'PostgreSQL', 6 => 'MSSQL', 7 => 'SQLite']
```

Удалить элемент массива можно посредством функции `unset`, описанной в *разд. 3.1.5*:

```
// Удаляем из массива $arr3 элемент с индексом 2
unset($arr3[2]);
```

При этом нумерация элементов, у которых не были явно указаны индексы, соответственно сдвинется:

```
echo $arr3;
// [1 => 'HTML', 2 => 'JavaScript', 5 => 'PHP', 6 => 'MySQL']
// Элемент 'JavaScript' ранее имел индекс 3, а теперь получил индекс 2,
// поскольку предыдущий элемент был удален
```

4.2. Ассоциативные массивы

Элементы *ассоциативного массива* доступны не по целочисленным индексам, а по строковым (*ключам*). Такие массивы хорошо подходят для хранения набора значений, к которым нужно иметь, в первую очередь, произвольный доступ — слова запоминаются лучше, чем числа.

Ассоциативный массив создается так же, как и индексированный, за тем исключением, что индексы следует указывать у всех элементов:

```
$platforms = ['application' => 'PHP', 'data' => 'MySQL'];
echo $platforms;
// ['application' => 'PHP', 'data' => 'MySQL']
```

Работа с элементами таких массивов выполняется аналогично:

```
echo $platforms['application'];                    // PHP
$platforms['data'] = 'MariaDB';
// Создаем новый элемент с ключом 'view'
$platforms['view'] = 'HTML, CSS, JavaScript';
// Удаляем элемент 'data'
unset($platforms['data']);
echo $platforms;
// ['application' => 'PHP', 'view' => 'HTML, CSS, JavaScript']
```

4.3. Комбинированные массивы

> *Комбинированный массив* содержит элементы как с целочисленными индексами, так и со строковыми ключами, объединяя в себе индексированный и ассоциативный массивы.

Комбинированные массивы создаются и используются так же, как и рассмотренные ранее:

```
$arr = ['HTML', 'CSS', 'JavaScript', 'application' => 'PHP',
    'data' => 'MySQL'];
echo $arr;
// [0 => 'HTML', 1 => 'CSS', 2 => 'JavaScript', 'application' => 'PHP',
    'data' => 'MySQL']
echo $arr[0];                                          // HTML
echo $arr['data'];                                     // MySQL
```

Полезно знать...

Любопытно, что сама платформа PHP не «различает» между собой индексированные, ассоциативные и комбинированные массивы, считая их принадлежащими к одной разновидности — комбинированным. Разделение массивов на разновидности введено лишь для удобства программистов.

4.4. Вложенные массивы

> *Вложенный массив* — массив, являющийся элементом другого массива (*внешнего*).

Пример создания вложенного массива:

```
$platforms = [
    ['HTML', 'CSS', 'JavaScript'],
    ['application' => 'PHP', 'data' => 'MySQL']
];
```

Чтобы обратиться к элементу вложенного массива, нужно записать две пары квадратных скобок, указав в первой паре индекс элемента внешнего массива, в котором хранится вложенный массив, а во второй паре — индекс требуемого элемента вложенного массива.

Примеры:

```
echo $platforms[0][2];                                 // JavaScript
echo $platforms[1]['application'];                     // PHP
$platforms[1]['data'] = 'MariaDB';
```

4.5. Работа с массивами

- Получение размера *массива* — функция `count(<массив>[, <режим>])`:

```
$arr = ['HTML', 'CSS', 'JavaScript', 'application' => 'PHP',
    'data' => 'MySQL'];
echo count($arr);                                    // 5
```

Параметр *режим* указывает, подсчитывать ли еще и элементы вложенных массивов (если такие есть). Он задается в виде одной из констант:

- `COUNT_NORMAL` — подсчитывать только элементы самого массива (поведение функции по умолчанию, если *режим* не указан):

```
$platforms = [
    ['HTML', 'CSS', 'JavaScript'],
    ['application' => 'PHP', 'data' => 'MySQL']
];
echo count($platforms);                          // 2
// 2 элемента в самом массиве $platform.
// Элементы во вложенных массивах не считаются.
```

- `COUNT_RECURSIVE` — подсчитывать и элементы вложенных массивов:

```
echo count($platforms, COUNT_RECURSIVE);      // 7
// 2 элемента в самом массиве $platform, 3 элемента в первом
// вложенном массиве и 2 — во втором
```

Можно использовать и функцию `sizeof`, которая вызывается точно так же.

- Проверка, присутствует ли в *массиве* элемент с заданным *индексом*, — функция `key_exists(<индекс>, <массив>)`. Если такой элемент есть, возвращается `TRUE`, в противном случае — `FALSE`:

```
echo key_exists(1, $arr);                        // TRUE
echo key_exists('application', $arr);            // TRUE
echo key_exists('special', $arr);                // FALSE
```

Можно использовать и функцию `array_key_exists`, которая вызывается точно так же.

- Проверка, присутствует ли в *массиве* элемент с заданным *значением*, — функция `in_array`. Если такой элемент есть, возвращается `TRUE`, в противном случае — `FALSE`:

```
in_array(<искомое значение>, <массив>[,
        <запретить преобразование типов?>])
```

Если третьим параметром передать значение `FALSE` или вообще не указать его, функция будет выполнять преобразование типов перед сравнением (как если бы применялся оператор «равно» ==). Если же указать значение `TRUE`, преобразование типов выполняться не будет (как при использовании оператора «строго равно» ===).

Примеры:

```
$arr2 = [1, 4, 78, '2', -900];
echo in_array(4, $arr2);                          // TRUE
echo in_array(22, $arr2);                         // FALSE
echo in_array('2', $arr2);                        // TRUE
echo in_array(2, $arr2);                          // TRUE
echo in_array('2', $arr2, TRUE);                  // TRUE
echo in_array(2, $arr2, TRUE);                    // FALSE
```

- Поиск в *массиве* элемента с заданным *значением* — функция `array_search`:

```
array_search(<искомое значение>, <массив>[,
             <запретить преобразование типов?>])
```

В качестве результата возвращается индекс найденного элемента или `FALSE`, если поиск не увенчался успехом.

Третий параметр имеет то же назначение, что и у функции `in_array` (см. ранее). Примеры:

```
echo array_search(4, $arr2);                      // 1
echo array_search(22, $arr2);                     // FALSE
echo array_search('2', $arr2);                    // 3
echo array_search(2, $arr2);                      // 3
echo array_search('2', $arr2, TRUE);              // 3
echo array_search(2, $arr2, TRUE);                // FALSE
```

- Добавление заданных *элементов* в конец *массива* — функция `array_push`:

```
array_push(<массив>, <элемент 1>, <элемент 2> . . . <элемент n>)
```

В качестве результата возвращается новый размер *массива*. Пример:

```
$arr3 = ['PHP', 'Python'];
echo array_push($arr3, 'Ruby', 'Visual Basic');    // 4
echo $arr3;
// [0 => 'PHP', 1 => 'Python', 2 => 'Ruby', 3 => 'Visual Basic']
```

- Добавление заданных *элементов* в начало *массива* — функция `array_unshift`:

```
array_unshift(<массив>, <элемент 1>, <элемент 2> . . . <элемент n>)
```

В качестве результата возвращается новый размер *массива*. Пример:

```
echo array_unshift($arr3, 'C++');                 // 5
echo $arr3;
// [0 => 'C++', 1 => 'PHP', 2 => 'Python', 3 => 'Ruby',
    4 => 'Visual Basic']
```

- Получение первого элемента *массива* с его одновременным удалением — функция `array_shift(<массив>)`:

```
echo array_shift($arr3);                          // C++
echo $arr3;
// [0 => 'PHP', 1 => 'Python', 2 => 'Ruby', 3 => 'Visual Basic']
```

- Получение последнего элемента *массива* с его одновременным удалением — функция `array_pop(<массив>)`:

```
echo array_pop($arr3);                                    // Visual Basic
echo $arr3;
// [0 => 'PHP', 1 => 'Python', 2 => 'Ruby']
```

- Преобразование элементов заданного *массива* в обычные переменные — функция `extract`:

```
extract(<массив>[, <режим преобразования>[, <префикс>]])
```

В результате каждый элемент *массива* будет преобразован в обычную переменную с именем, совпадающим с ключом этого элемента. Элементы с числовыми индексами по умолчанию в переменные не преобразуются.

В качестве результата функция возвращает количество преобразованных таким образом элементов.

Пример:

```
$arr = ['HTML', 'CSS', 'JavaScript', 'application' => 'PHP',
    'data' => 'MySQL'];
echo extract($arr);                                       // 2
// Были преобразованы только элементы с ключами
echo $application;                                        // PHP
echo $data;                                               // MySQL
```

Параметр *режим преобразования* указывает, что делать в случае, если уже существует переменная с именем, совпадающим с ключом элемента *массива*, или если ключ элемента не удовлетворяет требованиям к именам переменных (например, начинается с цифры, — подробно об именах переменных рассказано в *разд. 3.1.1*). Параметр *режим* указывается в виде одной из следующих констант:

- `EXTR_OVERWRITE` — если существует переменная с именем, совпадающим с ключом элемента, перезаписать ее значение (поведение по умолчанию, если *режим* не указан);
- `EXTR_SKIP` — не перезаписывать переменную с совпадающим именем;
- `EXTR_PREFIX_SAME` — если существует переменная с совпадающим именем, создать новую версию этой переменной с добавленным к ее имени *префиксом*;
- `EXTR_PREFIX_INVALID` — добавить *префикс* только к некорректным именам. Этот режим позволяет преобразовать в переменные также элементы с индексами;
- `EXTR_PREFIX_ALL` — добавить *префикс* к именам всех создаваемых переменных. Этот режим позволяет преобразовать в переменные также элементы с индексами;
- `EXTR_IF_EXISTS` — только перезаписать переменные с совпадающими именами;

- EXTR_PREFIX_IF_EXISTS — только создать новые версии переменных с совпадающими именами, добавив к их именам заданный *префикс*;
- EXTR_REFS — создать ссылки на элементы *массива*. Значение этой константы можно объединить с любой другой из ранее упомянутых констант с помощью оператора побитового ИЛИ | (например: EXTR_SKIP | EXTR_REFS).

Добавляемый *префикс* отделяется от имени создаваемой переменной символом подчеркивания.

Примеры:

```
echo extract($arr, EXTR_PREFIX_INVALID, 'a');       // 5
// Будут также преобразованы элементы с числовыми ключами
echo $a_0;                                          // HTML
echo $a_1;                                          // CSS
echo $a_2;                                          // JavaScript
```

4.6. Упражнение. Вывод описаний к изображениям в фотогалерее

Сделаем так, чтобы на сайте фотогалереи при наведении курсора мыши на миниатюру на экране появлялась всплывающая подсказка с описанием этого изображения.

Файлы сайта фотогалереи можно найти в папке 1\1.4\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. В папке 4\!sources электронного архива найдем файл data.php. Скопируем его в папку modules, находящуюся в корневой папке веб-сервера, заменив имеющуюся там старую копию.
2. Откроем копию файла modules\data.php в текстовом редакторе и посмотрим на код, создающий массив $arr_images с изображениями (здесь показан только первый элемент этого массива):

```
$arr_images = [
    ['src' => '200804282020.jpg', 'desc' => 'Цветы кактуса'],
    . . .
];
```

В новой редакции файла modules\data.php каждое изображение из фотогалереи описывается вложенным ассоциативным массивом из двух элементов с ключами:

- 'src' — интернет-адрес файла с изображением;
- 'desc' — описание к изображению.

Сразу же исправим серверное приложение index.php, выводящее страницу со списком изображений, иначе оно не заработает с новым массивом.

3. Откроем для этого файл index.php, найдем PHP-фрагмент, выводящий интернет-адрес файла с изображением, и исправим его следующим образом (добавления и правки в написанном ранее коде выделены здесь полужирным шрифтом):

```
<img src="/images/<?php echo $arr_images[$i]['src'] ?>">
```

   Единственное, что мы добавили, — обращение к элементу с ключом `'src'` вложенного массива, где хранится интернет-адрес файла.

4. Текст всплывающей подсказки, которая будет выводиться при наведении курсора мыши на элемент страницы, указывается в атрибуте тега `title`. Впишем в тег `<img>`, выводящий изображение, HTML- и PHP-код, который сформирует этот атрибут тега:

```
<img src="/images/<?php echo $arr_images[$i]['src'] ?>"
title="<?php echo $arr_images[$i]['desc'] ?>">
```

Для проверки исправленного сайта перейдем по интернет-адресу: **http://localhost/**. Наведем курсор мыши на самое первое изображение — цветущий кактус — и увидим всплывающую подсказку (рис. 4.1).

Рис. 4.1. Всплывающая подсказка, появившаяся при наведении курсора мыши на миниатюру

4.7. Упражнение. Веб-страница для просмотра изображений

На странице списка изображений нашего сайта фотогалереи выводятся небольшие миниатюры картинок. Сделаем еще одно серверное приложение, генерирующее страницу с полноразмерным изображением. Посетитель будет переходить на эту страницу, щелкнув на нужной миниатюре в списке.

Файлы сайта фотогалереи можно найти в папке 4\4.6\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. В папке 4\!sources электронного архива найдем файлы styles.css (с исправленной таблицей стилей) и item.php (хранящий заготовку для написания нового приложения). Скопируем их в корневую папку веб-сервера, заменив имеющуюся там старую копию файла styles.css.

Сначала допишем приложение index.php, выводящее список изображений. Поместим каждую миниатюру в гиперссылку, в которой пропишем интернет-адрес приложения item.php. В результате посетитель, щелкнув на миниатюре, запустит приложение item.php, которое выведет страницу с изображением.

2. Откроем для этого файл index.php, найдем фрагмент, выводящий тег `<img>` с очередным изображением из массива, и заключим его в тег `<a>` (добавления и правки в написанном ранее коде выделены здесь полужирным шрифтом):

```
<a href="item.php">
    <img src="/images/<?php echo $arr_images[$i]['src'] ?>"
    title="<?php echo $arr_images[$i]['desc'] ?>">
</a>
```

3. Передать приложению item.php индекс нужного изображения проще всего через GET-параметр с именем, например, `index`. Мы поставим в конце интернет-адреса приложения item.php, записанного в теге гиперссылки, вопросительный знак, имя GET-параметра `index`, знак равенства и PHP-код, выводящий индекс текущего элемента массива из переменной `$i`:

```
<a href="item.php?index=<?php echo $i ?>">
    <img src="/images/<?php echo $arr_images[$i]['src'] ?>"
    title="<?php echo $arr_images[$i]['desc'] ?>">
</a>
```

4. Откроем копию файла item.php в текстовом редакторе и вставим в самое начало кода PHP-фрагмент, который выполнит включение модуля modules\data.php, получит с GET-параметром `index` индекс выбранного посетителем изображения, извлечет из массива `$arr_images` элемент с этим индексом и сохранит его в переменной `$item` для дальнейшего использования:

```
<?php
    require_once 'modules\data.php';
    $item = $arr_images[$_GET['index']];
?>
<!doctype html>
```

5. Вставим под HTML-кодом, создающим заголовок первого уровня, код, который выведет описание изображения:

```
<h1>Фотогалерея</h1>
<h2><?php echo $item['desc'] ?></h2>
```

6. Вставим в пустой тег `<section>` код, который выведет само изображение:

```
<section id="gallery-item">
    <img src="/images/<?php echo $item['src'] ?>">
</section>
```

Проверим сайт. Перейдя по интернет-адресу: **http://localhost/**, щелкнем на какой-либо миниатюре — и увидим страницу с полноразмерным изображением (рис. 4.2).

Рис. 4.2. Веб-страница для просмотра выбранного изображения

Сделайте сами

Сейчас на странице просмотра изображения в качестве названия (оно задается в теге `<title>` и отображается на корешке вкладки, в которой открыта страница) выводится слово **Фотогалерея**. Сделайте так, чтобы там выводился текст формата *<описание к изображению>* `::` `Фотогалерея` (рис. 4.3).

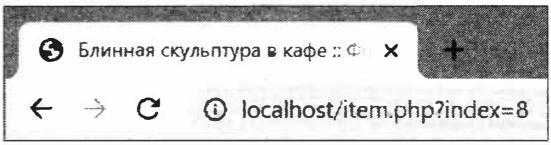


Рис. 4.3. Название исправленной веб-страницы для просмотра изображения

Урок 5
Управляющие конструкции

PHP предлагает множество средств для управления исполнением кода, два из которых: условное выражение и цикл со счетчиком — нам уже знакомы.

5.1. Условные выражения и операторы

PHP поддерживает две разновидности условных выражений: простое и множественное.

5.1.1. Простое условное выражение

> *Простое условное выражение* выполняет одно выражение, если заданное условие истинно, и другое — если условие ложно.

Формат простого условного выражения:

```
if (<условие>)
    <выражение TRUE>            // Выполняется, если условие истинно
                                // (то есть в качестве результата
                                // возвращает TRUE)
else
    <выражение FALSE>           // Выполняется, если условие ложно
                                // (возвращает FALSE)
```

Пример:

```
if ($n != 0)
    // Если делитель, хранящийся в переменной $n, не равен нулю,
    // выполняем деление
    $result = $t / $n;
else
    // В противном случае выводим предупреждение
    $result = 'Делитель не может быть равным нулю!';
```

> Сокращенная форма простого условного выражения не содержит выражения FALSE.

Его формат:

```
if (<условие>)
    <выражение TRUE>            // Выполняется, если условие истинно

// Если условие ложно, ничего не происходит
```

Пример:

```
if (isset($_POST['inches']))
    // Если в ассоциативном массиве $_POST есть элемент с ключом
    // 'inches', пересчитываем эту величину в сантиметры и выводим
    // на экран результат
    echo $_POST['inches'] * 2.54, ' сантиметров';
// В противном случае ничего не делаем
```

5.1.2. Множественное условное выражение

> *Множественное условное выражение* — это набор произвольного количества простых условных выражений.

Формат множественного условного выражения:

```
if (<условие 1>)
    <выражение TRUE 1>          // Выполняется, если условие 1 истинно
else if (<условие 2>)
    <выражение TRUE 2>          // Выполняется, если условие 2 истинно
else if (<условие 3>)
    <выражение TRUE 3>          // Выполняется, если условие 3 истинно
. . .
else if (<условие n>)
    <выражение TRUE n>          // Выполняется, если условие n истинно
[else
    <выражение FALSE>]          // Выполняется, если все условия ложны
```

Слова `else if` можно записать слитно: `elseif`.

Пример:

```
if ($n == 0)
    $s = 'Совсем ничего нет';
else if ($n == 1)
    $s = 'Всего один. Негусто...';
else if ($n == 2)
    $s = 'Пара';
else if ($n == 3)
    $s = 'Тройка';
else
    $s = 'Чем больше — тем лучше';
```

5.1.3. Условные операторы

> Условный оператор `?` возвращает *результат TRUE*, если заданное *условие* истинно, и *результат FALSE* — в противном случае.

Его формат:

```
(<условие>) ? <результат TRUE> : <результат FALSE>
```

Пример:

```
$n = 0;
echo ($n == 0) ? 'Ноль' : 'Не ноль';                    // Ноль
$n = 1;
echo ($n == 0) ? 'Ноль' : 'Не ноль';                    // Не ноль
```

> Оператор сравнения с NULL ?? возвращает *результат 1*, если *результат 1* не равен NULL, и *результат 2* — в противном случае.

Его формат:

```
<результат 1> ?? <результат 2>
```

Пример:

```
$n = 123;
echo $n ?? 0;                                           // 123
$n = NULL;
echo $n ?? 0;                                           // 0
```

5.2. Блоки

Условное выражение позволяет использовать только единичные *выражения* TRUE и *FALSE*.

> Если нужно подставить в условное выражение несколько выражений, следует объединить их в *блок*, заключив в фигурные скобки { и }.

Пример использования блока:

```
if ($n != 0) {
    $result = $t / $n;
    $s = 'Деление выполнено успешно';
} else
    $s = 'Делитель не может быть равным нулю!';
```

5.3. Выражения выбора. Прерывание

> *Выражение выбора* последовательно сравнивает заданное *значение* с набором указанных *величин* и при совпадении с какой-либо *величиной* выполняет соответствующий *фрагмент* кода.

Его формат:

```
switch (<значение>) {
    case <величина 1>:
        <фрагмент 1>     // Выполняется, если значение == величине 1
        break;
    case <величина 2>:
        <фрагмент 2>     // Выполняется, если значение == величине 2
        break;
```

```
    case <величина 3>:
        <фрагмент 3>      // Выполняется, если значение == величине 3
        break;
    . . .
    case <величина n>:
        <фрагмент n>      // Выполняется, если значение == величине n
        break;
    [default:
        <фрагмент default>]      // Выполняется, если значение != ни
                                 // одной из заданных величин
}
```

Как показывает практика, между конструкцией `switch` и круглыми скобками со *значением* ставить пробел необязательно.

Выражение выбора выглядит нагляднее, чем аналогичное множественное условное выражение, и в ряде случаев выполняется быстрее.

Пример:

```
switch ($n) {
    case 0:
        $s = 'Совсем ничего нет';
        break;
    case 1:
        $s = 'Всего один. Негусто...';
        break;
    case 2:
        $s = 'Пара';
        break;
    case 3:
        $s = 'Тройка';
        break;
    default:
        $s = 'Чем больше — тем лучше';
}
```

При использовании выражений выбора следует иметь в виду два момента:

- выражение выбора «за кулисами» использует обычный оператор «равно» `==`. Это значит, что в случае несовпадения типов *значения* и очередной *величины* PHP выполнит преобразование типов.

  В нашем случае, если переменная `$n` хранит строку `'1'`, то, поскольку все величины заданы в виде чисел, PHP при каждом сравнении будет преобразовывать ее в число. В результате выполнится второй по счету фрагмент, и переменной `$s` будет присвоена строка `'Всего один. Негусто...'`;

- в конце каждого фрагмента, за исключением самого последнего, в большинстве случаев ставится оператор `break`, выполняющий прерывание.

> *Прерывание* производится оператором `break`, немедленно прекращает исполнение управляющей конструкции и начинает выполнять код, следующий за ней.

Если операторы `break` во всех *фрагментах* опустить, после выполнения *фрагмента*, соответствующего совпавшей *величине*, станут выполняться последующие *фрагменты*, пока не будет достигнут конец выражения выбора. Понятно, что результат исполнения кода окажется не тем, на который мы рассчитывали.

Например, если переменная `$n` хранит `1`, выполнится сначала второй по счету *фрагмент* (соответствующий совпавшей *величине*), потом — все последующие *фрагменты*. В результате в переменной `$s` окажется строка `'Чем больше – тем лучше'`, совершенно не соответствующая заданному *значению*.

Такую особенность можно использовать, если при совпадении *значения* с любой *величиной* из некоторого набора требуется выполнить один и тот же *фрагмент* кода. Это позволит несколько сократить код, не потеряв в наглядности.

В табл. 5.1 приведены два примера, выдающие одинаковый результат. При этом левый пример написан без использования описанной ранее особенности, а правый — использует ее.

***Таблица 5.1.** Примеры, иллюстрирующие использование особенности применения оператора `break`*

Пример 1	Пример 2

`switch ($n) {`
`  case 0:`
`  $s = 'Ноль или один';`
`  break;`
`  case 1:`
`  $s = 'Ноль или один';`
`  break;`
`  case 2:`
`  $s = 'Два';`
`}`
	`switch ($n) {`
	`  case 0:`
	`  case 1:`
	`  $s = 'Ноль или один';`
	`  break;`
	`  case 2:`
	`  $s = 'Два';`
	`}`

Полезно знать...

Язык программирования JavaScript, очень похожий на PHP, при исполнении выражений выбора «за кулисами», напротив, использует оператор «строго равно» `===`.

5.4. Циклы

Цикл выполняет заданное выражение или блок многократно, пока остается истинным указанное условие.

5.4.1. Цикл со счетчиком

> *Цикл со счетчиком* выполняется строго определенное количество раз. Назван он так потому, что использует переменную-счетчик для хранения порядкового номера текущей итерации цикла.

Формат записи цикла со счетчиком:

```
for (<установка>; <условие>; <приращение>)     // Заголовок цикла
    <тело цикла>                               // «Полезная нагрузка» цикла
```

Алгоритм исполнения цикла со счетчиком представлен на рис. 5.1.

Здесь:

- *Установка* — присваивает счетчику номер самой первой итерации цикла (обычно 0);
- *Условие* — удостоверяется, что номер текущей итерации не превысил заданное количество итераций (то есть цикл еще не выполнился до конца);
- *Тело цикла* — можно записать в виде единичного выражения или блока;
- *Приращение* — изменяет (обычно инкрементирует) номер текущей итерации, хранящийся в счетчике.

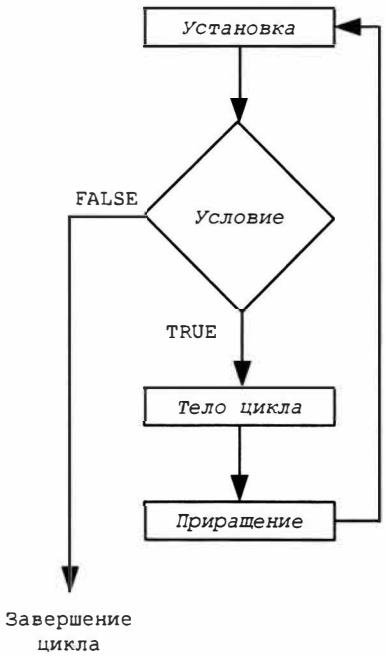


Рис. 5.1. Алгоритм исполнения цикла со счетчиком

Пример:

```
$arr = ['HTML', 'CSS', 'JavaScript', 'PHP', 'MySQL'];
$arr_cnt = count($arr);
for ($i = 0; $i < $arr_cnt; $i++)
    echo $arr[$i], ' ';
                                    // HTML CSS JavaScript PHP MySQL
```

Приращение может не инкрементировать, а декрементировать счетчик (что может понадобиться, например, для перебора элементов массива в обратном порядке):

```
for ($i = $arr_cnt - 1; $i >= 0; $i--)
    echo $arr[$i] , ' ';
                                        // MySQL PHP JavaScript CSS HTML
```

Приращение может изменять значение счетчика на любую другую величину — например, на 2:

```
for ($i = 0; $i < $arr_cnt; $i += 2)
    echo $arr[$i], ' ';
                                        // HTML JavaScript MySQL
```

В *установке* и *приращении* можно записать несколько выражений, разделив их запятыми:

```
for ($i = 0, $j = 0; $i < $arr_cnt; $i++, $j += 2)
    . . .
```

И *установка*, и *условие*, и *приращение* не являются обязательными для указания (однако символы точки с запятой в заголовке цикла указывать обязательно):

- можно опустить *установку* — тогда начальное значение счетчику следует присвоить перед выполнением цикла:

```
$i = 0;
for (; $i < $arr_cnt; $i++)
    echo $arr[$i], ' ';
```

- можно опустить *условие* — тогда придется производить необходимую проверку в начале *тела цикла* и прерывать выполнение цикла оператором `break`, знакомым нам по *разд. 5.3*:

```
for ($i = 0; ; $i++) {
    if ($i >= $arr_cnt)
        break;
    echo $arr[$i], ' ';
}
```

- можно не указывать *приращение* — тогда его придется выполнять в конце *тела цикла*:

```
for ($i = 0; $i < $arr_cnt;) {
    echo $arr[$i], ' ';
    $i++;
}
```

5.4.2. Цикл с предусловием

Цикл с предусловием выполняется, пока остается истинным заданное *условие*. *Условие* вычисляется перед *каждой* итерацией цикла (рис. 5.2).

Формат записи:

```
while (<условие>)
    <тело цикла>          // Выражение или блок
```

Пример:

```
$i = 0;
while ($i < $arr_cnt) {
    echo $arr[$i], ' ';
    $i++;
}
```

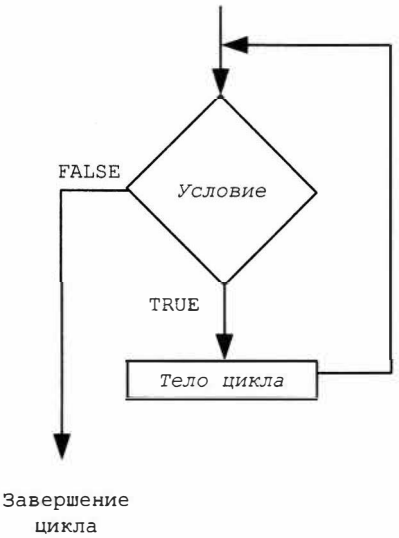


Рис. 5.2. Алгоритм выполнения цикла с предусловием

5.4.3. Цикл с постусловием

Цикл с постусловием аналогичен циклу с предусловием (см. *разд. 5.4.2*), только *условие* вычисляется *после* каждой итерации (рис. 5.3). Такой цикл выполнится хотя бы раз, даже если условие изначально ложно.

Формат записи:

```
do
    <тело цикла>          // Выражение или блок
while (<условие>)
```

Пример:

```
$i = $arr_cnt - 1;
do {
    echo $arr[$i], ' ';
    $i--;
} while ($i >= 0);
```

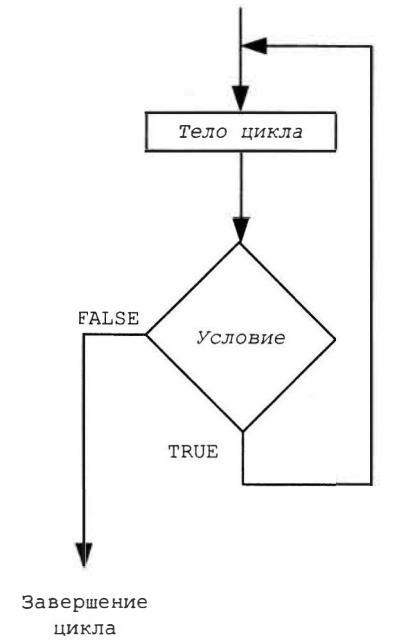


Рис. 5.3. Алгоритм выполнения цикла с постусловием

5.4.4. Цикл по массиву

Цикл по массиву предназначен для перебора элементов заданного *массива*.

Полный формат написания:

```
foreach (<массив> as <переменная-индекс> => <переменная-элемент>)
    <тело цикла>
```

На каждой итерации цикла индекс (или ключ) очередного элемента *массива* будет присвоен *переменной-индексу*, а значение элемента — *переменной-элементу*. Обе эти переменные могут быть использованы в *теле цикла*.

Пример:

```
foreach ($arr as $index => $val)
    echo $index, ': ', $val, '   ';
                  // 0: HTML   1: CSS   2: JavaScript   3: PHP   4: MySQL
```

Поддерживается также сокращенный формат написания:

```
foreach (<массив> as <переменная-элемент>)
    <тело цикла>
```

В этом случае индекс (ключ) очередного элемента массива станет недоступным *в теле цикла*, однако такой цикл выполнится быстрее.

Пример:

```
foreach ($arr as $val)
    echo $val, ' ';
```

5.4.5. Прерывание цикла

Для немедленного прерывания цикла можно применять оператор `break`, описанный в *разд. 5.3*.

Например, показанный далее цикл прервет выполнение, если длина очередного элемента массива `$arr` превысит 7 символов:

```
for ($i = 0; $i < $arr_cnt; $i++) {
    if (strlen($arr[$i]) > 7)
        break;
    echo $arr[$i], ' ';
}
                                                    // HTML CSS
```

Оператор `break` может принимать один целочисленный операнд.

Его формат в таком случае:

```
break <количество прерываемых вложенных друг в друга управляющих конструкций>
```

Если операнд не указан, будет прервана только текущая управляющая конструкция — цикл или выражение выбора.

Примеры использования оператора `break` с операндом:

```
// Внешний цикл
for ( . . . ) {
    . . .
    // Вложенный цикл
    for ( . . . ) {
        if ( . . . )
            // Будет прерван только один цикл — вложенный
            break;
    }
}

// Внешний цикл
for ( . . . ) {
    . . .
    // Вложенный цикл
    for ( . . . ) {
        if ( . . . )
            // Будут прерваны два цикла — и вложенный, и внешний
            break 2;
    }
}
```

5.4.6. Прерывание текущей итерации цикла

> Возможно прервать текущую итерацию цикла, после чего сразу же начнет исполняться следующая итерация (разумеется, если *условие* цикла все еще истинно). Прерывание итерации производится оператором `continue`.

Пример цикла, который выведет только те элемента массива `$arr`, чья длина превышает 4 символа:

```
for ($i = 0; $i < $arr_cnt; $i++) {
    if (strlen($arr[$i]) <= 4)
        continue;
    echo $arr[$i], ' ';
}
                                                    // JavaScript MySQL
```

Аналогично оператору `break` (см. ранее), оператор `continue` может принимать один целочисленный операнд:

```
continue <количество вложенных друг в друга циклов, в которых будет выполнено прерывание итерации>
```

Если операнд не указан, итерация будет прервана только в текущем цикле.

5.5. Безусловный переход

> При *безусловном переходе* выполнение кода переносится на выражение, помеченное особой *меткой*.

Метка записывается на отдельной строке в формате *<имя метки>*:. *Имя метки* должно удовлетворять тем же правилам, что и имя переменной (см. *разд. 3.1.1*).

Сам безусловный переход выполняется оператором `goto`, записываемым в формате:

```
goto <имя целевой метки>
```

Пример реализации посредством безусловного перехода программной конструкции, работающий аналогично циклу:

```
$i = 0;
start:
echo $arr[$i], ' ';
$i++;
if ($i < $arr_cnt)
    goto start;
```

5.6. Завершение работы модуля

Для принудительного завершения работы модуля применяется функция `exit`:

```
exit([<текст, который будет выведен перед завершением>])
```

Если *текст* не указан, ничего выведено не будет.

Пример:

```
if (all_work_done)
    // Если вся работа выполнена, завершаем работу модуля
    exit();
```

Аналогичное действие выполняет функция `die`.

5.7. Упражнение. Заставляем приложение convertor3.php соблюдать правила русского языка

Написанное на *уроке 1* веб-приложение convertor3.php пока что не «умеет» согласовывать имена числительных по правилам русского языка. Пора «научить» его грамматике, хотя бы в отношении дюймов (с которыми дело будет обстоять сложнее, поскольку пользователь может ввести величину в дюймах в виде вещественного числа, а сантиметры у нас всегда целочисленные).

Мы будем следовать такому алгоритму:

- преобразуем величину в дюймах в строковый вид;
- проверим, есть ли в ней десятичная точка (то есть является ли величина дробной) или является ли ее предпоследний символ цифрой «1» (на тот случай, если это число 11, 12, 111 и т. п.):
  - если это так — выведем слово **дюймов**;
  - если нет, извлечем последний символ величины в дюймах:
    - если это «1» — выведем слово **дюйм**;
    - если «2», «3» или «4» — слово **дюйма**;
    - в противном случае — слово **дюймов**.

Файл convertor3.php находится в папке 2\2.8\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Откроем файл convertor3.php в текстовом редакторе и вставим под выражением, вычисляющим величину в сантиметрах, выражение, явно преобразующее результат в строку и присваивающее его переменной `$is` (добавления и правки в написанном ранее коде здесь и далее выделены полужирным шрифтом)

```
$cents = round($ins * 2.54);
$is = (string)$ins;
```

2. Добавим под только что написанным кодом условное выражение, проверяющее, есть ли в полученной ранее строке десятичная точка или является ли ее предпоследний символ цифрой «1», и, если это так, заносящее в переменную `$ins_ending` строку `'ов'`:

```
$is = (string)$ins;
if (strpos($is, '.') !== FALSE || (strlen($is) > 1 && $is[-2] == '1'))
    $ins_ending = 'ов';
else {
}
```

   Перед проверкой, является ли предпоследний символ величины в дюймах цифрой «1», следует убедиться, что длина этой величины больше одного символа. В противном случае, попытавшись извлечь несуществующий второй с конца символ, мы получим ошибку.

   В переменной `$ins_ending` мы будем хранить лишь падежные окончания (например: `ов`), а не целые слова, чтобы сэкономить оперативную память.

3. Вставим между фигурными скобками после языковой конструкции `else` выражение, получающее последний символ величины в дюймах и присваивающее его той же переменной `$is`:

```
if (strpos($is, '.') !== FALSE || (strlen($is) > 1 && $is[-2] == '1'))
    . . .
else {
    $is = $is[-1];
}
```

4. Добавим к нему множественное условное выражение, выясняющее, что за символ хранится в переменной `$is`, и заносящее в переменную `$ins_ending` соответствующее падежное окончание:

```
if (strpos($is, '.') !== FALSE || (strlen($is) > 1 && $is[-2] == '1'))
    . . .
else {
    $is = $is[-1];
    if ($is == 1)
        $ins_ending = '';
    else if ($is >= 2 && $is <= 4)
        $ins_ending = 'а';
    else
        $ins_ending = 'ов';
}
```

5. Найдем выражение, выводящее результат, и перепишем заново следующим образом:

```
echo "<p>{$ins} дюйм{$ins_ending} = {$cents} сантиметров.</p>";
```

   Мы здесь воспользовались тем, что в строках, взятых в двойные кавычки, выполняется обработка переменных (подробно об этом рассказано в *разд. 2.2.2*). Получилось заметно нагляднее, чем ранее.

Запустим приложение, перейдя по интернет-адресу: **http://localhost/convertor3.php**. Введем величину, например 4 дюйма, нажмем кнопку **Преобразовать** и посмотрим, что получилось (рис. 5.4).

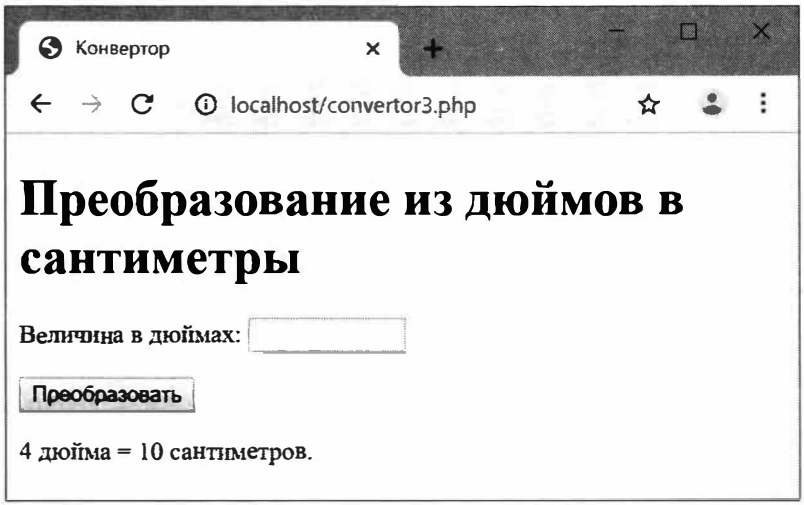


Рис. 5.4. Веб-приложение convertor3.php теперь соблюдает правила русского языка, хотя бы отчасти

Сделайте сами

Заставьте приложение convertor3.php соблюдать правила согласования числительных и в отношении сантиметров. Это будет проще, так как сантиметры у нас всегда целочисленные.

5.8. Упражнение. Реализуем единую точку входа на веб-сайт

5.8.1. Теоретическое обоснование

Наш сайт фотогалереи сейчас состоит из двух серверных веб-приложений: index.php — генерирующего страницу со списком изображений, и item.php — выдающего страницу с выбранным изображением. Однако рациональнее сделать так, чтобы все страницы сайта генерировались одним приложением.

> *Единая точка входа* на сайт — единственное веб-приложение, генерирующее все страницы сайта. Любой запрос, отправленный сайту, вызывает запуск этого приложения, которое обрабатывает поступивший запрос и запускает тот или иной программный модуль, собственно и выдающий требуемую страницу.

Какой именно программный модуль требуется запустить, единая точка входа выясняет, проанализировав путь, указанный в составе интернет-адреса из полученного запроса (на следующем примере этот путь помечен рамкой):

http://localhost/item.php?index=2

> *Маршрутизация* — анализ пути, извлеченного из полученного в запросе интернет-адреса, с целью определить, какую результирующую страницу следует сгенерировать.

> *Маршрутизатор* — программный модуль, выполняющий маршрутизацию.

Использование единой точки входа имеет следующие преимущества:

- код, выполняющий какие-либо подготовительные и завершающие операции (включение библиотек, создание переменных, установку и последующий разрыв соединения с базой данных и др.), записывается только в одном месте — в само́й единой точке входа, что упрощает программирование и сопровождение.

  В противном случае его пришлось бы писать во всех приложениях, составляющих сайт, а это трудоемко и чревато ошибками;
- можно установить свой собственный формат путей для вывода тех или иных страниц. Например, в нашем сайте для вывода третьей по счету картинки вместо громоздкого и неуклюжего интернет-адреса: **http://localhost/item.php?index=2** мы могли бы набирать более короткий и понятный: **http://localhost/2/**;
- можно структурировать код, разделив его отдельные части, выполняющие какие-либо законченные действия, на независимые программные модули. Это заметно упростит последующее программирование и сопровождение сайта.

В нашем случае можно вынести в отдельные модули код, выполняющий подготовительные задачи, код маршрутизатора, код, выводящий страницы, и т. п.

Для того чтобы при получении любого запроса запускалось одно и то же веб-приложение (выполнялось перенаправление на это приложение), веб-сервер должен быть соответственно сконфигурирован. Нужные настройки указываются в файле локальной конфигурации .htaccess (точка в начале имени файла обязательна!), который помещается в корневую папку сайта.

5.8.2. Собственно упражнение

Перепишем наш сайт фотогалереи таким образом, чтобы он использовал единую точку входа.

Файлы этого сайта можно найти в папке 4\4.7\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. В папке 5\!sources найдем файл локальной конфигурации .htaccess и скопируем его в корневую папку сайта.

   Откроем копию этого файла и посмотрим на хранящийся там конфигурационный код (листинг 5.1).

Листинг 5.1. Файл .htaccess

```
RewriteEngine On
RewriteCond %{SCRIPT_FILENAME} !-f
RewriteRule ^(.*)$ ./index.php?route=$1 [L,QSA]
```

   Первая строка активизирует встроенный в Apache механизм перенаправлений. Вторая строка указывает, чтобы запросы к конкретным файлам не перенаправлялись (благодаря чему запросы на загрузку файлов с таблицей стилей и изображениями будут успешно обрабатываться). Третья строка предписывает перенаправить любой другой запрос на единую точку входа — веб-приложение index.php, передав ему через GET-параметр `route` извлеченный из интернет-адреса путь. Все GET-параметры, присутствующие в полученном в запросе интернет-адресе, также будут переданы приложению index.php — за это «отвечает» языковая конструкция `[L,QSA]`. Отметим, что путь передается без начального символа слеша.

   Единую точку входа index.php мы скоро напишем. Но сначала подготовим почву...

2. Переименуем имеющийся файл index.php (это веб-приложение, выдающее список картинок) в list.php.

3. Переместим файлы list.php и item.php в папку modules.

   Ранее мы указывали в языковых конструкциях `require` и `require_once` относительные пути к включаемым файлам, отсчитываемые относительно пути вклю-

чающего файла. Поскольку мы позже «разложим» все файлы сайта по разным папкам, удобнее записывать в этих директивах абсолютные пути. А чтобы указать абсолютные пути, нам понадобится полный путь к корневой папке сайта.

Этот путь мы вычислим в единой точке входа index.php — так проще. Условимся сохранить его в переменной `$base_path`.

И пора бы структурировать код сайта, разнеся его по разным модулям. Там нам будет проще в дальнейшем.

4. Напишем модуль-маршрутизатор для нашего сайта. Он будет обрабатывать получаемые в составе запросов пути двух видов:
   - пустая строка — обращение к главной странице сайта, на которой выводятся миниатюры;
   - ***<индекс изображения>/*** — обращение к странице вывода изображения с указанным *индексом*.

   Создадим в папке modules модуль-маршрутизатор router.php (его код показан в листинге 5.2).

Листинг 5.2. Модуль modules\router.php

```
<?php
$request_path = $_GET['route'];
if ($request_path == '')
    require $base_path . 'modules\list.php';
else {
    $index = (integer)$request_path;
    require $base_path . 'modules\item.php';
}
```

Сам путь мы извлекаем из GET-параметра `route`. Если он не пуст, мы можем преобразовать его к целочисленному типу, получив тем самым индекс нужного изображения, который сохраним в переменной `$index` для дальнейшего использования.

5. Настала пора единой точки входа. Она будет вычислять путь к корневой папке сайта и включать написанные ранее модули modules\data.php и modules\router.php.

   Создадим в корневой папке сайта модуль index.php, который и станет единой точкой входа (листинг 5.3).

Листинг 5.3. Модуль index.php

```
<?php
$base_path = __DIR__ . '\\';
require_once $base_path . 'modules\data.php';
require_once $base_path . 'modules\router.php';
```

Константа `__DIR__`, создаваемая исполняющей средой PHP, хранит полный путь к папке, где хранится текущий модель, в нашем случае — путь к корневой папке сайта. Он не включает конечного разделителя (в Windows — обратный слеш \), поэтому мы добавляем таковой (обязательно продублировав, иначе PHP посчитает его и следующую за ним одинарную кавычку, завершающую строку, литералом `\'`, и мы получим фатальную ошибку).

6. Откроем модуль modules\list.php и внесем в него следующие исправления (добавления и правки в уже написанном коде здесь и далее помечены полужирным шрифтом, удаленный код зачеркнут):

```
<?php
    ~~require_once 'modules\data.php';~~
    . . .
?>
. . .
<a href="/<?php echo $i ?>/">
    <img . . . >
</a>
```

Мы удалили включение модуля modules\data.php (поскольку теперь оно выполняется в единой точке входа) и исправили интернет-адрес в теге `<a>` соответственно формату ***/<индекс изображения>/***.

7. Откроем модуль modules\item.php и внесем правки в самый первый PHP-фрагмент:

```
<?php
    ~~require_once 'modules\data.php';~~
    $item = $arr_images[$index];
?>
```

Здесь мы также удалили включение модуля modules\data.php. А индекс теперь будем брать не из GET-параметра `index`, а из переменной `$index`, созданной в модуле modules\router.php.

Проверим сайт в действии, выполнив переход по интернет-адресу: **http://localhost/**. Щелкнем на какой-либо миниатюре, чтобы перейти на страницу показа изображения, и вновь вернемся на главную страницу.

Урок 6
Функции

Функция выполняет над переданными ей параметрами какое-либо сложное действие. При этом она может возвращать результат.

PHP поддерживает много встроенных функций, выполняющих математические операции, обработку строк, манипуляции с файлами и др. Программист также может написать свои функции, тем самым расширив функциональность платформы.

6.1. Объявление и вызов функций

> *Объявление функции* — описание функции, включая ее имя и набор принимаемых ею параметров (если такие есть).

Формат объявления функции:

```
function <имя функции>([<список имен принимаемых параметров через запятую>]) {
    <тело функции>                    // Исполняемый код функции
}
```

- *Имя функции* должно удовлетворять тем же требованиям, что и имя переменной (см. *разд. 3.1.1*).
- Имена функций *нечувствительны* к регистру символов.
- Для каждого из параметров, чье *имя* указано в круглых скобках, создается переменная с тем же *именем*, доступная только в *теле функции*. Из этой переменной можно получить значение переданного функции параметра.

Для возврата результата из функции применяется оператор `return`, записываемый в формате:

```
return <возвращаемое значение>;
```

Исполнение этого оператора прерывает работу функции.

Пример функции, переводящей дюймы в сантиметры:

```
function ins_to_cents($ins) {
    $cents = $ins * 2.54;
    return $cents;
}
```

> Вызов функции, объявленной программистом, выполняется в том же формате, что и вызов встроенной функции:
>
> `<имя функции>([<список значений параметров через запятую>])`

Пример:

```
echo ins_to_cents(1);                    // 2.54
echo ins_to_cents(123);                  // 312.42
```

Объявление функции может располагаться в любом месте программного кода, даже после выражений, в которых вызывается эта функция. Однако традиционно объявления всех функций, используемых в программном модуле, записывают в его начале.

Полезно знать...

- Функции подготавливаются к выполнению во время компиляции программного модуля. Именно поэтому вызов функции может располагаться перед ее объявлением — исполняющая среда уже «знает», где искать это объявление.
- В старых версиях PHP, не компилировавших модуль перед исполнением, объявление функции обязательно должно было предшествовать ее первому вызову.

6.1.1. Область видимости функций

> Будучи объявленной, функция, как и переменная, доступна во всем программном модуле, во всех включаемых программных модулях и в модуле, выполнившем включение текущего модуля.

Примеры:

```
// Используем функцию еще до ее объявления
$cents = ins_to_cents(321);
// Объявляем функцию
function ins_to_cents($ins) { . . . }
// Здесь функция тоже доступна
$cents2 = ins_to_cents(45);

// Пример с включаемым модулем
// Модуль module.php
$ins = cents_to_ins(34);
function ins_to_cents($ins) { . . . }

// Модуль main.php
function cents_to_ins($cents) { . . . }
require 'module.php';
$cents = ins_to_cents(12);
```

PHP позволяет объявить одну функцию внутри другой.

> *Вложенная функция* — функция, объявленная в теле другой функции.

Вложенная функция имеет ту же область видимости, что и обычная, но вызвать ее можно *только после вызова «внешней» функции*, в которой она объявлена.

Пример:

```
function outer() {
    // Объявляем вложенную функцию
    function inner() { . . . }

    . . .
}
// Вызываем «внешнюю» функцию
outer();
// Теперь можно вызвать вложенную функцию
inner();
```

Полезно знать...

В других С-подобных языках программирования (например, в JavaScript) вложенные функции доступны только в тех функциях, в которых они объявлены.

6.1.2. Локальные, глобальные и статические переменные

В функциях можно создавать переменные для оперативного хранения каких-либо значений.

> *Локальная переменная* — переменная, созданная в теле функции. Доступна только в теле функции и после ее выполнения уничтожается.

Пример:

```
function ins_to_cents($ins) {
    $cents = $ins * 2.54;
    return $cents;
}
// Вызываем функцию
echo ins_to_cents(1);                                    // 2.54
// Проверяем, существует ли локальная переменная $cents, созданная
// в теле функции ins_to_cents
echo isset($cents);                                      // FALSE
```

> *Глобальная переменная* — переменная, созданная вне функции.

Глобальные переменные в теле функции недоступны. При попытке обратиться к глобальной переменной в теле функции PHP выполнит обращение к одноименной локальной переменной.

Пример:

```
$var = 0;
function func() {
    // В функции пытаемся присвоить переменной $var новое значение.
    // Будет создана локальная переменная с тем же именем — $var.
    $var = 10000;
}
```

```
// Проверяем значение переменной $var
echo $var;                                                    // 0
// Вызываем функцию func
func();
// Снова проверяем значение переменной $var — оно не изменилось
echo $var;                                                    // 0
```

> **Чтобы в теле функции обращаться к глобальным переменным, их следует явно объявить как глобальные, применив языковую конструкцию** `global`:
>
> `global <`*список имен глобальных переменных через запятую*`>;`

Пример:

```
$var2 = 0;
function func2() {
    global $var2;
    $var2 = 10000;
}
echo $var2;                                                   // 0
func2();
echo $var2;                                                   // 10000
```

К глобальным переменным также можно обратиться через элементы ассоциативного массива, хранящегося в суперглобальной переменной `$GLOBALS` (суперглобальные переменные описывались в *разд. 3.1.2*). Ключи элементов этого массива совпадают с именами глобальных переменных, указанными без знака доллара:

```
function func2() {
    $GLOBALS['var2'] = 10000;
}
```

> *Статическая переменная* — локальная переменная, не уничтожающаяся и сохраняющая свое значение после завершения работы функции.

Статическая переменная создается с применением языковой конструкции `static`:

```
static <имя создаваемой переменной> = <начальное значение>;
```

Начальное значение будет присвоено статической переменной после создания.

Пример:

```
function counter() {
    static $cnt = 0;
    return $cnt++;
}
echo counter();                                               // 0
echo counter();                                               // 1
echo counter();                                               // 2
```

6.1.3. Указание типов для параметров и возвращаемого результата

Если функция должна принимать параметры строго определенного типа (например, вещественного или строкового), мы можем указать тип этих параметров при объявлении функции.

> *Тип параметра* указывается вместе с его именем в формате:
> *<обозначение типа данных> <имя параметра>*

Поддерживаются следующие обозначения типов:

- `int` — целочисленный;
- `float` — вещественный;
- `string` — строковый;
- `boolean` — логический;
- `array` — массив;
- `object` — объект;
- *<имя класса или интерфейса>* — объект указанного *класса* или класса, реализующего заданный *интерфейс* (о классах будет рассказано на *уроке 7*, об интерфейсах — на *уроке 8*);
- `self` — объект того же класса, в котором объявлен текущий метод (применяется только в методах класса, о чем рассказывается на *уроке 7*).

Пример:

```
function ins_to_cents(float $ins) {
    $cents = $ins * 2.54;
    return $cents;
}
echo ins_to_cents(2);                                    // 5.08
```

> Если при вызове функции параметр получил значение, относящееся к другому типу, PHP преобразует его в нужный тип.

Если преобразование типов выполнить не удается (например, вместо вещественного параметра был указан массив), возникает фатальная ошибка с генерированием исключения `TypeError` (исключениям посвящен *урок 12*).

Пример:

```
echo ins_to_cents('2');                                  // 5.08
echo ins_to_cents([2]);                                  // Ошибка!
```

Аналогично можно указать тип возвращаемого функцией результата — следующим образом (выделен рамкой):

```
function <имя функции>( . . . ): <обозначение типа данных> { ... }
```

Если функция возвращает результат другого типа, PHP преобразует его к типу, указанному в объявлении функции. В случае невозможности выполнить преобразование типов возникает фатальная ошибка с генерированием исключения `TypeError`.

Пример:

```
function ins_to_cents(float $ins): int {
    $cents = $ins * 2.54;
    return $cents;
}
echo ins_to_cents(2);                                    // 5
```

6.2. Параметры функций: особые случаи

6.2.1. Необязательные параметры

> *Необязательный параметр* при вызове функции можно не указывать — в этом случае он получит значение по умолчанию, заданное в объявлении функции.

Необязательный параметр объявляется в формате:

```
<имя параметра> = <значение по умолчанию>
```

Пример функции с необязательным вторым параметром:

```
function get_tag(string $text, string $tag = 'p'): string {
    return "<{$tag}>{$text}</{$tag}>";
}
echo get_tag('Привет!', 'h1');                           // <h1>Привет!</h1>
echo get_tag('Пока!');                                   // <p>Пока!</p>
```

В функции может быть сколько угодно необязательных параметров.

> Все необязательные параметры должны располагаться в самом конце списка принимаемых функцией параметров. В противном случае PHP не сможет «понять», каким параметрам соответствуют указанные при вызове функции значения, и мы, скорее всего, получим ошибку.

6.2.2. Функции с произвольным количеством параметров

Если функция должна принимать произвольное количество параметров, в ее объявлении необходимо:

- записать параметры, которые обязательно должны быть указаны;
- после них, через запятую, записать конструкцию вида (три точки в начале обязательны):

  `...<имя переменной>`

Все остальные параметры, которых может быть произвольное количество, будут помещены в массив, который PHP присвоит локальной переменной с указанным *именем*.

Пример функции, принимающей два параметра, обязательных к указанию, и произвольное количество необязательных параметров:

```
function get_tag2(string $text, string $tag, ...$classes): string {
    $s = implode(' ', $classes);
    if ($s)
        $s = ' class="' . $s . '"';
    return "<{$tag}{$s}>{$text}</{$tag}>";
}
echo get_tag2('Привет!', 'h1');                          // <h1>Привет!</h1>
echo get_tag2('Привет!', 'h1', 'greeting', 'header');
                                // <h1 class="greeting header">Привет!</h1>
```

6.2.3. Параметры с изменяемыми значениями. Передача по ссылке

> Если в теле функции изменить значение какого-либо из полученных ей параметров, вне функции это значение останется прежним. Это происходит потому, что в функцию передаются копии значений параметров (*передача по значению*).

Пример:

```
$s = 'PHP';
function add_string($str) {
    $str .= ' и MySQL';
}
add_string($s);
echo $s;                                                 // PHP
```

> Чтобы разрешить функции менять значение какого-либо параметра, следует *передать его по ссылке*, предварив имя параметра, указанное в объявлении функции, символом & (амперсанд).

В качестве значений параметров, передаваемых по ссылке, можно указывать лишь переменные.

Пример:

```
$s = 'PHP';
function add_string2(&$str) {
    $str .= ' и MySQL';
}
add_string2($s);
echo $s;                                                 // PHP и MySQL
```

> Все здесь сказанное касается только величин, относящихся к значащим типам: целочисленному, вещественному, строковому, логическому, а также к массивам. Величины ссылочных типов (о которых речь пойдет на *уроке 7*) могут передаваться по значению.

Полезно знать...

- По умолчанию значение параметра, указанного при вызове функции, *копируется* в локальную переменную, неявно созданную PHP для хранения этого параметра. Это и есть *передача параметра по значению*.
- При передаче параметра по ссылке локальная переменная получит в качестве значения ссылку на переменную, хранящую значение параметра. Это и есть *передача по ссылке*.

6.3. Переменные функций

> *Переменная функции* — переменная, хранящая имя функции в виде строки и используемая для вызова этой функции.

Вызов функции, хранящейся в переменной функции, выполняется в следующем формате:

```
<имя переменной функции>([<список значений параметров через запятую>])
```

Пример:

```
function ins_to_cents($ins) {
    return $ins * 2.54;
}
function cents_to_ins($cents) {
    return $cents / 2.54;
}
$func = 'ins_to_cents';
echo $func(100);                                    // 254
$func = 'cents_to_ins';
echo $func(254);                                    // 100
```

6.4. Анонимные функции

> *Анонимная функция* — функция, не имеющая имени. Объявляется так же, как обычная (*именованная*) функция (см. *разд. 6.1*), но без указания имени.

Анонимную функцию сразу после объявления необходимо присвоить какой-либо переменной.

Объявление анонимной функции должно завершаться символом точки с запятой.

Пример анонимной функции:

```
$itc = function ($ins) {
    $cents = $ins * 2.54;
    return $cents;
};
```

> Вызов анонимной функции выполняется через переменную, которой она присвоена (как и вызов функции через переменную функции).

Пример:

```
echo $itc(12);                                    // 30.48
```

Переменная с анонимной функцией может быть передана в качестве параметра другой функции:

```
otherfunc(4, 'дюйм', $itc);
```

Анонимную функцию можно передать в качестве параметра и непосредственно:

```
otherfunc(4, 'дюйм', function ($ins) {
    $cents = $ins * 2.54;
    return $cents;
});
```

Анонимные функции могут использовать глобальные переменные:

```
$var = 0;
$func = function () {
    global $var;
    $var = 10000;
};
echo $var;                                        // 0
$func();
echo $var;                                        // 10000
```

> Анонимные функции могут наследовать переменные из области видимости, в которой они объявлены.

Так, если функция объявлена вне других функций, она может наследовать глобальную переменную, а если объявлена внутри другой функции — то локальную переменную из тела «внешней» функции.

Наследуемые анонимной функцией переменные записываются в ее объявлении с применением следующего формата:

```
function ( . . . ) use (<список наследуемых переменных через запятую>)
```

Пример:

```
function convert_and_echo($ins) {
    $ic = 2.54;
    // Эта анонимная функция наследует локальную переменную $ic
    // из "внешней" функции
```

```
    $conv = function ($ins) use ($ic) {
        return $ins * $ic;
    };
    $cents = $conv($ins);
    echo "{$ins} дюйм. = {$cents} см.";
}
echo convert_and_echo(12);                    // 12 дюйм. = 30.48 см.
```

Содержимое наследуемых переменных передается в анонимную функцию по значению. Поэтому, если в теле анонимной функции изменить значение какой-либо унаследованной переменной, ее значение вне функции не изменится.

Чтобы разрешить анонимной функции менять значение унаследованной переменной, его следует передать по ссылке.

Пример:

```
$conv = function ($ins) use (&$ic) {
    . . .
};
```

6.5. Рекурсия

Рекурсия — вызов функцией самой себя.

Классический пример рекурсии — функция, вычисляющая факториал числа:

```
function factorial($val) {
    if ($val == 1)
        return 1;
    else
        return $val * factorial($val - 1);
}
```

Если в качестве параметра функция получит `1`, то вернет значение факториала от единицы, также равное `1`. В противном случае она умножит полученное число на факториал числа, меньшего на единицу, который получит, вызвав себя.

Примеры вызова этой функции:

```
echo factorial(1);                            // 1
echo factorial(5);                            // 120
echo factorial(20);                           // 2432902008176640000
```

> При реализации рекурсии следует предусмотреть условие, при котором цикл вызова функцией самой себя прервется. В противном случае возникнет *бесконечная рекурсия*, которая приведет к фатальной ошибке и останову программы.

Полезно знать

При каждом вызове функции из тела другой функции веб-обозреватель заносит сведения об этом вызове в особую область памяти, называемую *стеком вызовов*.

Эта область имеет ограниченный объем — несколько тысяч позиций, и по ее исчерпанию выполнение программы будет прервано с выдачей сообщения о фатальной ошибке.

6.6. Упражнение. Объявляем функцию, определяющую падежное окончание по числу

В *разд. 5.6* мы научили приложение convertor3.php правильно согласовывать имена числительных. Код, определяющий требуемое падежное окончание по числу, — идеальный кандидат на оформление в качестве функции. Объявим эту функцию и дадим ей имя `case_ending`. Она будет принимать единственный параметр — число, по которому нужно определить падежное окончание, которое и станет возвращать в качестве результата.

Файл convertor3.php находится в папке 5\5.7\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Откроем файл convertor3.php в текстовом редакторе и поместим в самое начало кода (перед тегом `<!doctype>`) PHP-код, объявляющий функцию `case_ending`:

```
<?php
function case_ending(string $is): string {
    if (strpos($is, '.') !== FALSE ||
        (strlen($is) > 1 && $is[-2] == '1'))
        return 'ов';
    else {
        $is = $is[-1];
        if ($is == 1)
            return '';
        else if ($is >= 2 && $is <= 4)
            return 'а';
        else
            return 'ов';
    }
}
?>
```

   Указав для параметра строковый тип, мы предпишем PHP сразу же преобразовать полученное число в строковый тип, и нам не придется делать это самостоятельно.

2. Удалим ранее написанный код, определяющий падежное окончание, — он нам более не нужен (удаленный код зачеркнут):

```
$is = (string)$ins;
if (strpos($is, '.') !== FALSE || (strlen($is) > 1 && $is[-2] == '1'))
    $ins_ending = 'ов';
else {
    . . .
}
```

3. Запишем на место удаленного кода два выражения, определяющие падежные окончания для дюймов и сантиметров посредством вызова функции `case_ending` (добавленный и исправленный код здесь и далее выделен полужирным шрифтом):

```
$cents = round($ins * 2.54);
$ins_ending = case_ending($ins);
$cents_ending = case_ending($cents);
echo . . . ;
```

4. Исправим выражение, выводящее результат, так, чтобы оно ставило правильное падежное окончание для сантиметров:

```
echo "<p>{$ins} дюйм{$ins_ending} = {$cents} ↳
сантиметр{$cents_ending}.</p>";
```

Проверим приложение, перейдя по интернет-адресу: **http://localhost/convertor3.php**, и убедимся, что оно работает.

Сделайте сами

- Вынесите объявление функции `case_ending` в отдельный модуль library.php, создав тем самым библиотеку. Не забудьте выполнить ее включение в модуле convertor3.php, иначе приложение перестанет работать.
- Переделайте приложения convertor.php и convertor2.php так, чтобы они правильно согласовывали числительные, применяя библиотеку library.php.

  Файл convertor.php находится в папке 1\1.1\ex, а файл convertor2.php — в папке 1\1.2\ex сопровождающего книгу электронного архива (см. *приложение 3*).

Урок 7
Объектное программирование: классы и объекты

Для представления сложных сущностей в PHP лучше использовать объекты. Это гораздо удобнее, чем манипулировать разрозненным набором переменных, хранящих различные характеристики такой сущности, и функций, манипулирующих этими характеристиками.

7.1. Введение в объектное программирование

> *Объект* — сложная сущность, включающая в свой состав набор переменных, хранящих различные характеристики этой сущности, и функций, выполняющих над ней разные действия.

> *Свойство* — переменная, входящая в состав объекта.

> *Метод* — функция, входящая в состав объекта.

В качестве объекта мы можем представить, например, изображение из фотогалереи. Такой объект может иметь два свойства: хранящие интернет-адрес файла и его описание, и метод, возвращающий HTML-код для вывода изображения на экран.

> *Класс* — образец для создания объектов определенного вида. Задает набор свойств и методов, которые будут поддерживаться всеми объектами, относящимися к этому виду.

Продолжая пример с объектом-изображением, примем в качестве образца для создания таких объектов-изображений, например, класс `Image`. Он может включать свойства: `src` и `description`, хранящие, соответственно, имя файла с изображением и описание к нему, и метод `get_img_tag`, возвращающий HTML-код для вывода изображения.

> Создание объекта на основе заданного *класса* производится оператором `new`, который записывается в формате:
>
> `new <имя класса>([<список значений параметров через запятую>])`

Указываемые в круглых скобках *параметры* передаются объекту при его создании и могут, например, сразу же заноситься в его свойства.

Созданный объект должен быть присвоен какой-либо переменной.

Пример создания двух объектов класса `Image`:

```
// Сразу при создании объекта указываем параметры, которые будут занесены
// в его свойства: имя файла и описание к изображению
$img1 = new Image('1234.png', 'Осенний парк');
// А при создании этого объекта параметры не указываем, в результате чего
// его свойства будут иметь определяемые классом значения по умолчанию
$img2 = new Image();
```

> Обращение к свойству объекта записывается в формате:
>
> *<переменная с объектом>->*<имя свойства>

Имя свойства указывается без начального символа `$`.

Примеры:

```
// Получаем значение свойства src объекта $img1
echo $img1->src;                                    // 1234.png
// Присваиваем тому же свойству объекта $img2 имя файла
// (поскольку не сделали этого при создании объекта)
$img2->src = 'scheme.gif';
// Присваиваем свойству description описание изображения
$img2->description = 'Схема проезда к парку';
```

> Вызов метода объекта записывается в формате:
>
> *<переменная с объектом>->*<имя метода>([*<список значений параметров через запятую>*])

Пример:

```
// Выводим на страницу HTML-код для вставки обоих изображений
echo $img1->get_img_tag();
echo $img2->get_img_tag();
```

Удаление свойств

Можно удалить любое свойство посредством функции `unset` (см. *разд. 3.1.5*):

```
unset($img2->description);
```

> Объекты, даже созданные на основе одного класса, абсолютно независимы друг от друга: их свойства могут хранить разные значения, а методы — выполнять на основе значений этих свойств действия с разным результатом.

PHP предлагает относительно небольшой набор встроенных классов, объекты которых используются в специфических случаях (в частности, при работе с базами данных). Помимо этого, программист может объявить свои классы.

7.2. Объявление классов

> *Объявление класса* — описание класса, включая его имя, набор поддерживаемых свойств и методов, имя суперкласса (о суперклассах и наследовании будет рассказано в *разд. 7.3*) и имена реализуемых интерфейсов (о них разговор пойдет на *уроке 8*).

Формат объявления класса в самом простом случае:

```
class <имя класса> {
    <объявление свойств>
    <объявление методов>
}
```

Имя класса должно удовлетворять тем же требованиям, что и имя переменной (см. *разд. 3.1.1*).

7.2.1. Объявление свойств и методов

> Формат объявления свойства:
>
> `<модификатор> <имя свойства> = <начальное значение свойства>;`
>
> Здесь *имя свойства* записывается с начальным символом `$`.

Поддерживаются следующие *модификаторы*:

- `public` — *общедоступное* свойство. Доступно отовсюду: внутри текущего класса, в производных классах и вне класса;
- `protected` — *защищенное* свойство. Доступно внутри текущего класса и в производных классах, но не вне класса (о производных классах и наследовании речь пойдет в *разд. 7.3*);
- `private` — *закрытое* свойство. Доступно только внутри текущего класса, но не в производных классах и не вне класса.

> Формат объявления метода:
>
> `[<модификатор>] <обычное объявление функции>`

При объявлении метода можно указать один из модификаторов, приведенных для объявления свойства: `public`, `protected` или `private`, — в результате чего метод станет соответственно *общедоступным*, *защищенным* или *закрытым*.

Если *модификатор* не указан, объявляемый метод станет общедоступным (как если бы был использован модификатор `public`).

В теле объявляемого метода будет доступна переменная `$this`, создаваемая самой исполняющей средой PHP и хранящая ссылку на объект, у которого вызван этот метод (*текущий* объект).

Пример объявления класса `Image`:

```
class Image {
    // Свойства src и description — общедоступные
    public $src = '';
    public $description = '';

    // Свойство prefix, хранящее префикс интернет-адреса, — закрытое
    private $prefix = '/images/';
```

```
    // Метод get_url, возвращающий полный интернет-адрес файла
    // с изображением, — защищенный
    protected function get_url() {
        // Используем переменную $this, чтобы обратиться к текущему
        // объекту, получить значения его свойств и вызвать его методы
        return $this->prefix . $this->src;
    }
    // Поскольку модификатор доступа не указан, метод get_img_tag станет
    // общедоступным
    function get_img_tag() {
        $url = $this->get_url();
        return "<img src=\"{$url}\" title=\"{$this->description}\">";
    }
}
```

Примеры использования класса `Image`:

```
// На данном этапе указать параметры при создании объекта еще нельзя
// (но мы это скоро исправим)
$img = new Image();
// Обращаемся к общедоступным свойствам и вызываем общедоступные методы —
// успешно
$img->src = '1234.png';
$img->description = 'Осенний парк';
echo $img->get_img_tag();
                    // <img src="/images/1234.png" title="Осенний парк">
// Пытаемся обратиться к закрытому свойству — неуспешно
$img->prefix;                                          // Ошибка!
// Пытаемся обратиться к защищенному методу — неуспешно
$img->get_url();                                       // Ошибка!
```

В закрытых свойствах удобно хранить значения, которые должны использоваться и изменяться только текущим объектом и к которым не должен иметь доступ любой внешний по отношению к текущему объекту код. Аналогично, закрытыми следует делать методы, которые вызываются исключительно внутри текущего объекта, и доступ к которым извне нежелателен.

7.2.2. Конструкторы и деструкторы

> *Конструктор* — метод класса, выполняемый при создании каждого объекта этого класса.

Конструктор должен иметь имя `__construct`, быть общедоступным и не должен возвращать результата.

Все параметры, указанные при создании объекта оператором `new` (см. *разд. 7.1*), передаются конструктору. Как правило, значения этих параметров присваиваются свойствам создаваемого объекта.

Пример объявления в классе `Image` конструктора с двумя необязательными параметрами, через которые передаются имя файла и описание к изображению:

```
class Image {
    . . .
    function __construct(string $src = '', string $description = '') {
        $this->src = $src;
        $this->description = $description;
    }
}
```

Теперь мы можем указать имя файла и описание непосредственно при создании объекта этого класса:

```
$img = new Image('1234.png', 'Осенний парк');
echo $img->get_img_tag();
                    // <img src="/images/1234.png" title="Осенний парк">
```

> *Деструктор* — метод класса, выполняемый при уничтожении текущего объекта.

Деструктор должен иметь имя `__destruct`, быть общедоступным, не должен принимать параметров и возвращать результат.

Пример деструктора:

```
class Image {
    . . .
    function __destruct() {
        echo 'Сейчас объект класса Image будет уничтожен';
    }
}
```

В деструкторах выполняют различные завершающие действия: закрытие открытых файлов, разрыв соединений с базами данных и т. п.

7.2.3. Статические свойства и методы

> *Статическое свойство* — принадлежит не объекту, а классу.

Формат объявления статического свойства:

```
<модификатор> static <имя свойства> = <начальное значение свойства>;
```

Формат обращения к статическому свойству:

```
<имя класса>::<имя свойства>
```

Имя свойства указывается с начальным символом `$`.

Статические свойства применяются для хранения значений, относящихся ко всем объектам класса.

> *Статический метод* — также принадлежит классу.

Формат объявления статического метода:

```
[<модификатор>] static <обычное объявление функции>
```

Формат вызова статического метода:

```
<имя класса>::<имя метода>([<список значений параметров через запятую>])
```

Обратиться к статическому свойству или методу класса из статического метода того же класса можно, указав в качестве *имени класса* языковую конструкцию `self` (без начального знака `$`).

Статические методы применяются для выполнения каких-либо действий над отвлеченными значениями, а также для создания объектов класса с выполнением каких-либо дополнительных действий.

Пример класса `HTMLHelper` со статическим свойством `tag_name` (хранит имя HTML-тега) и статическим методом `get_tag` (принимает в качестве параметра строку и возвращает ее же, заключенную в тег, чье имя указано в свойстве `tag_name`):

```
class HTMLHelper {
    public static $tag_name = 'p';

    public static function get_tag(string $content): string {
        return '<' . self::$tag_name . ">{$content}</" .
            self::$tag_name . '>';
    }
}
```

Пример использования этого класса:

```
// Вызываем статический метод get_tag
echo HTMLHelper::get_tag('Привет, мир!');          // <p>Привет, мир!</p>
// Заносим новое значение в свойство tag_name
HTMLHelper::$tag_name = 'div';
echo HTMLHelper::get_tag('Привет, мир!');     // <div>Привет, мир!</div>
```

7.2.4. Константы классов

> *Константа класса* — в отличие от обычной константы (см. *разд. 3.3*) принадлежит классу (как и статическое свойство или метод).

Формат создания константы класса:

```
[<модификатор>] const <имя константы> = <значение константы>;
```

Если *модификатор* не указан, константа станет общедоступной (`public`).

Обращение к константе класса записывается так же, как обращение к статическому свойству, только имя константы пишется без начального символа `$`.

Пример объявления в классе `HTMLHelper` константы `HEADER1_TAG_NAME` (хранит имя тега `<h1>`) и статического метода `get_header1_tag`, обращающегося к этой константе:

```
class HTMLHelper {
    . . .
    public const HEADER1_TAG_NAME = 'h1';
```

```
    public static function get_header1_tag(string $content): string {
        return '<' . self::HEADER1_TAG_NAME . ">{$content}</" .
            self::HEADER1_TAG_NAME . '>';
    }
}
```

Пример обращения к константе и вызова метода:

```
echo HTMLHelper::HEADER1_TAG_NAME;                    // h1
echo HTMLHelper::get_header1_tag('Константы классов');
                                          // <h1>Константы классов</h1>
```

7.3. Наследование классов

> *Наследование* — создание одного класса на основе другого. Наследующий класс (*производный*, или *подкласс*) получает все свойства, методы, как обычные, так и статические, а также константы того, от которого он наследован (*базового*, или *суперкласса*). Также производный класс может объявить свои свойства, методы и константы.

Класс может быть производным только от одного базового класса.

Объявление производного класса записывается в формате:

```
class <имя производного класса> extends <имя базового класса> {
    <объявление свойств производного класса>
    <объявление методов производного класса>
}
```

Пример объявления класса `Image2`, производного от `Image`, с методом `get_tag_with_bg` (принимает в качестве параметров имя и текстовое содержимое тега и возвращает HTML-код, создающий элемент на основе указанного тега с графическим фоном):

```
class Image2 extends Image {
    public function get_tag_with_bg(string $tag_name, string $content):
        string {
        // Защищенный метод get_url доступен и в производных классах,
        // так что мы можем его вызвать
        $url = $this->get_url();
        return "<{$tag_name} style=\"background-image: url({$url});\">" .
            "{$content}</{$tag_name}>";
    }
}
```

Применение этого класса:

```
$img = new Image2('1234.png', 'Осенний парк');
// Вызываем метод get_img_tag, унаследованный у базового класса Image
echo $img->get_img_tag();
                    // <img src="/images/1234.png" title="Осенний парк">
```

```
// Вызываем метод get_tag_with_bg, объявленный в классе Image2
echo $img->get_tag_with_bg('div', 'Элемент с фоном');
// <div style="background-image: url(/images/1234.png);">Элемент с
// фоном</div>
```

> Производный класс наследует только общедоступные и защищенные свойства и методы. Закрытые свойства и методы остаются «внутри» базового класса.

Полезно знать...

В других языках программирования — в частности, в C++ и Python, производный класс может наследоваться сразу от нескольких базовых классов.

7.3.1. Перекрытие и переопределение методов

> *Перекрытие метода* — объявление в производном классе метода, одноименного с методом, унаследованным от базового класса. При этом метод производного класса заменит (*перекроет*) метод базового класса.

Пример:

```
class A {
    function method() {
        echo 'Класс A';
    }
}

class B extends A {
    // Перекрываем метод method, объявленный в базовом классе
    function method() {
        echo 'Класс B';
    }
}

$objA = new A();
$objA->method();                                    // Класс A
$objB = new B();
$objB->method();                                    // Класс B
```

Однако на практике методы чаще не перекрываются, а переопределяются.

> *Переопределение метода* — дополнение и (или) изменение функциональности метода, унаследованного от базового класса, в производном классе.

При переопределении в производном классе объявляется метод, одноименный с методом из базового класса, в теле этого метода записывается код, реализующий дополнительную функциональность, и в нужное место кода ставится вызов метода из базового класса в формате:

```
parent::<имя метода из базового класса>([<значения параметров через запятую>])
```

Пример:

```
class C extends A {
    // Переопределяем метод method из базового класса
    function method() {
        parent::method();
        echo '   Класс C';
    }
}

$objC = new C();
$objC->method();                                    // Класс A   Класс C
```

Перекрывать и переопределять можно также конструкторы и деструкторы.

7.3.2. Решение проблем с наследованием статических методов

При перекрытии или переопределении статических методов может возникнуть проблема, связанная с особенностью работы конструкции `self`.

Языковая конструкция `self` *всегда* ссылается на класс, в котором объявлен использующий ее статический метод (даже если этот метод будет вызван из производного класса).

Пример:

```
class A {
    public static function class_name() {
        echo 'Класс A';
    }
    public static function say() {
        // Языковая конструкция self в этом месте всегда будет ссылаться
        // на класс A, даже если метод say будет вызван из производного
        // класса
        self::class_name();
    }
}

class B extends A {
    public static function class_name() {
        echo 'Класс B';
    }
}

// Вызываем метод say из производного класса и убеждаемся,
// что сказанное ранее — правда
B::say();                                           // Класс A
```

> Языковая конструкция `static`, напротив, ссылается на класс, чей код исполняется в настоящий момент.

Пример использования этой конструкции:

```
class A {
    public static function class_name() {
        echo 'Класс A';
    }
    public static function say() {
        // Заменяем конструкцию self на static — и проблема решена!
        static::class_name();
    }
}

class B extends A {
    public static function class_name() {
        echo 'Класс B';
    }
}

B::say();                                    // Класс B
```

Полезно знать...

- Ссылки на классы через `self` вычисляются в момент компиляции, когда исполняющая среда еще точно не «знает», на какие именно классы ссылаются эти конструкции (*раннее связывание*). Это и приводит к описанной здесь проблеме.
- Напротив, ссылки через `static` вычисляются при исполнении кода, когда исполняющей среде уже все доподлинно «известно» (*позднее связывание*). Эти вычисления отнимают часть системных ресурсов и несколько снижают быстродействие, но зато при выполнении кода не возникает непредвиденных ситуаций.

7.3.3. Абстрактные методы и классы

> *Абстрактный метод* — метод, предназначенный для перекрытия в производном классе.

Объявление абстрактного метода записывается так же, как объявление обычного метода, но без тела (и без фигурных скобок), с завершающим символом `;` и с добавлением в самом начале, перед модификатором доступа, языковой конструкции `abstract`.

Абстрактные методы — это своего рода шаблоны для объявления в производных классах «конкретных» методов.

Класс, содержащий абстрактные методы, сам должен быть объявлен как абстрактный.

> *Абстрактный класс* — класс, предназначенный исключительно для создания на его основе производных классов.

Объявление абстрактного класса записывается так же, как объявление обычного класса, но с добавлением в самом начале, перед языковой конструкцией `class`, конструкции `abstract`.

Абстрактные классы также можно рассматривать как шаблоны — для объявления производных классов.

> Попытка создания объекта абстрактного класса приводит к фатальной ошибке.

Пример абстрактного класса с абстрактным методом:

```
abstract class A {
    abstract function method();
}

class B extends A {
    function method() {
        return 1;
    }
}

// Создаем объект «конкретного» класса B и вызываем у него метод
$objB = new B();
$objB->method();
// Но попытка создания объекта абстрактного класса приведет к ошибке
$objA = new A();                                    // Ошибка!
```

7.3.4. Окончательные методы и классы

> *Окончательный метод* — метод, который не может быть перекрыт или переопределен в производных классах.

Чтобы превратить метод в окончательный, в самом начале его объявления нужно поставить языковую конструкцию `final`.

Пример:

```
class A {
    final function method() {
        echo 'Класс A';
    }
}

class B extends A {
    function method() {                             // Ошибка!
        echo 'Класс B';
    }
}
```

Переопределив метод в производном классе, сторонний программист может вмешаться в работу внутренних механизмов базового класса. Чтобы предотвратить это, некоторые методы можно объявить окончательными.

> Окончательный класс — класс, на основе которого невозможно создать производные классы.

Чтобы превратить класс в окончательный, в его объявлении, перед языковой конструкцией `class`, нужно поставить конструкцию `final`.

Пример:

```
final class A {
    function method() {
        echo 'Класс A';
    }
}

class B extends A {                                    // Ошибка!
}
```

Классы объявляются окончательными по той же причине, что и отдельные методы, — чтобы предотвратить вмешательство в их работу из производных методов.

7.4. Присваивание объектов. Ссылочные типы данных

> При присваивании переменной объекта, хранящегося в другой переменной, присваивается *ссылка* на этот объект. В результате будут получены две переменные, ссылающиеся на один и тот же объект.

Пример:

```
// Объявляем класс E со свойством prop
class E {
    public $prop = 10;
}
// Создаем объект этого класса и присваиваем его переменной $obj1
$obj1 = new E();
// Присваиваем объект $obj1 переменной $obj2
$obj2 = $obj1;
// Проверяем, что хранится в свойстве prop объекта, обратившись к нему
// через переменные $obj1 и $obj2
echo $obj1->prop;                                      // 10
echo $obj2->prop;                                      // 10
// Изменяем значение свойства prop объекта через переменную $obj1
$obj1->prop = 2000;
// Проверяем, что хранится в этом свойстве, обратившись к объекту через
// обе переменные
```

```
echo $obj1->prop;                                    // 2000
echo $obj2->prop;                                    // 2000
// Как видим, переменные $obj1 и $obj2 указывают на один и тот же объект
```

> *Ссылочный тип данных* — тип, у которого при присваивании другим переменным копируются ссылки на значения.

К ссылочным типам относятся объекты.

Полезно знать...

Значения ссылочных типов хранятся не в самих переменных, а в отдельной области памяти, а в переменных хранятся лишь ссылки на них. При присваивании другим переменным эти ссылки копируются.

7.5. Работа с объектами и классами

PHP поддерживает ряд инструментов для выполнения различных действий с объектами и классами.

7.5.1. Работа с объектами

- Проверка, создан ли указанный *объект* на основе заданного *класса*, — оператор `instanceof`: `<объект> instanceof <класс>`:

```
class A { . . . }
class B { . . . }
$objA = new A();
echo $objA instanceof A;                             // TRUE
echo $objA instanceof B;                             // FALSE
```

Если указанный *объект* создан на основе класса, являющегося производным от заданного *класса*, оператор `instanceof` также вернет `TRUE`:

```
class C extends B { . . . }
$objC = new C();
echo $objC instanceof B;                             // TRUE
```

Также можно использовать функцию `is_a(<объект>, <имя класса>)`, где *имя класса* задается в виде строки:

```
echo is_a($objA, 'A');                               // TRUE
echo is_a($objA, 'B');                               // FALSE
echo is_a($objC, 'B');                               // TRUE
```

- Проверка, является ли класс, на основе которого создан указанный *объект*, производным от заданного *класса*, — функция `is_subclass_of(<объект>, <имя класса>)`. *Имя класса* задается в виде строки:

```
echo is_subclass_of($objC, 'B');                     // TRUE
echo is_subclass_of($objC, 'A');                     // FALSE
```

- Проверка, поддерживает ли класс, на основе которого создан указанный *объект*, заданное *свойство*, — функция property_exists(<*объект*>, <*имя свойства*>). *Имя свойства* задается в виде строки:

```
$img = new Image();
echo property_exists($img, 'src');                    // TRUE
echo property_exists($img, 'alt');                    // FALSE
```

- Проверка, поддерживает ли класс, на основе которого создан указанный *объект*, заданный *метод*, — функция method_exists(<*объект*>, <*имя метода*>). *Имя метода* задается в виде строки:

```
echo method_exists($img, 'get_img_tag');              // TRUE
echo method_exists($img, 'get_div_tag');              // FALSE
```

- Имя класса, на основе которого был создан *объект*, — функция get_class(<*объект*>). Имя класса возвращается в виде строки:

```
echo get_class($objA);                                // A
```

- Имя базового класса для класса, на основе которого создан *объект*, — функция get_parent_class(<*объект*>). Результат возвращается в виде строки:

```
echo get_parent_class($objC);                         // B
```

7.5.2. Работа с классами

- Имя класса в виде строки — статическое свойство class, которое PHP добавляет ко всем классам:

```
echo A::class;                                        // A
```

- Уточненное имя класса, полученное путем позднего связывания, — функция get_called_class(). Возвращает имя класса в виде строки, если вызвана из тела статического метода, и FALSE — в случае вызова вне класса:

```
class D {
    static function method() {
        echo get_called_class();                      // D
    }
}
```

- Проверка, объявлен ли указанный *класс*, — функция class_exists:

```
class_exists(<имя класса>[, <выполнять автозагрузку?>])
```

Имя класса задается в виде строки. Если поиски класса завершились удачей, возвращается TRUE, в противном случае — FALSE.

Второй параметр используется только в том случае, если PHP не найдет объявление класса в текущем модуле или во включенных модулях. Если передать значение TRUE или вообще не указывать второй параметр, PHP далее попытается выполнить автозагрузку класса (подробно об автозагрузке рассказано в *разд. 7.7*)

и вернет TRUE или FALSE в случае, соответственно, удачи или неудачи. Если указать значение FALSE, автозагрузка выполняться не будет, и функция сразу вернет FALSE.

Примеры:

```
echo class_exists('A');                    // TRUE
echo class_exists('E');                    // FALSE
```

- Проверка, является ли *класс 1* производным от *класса 2* — функция is_subclass_of(<*имя класса 1*>, <*имя класса 2*>). *Имена классов* задаются в виде строк:

```
echo is_subclass_of('C', 'B');             // TRUE
echo is_subclass_of('C', 'A');             // FALSE
```

- Проверка, поддерживает ли указанный *класс* заданное *свойство*, — функция property_exists(<*имя класса*>, <*имя свойства*>). *Имена класса* и *свойства* задаются в виде строк:

```
echo property_exists('Image', 'src');      // TRUE
echo property_exists('Image', 'alt');      // FALSE
```

- Проверка, поддерживает ли указанный *класс* заданный *метод*, — функция method_exists(<*имя класса*>, <*имя метода*>). *Имена класса* и *метода* задаются в виде строк:

```
echo method_exists('Image', 'get_img_tag');   // TRUE
echo method_exists('Image', 'get_div_tag');   // FALSE
```

- Имя базового класса для заданного *класса* — функция get_parent_class(<*имя класса*>). *Имя класса* задается в виде строки, результат также возвращается в виде строки:

```
echo get_parent_class('C');                // B
```

7.6. Магические методы

> *Магический метод* вызывается в особых ситуациях — например, при обращении к неподдерживаемому свойству, вызове неподдерживаемого метода, попытке преобразования объекта в строку.

Магические методы объявляются в классе так же, как и обычные методы, однако должны иметь строго определенные имена.

Список ситуаций и соответствующих им магических методов приведен далее:

- получение значения неподдерживаемого свойства: метод __get. Формат объявления метода:

```
__get(<имя свойства, к которому было выполнено обращение>)
```

Имя свойства передается в виде строки. Метод должен возвращать значение, которое будет возвращено коду, обратившемуся к этому свойству;

- присваивание нового значения неподдерживаемому свойству: метод __set. Формат объявления:

```
__set(<имя свойства, которому присваивается новое значение>,
      <присваиваемое значение>)
```

Имя свойства передается в виде строки. Метод не должен возвращать результата;

- проверка существования неподдерживаемого свойства: метод _isset. Формат объявления:

```
__isset(<имя свойства, чье существование проверяется>)
```

Имя свойства передается в виде строки. Метод должен возвращать TRUE, чтобы имитировать существование такого свойства в классе, или FALSE, если свойство не должно «существовать».

Метод __isset вызывается при использовании функций isset и empty;

- удаление неподдерживаемого свойства: метод __unset:

```
__unset(<имя удаляемого свойства>)
```

Имя свойства передается в виде строки. Метод не должен возвращать результата;

- вызов неподдерживаемого метода: метод __call:

```
__call(<имя вызываемого метода>,
       <массив с параметрами, переданными вызываемому методу>)
```

Имя метода передается в виде строки. Метод может возвращать результат;

- вызов неподдерживаемого статического метода: статический метод __callStatic:

```
__callStatic(<имя вызываемого метода>,
             <массив с параметрами, переданными вызываемому методу>)
```

Имя метода передается в виде строки. Метод может возвращать результат;

> ***Замечание о закрытых и защищенных свойствах и методах***
>
> Шесть указанных здесь методов вызываются также в случае обращения к закрытому или защищенному свойству или методу извне объекта.

- попытка преобразования объекта в строку: метод __toString. Он не должен принимать параметров и должен возвращать в качестве результата строку;
- попытка вызова объекта как функции: метод __invoke. Количество принимаемых им параметров может быть произвольным. Метод может возвращать результат.

Пример реализации методов: __get, __set, __isset, __unset, __call и __callStatic:

```
class Demo {
    function __get($name) {
        echo "Свойство {$name}: получение значения";
    }
    function __set($name, $val) {
        echo "Свойство {$name}: присваивание значения {$val}";
    }
```

```
    function __isset($name) {
        echo "Свойство {$name}: проверка существования";
        return TRUE;
    }
    function __unset($name) {
        echo "Свойство {$name}: удаление";
    }
    function __call($name, $params) {
        $sp = implode(', ', $params);
        echo "Метод {$name}: вызов с параметрами {$sp}";
    }
    static function __callStatic($name, $params) {
        $sp = implode(', ', $params);
        echo "Метод {$name} (статический): вызов с параметрами {$sp}";
    }
}

$obj = new Demo();
$a = $obj->a;                  // Свойство a: получение значения
$obj->b = 4;                   // Свойство b: присваивание значения 4
isset($obj->c);                // Свойство c: проверка существования
unset($obj->def);              // Свойство def: удаление
$obj->xyz(1, 2, 3, 4);         // Метод xyz: вызов с параметрами 1, 2, 3, 4
Demo::xyzStatic('Тест', '!!!');
          // Метод xyzStatic (статический): вызов с параметрами Тест, !!!
```

Пример реализации методов: `__toString` и `__invoke`:

```
class Demo2 {
    function __toString() {
        return 'Объект класса Demo2';
    }
    function __invoke(...$params) {
        $sp = implode(', ', $params);
        echo "Объект класса Demo2: вызов с параметрами {$sp}";
    }
}

$obj = new Demo2();
echo (string)$obj;                                  // Объект класса Demo2
$obj(7, 8, 9);         // Объект класса Demo2: вызов с параметрами 7, 8, 9
```

7.7. Автозагрузка классов

В современном PHP-программировании принято соглашение, согласно которому объявление каждого класса хранится в отдельном модуле. Чтобы не писать каждый раз в начале кода длинный список выражений, включающих модули с нужными классами, рекомендуется использовать автозагрузку классов.

> *Автозагрузка классов* — механизм, автоматически загружающий и включающий модуль с объявлением требуемого класса.

Чтобы активизировать автозагрузку классов, необходимо перед самым первым обращением к какому-либо классу зарегистрировать, по крайней мере, один обработчик автозагрузки.

> *Обработчик автозагрузки классов* — функция, непосредственно выполняющая поиск и включение нужного модуля. В качестве параметра она должна принимать имя требуемого класса в виде строки и не должна возвращать результат.

Регистрация обработчика автозагрузки выполняется вызовом функции `spl_autoload_register`:

```
spl_autoload_register(<обработчик>[, <сгенерировать исключение?>[,
                      <вставить обработчик в начало очереди?>]])
```

Обработчик указывается в виде строки с именем функции или переменной, хранящей анонимную функцию.

Если вторым параметром передать значение `TRUE` или вообще не указывать его, при невозможности зарегистрировать обработчик функция сгенерирует исключение (об исключениях мы поговорим на *уроке 12*). Если передать значение `FALSE`, исключение генерироваться не будет.

Вызовом функции `spl_autoload_register` можно зарегистрировать несколько обработчиков автозагрузки, которые будут помещены в очередь и станут вызываться друг за другом: если первый обработчик не сможет найти нужный модуль, будет вызван второй, и т. д.

Если третьим параметром передать значение `FALSE` или вообще опустить его, регистрируемый обработчик будет помещен в конец очереди, если же передать `TRUE` — в ее начало.

Функция возвращает `TRUE`, если обработчик успешно зарегистрирован, и `FALSE` — в противном случае.

Пример реализации автозагрузки классов:

```
// Не забываем дублировать обратный слеш, если он находится в конце
// строки, иначе PHP посчитает его литералом \'
$base_path = 'c:\xampp\htdocs\\';
function autoloader(string $class_name) {
    require_once 'modules\classes\\' . $class_name . '.php';
}
spl_autoload_register('autoloader');

// Будет выполнена автозагрузка модуля
// c:\xampp\htdocs\modules\classes\A.php
$objA = new A();
// Будет выполнена автозагрузка модуля
// c:\xampp\htdocs\modules\classes\SuperClass.php
$objSC = new SuperClass();
```

Урок 8
Объектное программирование: трейты, интерфейсы и пространства имен

8.1. Трейты

> *Трейт* — набор свойств и методов, который можно добавить в произвольный класс.

Если необходимо наделить сходной функциональностью сразу несколько классов, не делая их производными от одного суперкласса, эту функциональность удобно реализовать в трейте.

8.1.1. Объявление и использование трейтов

Объявление трейта записывается в формате:

```
trait <имя трейта> {
    <объявления свойств>
    <объявления методов>
}
```

> В трейтах допускается объявлять обычные свойства, обычные методы, статические методы и абстрактные методы. Абстрактные методы должны быть перекрыты в классе, использующем трейт.

Пример трейта:

```
trait SomeTrait {
    // Обычное свойство
    public $prop = 1234;
    // Обычный метод
    function method1() {
        echo 'Трейт SomeTrait: обычный метод';
    }
    // Статический метод
    static function method2() {
        echo 'Трейт SomeTrait: статический метод';
    }
    // Абстрактный метод. Должен быть перекрыт в классе.
    abstract function method3();
}
```

Чтобы использовать трейты в классе, в его объявлении, внутри фигурных скобок, следует записать конструкцию:

```
use <список имен используемых трейтов через запятую>;
```

Класс может использовать произвольное количество трейтов.

Пример использования трейта в классе:

```
class SomeClass {
    use SomeTrait;
    function method3() {
        echo 'Класс SomeClass: абстрактный метод, перекрытый в классе';
    }
}

$obj = new SomeClass();
echo $obj->prop;                                    // 1234
$obj->method1();                    // Трейт SomeTrait: обычный метод
SomeClass::method2();               // Трейт SomeTrait: статический метод
$obj->method3();
              // Класс SomeClass: абстрактный метод, перекрытый в классе
```

Трейты могут использоваться и другими трейтами.

Пример:

```
trait OtherTrait {
    use SomeTrait;
    . . .
}
```

8.1.2. Разрешение конфликтов

Если и в производном классе, и в используемом им трейте, и в базовом классе объявлены свойства или методы с одинаковыми именами, возникает конфликт, разрешаемый по описанным далее правилам.

Простые случаи: автоматическое разрешение конфликтов

> Если в используемом трейте объявлено свойство, класс не может объявить свойство с тем же именем, но с другими модификатором и (или) начальным значением. Попытка сделать это вызовет фатальную ошибку.

Пример:

```
trait SomeTrait2 {
    public $prop1 = 4321;
    public $prop2 = 8.4;
}
class SomeClass2 {
    use SomeTrait2;
```

```
    // Повторно объявляем свойство prop1 из трейта с теми же
    // модификатором и начальным значением — успешно
    public $prop1 = 4321;
    // Повторно объявляем свойство prop2 из трейта,
    // но с другим модификатором — неуспешно
    protected $prop2 = 8.4;                          // Ошибка!
}
```

> Если метод с одним и тем же именем объявлен в производном классе, трейте, используемом классом, и базовом классе, применяется следующий порядок разрешения возникшего конфликта:
>
> - метод, объявленный в трейте, перекроет метод из базового класса;
> - метод, объявленный в производном классе, перекроет метод из трейта.

Пример:

```
class BaseClass {
    function method1() {
        echo 'Базовый класс: метод method1';
    }
    function method2() {
        echo 'Базовый класс: метод method2';
    }
    function method3() {
        echo 'Базовый класс: метод method3';
    }
}
trait SomeTrait3 {
    function method2() {
        echo 'Трейт: метод method2';
    }
    function method3() {
        echo 'Трейт: метод method3';
    }
}
class DerivedClass extends BaseClass {
    use SomeTrait3;
    function method3() {
        echo 'Производный класс: метод method3';
    }
}

$obj = new DerivedClass();
$obj->method1();                    // Базовый класс: метод method1
$obj->method2();                    // Трейт: метод method2
$obj->method3();                    // Производный класс: метод method3
```

Сложные случаи: разрешение конфликтов вручную

> Если класс использует несколько трейтов, в которых объявлен метод с одним и тем же именем, возникает фатальная ошибка. В этом случае надо явно указать, метод какого трейта следует использовать, записав необходимые указания в дополнительных параметрах конструкции `use`.

Дополнительные параметры конструкции `use` записываются в формате:

```
use <список имен используемых трейтов через запятую> {
    <набор дополнительных параметров>
}
```

Указание, *метод* из какого *трейта* следует вызывать, пишется в составе *дополнительных параметров* в формате:

```
<целевой трейт>::<метод> insteadof <остальные трейты через запятую>;
```

Пример:

```
trait A {
    function method1() {
        echo 'Трейт A: метод method1';
    }
    function method2() {
        echo 'Трейт A: метод method2';
    }
}
trait B {
    function method1() {
        echo 'Трейт B: метод method1';
    }
    function method2() {
        echo 'Трейт B: метод method2';
    }
}
class C {
    use A, B {
        // Используем метод method1 из трейта A, а не B
        A::method1 insteadof B;
        // Используем метод method2 из трейта B, а не A
        B::method2 insteadof A;
    }
}

$obj = new C();
$obj->method1();                    // Трейт A: метод method1
$obj->method2();                    // Трейт B: метод method2
```

Языковая конструкция `insteadof` делает методы из других трейтов недоступными для вызова. Если же эти методы должны оставаться доступными, следует указать для них псевдонимы.

> *Псевдоним метода* из трейта указывается в дополнительных параметрах конструкции use в формате:
>
> `<трейт>::<метод> as <псевдоним>;`

Пример:

```
class D {
    use A, B {
        // Используем метод method1 из трейта B, а не A
        B::method1 insteadof A;
        // Используем метод method2 из трейта A, а не B
        A::method2 insteadof B;
        // Делаем метод method2 из трейта B доступным через псевдоним mb
        B::method2 as mb;
    }
}

$obj = new D();
$obj->method1();                    // Трейт B: метод method1
$obj->method2();                    // Трейт A: метод method2
$obj->mb();                         // Трейт B: метод method2
```

8.1.3. Дополнительные инструменты для работы с трейтами

Для проверки, объявлен ли заданный *трейт*, следует вызвать функцию trait_exists:

```
trait_exists(<имя трейта>[, <выполнять автозагрузку?>])
```

Имя трейта задается в виде строки. Если поиски трейта завершились удачей, возвращается TRUE, в противном случае — FALSE.

Если вторым параметром передать значение TRUE или вообще не указывать его, PHP далее попытается выполнить автозагрузку трейта и вернет TRUE или FALSE в случае, соответственно, удачи или неудачи. Если указать значение FALSE, автозагрузка выполняться не будет, и функция сразу вернет FALSE.

Примеры:

```
echo trait_exists('SomeTrait');          // TRUE
echo trait_exists('SpecialTrait');       // FALSE
```

8.2. Интерфейсы

> *Интерфейс* — список методов, которые должны быть объявлены в классах, реализующих этот интерфейс.

Интерфейсы хорошо подходят для унификации классов, в том числе и унаследованных от разных базовых классов.

8.2.1. Объявление и реализация интерфейсов

Интерфейс объявляется так же, как и класс, за следующими исключениями:

- вместо языковой конструкции `class` применяется `interface`;
- интерфейсы могут содержать только объявления методов и констант;
- методы в интерфейсе объявляются без тела и без фигурных скобок (как и абстрактные методы в классе);
- все методы должны быть общедоступными.

Имена интерфейсов традиционно начинаются с прописной буквы `I`.

Пример объявления интерфейса:

```
interface ITag {
    public function get_tag($content);
}
```

Список реализуемых интерфейсов записывается в классе с применением конструкции `implements`:

```
class <имя класса> . . . implements <список интерфейсов через запятую> {
    . . .
}
```

> Класс, реализующий интерфейсы, должен объявлять *все* методы из этих интерфейсов. Несоблюдение этого требования приведет к фатальной ошибке.

Пример:

```
class PTag implements ITag {
    public function get_tag($content) {
        return "<p>{$content}</p>";
    }
}
class DivTag implements ITag {
    public function get_tag($content) {
        return "<div>{$content}</div>";
    }
}

$objP = new PTag();
$objDiv = new DivTag();
echo $objP->get_tag('PHP');                          // <p>PHP</p>
echo $objDiv->get_tag('MySQL');                      // <div>MySQL</div>
// Класс PTag, на основе которого создан объект $objP, реализует
// интерфейс ITag?
echo $objP instanceof ITag;                          // TRUE
```

8.2.2. Наследование интерфейсов

> Интерфейсы могут наследоваться, причем у производного интерфейса может быть несколько базовых интерфейсов.

Пример:

```
// Объявляем базовые интерфейсы
interface I1 {
    function method1{};
}
interface I2 {
    function method2{};
}
interface I3 {
    function method3{};
}
// Объявляем наследующий их производный интерфейс
interface IAll extends I1, I2, I3 {
    function method4{};
}
```

> Класс, реализующий производный интерфейс, должен объявить все методы, имеющиеся как в производном, так и во всех базовых интерфейсах.

Пример:

```
class E implements IAll {
    function method1() { . . . }
    function method2() { . . . }
    function method3() { . . . }
    function method4() { . . . }
}
```

8.2.3. Дополнительные инструменты для работы с интерфейсами

Для проверки, объявлен ли заданный *интерфейс*, следует вызвать функцию `interface_exists`:

```
interface_exists(<имя интерфейса>[, <выполнять автозагрузку?>])
```

Имя интерфейса задается в виде строки. Если поиски завершились удачей, возвращается `TRUE`, в противном случае — `FALSE`.

Если вторым параметром передать значение `TRUE` или вообще не указывать его, PHP попытается выполнить автозагрузку интерфейса и вернет `TRUE` или `FALSE` в случае, соответственно, удачи или неудачи. Если указать значение `FALSE`, автозагрузка выполняться не будет, и функция сразу вернет `FALSE`.

Примеры:

```
echo interface_exists('ITag');                    // TRUE
echo interface_exists('ITable');                  // FALSE
```

8.3. Пространства имен

> *Пространство имен* — сущность, объединяющая набор классов, интерфейсов, трейтов, функций и констант.

Пространства имен PHP можно сравнить с папками файловой системы. Как папки предназначены для объединения файлов, составляющих одну программу или хранящих части одного документа, так и пространства имен служат для группировки программных сущностей, реализующих какую-либо часть функциональности.

> Разные пространства имен могут содержать сущности (например, классы) с одинаковыми именами.

8.3.1. Объявление пространств имен

Пространство имен можно объявить с применением двух видов синтаксиса: однострочного и с фигурными скобками.

> Однострочный синтаксис:
>
> ```
> namespace <имя пространства имен>;
> <объявления сущностей, входящих в это пространство имен>
> ```

Языковая конструкция `namespace` должна быть самой первой в программном модуле. Перед ней не должно быть никакого другого кода, даже написанного на HTML.

Пример:

```
<?php
// Объявляем пространство имен Core
namespace Core;
// Объявляем константу, функцию и класс, которые войдут в это
// пространство имен
const MODEL_PREFIX = 'M';
function load_model($model_name) { . . . }
class Model { . . . }
```

> Синтаксис с фигурными скобками:
>
> ```
> namespace <имя пространства имен> {
>     <объявления сущностей, входящих в это пространство имен>
> }
> ```

Этот синтаксис более предпочтителен, поскольку, во-первых, нагляднее (сразу видно, какие элементы относятся к пространству имен), а во-вторых, позволяет объявить несколько пространств имен в одном модуле.

Пример:

```
namespace Core {
    const MODEL_PREFIX = 'M';
    function load_model($model_name) { . . . }
    class Model { . . . }
}
```

Как и папки файловой системы, пространства имен можно вкладывать друг в друга:

- для объявления вложенного пространства имен в языковой конструкции namespace достаточно записать его путь;
- в качестве разделителя в путях пространств имен применяется символ обратного слеша \.

Пример:

```
// Объявляем пространство имен Models, вложенное в Core
namespace Core\Models {
    class User { . . . }
    class Image { . . . }
    class Comment { . . . }
}
```

8.3.2. Обращение к сущностям, объявленным в других пространствах имен

Обратиться к сущности (например, классу), объявленной в одном пространстве имен, из другого пространства имен можно двумя способами.

Прямое обращение по пути

Проще всего для обращения к сущности из другого пространства имен — записать полный путь этой сущности.

> *Полный путь* к сущности должен завершаться именем этой сущности. Не забываем, что в качестве разделителей в путях применяются символы обратного слеша \.

> *Относительные пути* отсчитываются от текущего пространства имен. Их признаком является отсутствие в начале символа обратного слеша.

Пример использования относительного пути (выделен рамкой):

```
namespace Core\Utilities\DB {
    function connect_to_database() { . . . }
}
namespace Core {
    class Model {
        . . .
        function __construct() {
            . . .
```

```
            // Обращение к функции connect_to_database
            // из пространства имен Core\Utilities\DB
            Utilities\DB\connect_to_database();
            . . .
        }
    }
}
```

> *Абсолютные пути* отсчитываются от *корневого пространства имен*, аналогичного корневой папке в файловой системе. Абсолютные пути должны начинаться с обратного слеша.

Пример использования абсолютного пути (выделен рамкой):

```
namespace Settings\Database {
    const db_host = '192.168.1.3'
}
namespace Core\Utilities\DB {
    function connect_to_database($db_id) {
        . . .
        $host = \Settings\Database\db_host;
        . . .
    }
}
```

> Все встроенные в PHP функции, константы и классы объявлены в корневом пространстве имен. Чтобы обратиться к сущности из корневого пространства имен, достаточно поставить перед именем этой сущности обратный слеш.

Пример обращения к встроенной функции `trim`, объявленной в корневом пространстве имен (выделено рамкой):

```
namespace Core\Utilities\DB {
    function connect_to_database($db_id) {
        . . .
        $host = \Settings\Database\db_host;
        $host = \trim($host);
        . . .
    }
}
```

Выполнение импорта

> После выполнения *импорта* сущность из другого пространства имен становится доступной в текущем пространстве имен по одному только имени, без указания пути.

Выражение импорта сущности записывается с помощью конструкции `use`:

```
use [const|function] <полный путь к сущности> [as <псевдоним>];
```

Если импортируется константа, после use следует указать конструкцию const, если импортируется функция — конструкцию function. При импорте класса, трейта или интерфейса между use и *путем* ничего вставлять не нужно.

Если *псевдоним* не указан, сущность будет доступна под ее изначальным именем.

Пример:

```
namespace Core {
    const MODEL_PREFIX = 'M';
    class Model { . . . }
}
namespace Core\Utilities\DB {
    function connect_to_database() { . . . }
}

// Импортируем функцию Core\Utilities\DB\connect_to_database
use function \Core\Utilities\DB\connect_to_database;
// Теперь для вызова этой функции достаточно записать лишь ее имя
connect_to_database();

// Импортируем константу Core\MODEL_PREFIX под псевдонимом MP
use const \Core\MODEL_PREFIX as MP;
$pr = MP;

// Импортируем класс Core\Model
use \Core\Model;
class userModel extends Model { . . . }
```

> В одной языковой конструкции use можно указать несколько путей к импортируемым сущностям *одного типа* (например, только классам или только функциям), перечислив их через запятую.

Пример:

```
namespace Core\Models {
    class Image { . . . }
    class Comment { . . . }
}
namespace Core\Controllers {
    class Images { . . . }
    class Comments { . . . }
}

use \Core\Models\Image, \Core\Models\Comment, \Core\Controllers\Images,
    \Core\Controllers\Comments;
```

> Если импортируемые сущности, вдобавок, находятся в одном пространстве имен, можно указать их имена через запятую, взяв в фигурные скобки.

Пример:

```
use \Core\Models\{Image, Comment}, \Core\Controllers\{Images, Comments};
```

8.3.3. Дополнительные инструменты для работы с пространствами имен

- Полный путь к классу, на основе которого был создан *объект*, — функция `get_class(<объект>)`. Путь возвращается в виде строки без начального обратного слеша:

```
$img = new \Core\Models\Image();
echo get_class($img);                          // Core\Models\Image
```

- Полный путь к классу — статическое свойство `class`, которое PHP добавляет ко всем классам. Путь возвращается также в виде строки без начального обратного слеша:

```
use \Core\Controllers\Comments;
echo Comments::class;                  // Core\Controllers\Comments
```

Урок 9
Архитектура «модель-шаблон-контроллер»

Еще в упражнении из *разд. 5.8* мы начали структурировать программный код сайта фотогалереи, разделяя его на отдельные модули. В дальнейшем это значительно упростит сопровождение сайта (хотя бы потому, что мы будем знать, какой модуль за что «отвечает»).

Оптимальнее структурировать код позволяет объектное программирование. В *разд. 7.3* мы узнали, что классы, на основе которых создаются объекты, могут наследовать функциональность друг у друга. Это значит, что общую функциональность можно реализовать в базовых классах, а на долю производных оставить только специфическую. Используя готовые базовые классы при разработке последующих сайтов, мы значительно упростим себе жизнь.

9.1. Введение в архитектуру «модель-шаблон-контроллер»

И, наконец, наилучшим вариантом является разделение программных модулей, составляющих код сайта, на модели, шаблоны и контроллеры.

> *Модель* — хранит и выдает по запросам данные, необходимые для генерирования результирующих веб-страниц.

Модель пишется на PHP и практически всегда оформляется в виде класса.

Как правило, данные, которыми манипулирует модель, хранятся в информационной базе, однако в самых простых случаях они могут быть записаны в текстовом файле или даже обычном массиве PHP.

> *Шаблон* — описывает генерируемую сайтом результирующую веб-страницу.

Шаблон пишется, в основном, на HTML с отдельными фрагментами PHP-кода (то есть представляет собой серверную страницу).

> *Контроллер*:
>
> - выполняется при получении клиентского запроса;
> - извлекает нужные данные из модели;
> - возможно, сохраняет в модели данные, полученные от посетителя сайта;

- генерирует в ответ результирующую страницу на основе шаблона, используя полученные на предыдущих шагах данные.

Контроллер пишется на PHP и обычно оформляется в виде класса (хотя в наиболее простых случаях его можно написать в виде функции).

Контроллер запускается непосредственно маршрутизатором при получении запроса с путем, совпадающим с определенным шаблоном (о маршрутизаторе рассказывалось в *разд. 5.8*). Модели и шаблоны запускаются контроллером, как только в них появится нужда. Можно сказать, что контроллер играет главную роль, модель и шаблон — подчиненную.

Модели, шаблоны и контроллеры программируются таким образом, чтобы не зависеть друг от друга, быть универсальными. Любой контроллер может использовать произвольные модели и шаблоны, и, наоборот, любая модель и любой шаблон могут быть использованы произвольным контроллером.

Архитектура *«модель-шаблон-контроллер»* — предусматривает разделение программного кода на модели, шаблоны и контроллеры.

Преимущества этой архитектуры:

- удобство изменения кода, выполняющего разные задачи и разнесенного по разным модулям:
  - чтобы изменить формат хранения данных — достаточно исправить модель, не затрагивая контроллер и шаблон;
  - чтобы изменить логику работы сайта — достаточно исправить контроллер, не затрагивая модель и шаблон;
  - чтобы изменить генерируемую веб-страницу — достаточно исправить шаблон, не затрагивая модель и контроллер;
- упрощение программирования и тестирования модулей — по той же самой причине;
- возможность повторного использования кода — поскольку каждый модуль универсален и независим от остальных. В результате, например, одна и та же модель может использоваться разными контроллерами, даже входящими в состав разных сайтов;
- написание разных модулей можно поручить разным разработчикам. Обычно модели и контроллеры пишут PHP-программисты, а шаблоны — веб-верстальщики.

На этом уроке мы займемся переделкой сайта фотогалереи с использованием архитектуры «модель-шаблон-контроллер». Процесс этот будет долгим, поэтому разобьем его на три упражнения.

9.2. Упражнение. Пишем класс модели и реализуем автозагрузку классов

Напишем класс модели `Image`, предоставляющий сведения об опубликованных на сайте изображениях. Этот класс будет поддерживать два статических метода:

- `get_count()` — возвращает количество опубликованных изображений;
- `get_image(<индекс>)` — возвращает вложенный массив, в котором хранятся сведения об изображении с заданным *индексом*.

Класс модели поместим в пространство имен `Models`. Объявление этого класса сохраним в модуле image.php, расположенном в папке modules\models корневой папки сайта.

Наконец, реализуем автозагрузку модулей, в которых хранятся объявления запрашиваемых классов.

Файлы фотогалереи можно найти в папке 5\5.8\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Нам понадобится отдельный модуль для хранения настроек сайта — в частности, названия сайта и имени папки, в которой хранятся графические файлы.

   Храните настройки веб-сайта в отдельном модуле!
   Особенно если сайт сложный, и настроек, влияющих на его работу, много.

   Создадим в папке modules настроечный модуль settings.php и запишем в него код из листинга 9.1.

Листинг 9.1. Модуль modules\settings.php

```
<?php
namespace Settings {
    const SITE_NAME = 'Фотогалерея';

    const IMAGE_PATH = '/images/';
}
```

   Константы `SITE_NAME` (с названием сайта) и `IMAGE_PATH` (с относительным путем к папке с изображениями) мы поместили в пространство имен с «говорящим» именем `Settings`.

2. Теперь займемся автозагрузкой классов и заодно выполним включение модуля с настройками. Необходимый код запишем в единой точке входа.

   Откроем модуль единой точки входа index.php и заменим хранящийся в нем код на приведенный в листинге 9.2.

Листинг 9.2. Модуль index.php

```
<?php
$base_path = __DIR__ . '\\';
require_once $base_path . 'modules\settings.php';

function my_autoloader(string $class_name) {
    global $base_path;
    require_once $base_path . 'modules\\' . $class_name . '.php';
}
spl_autoload_register('my_autoloader');

require_once $base_path . 'modules\router.php';
```

Мы удалили выражение, включающее модуль modules\data.php, поскольку хранить список изображений отныне будет модель, которую мы сейчас напишем. Затем вставили регистрацию обработчика автозагрузки классов — функции `my_autoloader` — и выражение, включающее модуль маршрутизатора modules\router.php.

Обработчик автозагрузки поступит очень просто: добавит к пути к классу, полученному с параметром, путь к папке modules — в начале и расширение .php — в конце. В результате объявление класса, например, `Models\Image` будет загружено из файла *<путь к корневой папке>*\modules\models\image.php.

> ***Пространства имен и структура папок***
>
> Очень часто структуру папок, в которых хранятся модули с объявлениями классов, приводят в соответствие со структурой вложенных друг в друга пространств имен (как поступили мы). Это значительно упрощает программирование.

3. Создадим в папке modules папку models.
4. Создадим в папке modules\models модуль image.php, в который запишем объявление класса модели `Models\Image` из листинга 9.3.

Листинг 9.3. Модуль modules\models\image.php

```
<?php
namespace Models;
class Image {
    private const DATA = [
        ['src' => '200804282020.jpg', 'desc' => 'Цветы кактуса'],
        . . .
        ['src' => '201802131844.jpg',
            'desc' => 'Блинная скульптура в кафе']
    ];

    static function get_count(): int {
        return count(self::DATA);
    }
```

```
    static function get_image(int $index): array {
        return self::DATA[$index];
    }
}
```

Массив со сведениями об изображениях, хранящийся в константе `DATA` класса `Models\Image`, можно скопировать из модуля modules\data.php.

5. Удалим модуль modules\data.php.

На этом написание модели и обработчика автозагрузки закончено. Контроллер и шаблоны мы напишем в следующих упражнениях.

Сделайте сами

Измените модули modules\list.php и modules\item.php, чтобы они извлекали данные из модели `Models\Image`. Так вы сразу сможете проверить модель в работе.

9.3. Упражнение. Пишем контроллер

Приступим к работе над контроллером, выводящим страницы со списком изображений и с изображением, выбранным посетителем.

Сначала создадим модуль modules\helpers.php, в котором объявим функцию `Helpers\render`. Она будет генерировать страницу на основе указанного файла шаблона и контекста шаблона.

> *Контекст шаблона* — данные, необходимые для генерирования страницы на основе шаблона. Может представлять собой набор обычных переменных или ассоциативный массив.

В качестве параметров функция примет имя шаблона без расширения и контекст шаблона в виде ассоциативного массива. Она превратит контекст-массив в набор обычных переменных (чтобы упростить программирование шаблонов), вызвав функцию `extract`, после чего сформирует полный путь к файлу шаблона и выполнит включение этого файла.

Условимся, что файлы шаблонов будем хранить в папке modules\templates.

Далее напишем два класса:

- `Controllers\BaseController` — базовый класс, реализующий единую для всех контроллеров функциональность. В этом классе объявим два метода:
  - `context_append` — добавит в контекст шаблона дополнительные данные. В качестве параметра будет принимать ссылку на контекст шаблона и не станет возвращать результат.

    Сейчас пока этот метод останется «пустым» (на *уроке 21* мы напишем в нем код, добавляющий в контекст шаблона сведения о текущем пользователе);

- `render` — сгенерирует страницу на основе указанного шаблона. В качестве параметра будет принимать имя шаблона и контекст шаблона и не станет возвращать результат.

  Тело этого метода будет содержать лишь вызов метода `context_append` того же класса и функции `Helpers\render`.

Поскольку оба метода станут вызываться лишь изнутри производных классов, сделаем их защищенными.

Объявление класса сохраним в модуле modules\controllers\basecontroller.php;

- `Controllers\Images` — класс нашего первого контроллера с двумя общедоступными методами:
  - `list()` — выведет страницу со списком изображений, используя шаблон, который мы назовем `list`;
  - `item(<индекс>)` — выведет страницу с изображением, имеющим указанный *индекс*, посредством шаблона `item`.

  Сохраним его в модуле modules\controllers\images.php.

Файлы сайта фотогалереи можно найти в папке 9\9.2\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Создадим в папке modules модуль helpers.php, предназначенный для хранения вспомогательного кода, и запишем в него объявление функции `Helpers\render` из листинга 9.4.

Листинг 9.4. Модуль modules\helpers.php

```
<?php
namespace Helpers {
    function render(string $template, array $context) {
        global $base_path;
        extract($context);
        require $base_path . '\modules\templates\\' . $template .
            '.php';
    }
}
```

2. Создадим папку modules\controllers.
3. Создадим в папке modules\controllers модуль basecontroller.php и запишем в него объявление базового для всех контроллеров класса `Controllers\BaseController` из листинга 9.5.

Листинг 9.5. Модуль modules\controllers\basecontroller.php

```
<?php
namespace Controllers;
require_once $base_path . 'modules\helpers.php';
```

```
class BaseController {
    protected function context_append(array &$context) {}

    protected function render(string $template, array $context) {
        $this->context_append($context);
        \Helpers\render($template, $context);
    }
}
```

4. Создадим в папке modules\controllers модуль images.php и запишем в него объявление класса контроллера `Controllers\Images` из листинга 9.6.

Листинг 9.6. Модуль modules\controllers\images.php

```
<?php
namespace Controllers;
class Images extends BaseController {
    function list() {
        $ctx = ['cnt' => \Models\Image::get_count()];
        $this->render('list', $ctx);
    }

    function item(int $index) {
        $item = \Models\Image::get_image($index);
        $ctx = ['item' => $item];
        $this->render('item', $ctx);
    }
}
```

При выводе страницы со списком картинок (метод `list`) в контекст шаблона под именем `cnt` заносится общее количество картинок. Чтобы в коде шаблона получить сведения об отдельных изображениях, придется обращаться непосредственно к модели `Models\Image`.

> ***Изолируйте шаблоны от моделей!***
>
> Извлечение данных непосредственно из модели в коде шаблона — плохой стиль программирования. Шаблон должен получать необходимые данные исключительно от контроллера и исключительно через контекст шаблона.
>
> Этот недочет мы исправим на *уроке 10*. А пока что не будем обращать на него внимание.

При выводе страницы с картинкой (метод `item`) в контекст шаблона под именем `item` заносится ассоциативный массив, содержащий все необходимые данные. Обращаться к модели из шаблона нам в этом случае не придется.

5. Откроем модуль маршрутизатора modules\router.php и исправим код, определяющий вызываемый контроллер по пути, полученному в составе запроса (правки выделены полужирным шрифтом):

```
if ($request_path == '') {
    $ctr = new \Controllers\Images();
    $ctr->list();
} else {
    $index = (integer)$request_path;
    $ctr = new \Controllers\Images();
    $ctr->item($index);
}
```

Мы создаем объект класса `Controllers\Images` и вызываем у него метод `list` или `item`.

Осталось написать шаблоны.

9.4. Упражнение. Создаем шаблоны

На основе уже имеющихся одноименных модулей напишем шаблон list.php, описывающий страницу со списком изображений, и шаблон item.php, который формирует страницу с выбранным изображением.

Оба модуля содержат два одинаковых фрагмента HTML-кода:

- самое начало, включающее пролог, секцию заголовка, открывающий тег `<body>` и заголовок первого уровня с названием сайта, — *начальный* код;
- самый конец — закрывающие теги `</body>` и `</html>` — *конечный* код.

Чтобы не дублировать их раз за разом в каждом последующем шаблоне, вынесем их в шаблоны-фрагменты: __header.inc.php и __footer.inc.php соответственно.

> *Шаблон-фрагмент* — шаблон, предназначенный для включения в состав другого шаблона.

Шаблоны-фрагменты традиционно сохраняются в файлах с расширением inc.php.

Для вычисления путей к шаблонам-фрагментам в модуле modules\helpers.php объявим функцию `\Helpers\get_fragment_path`, принимающую в качестве параметра имя включаемого шаблона-фрагмента и возвращающую полный путь к нему.

Файлы сайта фотогалереи можно найти в папке 9\9.3\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. В *разд. 9.3* мы условились, что сохраним шаблоны в папке modules\templates.

   Создадим папку modules\templates.

2. Переместим из папки modules в папку modules\templates файлы list.php и item.php.
3. Создадим в папке modules\templates модуль шаблона-фрагмента __header.inc.php и запишем в него общий для всех шаблонов начальный код (листинг 9.7).

Листинг 9.7. Модуль modules\templates__header.inc.php

```
<!doctype html>
<html>
```

```
<head>
    <meta charset="UTF-8">
    <link href="/styles.css" rel="stylesheet" type="text/css">
    <title><?php echo ((isset($site_title)) ?
    $site_title . ' :: ' : '') ?>
    <?php echo \Settings\SITE_NAME ?></title>
</head>
<body>
<h1><?php echo \Settings\SITE_NAME ?></h1>
```

В переменной `$site_title` будет храниться префикс для названия сайта. Если эта переменная существует, ее содержимое выводится в теге `<title>` перед названием, взятым из ранее созданной константы `Settings\SITE_NAME`, и отделяется от него двойным двоеточием.

Название сайта, выводящееся в теге `<h1>`, также берется из константы `Settings\SITE_NAME`.

4. Создадим в папке modules\templates модуль шаблона-фрагмента __footer.inc.php и занесем в него общий для всех шаблонов конечный код (листинг 9.8).

Листинг 9.8. Модуль modules\templates__footer.inc.php

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

5. Откроем модуль modules\templates\list.php в текстовом редакторе и запишем в него код шаблона списка изображений (листинг 9.9).

Листинг 9.9. Модуль modules\templates\list.php

```
<?php require \Helpers\get_fragment_path('__header') ?>
<section id="gallery">
    <?php
        for ($i = 0; $i < $cnt; $i++) {
            $img = \Models\Image::get_image($i);
    ?>
    <a href="/<?php echo $i ?>/">
        <img src="<?php echo \Settings\IMAGE_PATH . $img['src'] ?>"
        title="<?php echo $img['desc'] ?>">
    </a>
    <?php } ?>
</section>
<?php require \Helpers\get_fragment_path('__footer') ?>
```

В начале кода модуля не забываем выполнить включение шаблона с начальным кодом modules\templates__header.inc.php, а в конце — шаблона с конечным кодом

modules\templates__footer.inc.php. Полные пути к этим шаблонам мы получим, вызвав функцию Helpers\get_fragment_path, которую напишем позже.

Количество изображений берем из переменной $cnt, созданной функцией Helpers\render, а за списком изображений обращаемся непосредственно к модели Models\Image. Относительный путь к папке с изображениями берем из константы Settings\IMAGE_PATH.

6. Откроем модуль modules\templates\item.php в текстовом редакторе и запишем в него код шаблона выбранного изображения (листинг 9.10).

Листинг 9.10. Модуль modules\templates\item.php

```
<?php require \Helpers\get_fragment_path('__header') ?>
<h2><?php echo $item['desc'] ?></h2>
<p><a href="/">На главную</a></p>
<section id="gallery-item">
    <img src="<?php echo \Settings\IMAGE_PATH . $item['src'] ?>">
</section>
<p><a href="/">На главную</a></p>
<?php require \Helpers\get_fragment_path('__footer') ?>
```

Ассоциативный массив со сведениями об изображении берем из переменной $item, созданной функцией Helpers\render.

7. В методе item контроллера Controllers\Images, выводящем страницу с изображением, нужно добавить в контекст шаблона элемент site_title с описанием изображения. Это описание будет добавлено к названию страницы, выводимому в теге <title>.

Откроем файл modules\controllers\images.php и внесем в код контроллера Controllers\Images следующие правки (добавленный и исправленный код здесь и далее выделен полужирным шрифтом):

```
class Images extends BaseController {
    . . .
    function item(int $index) {
        $item = \Models\Image::get_image($index);
        $ctx = ['item' => $item, 'site_title' => $item['desc']];
        $this->render('item', $ctx);
    }
}
```

8. Осталось объявить в модуле modules\helpers.php функцию Helpers\get_fragment_path, которая будет формировать полный путь к файлу с заданным шаблоном.

Откроем модуль modules\helpers.php в текстовом редакторе и добавим объявление этой функции:

```
namespace Helpers {
```

```
    function get_fragment_path(string $fragment): string {
        global $base_path;
        return $base_path . '\modules\templates\\' . $fragment .
            '.inc.php';
    }
}
```

Проверим переделанный сайт в работе и убедимся, что он функционирует без сбоев.

Архитектура «модель-шаблон-контроллер» — для больших сайтов

Реализовывать ее имеет смысл, если разрабатываемый сайт достаточно велик — содержит более десятка страниц. В противном случае удобнее и быстрее программировать сайты «по старинке», как мы делали это на *уроке 1*.

Урок 10
Генераторы и итераторы

PHP предлагает развитые средства для обработки массивов (часть из них была рассмотрена на *уроке 4*). А еще этот язык позволяет превратить в нечто подобное массивам обычные функции и объекты.

10.1. Генераторы

> *Генератор* — функция, возвращающая при каждом вызове очередной элемент заданной последовательности.

Генератор объявляется так же, как и обычная функция (см. *разд. 6.1*). Но для возврата результата используется языковая конструкция `yield`:

```
yield <возвращаемое значение>;
```

Конструкция `return` (также описанная в *разд. 6.1*) прерывает выполнение функции полностью, и при последующем вызове функция начинает выполняться с начала. Напротив, конструкция `yield` лишь приостанавливает выполнение функции, возобновляемое при последующем вызове.

> Конструкция `yield`, а следовательно, и сам генератор возвращают в качестве результата объект класса `Generator`, встроенного в PHP. Этот класс объявлен в корневом пространстве имен (о пространствах имен рассказывалось в *разд. 8.3*).

Генератор может быть подвергнут перебору в цикле по массиву (см. *разд. 5.4.4*), как обычный массив. К сожалению, указывать его в вызовах функций, обрабатывающих массивы (и описанных в *разд. 4.5*), нельзя, поскольку в генераторе отсутствует необходимая для этого функциональность.

Пример генератора `int_gen`, выдающего последовательность целых чисел от `$begin` до `$end` с шагом `$step` (если шаг не указан, принимается равным `1`):

```
function int_gen(int $begin, int $end, int $step = 1): Generator {
    if ($end > $begin)
        for ($i = $begin; $i <= $end; $i += $step)
            yield $i;
    else
        for ($i = $begin; $i >= $end; $i += $step)
            yield $i;
}
```

Использование генератора `int_gen`:

```
$gen1 = int_gen(1, 10);
foreach ($gen1 as $el)
    echo $el . ' ';                                  // 1 2 3 4 5 6 7 8 9 10

$gen2 = int_gen(1, 20, 2);
foreach ($gen2 as $el)
    echo $el . ' ';                                // 1 3 5 7 9 11 13 15 17 19

$gen3 = int_gen(100, 50, -5);
foreach ($gen3 as $el)
    echo $el . ' ';                  // 100 95 90 85 80 75 70 65 60 55 50
```

10.2. Итераторы

Итератор — объект, имеющий функциональность массива.

Итератор можно подвергнуть перебору в цикле по массиву (см. *разд. 5.4.4*), а также выяснить количество имеющихся в нем элементов, вызвав функцию `count`.

Класс объекта-итератора (*итерируемый класс*) должен реализовать ряд магических методов, объявленных в двух рассматриваемых далее интерфейсах.

10.2.1. Интерфейс *Iterator*

Интерфейс `Iterator` — обеспечивает перебор объекта в цикле по массиву. Он объявлен в корневом пространстве имен.

Класс объекта-итератора должен реализовать этот интерфейс и объявить имеющиеся в нем магические методы.

Таких методов пять:

- `current()` — возвращает текущий элемент;
- `key()` — возвращает ключ текущего элемента;
- `next()` — сдвигает внутренний указатель на следующий элемент;
- `rewind()` — возвращает внутренний указатель на первый элемент;
- `valid()` — возвращает `TRUE`, если текущий элемент действителен, и `FALSE`, если элементов больше нет. Вызывается после вызовов методов `next` и `rewind`.

Пример класса итератора `IntIter`, выдающего последовательность целых чисел от `$begin` до `$end` с шагом `$step` (если шаг не указан, принимается равным `1`):

```
class IntIter implements Iterator {
    private $begin = 0;
    private $end = 0;
    private $step = 1;
    private $current_index = 0;
    private $current_value = 0;
```

```
    function __construct(int $begin, int $end, int $step = 1) {
        $this->begin = $begin;
        $this->end = $end;
        $this->step = $step;
        $this->rewind();
    }

    function current() {
        return $this->current_value;
    }

    function key() {
        return $this->current_index;
    }

    function next() {
        $this->current_index++;
        $this->current_value += $this->step;
    }

    function rewind() {
        $this->current_index = 0;
        $this->current_value = $this->begin;
    }

    function valid() {
        if ($this->end > $this->begin)
            return ($this->current_value >= $this->begin &&
                $this->current_value <= $this->end);
        else
            return ($this->current_value <= $this->begin &&
                $this->current_value >= $this->end);
    }
}
```

Использование итератора `IntIter`:

```
$iter1 = new IntIter(1, 10);
foreach ($iter1 as $el)
    echo $el . ' ';                                  // 1 2 3 4 5 6 7 8 9 10

$iter2 = new IntIter(100, 10, -7);
foreach ($iter2 as $el)
    echo $el . ' ';              // 100 93 86 79 72 65 58 51 44 37 30 23 16
```

10.2.2. Интерфейс *Countable*

> Интерфейс `Countable` — обеспечивает получение количества элементов в итераторе посредством функции `count`. Он объявлен в корневом пространстве имен.

Класс объекта-итератора должен реализовать этот интерфейс и объявить имеющийся в нем метод `count()`, возвращающий количество элементов.

Пример реализации интерфейса `Countable` в классе `IntIter`:

```
class IntIter implements Iterator, Countable {
    . . .
    function count() {
        return abs(intdiv($this->end - $this->begin, $this->step)) + 1;
    }
}
```

Примеры получения количества элементов в объектах этого класса:

```
$iter1 = new IntIter(1, 10);
echo count($iter1);                                    // 10

$iter2 = new IntIter(100, 10, -7);
echo count($iter2);                                    // 13
```

10.3. Упражнение. Превращаем модель в итерируемый класс

Написанная на *уроке 9* модель `Models\Image` требует обращаться из шаблона непосредственно к ее статическим методам, что не приветствуется в архитектуре «модель-шаблон-контроллер». Превратим эту модель в итерируемый класс — и тогда сможем перебрать ее объект в цикле, как обычный массив.

Файлы сайта фотогалереи можно найти в папке 9\9.4\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Откроем модуль modules\models\image.php с кодом модели `Models\Image` и добавим в объявление класса следующий код (добавления и правки в написанном ранее коде здесь и далее выделены полужирным шрифтом):

```
class Image implements \Iterator {
    . . .
    private $current_index = 0;

    function current() {
        return self::DATA[$this->current_index];
    }

    function key() {
        return $this->current_index;
    }
```

```
    function next() {
        $this->current_index++;
    }

    function rewind() {
        $this->current_index = 0;
    }

    function valid() {
        return isset(self::DATA[$this->current_index]);
    }
}
```

После языковой конструкции `implements` следует указать полный путь к интерфейсу `Iterator`: `\Iterator` (в противном случае PHP попытается выполнить его автозагрузку, что приведет к фатальной ошибке).

2. Удалим из класса модели `Models\Image` не нужный более статический метод `get_count` (удаленный код здесь и далее зачеркнут):

```
class Image implements \Iterator {
    . . .
    static function get_count(): int {
        return count(self::DATA);
    }
    . . .
}
```

3. Откроем модуль modules\controllers\images.php с кодом контроллера `Controllers\Images` и внесем следующие правки:

```
class Images extends BaseController {
    function list() {
        $ctx = ['list' => new \Models\Image()];
        $this->render('list', $ctx);
    }
    . . .
}
```

Мы поместили в контекст шаблона под именем `list` объект класса модели.

4. Откроем файл modules\templates\list.php с шаблоном списка изображений и внесем следующие правки:

```
<section id="gallery">
    <?php
        for ($i = 0; $i < $cnt; $i++) {
        foreach ($list as $i => $img) {
            $img = \Models\Image::get_image($i);
    ?>
    . . .
</section>
```

Осталось только проверить, как работает сайт.

Урок 11
Регулярные выражения

В веб-программировании часто приходится проверять строки на соответствие какому-либо шаблону или искать в них совпадающие с шаблоном фрагменты. Такого рода операции лучше выполнять с помощью регулярных выражений.

> *Регулярное выражение* — шаблон, применяемый для поиска совпадающих с ним фрагментов текста и проверки на совпадение с ним заданных величин. Может включать в себя как обычные символы (которые обрабатываются как есть), так и литералы.

> *Литерал регулярного выражения* — символ или последовательность символов, имеющих особое значение.

Литерал может обозначать символ определенной категории (например, любую букву, любую цифру, пробельный символ, букву из заданного набора и пр.), указывать количество повторений символа и др. Существует несколько разновидностей литералов.

11.1. Введение в регулярные выражения

11.1.1. Написание регулярных выражений

> В PHP регулярные выражения записываются в виде строк в формате:
> */<собственно шаблон>/*[*<модификаторы>*]

> *Модификатор* — обозначение специфического режима обработки регулярного выражения, присутствие которого активирует этот режим. В качестве модификаторов используются буквы латиницы.

Созданное регулярное выражение присваивается какой-либо переменной.

Пример регулярного выражения:

```
$rex = '/PHP/';
```

Шаблон этого регулярного выражения содержит только обычные символы и совпадет с любой последовательностью «PHP». Поиск совпадающих фрагментов ведется *с учетом регистра символов*:

```
// Здесь и далее фрагменты строк, совпадающие с регулярным выражением,
// выделены рамками
                // Результат поиска: PHP PHP7 php MySQL
```

Еще одно регулярное выражение

```
$rex = '/PHP\d/';
```

Литерал `\d`, присутствующий в нем, совпадает с любой цифрой от 0 до 9. Следовательно, весь шаблон совпадет с любым фрагментом вида: «PHP<*цифра*>».

```
// Результат: PHP PHP7 php MySQL
```

Модификатор `i` включает поиск *без учета регистра* символов.

Пример поиска без учета регистра:

```
$rex = '/PHP/i';                    // Результат: PHP PHP7 php MySQL
```

Модификатор `u` предписывает при поиске обрабатывать и строку, в которой выполняется поиск, и сам шаблон в *кодировке UTF-8*. Этот модификатор необходимо использовать при обработке строк на русском языке.

Пример поиска в кодировке UTF-8:

```
$rex = '/Язык/u';
// Результат: Язык программирования PHP. Это отличный язык!
```

В регулярном выражении можно указать несколько модификаторов, записав их вплотную друг к другу.

Пример указания нескольких модификаторов (поиск в кодировке UTF-8 без учета регистра):

```
$rex = '/Язык/ui';
// Результат: Язык программирования PHP. Это отличный язык!
```

11.1.2. Поиск с применением регулярных выражений

Поиск фрагментов строк, совпадающих с регулярными выражениями, можно выполнить вызовом функции `preg_match`. Вот наиболее простой формат ее вызова:

```
preg_match(<регулярное выражение>, <строка>)
```

Возвращаемый результат: `1` — в случае успеха, `0` — если совпадающий фрагмент не был найден, или `FALSE` — если в регулярном выражении была обнаружена ошибка:

```
$str = 'Язык программирования PHP. Это отличный язык!';
$rex1 = '/PHP/u';
echo preg_match($rex1, $str);                    // 1
$rex2 = '/HTML/u';
echo preg_match($rex2, $str);                    // 0
```

Более подробно функцию `preg_match` мы рассмотрим в *разд. 11.3.1*.

11.1.3. Тестирование регулярных выражений

Для тестирования шаблонов регулярных выражений можно использовать интернет-службу **Мастер составления регулярного выражения для PHP** (рис. 11.1), дос-

тупную по адресу: **https://uvsoftium.ru/php/regexp_all.php**. В поле ввода **Регулярное выражение** заносится шаблон регулярного выражения, в поле, расположенном правее, — модификаторы, а в область редактирования **Тестовые строки** — обрабатываемая строка. Совпадающие фрагменты, найденные в обрабатываемой строке, указываются справа от области редактирования.

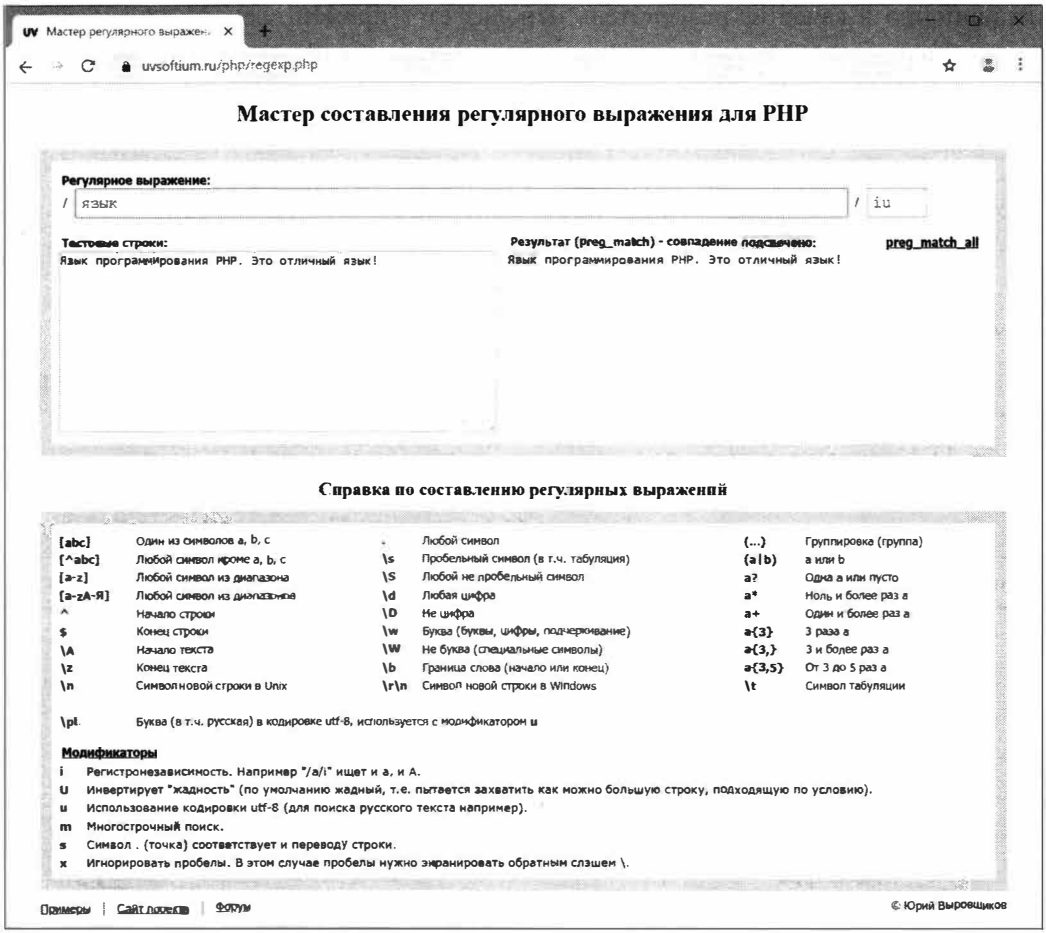


Рис. 11.1. Интернет-служба **Мастер составления регулярного выражения для PHP**

По умолчанию в заданной строке ищется первый фрагмент, совпавший с указанным регулярным выражением, после чего поиск прекращается. Чтобы найти все совпадающие фрагменты, достаточно щелкнуть на гиперссылке **preg_match_all**, расположенной под полем для ввода модификаторов. Чтобы вернуться к поиску первого совпавшего фрагмента, нужно щелкнуть на гиперссылке **preg_match**.

В нижней части страницы находится краткий справочник по литералам, используемым в регулярных выражениях PHP.

11.2. Литералы регулярных выражений

11.2.1. Разделители

> *Разделитель* — литерал, ограничивающий шаблон регулярного выражения слева и справа.

Традиционно в качестве разделителя используется прямой слеш /, однако можно применять любые символы, отличные от букв, цифр, обратного слеша \ и пробела.

Примеры:

```
// Разделитель – символ решетки #
echo preg_match('#A#', 'ABCDE');                    // 1
// Разделитель – тильда ~
echo preg_match('~\d~', 'PHP7');                    // 1
```

11.2.2. Метасимволы

> *Метасимвол* — литерал, обозначающий любой символ из какой-либо группы или символ с указанным кодом.

В табл. 11.1 приведен список поддерживаемых метасимволов.

***Таблица 11.1.** Список поддерживаемых метасимволов*

Литерал	Описание	Пример	Результат
. (точка)	Любой символ, кроме возврата каретки (\r) и перевода строки (\n)	/./	PHP!
\w	Алфавитно-цифровой символ (буква, цифра или подчеркивание)	/\w/	PHP!
\W	Не алфавитно-цифровой символ	/\W/	PHP!
\d	Цифра	/\d/	PHP7
\D	Не цифра	/\D/	PHP7
\s	Пробельный символ (пробел, табуляция, возврат каретки или перевод строки)	/\s/	PHP MySQL
\S	Не пробельный символ	/\S/	PHP MySQL
\b	Начало или конец слова (позволяет указать, что какой-либо символ должен находиться в начале слова или его конце)	/H/ /\bH/	HTML PHP HTML PHP
\B	Не начало и не конец слова	/\BH/	HTML PHP
\x{<*код*>}	Символ с указанным Unicode-кодом. Используется с модификатором u	/\x{043c}/u	Код символа
\R	Разрыв строки (символы \r, \n или последовательность \r\n)		

Примеры:

```
// Ищем отдельное слово вида: «PHP<цифра>»
$rex = '/\bPHP\d\b/u';
echo preg_match($rex, 'Язык PHP7');                    // 1
echo preg_match($rex, 'Язык PHP');                     // 0
// Ищем название языка программирования из четырех букв
$rex = '/\b\w\w\w\w\b/i';
echo preg_match($rex, 'PHP Python Ruby');              // 1
// Проверяем, присутствует ли в строке разрыв
$rex = '/\R/';
echo preg_match($rex, "PHP\nMySQL");                   // 1
```

11.2.3. Поднаборы

Поднабор — литерал, обозначающий любой символ из заданного набора.

В табл. 11.2 приведен список поддерживаемых поднаборов.

***Таблица 11.2.** Список поддерживаемых поднаборов*

Литерал	Описание	Пример	Результат
[<*символы*>]	Любой из *символов*, приведенных в скобках	/[MPQ]/	PHP MySQL
[^<*символы*>]	Любой из *символов*, *не* приведенных в скобках	/[^MPQ]/	PHP MySQL
[<*c1*>-<*c2*>]	Любой из символов из диапазона от *c1* до *c2*	/[P-T]/	PHP MySQL
[^<*c1*>-<*c2*>]	Любой из символов *не* из диапазона от *c1* до *c2*	/[^P-T]/	PHP MySQL

Примеры:

```
$rex = '/[RST][i-m][^a-w]/';
echo preg_match($rex, 'Sky');                          // 1
echo preg_match($rex, 'Tin');                          // 0
```

11.2.4. Вариант

Вариант — литерал, обозначающий любой фрагмент из перечисленных.

Он записывается в формате:

<фрагмент 1>|<фрагмент 2>| . . . |<фрагмент n-1>|<фрагмент n>

В состав *фрагментов* включаются все символы и литералы, расположенные между вертикальными линиями, началом и концом регулярного выражения.

Примеры:

```
// Ищем фрагменты «jpg», «jpeg» или «jpe»
$rex = '/jpg|jpeg|jpe/i';
```

```
echo preg_match($rex, 'logo.jpg');                    // 1
echo preg_match($rex, 'background.jpeg');             // 1
echo preg_match($rex, 'archive.zip');                 // 0
// Ищем либо «PHP<цифра>», либо «MySQL»
$rex = '/PHP\d|MySQL/u';
echo preg_match($rex, 'Язык PHP7');                   // 1
echo preg_match($rex, 'Изучайте MySQL!');             // 1
echo preg_match($rex, 'Язык PHP');                    // 0
```

11.2.5. Квантификаторы

> *Квантификатор* — литерал, указывающий количество повторений или местоположение символа.

Вот все поддерживаемые квантификаторы:

- <*символ*>+ — *символ* должен присутствовать в произвольном количестве экземпляров:

```
$rex = '/\bPH+P\b/';              // Результат: PP PHP PHHP PHHHHHHHHP
```

- <*символ*>* — *символ* может присутствовать в произвольном количестве экземпляров или вообще отсутствовать:

```
$rex = '/\bPH*P\b/';              // Результат: PP PHP PHHP PHHHHHHHHP
```

- <*символ*>? — *символ* может присутствовать в одном экземпляре или отсутствовать:

```
$rex = '/\bPH?P\b/';              // Результат: PP PHP PHHP PHHHHHHHHP
```

- <*символ*>{<*n*>} — *символ* должен присутствовать в количестве *n* экземпляров:

```
$rex = '/\bPH{2}P\b/';            // Результат: PP PHP PHHP PHHHHHHHHP
```

- <*символ*>{<*n*>,} — *символ* должен присутствовать в количестве не менее *n* экземпляров:

```
$rex = '/\bPH{2,}P\b/';           // Результат: PP PHP PHHP PHHHHHHHHP
```

- <*символ*>{<*n1*>,<*n2*>} — *символ* должен присутствовать в количестве от *n1* до *n2* экземпляров:

```
$rex = '/\bPH{1,2}P\b/';          // Результат: PP PHP PHHP PHHHHHHHHP
```

- ^<*символ*> — *символ* должен находиться в начале строки:

```
$rex = '/^Py/';                   // Результат: Python PyQT\r\nPyCharm
```

- \A<*символ*> — если указан модификатор `m` (будет рассмотрен далее), *символ* должен находиться в начале текста, в противном случае — аналогичен квантификатору ^;

- <*символ*>$ — *символ* должен находиться в конце строки:

```
$rex = '/jpg$/';          // Результат: image.jpg\r\nmap.jpg picture.jpg
```

- *<символ>*\z — если указан модификатор m (будет рассмотрен далее), *символ* должен находиться в конце текста, в противном случае — аналогичен квантификатору $;
- *<символ 1>*(?=*<символ 2>*) — за *символом 1* должен следовать *символ 2*, причем последний не включается в состав фрагмента, совпадающего с регулярным выражением:

```
// Ищем последовательность цифр, запятых и точек, за которой следует
// произвольное количество пробелов и фрагмент «руб»,
// то есть цену в рублях.
// Чтобы поместить в регулярное выражение символ точки, предварим
// его обратным слешем. Если этого не сделать, веб-обозреватель
// посчитает точку за литерал ., совпадающий с любым символом.
$rex = '/[\d,\.]+(?=\s*руб)/u';
                        // Результат: 12 [12] руб [34,67]руб [89.8] руб 89.8
```

- *<символ 1>*(?!*<символ 2>*) — за *символом 1 не* должен следовать *символ 2*, причем последний не включается в состав фрагмента, совпадающего с регулярным выражением:

```
// Ищем произвольное количество цифр, за которыми не следует
// произвольное количество букв, то есть «чистые» числа
$rex = '/\d+(?!\w+)/u';          // Результат: [10] 12abc [6790]
```

11.2.6. Подквантификатор. «Жадный» и «щедрый» режимы поиска

По умолчанию квантификаторы + и * работают так, чтобы совпавшей с регулярным выражением оказалась строка максимальной длины (*«жадный» режим поиска*).

В качестве примера рассмотрим регулярное выражение, которое совпадает с последовательностью любых символов, взятых в апострофы:

```
$rex = "/'.*'/";          // Результат: PHP ['HTML' 'CSS' 'JavaScript'] MySQL
```

Видно, что регулярное выражение, работая в «жадном» режиме, «захватило» самый длинный фрагмент — расположенный между первым и последним апострофами. Чтобы оно «захватывало», напротив, наиболее короткий фрагмент, его следует переключить в *«щедрый» режим поиска*.

> *Подквантификатор* — литерал, указываемый после квантификатора и изменяющий режим его работы.

Подквантификатор ?, указанный после квантификаторов + и *, переключает поиск в «щедрый» режим:

```
$rex = "/'.*?'/";         // Результат: PHP ['HTML'] ['CSS'] ['JavaScript'] MySQL
```

Будучи указанным в другом месте регулярного выражения, символ вопросительного знака считается квантификатором ?, описанным в *разд. 11.2.5*.

11.2.7. Группы и обратные ссылки

Группа — часть регулярного выражения, ведущая себя как один символ.

PHP поддерживает три вида групп:

- `(?:<шаблон>)`:

```
// Регулярное выражение, совпадающее с фрагментами «j pg», «j peg»
// и «j pe». При написании группа не использовалась.
$rex = '/j pg|j peg|j pe/';
// То же самое регулярное выражение, но записанное с применением
// группы. Получилось короче.
$rex = '/j p(?:g|eg|e)/';
```

- `(<шаблон>)` — фрагмент, совпавший с *шаблоном*, запоминается исполняющей средой и может быть извлечен с помощью обратной ссылки (будет описана далее):

```
// Регулярное выражение, совпадающее с последовательностью цифр,
// после которой присутствует фрагмент «руб».
// Последовательность цифр заключена в группу второго типа и,
// следовательно, будет запомнена исполняющей средой
// (на примере далее выделена черной заливкой).
$rex = '/(\d+)руб/';                    // Результат: Цена: 5 руб
```

Обратная ссылка — литерал регулярного выражения, ссылающийся на фрагмент, который совпадает с определенной группой второго вида.

Обратная ссылка на группу второго вида записывается в одном из следующих форматов:

```
\<порядковый номер группы второго вида>
\g<порядковый номер группы второго вида>
\g{<порядковый номер группы второго вида>}
```

Группы второго вида в регулярном выражении нумеруются в порядке слева направо, начиная с 1.

Далее приведен пример регулярного выражения, совпадающего с любым парным тегом. Внутри него созданы две группы: первая — для имени тега, вторая — для его содержимого. В записи закрывающего тега применяется обратная ссылка, хранящая имя тега из первой группы. Чтобы поместить в регулярное выражение символ прямого слеша, мы предварили его обратным слешем:

```
$rex = '/<(\w+)>(.*?)<\/\1>/';      // Результат: <strong>HTML</strong>
```

- `(?P<<имя>><шаблон>)` — то же самое, что группа второго вида, только обращение к ней выполняется не по порядковому номеру, а по заданному *имени*. Также поддерживаются альтернативные форматы `(?<<имя>><шаблон>)` и `(?'<имя>'<шаблон>)`.

Обратная ссылка на группу третьего вида записывается в одном из следующих форматов:

```
(?P=<имя группы третьего вида>)
\k<<имя группы третьего вида>>
\k'<имя группы третьего вида>'
\k{<имя группы третьего вида>}
\g{<имя группы третьего вида>}
\g<<имя группы третьего вида>>
\g'<имя группы третьего вида>'
```

Пример:

```
$rex = '/<(?P<tag>\w+)>(.*?)<\/\k<tag>>/';
                                    // Результат: <strong>HTML</strong>
```

11.2.8. Обычные символы

Все прочие символы в шаблонах регулярных выражений обозначают сами себя.

Если требуется вставить в шаблон символ, совпадающий с каким-либо литералом, его следует предварить знаком обратного слеша \:

```
// Точка в шаблоне регулярного выражения — это литерал,
// обозначающий любой символ
$rex2 = '/bhv.ru/';                 // Результат: bhv.ru bhv@ru bhv ru
// Чтобы вставить в шаблон символ точки, его нужно предварить
// обратным слешем
$rex3 = '/bhv\.ru/';                // Результат: bhv.ru bhv@ru bhv ru
// Чтобы вставить в регулярное выражение прямой слеш, обратный слеш или
// звездочку, обозначающих различные литералы, их также предваряют
// обратным слешем: \/, \\, \*
```

11.3. Поиск и обработка фрагментов, совпадающих с регулярными выражениями

11.3.1. Обычный режим поиска

> В *обычном режиме*, используемом по умолчанию, поиск останавливается после нахождения первого фрагмента, совпадающего с заданным регулярным выражением.

Обычный поиск первого фрагмента в *строке*, совпадающего с *регулярным выражением*, выполняет уже знакомая нам функция `preg_match`. Вот полный формат ее вызова:

```
preg_match(<регулярное выражение>, <строка>[, <массив для результатов>[,
          <режим>]])
```

Функция возвращает `1` в случае успеха, `0` — если подходящий фрагмент не был найден, или `FALSE` — если в регулярном выражении была обнаружена ошибка.

Примеры:

```
$str = 'Язык программирования PHP. Это отличный язык!';
$rex = '/PHP/u';
echo preg_match($rex, $str);                    // 1
$rex = '/MySQL/u';
echo preg_match($rex, $str);                    // 0
```

Если был указан *массив для результатов*, он будет заполнен результатами поиска. Что это будут за результаты, зависит от указанного *режима*:

- 0 — первый элемент *массива* будет хранить фрагмент, совпавший с самим регулярным выражением, второй элемент — фрагмент, совпавший с первой группой, и т. д. (поведение по умолчанию, когда *режим* не задан):

```
$rex = '/(P)(H)*(P)/';
$arr = [];
$str = 'PHP';
preg_match($rex, $str, $arr);
echo $arr[0];                                   // PHP
echo $arr[1];                                   // P
echo $arr[2];                                   // H
echo $arr[3];                                   // P
```

Если для какой-либо группы совпавший фрагмент не найден, соответствующий элемент массива будет хранить пустую строку.

```
$str = 'PP';
preg_match($rex, $str, $arr);
echo $arr[0];                                   // PHP
echo $arr[1];                                   // P
echo $arr[2];                                   // пустая строка
echo $arr[3];                                   // P
```

- PREG_UNMATCHED_AS_NULL — то же самое, только элементы массива, соответствующие группам, для которых не были найдены совпавшие фрагменты, будут хранить NULL:

```
preg_match($rex, $str, $arr, PREG_UNMATCHED_AS_NULL);
echo $arr[0];                                   // PHP
echo $arr[1];                                   // P
echo $arr[2];                                   // NULL
echo $arr[3];                                   // P
```

- PREG_OFFSET_CAPTURE — каждый элемент заданного *массива* будет хранить вложенный массив с двумя элементами: совпавшим фрагментом и номером символа в *строке*, с которого начинается совпавший фрагмент:

```
$str = 'PHP';
preg_match($rex, $str, $arr, PREG_OFFSET_CAPTURE);
echo $arr[0][0], ' - ', $arr[0][1];             // PHP - 0
echo $arr[1][0], ' - ', $arr[1][1];             // P - 0
```

```
echo $arr[2][0], ' - ', $arr[2][1];                    // H - 1
echo $arr[3][0], ' - ', $arr[3][1];                    // P - 2
```

Элементы массива, соответствующего группе, для которой не был найден совпавший элемент, будут хранить пустую строку и число -1 соответственно.

Если в регулярном выражении присутствуют группы третьего вида (именованные), соответствующие элементы массива будут доступны через ключи, совпадающие с именами групп:

```
$rex = '/(?P<first>P)(H)*(?P<last>P)/';
$str = 'PHP';
preg_match($rex, $str, $arr);
echo $arr[0];                                          // PHP
echo $arr['first'];                                    // P
echo $arr[2];                                          // H
echo $arr['last'];                                     // P
```

11.3.2. Глобальный поиск

> При *глобальном поиске* исполняющая среда ищет в строке все совпадающие фрагменты.

Глобальный поиск выполняется функцией preg_match_all:

```
preg_match_all(<регулярное выражение>, <строка>[,
               <массив для результатов>[, <режим>]])
```

Функция возвращает количество фрагментов, совпавших с регулярным выражением, или FALSE — если в регулярном выражении была обнаружена ошибка.

Примеры:

```
$str = 'Язык программирования PHP. Изучайте PHP!';
$rex = '/PHP/u';
echo preg_match_all($rex, $str);                       // 2
$rex = '/MySQL/u';
echo preg_match_all($rex, $str);                       // 0
```

Если был указан *массив для результатов*, он будет заполнен результатами поиска. Что это будут за результаты, зависит от указанного *режима*:

- PREG_PATTERN_ORDER — первый элемент *массива* будет хранить вложенный массив фрагментов, совпавших с самим регулярным выражением, второй — массив фрагментов, совпавших с первой группой, третий — совпавших со второй группой, и т. д. (поведение по умолчанию, если *режим* не указан):

```
$rex = '/(P)(H)*(P)/';
$arr = [];
$str = 'PHP PP';
preg_match_all($rex, $str, $arr);
```

```
echo implode(', ', $arr[0]);                    // PHP, PP
echo implode(', ', $arr[1]);                    // P, P
echo implode(', ', $arr[2]);                    // H, пустая строка
echo implode(', ', $arr[3]);                    // P, P
```

- PREG_UNMATCHED_AS_NULL — то же самое, только элементы массива, соответствующие группам, для которых не были найдены совпавшие фрагменты, будут хранить NULL;
- PREG_OFFSET_CAPTURE — каждый элемент вложенного массива будет хранить вложенный массив с двумя элементами: самим совпавшим фрагментом и номером символа в *строке*, с которого он начинается:

```
preg_match_all($rex, $str, $arr, PREG_OFFSET_CAPTURE);
echo $arr[0][0][0], ' - ', $arr[0][0][1];          // PHP - 0
echo $arr[0][1][0], ' - ', $arr[0][1][1];          // PP - 4
echo $arr[1][0][0], ' - ', $arr[1][0][1];          // P - 0
echo $arr[1][1][0], ' - ', $arr[1][1][1];          // P - 4
echo $arr[2][0][0], ' - ', $arr[2][0][1];          // H - 1
echo $arr[2][1][0], ' - ', $arr[2][1][1];        // пустая строка - -1
echo $arr[3][0][0], ' - ', $arr[3][0][1];          // P - 2
echo $arr[3][1][0], ' - ', $arr[3][1][1];          // P - 5
```

- PREG_SET_ORDER — первый элемент *массива* будет содержать вложенный массив с первым набором вхождений, второй элемент — массив со вторым набором, и т. д.:

```
preg_match_all($rex, $str, $arr, PREG_SET_ORDER);
echo implode(', ', $arr[0]);                       // PHP, P, H, P
echo implode(', ', $arr[1]);                 // PP, P, пустая строка, P
```

Если в регулярном выражении присутствуют группы третьего вида (именованные), соответствующие элементы вложенных массивов будут доступны через ключи, совпадающие с именами групп.

11.3.3. Многострочный поиск

> При *многострочном поиске* квантификаторы \A и \z обозначают начало и конец всего текста, а не отдельной строки (квантификаторы ^ и $ все так же обозначают начало и конец отдельной строки).

Отдельные строки в строковом значении разделяются либо символом \n, либо комбинацией символов \r\n:

```
$str = 'Первая строка\r\nВторая строка\r\nТретья строка';
```

Многострочный поиск включается указанием в регулярном выражении модификатора m.

Примеры:

```
$rex = '/\APy/';          // Результат: Python PyQT\r\nPyCharm
$rex = '/\APy/m';         // Результат: Python PyQT\r\nPyCharm
```

```
$rex = '/jpg\z/';          // Результат: image.jpg\r\nmap.jpg picture.jpg
$rex = '/jpg\z/m';         // Результат: image.jpg\r\nmap.jpg picture.jpg
```

11.3.4. Замена совпавших фрагментов

Замену *заменяющей подстрокой* фрагментов *строки*, совпавших с *регулярным выражением*, выполняет функция `preg_replace`:

```
preg_replace(<регулярное выражение>, <заменяющая подстрока>, <строка>[,
             <количество замен>[,
             <переменная для количества замененных фрагментов>]])
```

Если *количество замен* не указано или задано значение `-1`, будут заменены все найденные фрагменты.

Вместо *строки* можно указать массив строк — тогда замена будет выполнена в каждой строке массива.

Функция возвращает строку или массив строк, в которых были выполнены замены.

Примеры:

```
$rex = '/PHP\d?/iu';
$n = 0;
$str = 'PHP PHP5 PHP3 php';
$str = preg_replace($rex, 'PHP7', $str, -1, $n);
echo $str;                                          // PHP7 PHP7 PHP7 PHP7
// Выясним количество замененных фрагментов
echo $n;                                              // 4
$arr = ['PHP', 'PHP5', 'PHP3', 'php'];
$arr = preg_replace($rex, 'PHP7', $arr);
echo implode(' ', $arr);                            // PHP7 PHP7 PHP7 PHP7
```

В *заменяющей подстроке* можно использовать следующие специализированные литералы, работающие только здесь:

- `${0}`, `$0` или `\\0` — обозначает весь фрагмент, совпавший с *регулярным выражением*;
- `${<n>}`, `$<n>` или `\\<n>` — фрагмент, совпавший с группой с порядковым номером *n* (то есть обратная ссылка);
- `\\\\` — символ обратного слеша.

Примеры:

```
$rex = '/PHP(\d)/iu';
$str = 'php5 pHp3 PhP7';
$str = preg_replace($rex, 'PHP${1}', $str);
echo $str;                                            // PHP5 PHP3 PHP7
```

Вместо *регулярного выражения* можно указать массив из регулярных выражений. В таком случае *заменяющая подстрока* заменит фрагменты, совпадающие со всеми регулярными выражениями из этого массива:

```
$rex = ['/PHP(\d)/iu', '/PHP\s(\d)/iu'];
$str = 'php5 pHp3 PhP7 php 5';
$str = preg_replace($rex, 'PHP${1}', $str);
echo $str;                                          // PHP5 PHP3 PHP7 PHP5
```

Если при этом и вместо *заменяющей подстроки* указать массив подстрок, то фрагмент, совпавший с первым регулярным выражением, будет заменен первой заменяющей подстрокой из массива, фрагмент, совпавший с вторым регулярным выражением, — второй подстрокой и т. д. Если заменяющих подстрок меньше, чем регулярных выражений в первом массиве, фрагменты, совпавшие с «лишними» регулярными выражениями, будут заменены пустыми строками (фактически удалены):

```
$rex = ['/PHP(\d)/iu', '/PHP\s(\d)/iu', '/Python/iu'];
$str = 'php5 python pHp3 PYTHON PhP7 php 5 Python PHP 1';
$str = preg_replace($rex, ['PHP${1}', 'PHP ${1}'], $str);
echo $str;                              // PHP5  PHP3  PHP7 PHP 5  PHP 1
```

11.3.5. Разбиение строк

Для разбиения указанной *строки* на фрагменты по *разделителю*, заданному регулярным выражением, применяется функция `preg_split`:

```
preg_split(<разделитель>, <строка>[, <количество фрагментов>[, <режим>]])
```

Если *количество фрагментов* не указано, задано равным `-1` или `0`, возвращаются все фрагменты, полученные в результате разбиения *строки*.

Функция возвращает массив, содержащий полученные фрагменты.

Примеры:

```
$rex = '/[\s,;]/';
$str = 'PHP,MySQL;Python Ruby';
$arr = preg_split($rex, $str);
echo $arr[0];                                       // PHP
echo $arr[1];                                       // MySQL
echo $arr[2];                                       // Python
echo $arr[3];                                       // Ruby
$arr = preg_split($rex, $str, 3);
echo $arr[0];                                       // PHP
echo $arr[1];                                       // MySQL
echo $arr[2];                                       // Python Ruby
```

В качестве *режима* можно указать следующие значения:

- `PREG_SPLIT_NO_EMPTY` — пустые фрагменты не включаются в возвращаемый массив:

```
$str = 'PHP,,MySQL';
$arr = preg_split($rex, $str);
echo $arr[0];                                       // PHP
```

```
echo $arr[1];                                          // пустая строка
echo $arr[2];                                          // MySQL
$arr = preg_split($rex, $str, -1, PREG_SPLIT_NO_EMPTY);
echo $arr[0];                                          // PHP
echo $arr[1];                                          // MySQL
```

- PREG_SPLIT_OFFSET_CAPTURE — каждый элемент возвращаемого массива будет хранить вложенный массив с двумя элементами: фрагментом и номером символа в строке, с которого он начинается:

```
$str = 'PHP,MySQL;Python Ruby';
$arr = preg_split($rex, $str, -1, PREG_SPLIT_OFFSET_CAPTURE);
echo $arr[0][0], ' - ', $arr[0][1];                    // PHP - 0
echo $arr[1][0], ' - ', $arr[1][1];                    // MySQL - 4
echo $arr[2][0], ' - ', $arr[2][1];                    // Python - 10
echo $arr[3][0], ' - ', $arr[3][1];                    // Ruby - 17
```

Полезно знать...

PHP поддерживает и другой, более старый, формат записи регулярных выражений и набор функций для работы с ними. Однако этот формат и эти функции объявлены устаревшими и не рекомендованными к применению.

11.4. Упражнение. Пишем маршрутизатор на основе регулярных выражений

Маршрутизатор, написанный нами для сайта фотогалереи, пока еще очень прост. Но в дальнейшем, когда мы добавим в состав сайта новые контроллеры, возникнут сложности с написанием условных выражений, проверяющих путь на соответствие заданным шаблонам. Уменьшить сложность кода нам позволят регулярные выражения.

Перепишем маршрутизатор, использовав регулярные выражения для сравнения пути, полученного в составе запроса, с заданными шаблонами.

Файлы сайта фотогалереи можно найти в папке 10\10.3\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Откроем модуль маршрутизатора modules\router.php в текстовом редакторе и перепишем его код согласно листингу 11.1.

Листинг 11.1. Модуль modules\router.php

```
<?php
$request_path = $_GET['route'];
if ($request_path && $request_path[-1] == '/')
    $request_path = substr($request_path, 0,
        strlen($request_path) - 1);
$result = [];
```

```
if (preg_match('/^(\d+)$/', $request_path, $result) === 1) {
    $index = (integer)$result[1];
    $ctr = new \Controllers\Images();
    $ctr->item($index);
} else if ($request_path == '') {
    $ctr = new \Controllers\Images();
    $ctr->list();
} else {
}
```

Символ слеша в конце полученного пути может как присутствовать, так и отсутствовать. Для простоты удалим этот слеш (если он присутствует, и если путь не пуст), выделив из пути подстроку, начинающуюся с первого символа и заканчивающуюся непосредственно перед слешем, и заменив ей «старый» путь.

Сначала проверим, соответствует ли полученный путь формату ***<числовой индекс>***/, и, если это так, выведем страницу показа изображения с полученным *индексом*.

Формат ***<числовой индекс>***/ задается регулярным выражением с шаблоном `^(\d+)$`. Отметим, что в его начале стоит литерал `^`, обозначающий начало строки, а в конце — литерал конца строки `$`. Это нужно для того, чтобы проверялось совпадение шаблона со всем путем, а не с его фрагментами (так мы в дальнейшем избежим ситуации, когда шаблон совпадет с путем формата, скажем, ***<индекс>***/edit/, и на экране вместо страницы правки изображения появится страница для его просмотра).

Если полученный путь пуст, выведем страницу со списком изображений. Если же путь не совпадает с регулярным выражением и не пуст, мы не ничего не будем делать. На *уроке 12* мы напишем код, который в таком случае выведет страницу с сообщением об ошибке 404.

Проверим переделанный сайт в действии и убедимся, что все работает.

Урок 12
Обработка ошибок. Исключения

Фатальная ошибка приведет к аварийному останову программы, результирующая веб-страница не будет сгенерирована, и мы не увидим сообщение об ошибке, вставленное в ее HTML-код. Но мы можем написать код, реализующий другую, нужную нам, реакцию на такие ошибки, оформив его как обработчик исключения.

> *Исключение* — объект, создаваемый исполняющей средой PHP в случае возникновения любой фатальной ошибки и хранящий сведения о ней.

Не все фатальные ошибки порождают исключения!

Ошибки, возникающие при компиляции кода, в частности при разрешении констант и выполнении включений посредством конструкций `require` и `require_once`, не приводят к возникновению исключений.

Нефатальные ошибки также не порождают исключения!

Поэтому всегда следует внимательно проверять, все ли на сгенерированной странице выведено как надо, и нет ли в ее коде «спрятанных» сообщений о нефатальных ошибках.

Исключение также может быть сгенерировано самим программистом, чтобы уведомить приложение о какой-либо нештатной ситуации.

> *Обработка исключений* — выполнение, в случае возникновения исключения, его *обработчика* — фрагмента кода, производящего какие-либо действия с целью устранить последствия ошибки или уведомить о ней посетителя сайта (например, выводящего страницу с сообщением).

12.1. Обработчики исключений

Обработчик исключений записывается в следующем формате:

```
try
    <блок try>          // код, в котором может возникнуть исключение
catch (<класс исключения 1> <переменная для объекта исключения>)
    <блок catch 1>      // выполняется при возникновении исключения,
                        // относящегося к классу 1
catch (<класс исключения 2> <переменная для объекта исключения>)
    <блок catch 2>      // выполняется при возникновении исключения,
                        // относящегося к классу 2
```

```
catch (<класс исключения n> <переменная для объекта исключения>)
    <блок catch n>      // выполняется при возникновении исключения,
                        // относящегося к классу n
[finally
    <блок finally>]     // выполняется всегда, независимо от того,
                        // возникло исключение или нет
```

> *Переменная для объекта исключения*, указанная в круглых скобках после языковой конструкции `catch`, доступна только внутри соответствующего *блока catch*.

Пример:

```
try {
    // Намеренно выполняем деление на 0, чтобы вызвать возникновение
    // исключения класса DivisionByZeroError
    echo 24 % 0;
} catch (DivisionByZeroError $e) {
    // Обрабатываем исключение класса DivisionByZeroError
    echo "Деление на 0!\r\n";
    // Метод getMessage, вызываемый у исключения, возвращает описание
    // возникшей ошибки
    echo $e->getMessage(), "\r\n";
} finally {
    // Этот код выполнится вне зависимости от того, возникло исключение
    // или нет
    echo 'Опасный участок пройден';
}
                                    // Результат: Деление на 0!
                                    //            Modulo by zero
                                    //            Опасный участок пройден
```

> В круглых скобках после конструкции `catch` можно указать сразу несколько классов исключений, записав их через знак вертикальной линии `|`.

Пример обработки исключений, принадлежащих классам `ArithmeticError` и `TypeError` (будут рассмотрены в *разд. 12.2*):

```
try {
    . . .
} catch (ArithmeticError | TypeError $e) {
    . . .
}
```

12.2. Классы исключений

Встроенные в PHP классы исключений приведены в табл. 12.1. Все они объявлены в корневом пространстве имен.

Таблица 12.1. Встроенные классы исключений PHP

Класс	Описание	Базовый класс
`Error`	Все фатальные ошибки PHP, за исключением указанных далее	
`ArithmeticError`	Ошибки в математических вычислениях (например, побитовый сдвиг на отрицательное количество разрядов)	`Error`
`DivisionByZeroError`	Деление на 0	`ArithmeticError`
`CompileError`	Некоторые ошибки компиляции	`Error`
`ParseError`	Ошибки в PHP-коде, переданном функции `eval` (см. *разд. 2.7*)	`CompileError`
`TypeError`	Несоответствие типа значения параметра в вызове функции типу, указанному в ее объявлении	`Error`
`ArgumentCountError`	Передача функции недостаточного количества параметров	`TypeError`
`Exception`	Базовый класс для всех исключений, создаваемых самим программистом	

Из объекта исключения можно извлечь сведения о возникшей ошибке, применив следующие методы, которые поддерживаются всеми классами исключений:

- текстовое описание ошибки — метод `getMessage()`;
- путь к модулю, в котором возникла ошибка, — метод `getFile()`;
- порядковый номер строки кода, в которой возникла ошибка, — метод `getLine()`.

Пример:

```
try {
    . . .
} catch (Error $e) {
    echo $e->getMessage(), "\r\n";
    echo 'Файл: ', $e->getFile(), "\r\n";
    echo 'Строка №: ', $e->getLine();
}
```

Можно создавать свои классы исключений, делая их производными от класса `Exception`:

```
class Page404Exception extends Exception {}
```

Как правило, дополнительных свойств и методов в новых классах исключений объявлять не нужно — достаточно унаследованных от базового класса.

12.3. Генерирование исключений

Исключения, являющиеся производными от класса `Error`, генерируются исполняющей средой PHP. Исключения, объявленные программистом, должны генерироваться самим программистом.

> Генерирование исключения выполняется языковой конструкцией `throw`:
>
> `throw <объект исключения>;`

При создании объекта исключения единственным необязательным параметром может быть указана строка с сообщением о возникшей ошибке.

Пример генерирования исключения `Page404Exception`:

```
throw new Page404Exception('Такая веб-страница не существует');
```

Сгенерированное таким образом исключение может быть обработано так же, как и встроенное в PHP;

```
try {
    . . .
    throw new Page404Exception();
} catch (Page404Exception $e) {
    // Обрабатываем исключение класса Page404Exception
}
```

12.4. Обработчик исключений по умолчанию

> *Обработчик исключений по умолчанию* — срабатывает, если исключение возникло вне блока `try` (см. *разд. 12.1*).

Обычно используется обработчик по умолчанию, встроенный в PHP и выводящий сообщение об ошибке в HTML-код генерируемой страницы. Однако мы можем указать свой обработчик.

> Задание обработчика исключений по умолчанию выполняется вызовом функции `set_exception_handler`:
>
> `set_exception_handler(<имя функции-обработчика исключений>)`

Имя функции-обработчика указывается в виде строки. Если указать `NULL`, будет установлен обработчик по умолчанию, встроенный в PHP.

Функция-обработчик исключений должна принимать в качестве параметра объект исключения и не возвращать результата.

Пример:

```
function exception_handler($e) {
    echo '<p class="error">', $e->getMessage(), '</p>';
}
set_exception_handler('exception_handler');
```

Обработчик исключений по умолчанию позволяет сосредоточить обработку исключений, возникающих во всем веб-приложении, в каком-либо одном модуле, что очень удобно.

Полезно знать...

В старых версиях PHP, не поддерживающих исключения, применялись обработчики ошибок по умолчанию, создаваемые аналогичным образом.

12.5. Упражнение. Создаем веб-страницы с сообщениями об ошибках

Для сайта фотогалереи создадим две веб-страницы, сообщающие о следующих ошибках:

- ошибка 404 — обращение к несуществующей странице (в нашем случае — попытка вывода изображения с несуществующим индексом);
- ошибка 503 — программный сбой в веб-приложении.

Для вывода страницы с сообщением об ошибке 404 объявим класс исключения `Page404Exception`, которое будем генерировать при обращении к изображению с несуществующим индексом. Ради простоты объявим исключение в корневом пространстве имен. Напишем также контроллер `Controllers\Error` с методами `page404` и `page503`, шаблоны **404.php** и **503.php**, а также обработчик исключений по умолчанию.

Файлы сайта фотогалереи хранятся в папке **11\11.4\ex** сопровождающего книгу электронного архива (см. *приложение 3*).

1. Начнем с объявления класса исключения `Page404Exception` и задания своего обработчика исключений по умолчанию. Необходимый код поместим в модуль единой точки входа.

   Откроем модуль единой точки входа **index.php** в текстовом редакторе и впишем сразу после кода, задающего обработчик автозагрузки классов, объявления класса исключения, функции-обработчика исключений по умолчанию и вызов функции `set_exception_handler` (дополнения и правки в ранее написанном коде здесь и далее выделены полужирным шрифтом):

```
spl_autoload_register('my_autoloader');

class Page404Exception extends Exception {}

function exception_handler($e) {
    $ctr = new \Controllers\Error();
    if ($e instanceof Page404Exception)
        $ctr->page404();
    else
        $ctr->page503($e);
}
set_exception_handler('exception_handler');
```

В теле функции создаем объект контроллера `Controllers\Error` (его напишем уже скоро) и проверяем, создан ли объект исключения на основе класса `Page404Exception`. Если это так, вызываем метод `page404` контроллера, в противном случае — метод `page503`, передав ему объект исключения в качестве параметра.

2. Создадим модуль **modules\controllers\error.php** и поместим в него код контроллера `Controllers\Error` (листинг 12.1).

Листинг 12.1. Модуль modules\controllers\error.php

```
<?php
namespace Controllers;
class Error extends BaseController {
    function page404() {
        $this->render('404', []);
    }

    function page503($e) {
        $ctx = ['message' => $e->getMessage(),
                'file' => $e->getFile(),
                'line' => $e->getLine()];
        $this->render('503', $ctx);
    }
}
```

Метод `page404` выведет страницу с сообщением об ошибке 404. Никаких вычисленных программно сведений на ней выводить не нужно, поэтому мы передаем шаблону пустой контекст.

Метод `page503` выведет страницу с сообщением об ошибке 503. Здесь мы передаем шаблону в составе контекста описание ошибки (`message`), путь к модулю, в котором она возникла (`file`), и номер строки кода (`line`).

3. Создадим модуль **modules\templates\404.php** с кодом шаблона страницы 404 (листинг 12.2).

Листинг 12.2. Модуль modules\templates\404.php

```
<?php require \Helpers\get_fragment_path('__header') ?>
<h2>Страница не найдена</h2>
<p>Запрошенный интернет-адрес не существует.</p>
<p><a href="/">На главную</a></p>
<?php require \Helpers\get_fragment_path('__footer') ?>
```

4. Создадим модуль **modules\templates\503.php** с кодом шаблона страницы 503 (листинг 12.3).

Листинг 12.3. Модуль modules\templates\503.php

```
<?php require \Helpers\get_fragment_path('__header') ?>
<h2>Внутренняя ошибка сервера</h2>
<pre><?php echo $message ?>

<?php echo $file, ', line ', $line ?></pre>
<p><a href="/">На главную</a></p>
<?php require \Helpers\get_fragment_path('__footer') ?>
```

Сведения об ошибке выводим моноширинным шрифтом в теге `<pre>` — это своего рода стандарт для страниц подобного рода. Путь к модулю и номер строки кода, где возникла ошибка, отделяем от описания ошибки пустой строкой.

5. Исправим маршрутизатор таким образом, чтобы при обращении по ошибочному интернет-адресу выводилась страница ошибки 404.

   Откроем модуль маршрутизатора modules\router.php и внесем необходимые дополнения:

```
if (preg_match('/^(\d+)\/$/', $request_path, $result) === 1) {
    . . .
} else if ($request_path == '') {
    . . .
} else {
    throw new Page404Exception();
}
```

6. Исправим модель `Models\Image`, чтобы при запросе изображения с несуществующим индексом выводилась страница ошибки 404. Для этого нам нужно внести правки в код статического метода `get_image`.

   Откроем модуль modules\models\image.php с кодом модели `Models\Image` и внесем нужные правки:

```
class Image implements \Iterator {
    . . .
    static function get_image(int $index): array {
        if (key_exists($index, self::DATA))
            return self::DATA[$index];
        else
            throw new \Page404Exception();
    }
    . . .
}
```

Для проверки существования в массиве `DATA` запрашиваемого индекса мы применили функцию `key_exists` (см. *разд. 4.5*).

7. Проверим, как сайт реагирует на обращение по ошибочному интернет-адресу.

   Запросим вывод изображения с очень большим индексом — например: 123456, перейдя по интернет-адресу: **http://localhost/123456/**. В ответ должна появиться страница с сообщением об ошибке 404 (рис. 12.1).

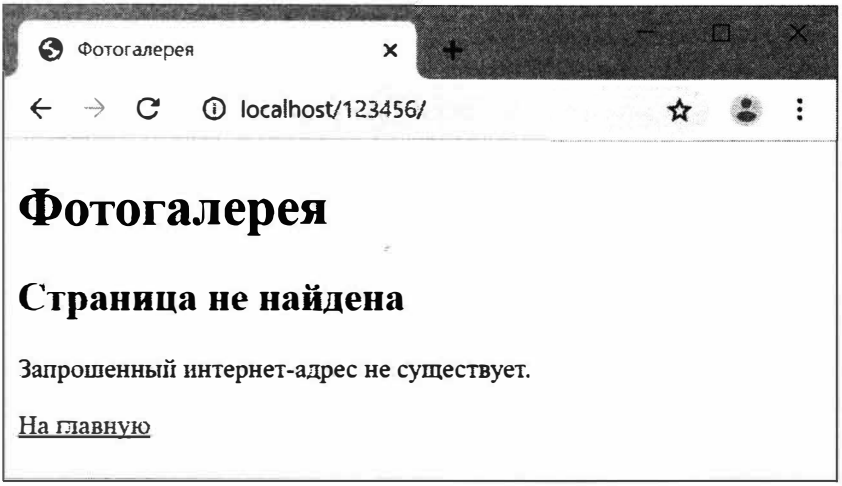


Рис. 12.1. Веб-страница с сообщением об ошибке 404

8. Запросим вывод изображения с некорректным индексом — например, строковым: «abcde». Перейдем по интернет-адресу **http://localhost/abcde/** — в ответ также будет выведена страница ошибки 404.

9. Проверим, как сайт реагирует на ошибки в собственном программном коде.

   Откроем модуль modules\controllers\images.php с кодом контроллера `Controllers\Images` и внесем ошибку в метод `item`:

```
class Images extends BaseController {
    . . .
    function item(int $index) {
        $item = \Models\Image::get_imag($index);
        $ctx = ['item' => $item, 'site_title' => $item['desc']];
        $this->render('item', $ctx);
    }
}
```

   Вместо метода `get_image` модели `Models\Image` теперь вызывается несуществующий метод `get_imag`.

10. Перейдем на главную страницу сайта, набрав интернет-адрес: **http://localhost/**, и щелкнем на каком-либо изображении. В ответ появится страница ошибки 503 с описанием возникшей ошибки (рис. 12.2).

11. Вернем код метода `item` класса `Controllers\Images` к первоначальному виду и еще раз проверим работу сайта.

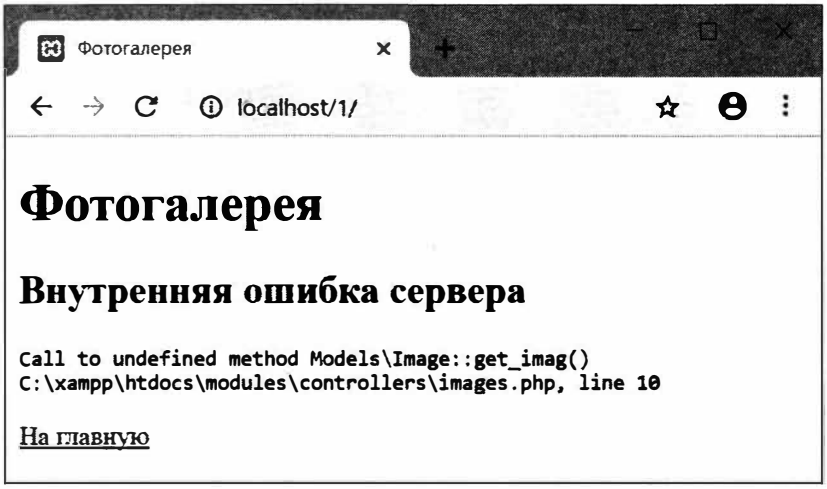


Рис. 12.2. Веб-страница с сообщением об ошибке 503

ЧАСТЬ II

Система управления базами данных MySQL

- *Урок 13*. Базы данных
- *Урок 14*. Написание запросов к базам данных. Язык SQL
- *Урок 15*. Выполнение запросов к базам данных MySQL средствами PHP

Урок 13
Базы данных

> *База данных* — организованная структура, предназначенная для хранения, изменения и обработки взаимосвязанной информации, преимущественно больших объемов.

Базы данных используются в составе веб-сайтов, обрабатывающих большие объемы данных: порталов, интернет-магазинов, корпоративных сайтов. Веб-приложения, составляющие такие сайты, извлекают данные из базы и генерируют страницы на их основе.

Существует несколько разновидностей баз данных, из которых в настоящее время наиболее популярной является реляционная.

> *Реляционная база данных* — база данных, которая организует хранящиеся в ней данные в виде набора связанных друг с другом таблиц.

В реляционных базах данных удобно хранить списки каких-либо сущностей: товаров, записей о продажах, электронных статей, изображений, опубликованных в фотогалерее, и пр.

> *Система управления базами данных* (СУБД) — комплекс программ, предназначенных для создания баз данных, их ведения и совместного использования многими пользователями.

Примеры СУБД: MySQL, MariaDB, Oracle, Microsoft SQL Server, PostgreSQL, Microsoft Access.

Преимущества использования баз данных для хранения информации, содержащейся в сайтах:

- компактность;
- мощные инструменты для работы с данными, предоставляемые СУБД;
- универсальность — база данных может использоваться не только веб-сайтом, но и другими программами (например, бухгалтерской и складской).

Полезно знать...

Существуют и другие разновидности баз данных: объектные, объектно-реляционные, иерархические, сетевые и др. Как правило, они используются в очень специфических приложениях и относительно редко.

13.1. Введение в реляционные базы данных

13.1.1. Таблицы, поля и записи

Реляционная база данных состоит из отдельных таблиц, каждая из которых хранит перечень сущностей какого-либо одного типа.

Каждая таблица имеет уникальное в пределах базы данных имя, по которому выполняется обращение к ней. К именам таблиц предъявляются те же требования, что и к именам переменных PHP (см. *разд. 3.1.1*).

На рис. 13.1 схематично показана база данных с тремя таблицами: `users`, `pictures` и `comments`.

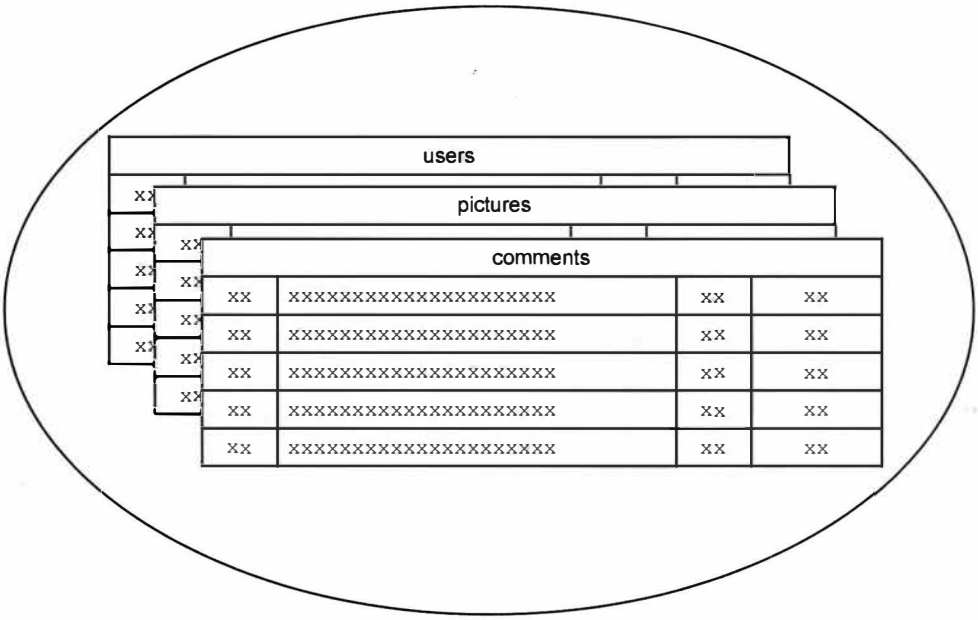


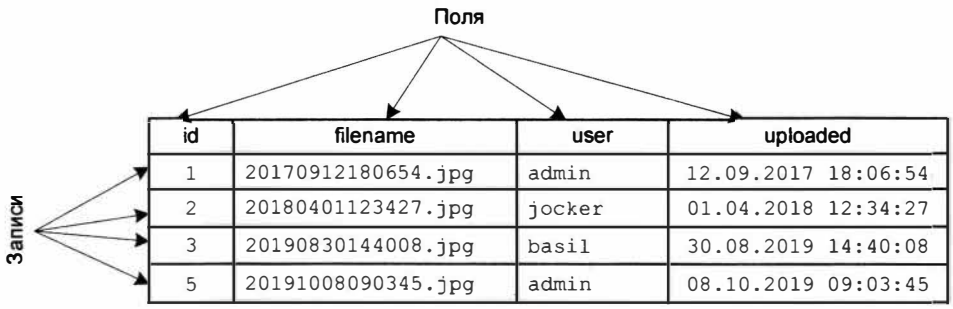
Рис. 13.1. Схематичное изображение базы данных с тремя таблицами

Запись — отдельная строка в таблице базы данных. Хранит одну сущность.

Поле — отдельный столбец в таблице базы данных. Содержит одну из характеристик сущности, сохраненной в записи.

Каждое поле имеет уникальное в пределах таблицы имя, по которому выполняется обращение к нему.

На рис. 13.2 схематично показана таблица `pictures`, хранящая список опубликованных в фотогалерее изображений. Она содержит поля `id` (уникальный номер записи), `filename` (имя файла с изображением), `user` (имя пользователя, опубликовавшего изображение) и `uploaded` (дата и время публикации). В таблице `pictures` хранятся четыре записи.



id	filename	user	uploaded
1	20170912180654.jpg	admin	12.09.2017 18:06:54
2	20180401123427.jpg	jocker	01.04.2018 12:34:27
3	20190830144008.jpg	basil	30.08.2019 14:40:08
5	20191008090345.jpg	admin	08.10.2019 09:03:45

Рис. 13.2. Таблица `pictures` с четырьмя полями и четырьмя записями

О номерах записей

Номера, хранящиеся в поле `id`, необязательно будут порядковыми, последовательно увеличивающимися на единицу. В них возможны «промежутки», которые могут образоваться после удаления каких-либо записей. На рис. 13.2 показан пример такого «промежутка» между записями № 3 и 5 — запись № 4 была удалена.

> Каждое поле может хранить значение строго определенного типа: строковое, целочисленное, вещественное, логическое, временну́ю отметку, большой фрагмент текста и др.

Некоторые типы данных имеют несколько разновидностей. Например, существуют четыре разновидности целочисленного типа: байт, короткое, длинное и сверхдлинное целое. Поле типа «сверхдлинное целое» способно хранить сколь угодно большое целое число, однако занимает объем в 8 байтов, а поле типа «байт» может хранить числа от 0 до 255, но занимает всего один байт. Это позволяет разработчику выбрать наиболее подходящий тип данных для поля.

Знаковые и беззнаковые целые числа

Для полей целочисленных типов можно указать, какие числа они могут хранить: знаковые или беззнаковые. Так, знаковое поле типа «байт» может хранить числа от –128 до 127, а беззнаковое поле — числа от 0 до 255.

> *Автоинкрементное поле* — поле, в которое при создании новых записей заносятся последовательно увеличивающиеся целые числа. Такие поля применяются для сохранения в записях порядковых номеров, по которым впоследствии можно идентифицировать эти записи.

Традиционно автоинкрементные поля с номерами записей носят имя `id` — в нашей таблице `pictures` тоже есть поле `id`.

Для поля можно задать значение по умолчанию, которое будет занесено при создании новой записи, если значение для поля явно указано не было.

Любому полю можно дать возможность хранить также значение `NULL`, обозначающее отсутствие любого «значащего» значения. Однако на практике это необходимо только в очень редких случаях.

13.1.2. Индексы

Хранящиеся в таблице записи могут иметь весьма большой объем (так, популярная СУБД MySQL позволяет хранить записи объемом до 64 килобайт). В результате, например, при выборке записей, в поле `user` которых хранится значение `jocker`, СУБД придется по частям считывать всю таблицу в память и последовательно просматривать все записи, что может отнять много времени.

> *Индекс* — компактный массив, составленный из упорядоченных значений какого-либо поля таблицы и ссылок на записи, в которых хранятся эти значения. Служит для ускорения поиска, фильтрации и сортировки записей.

При наличии индекса, составленного по полю `user`, СУБД быстро считает его в память, найдет в нем ячейки с искомым значением и по хранящимся там же ссылкам извлечет нужные записи.

На рис. 13.3 показан индекс `user_idx`, составленный по значениям поля `user` таблицы `pictures` и упорядоченный по алфавиту.

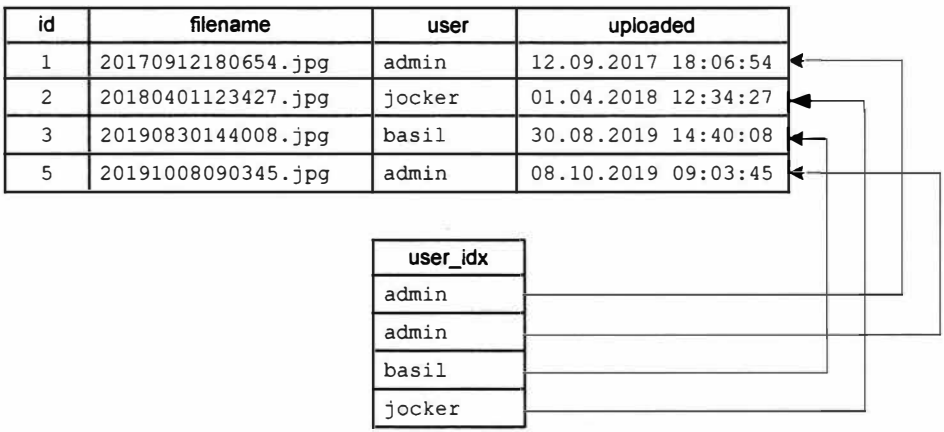


Рис. 13.3. Индекс `user_idx`, составленный по полю `user`

> *Составной индекс* — индекс, составленный по значениям двух или большего количества полей.

Составной индекс может пригодиться, если часто выполняется сортировка записей сразу по нескольким полям. Например, если записи в таблице `pictures` часто сортируются по полям `user` и `uploaded`, имеет смысл создать по этим полям составной индекс.

> *Уникальный индекс* — индекс, содержащий только уникальные значения. При попытке добавить в него повторяющееся значение, СУБД откажется сохранять запись и сообщит об ошибке.

Создав уникальный индекс по какому-либо полю, мы укажем, что это поле может хранить лишь неповторяющиеся значения. Это пригодится, скажем, при хранении списка зарегистрированных пользователей, чьи имена должны быть уникальны.

> *Ключевой индекс* (*ключ*) — уникальный индекс, составленный по полю, в котором хранятся какие-либо значения, однозначно идентифицирующие записи в таблице (*ключевое поле*). Практически всегда в качестве ключевого используется автоинкрементное поле с номерами записей.

Например, в таблице `pictures` по полю `id` можно создать ключевой индекс.

Ключевые индексы используются, чтобы гарантировать уникальность значений какого-либо поля, а также для установления межтабличных связей, о которых мы скоро поговорим.

> ***Используйте индексы с умом!***
>
> Индексы следует создавать лишь по тем полям, по которым часто выполняются фильтрация и сортировка записей. Не следует забывать, что каждый индекс занимает определенный объем в базе данных и отнимает время на модификацию при добавлении, правке и удалении записей.

13.1.3. Связи

В поле `user` таблицы `pictures` содержатся имена пользователей, опубликовавших изображения, и те же самые имена хранятся в таблице `users` (рис. 13.4).

pictures			
id	filename	user	uploaded_at
1	20170912180654.jpg	admin	12.09.2017 18:06:54
2	20180401123427.jpg	jocker	01.04.2018 12:34:27
3	20190830144008.jpg	basil	30.08.2019 14:40:08
5	20191008090345.jpg	admin	08.10.2019 09:03:45

users	
id	name
1	admin
3	jocker
4	basil

Рис. 13.4. Структура таблиц неоптимальна: имена пользователей хранятся и в поле `user` таблицы `pictures`, и в поле `name` таблицы `users`

Это неоптимально по двум причинам. Во-первых, одно и то же значение, находясь в двух местах, лишь зря занимает пространство на диске. Во-вторых, если понадобится исправить имя какого-либо пользователя, придется править его в двух таблицах: `pictures` и `users`.

Устранить обе проблемы можно, если записывать в поле `user` таблицы `pictures` не имя пользователя, а номер соответствующей записи таблицы `users` (рис. 13.5). Так мы создадим связь между двумя таблицами.

pictures			
id	filename	user	uploaded
1	20170912180654.jpg	1	12.09.2017 18:06:54
2	20180401123427.jpg	3	01.04.2018 12:34:27
3	20190830144008.jpg	4	30.08.2019 14:40:08
5	20191008090345.jpg	1	08.10.2019 09:03:45

users	
id	name
1	admin
3	jocker
4	basil

Рис. 13.5. Структура таблиц оптимальна: в поле `user` таблицы `pictures` хранятся номера, а не имена пользователей

> *Связь* устанавливается между записями первичной и вторичной таблицы. С одной записью *первичной таблицы* может быть связано произвольное количество записей во *вторичной таблице* (связь *«один-ко-многим»*).

У нас таблица списка пользователей `users` — первичная, а таблица изображений `pictures` — вторичная. Один пользователь может быть связан с произвольным количеством опубликованных им изображений.

> *Первичный ключ* — значение ключевого поля в первичной таблице (обычно поле уникального номера записи `id`).

> *Внешний ключ* — номер записи первичной таблицы, сохраненный в связанной записи вторичной таблицы, в *поле внешнего ключа*.

У нас поле `id` таблицы `users` хранит первичные ключи, а в таблице `pictures` имеется поле внешнего ключа `user`.

Связь указывает, что поле внешнего ключа может хранить только первичные ключи. Попытка записать в поле внешнего ключа значение, не совпадающее ни с одним первичным ключом, приведет к ошибке. Она также задает поведение при удалении записей первичной таблицы и изменении первичного ключа.

> *Каскадное удаление* — при удалении записи первичной таблицы производится автоматическое удаление связанных записей вторичной таблицы.

В нашем случае, если активно каскадное удаление, при удалении пользователя будут удалены все изображения, опубликованные этим пользователем.

> *Каскадное изменение* — при изменении первичного ключа осуществляется синхронная замена в полях внешнего ключа связанных записей вторичной таблицы.

В нашем случае, если активно каскадное изменение, при правке номера пользователя он будет автоматически заменен в полях внешнего ключа опубликованных им изображений, и связи между ними не разорвутся.

Рекомендуется разрешать и каскадное изменение, и каскадное удаление

Это пригодится на тот случай, если потребуется исправить значение ключевого поля в записи первичной таблицы.

Полезно знать...

Реляционные СУБД также поддерживают установление связей более редко используемых типов: «один-к-одному» и «многие-ко-многим».

13.1.4. Разграничение доступа

Ради обеспечения безопасности почти все СУБД реализуют разграничение доступа, позволяя подключаться к базам данных только «знакомым» им пользователям.

> *Разграничение доступа* — управление доступом пользователей к базе данных в зависимости от того, удовлетворяют ли они определенным требованиям.

В число этих требований входят следующие:

- пользователь должен иметь учетную запись в списке пользователей.

  > *Учетная запись* — сведения о пользователе: имя, пароль, сведения о привилегиях (о них — чуть позже) и служебные данные (например, интернет-адрес, с которого он может подключаться к СУБД).

  > *Список пользователей* хранит учетные записи пользователей, которым разрешено подключаться к базам данных, обслуживаемым текущей СУБД;

- пользователь должен иметь привилегии на работу с базой данных, к которой он подключается.

  > *Привилегии* — правила, определяющие операции, которые пользователь может выполнять с различными базами данных: считывание записей, их создание, правку и удаление, создание, правку, удаление таблиц, индексов, связей, самих баз данных и других пользователей.

  Привилегии могут назначаться на уровне самой СУБД (и, соответственно, действовать на все обслуживаемые ей базы данных — это *глобальные привилегии*), отдельной базы данных (*привилегии уровня баз данных*) и даже отдельной таблицы в базе данных (применяется редко);

- пользователь после подключения к СУБД должен пройти процедуру входа.

  > При выполнении *процедуры входа* пользователь вводит свои имя и пароль, СУБД ищет в списке учетную запись с такими именем и паролем и, если находит, допускает пользователя к работе. Если подходящей учетной записи в списке нет, подключения к базе данных не происходит;

- закончив работу с базой данных, пользователь должен выполнить процедуру выхода.

> После выполнения *процедуры выхода* СУБД «забудет», что такой пользователь ранее произвел вход, и не даст ему выполнить ни одной операции с базами данных. Чтобы получить доступ к базам, пользователь снова должен выполнить вход.

При выполнении выхода освобождаются системные ресурсы, расходуемые СУБД на «запоминание» пользователя, выполнившего вход, и сведений о нем (в первую очередь, привилегий). Поэтому по окончании работы с базой данных рекомендуется выходить из нее.

> *Администратор* — пользователь, имеющий максимальные привилегии на доступ либо ко всем базам данных, обрабатываемых текущей СУБД (*администратор СУБД*), либо только одной из баз данных (*администратор базы данных*).

Администратор создает и правит базы данных, таблицы, индексы и связи в них, следит за корректностью данных, хранящихся в базах, и выполняет задачи по их обслуживанию (например, резервное копирование). Требования безопасности предписывают, чтобы администраторов было как можно меньше, а лучше всего, если администратор будет всего один.

Полезно знать...

Список пользователей обычно хранится в таблице *системной базы данных*, используемой самой СУБД и недоступной для пользователей. Помимо этого, в системной базе хранится перечень всех баз данных, обслуживаемых текущей СУБД.

13.1.5. Серверные СУБД

> *Серверная СУБД* (*сервер баз данных*) — взаимодействует лишь с базами данных, но не с конечными пользователями. Серверная СУБД выдает хранящуюся в базах информацию другим программам — клиентам, возможно, работающим на других компьютерах и представляющим информацию пользователям.

В веб-программировании программой-клиентом выступает веб-сайт, обрабатывающий полученную от СУБД информацию и преобразующий ее в веб-страницы.

Все серверные СУБД в обязательном порядке реализуют разграничение доступа.

13.2. Серверные СУБД MySQL и MariaDB

> *MySQL* — свободно распространяемая серверная реляционная СУБД. В настоящее время является наиболее популярной в веб-программировании.

> *MariaDB* — свободно распространяемая серверная реляционная СУБД, ответвление от MySQL. Была создана, чтобы преодолеть некоторые ограничения в лицензировании MySQL. Полностью совместима с последней.

MariaDB === MySQL

Далее в книге речь будет идти о MySQL, однако все сказанное также справедливо и для MariaDB.

Наиболее востребованные типы данных, поддерживаемые MySQL, приведены в табл. 13.1.

Таблица 13.1. *Наиболее востребованные типы данных MySQL*

Название	Описание
TINYINT	Байт, целые числа от –128 до 127 (знаковое) или от 0 до 255 (беззнаковое)
SMALLINT	Короткое целое, числа от –32768 до 32767 (знаковое) или от 0 до 65535 (беззнаковое)
MEDIUMINT	Целое средней длины, числа от –8388608 до 8388607 (знаковое) или от 0 до 16777215 (беззнаковое)
INT	Длинное целое, числа от –2147483648 до 2147483647 (знаковое) или от 0 до 4294967295 (беззнаковое)
BIGINT	Сверхдлинное целое, числа от –9223372036854775808 до 9223372036854775807 (знаковое) или от 0 до 18446744073709551615 (беззнаковое)
FLOAT	Вещественное одинарной точности, числа от $-3{,}402823466^{38}$ до $-1{,}175494351^{-38}$, 0 и от $1{,}175494351^{-38}$ до $3{,}402823466^{38}$
DOUBLE	Вещественное двойной точности, числа от $-1{,}7976931348623157^{308}$ до $-2.2250738585072014^{-308}$, 0 и от $2{,}2250738585072014^{-308}$ до $1{,}7976931348623157^{308}$
DECIMAL	Вещественное заданной длины и с заданным количеством знаков после десятичной запятой. Часто применяется для хранения денежных сумм
VARCHAR	Строка фиксированной длины. Длина строки — не более 65535 символов — указывается при создании поля
TINYTEXT	Короткая строка переменной длины — до 255 символов
TEXT	Средняя строка переменной длины — до 65535 символов
MEDIUMTEXT	Длинная строка переменной длины — до 16777215 символов
LONGTEXT	Сверхдлинная строка переменной длины — до 4294967295 символов
BOOLEAN	Логическая величина
TIMESTAMP	Временна́я отметка в формате PHP, то есть количество секунд, прошедшее с «начала эры UNIX»

Пояснения к табл. 13.1:

- числовые поля, способные хранить бóльшие числа, занимают бóльший объем, нежели поля, способные хранить меньшие числа. Например, поле типа SMALLINT занимает объем 2 байта, а поле типа INT — уже 4 байта;
- поле типа VARCHAR (строка фиксированной длины) всегда занимает объем, равный указанной при создании поля длине строки, независимо от реальной длины

строкового значения, которое в нем сохранено. Создание поля слишком большой длины приведет к значительному увеличению объема базы данных. Поэтому поля фиксированной длины не рекомендуется применять для хранения больших фрагментов текста;

- поля типа TINYTEXT, TEXT, MEDIUMTEXT или LONGTEXT (строка переменной длины) занимают объем, строго равный длине хранящегося в поле значения. Такие поля являются более «экономными», нежели VARCHAR (хоть и обрабатываются медленнее), и прекрасно подходят для хранения длинных текстов;
- значения полей типа TIMESTAMP при извлечении преобразуются в строки формата:

  `<год>-<номер месяца от 1 до 12>-<число> <часы>:<минуты>:<секунды>`

  Такую строку можно преобразовать во временну́ю отметку PHP вызовом функции strtotime (см. *разд. 2.4.1*);
- у поля типа TIMESTAMP в качестве значения по умолчанию можно указать текущие дату и время.

Полезно знать...

MySQL также поддерживает тип данных CHAR. Это такие же строки фиксированной длины, как и типа VARCHAR, но в них слишком короткие строковые значения дополняются для достижения нужной длины пробелами, что весьма неудобно.

13.3. Программа для работы с базами данных phpMyAdmin

> *phpMyAdmin* — клиентская программа для работы с базами данных MySQL. Позволяет создавать, править и удалять базы данных, таблицы, индексы, связи и пользователей, извлекать данные и многое другое.

Программа phpMyAdmin фактически представляет собой веб-сайт, написанный на PHP, и требует для работы веб-сервер с установленной исполняющей средой PHP.

phpMyAdmin входит в состав XAMPP

Эта программа уже настроена нужным образом и готова к работе.

Внимание!

Перед тем как начинать работу с базами данных MySQL, необходимо запустить и веб-сервер Apache, и СУБД MySQL, входящие в состав пакета XAMPP (см. *приложение 2*).

13.3.1. Открытие phpMyAdmin и выполнение входа

Для открытия главной страницы phpMyAdmin нужно выполнить одно из следующих действий:

- запустить веб-обозреватель и выполнить переход по интернет-адресу: **http://localhost/phpmyadmin/**;

- нажать кнопку **Admin** в строке **MySQL** панели управления XAMPP (см. *приложение 2*). При этом будет запущен веб-обозреватель и выполнен переход по упомянутому ранее интернет-адресу.

Подключение к серверной СУБД и процедура входа выполнятся автоматически, и на экране откроется окно phpMyAdmin с начальной страницей, содержащей сведения о самой программе.

> ***Имя пользователя и пароль***
>
> phpMyAdmin выполняет вход в СУБД под именем пользователя `root` с пустым паролем, который присутствует в MySQL изначально.
>
> Имя пользователя и пароль для выполнения входа в СУБД хранятся в конфигурационном файле phpMyAdmin.

13.3.2. Навигация в phpMyAdmin

В левой части окна phpMyAdmin выводится иерархический список баз данных, хранящихся в них таблиц и индексов, а также полей, содержащихся в таблицах (рис. 13.6).

Пункты этого списка расположены по пяти вложенным друг в друга уровням:

- первый, самый верхний, — это базы данных. При выборе любой из них в правой части окна отображается страница со сведениями о выбранной базе, включающими список таблиц.

  Пункт **Создать БД** выводит страницу со списком баз данных, где можно создать новую базу и выполнить некоторые другие действия;

- второй — пункты **Представления** и **Таблицы** — содержит перечни, соответственно, представлений и таблиц, имеющихся в базе данных. Присутствует, только если в базе данных есть представления.

> ***Представления MySQL***
>
> *Представлением* называется SQL-запрос, сохраненный в самой базе данных (о запросах к базе данных и языке SQL мы узнаем на *уроке 14*). Такие запросы выполняются быстрее, чем сформированные программно, однако не могут быть изменены без правки базы данных. Описание представлений выходит за рамки этой книги.

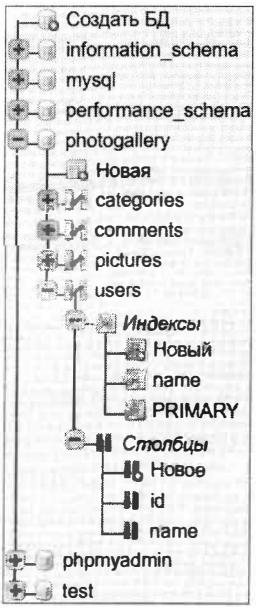


Рис. 13.6. Иерархический список баз данных и их составляющих

- третий, вложенный в пункт **Таблицы**, — это список таблиц в базе данных. При выборе любой из них выводятся сведения о таблице, включая хранящиеся в ней записи;

- четвертый — пункты **Индексы** и **Столбцы** — содержит перечни, соответственно, индексов и полей (в терминологии phpMyAdmin почему-то называемых *столбцами*) таблицы;

- пятый, вложенный в пункты **Индексы** и **Столбцы**, — содержит списки индексов и полей. При их выборе в правой части окна выводятся формы для правки индекса или поля.

В правой части окна отображаются страницы для работы с базами данных, таблицами, записями и др. Многие разбиты на отдельные вкладки, между которыми можно переключаться, щелкая на расположенных вверху корешках (рис. 13.7).

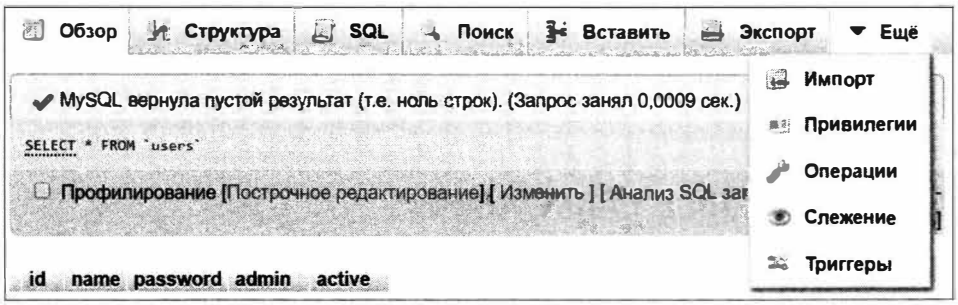


Рис. 13.7. Корешки вкладок у веб-страниц (активна вкладка **Обзор**) и корешок **Еще** с открытым меню

Если ширины окна недостаточно для вывода всех вкладок, справа появится корешок **Еще**. Если навести на него курсор мыши, отобразится меню, из которого можно переключиться на отсутствующие вкладки.

13.3.3. Работа с базой данных

Создание базы данных

1. Выберите пункт **Создать БД** в иерархическом списке (см. рис. 13.6) — появится страница со списком баз данных (рис. 13.8).

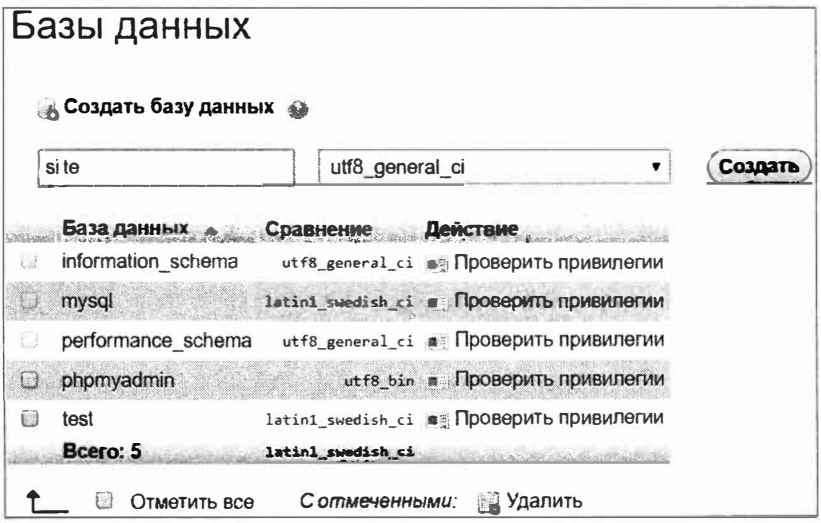


Рис. 13.8. Список баз данных

2. Укажите:
   - имя создаваемой базы данных — в поле ввода под надписью **Создать базу данных**;
   - кодировку UTF-8 и правила сравнения строк согласно этой кодировке — выбрав пункт **utf8_general_ci** в расположенном правее раскрывающемся списке.
3. Нажмите кнопку **Создать** — откроется страница со сведениями о новой базе данных.

Полезно знать...

В комплекте пакета XAMPP поставляются следующие базы данных:

- `mysql` — системная;
- `information_schema` — служебная, представляет сведения из системной базы данных;
- `performance_schema` — отладочная, хранит статистику работы СУБД;
- `phpMyAdmin` — служебная база phpMyAdmin, в которой хранятся, в частности, настройки программы;
- `test` — пустая тестовая база.

Переименование базы данных

1. Выберите в иерархическом списке нужную базу данных и переключитесь на вкладку **Операции** (см. рис. 13.7).
2. Занесите новое имя базы в поле ввода веб-формы **Переименовать базу данных в**.
3. Удостоверьтесь, что флажок **Настроить привилегии** установлен.

   Если он установлен, phpMyAdmin после переименования базы соответственно исправит привилегии пользователей на доступ к ней (в противном случае это придется делать вручную).
4. Нажмите кнопку **Вперед**.

Изменение кодировки у базы данных

1. Выберите в иерархическом списке нужную базу данных и переключитесь на вкладку **Операции**.
2. Выберите нужную кодировку (которая задаст и правила сравнения строк) в раскрывающемся списке веб-формы **Сравнение**.
3. Установите флажок **Изменить сортировки всех таблиц**.
4. Нажмите кнопку **Вперед**.

Удаление базы данных

1. Выберите пункт **Создать БД** в иерархическом списке.
2. Найдите нужную базу данных в списке и пометьте ее, установив расположенный в левом столбце флажок.

3. Щелкните гиперссылку **Удалить**, находящуюся под списком.
4. В открывшемся на экране предупреждении нажмите кнопку **ОК**.

13.3.4. Работа с таблицами

Создание таблицы

1. Выберите в иерархическом списке базу данных, в которую хотите добавить таблицу.
2. Укажите в веб-форме **Создать таблицу**:
   - имя создаваемой таблицы — в поле ввода **Имя**;
   - приблизительное количество полей в создаваемой таблице (может быть уточнено позднее) — в поле ввода **Количество столбцов**.
3. Нажмите кнопку **Вперед** — откроется страница с формой для создания полей у новой таблицы (рис. 13.9). Она организована в виде таблицы: строки представляют отдельные поля, а столбцы — параметры полей.

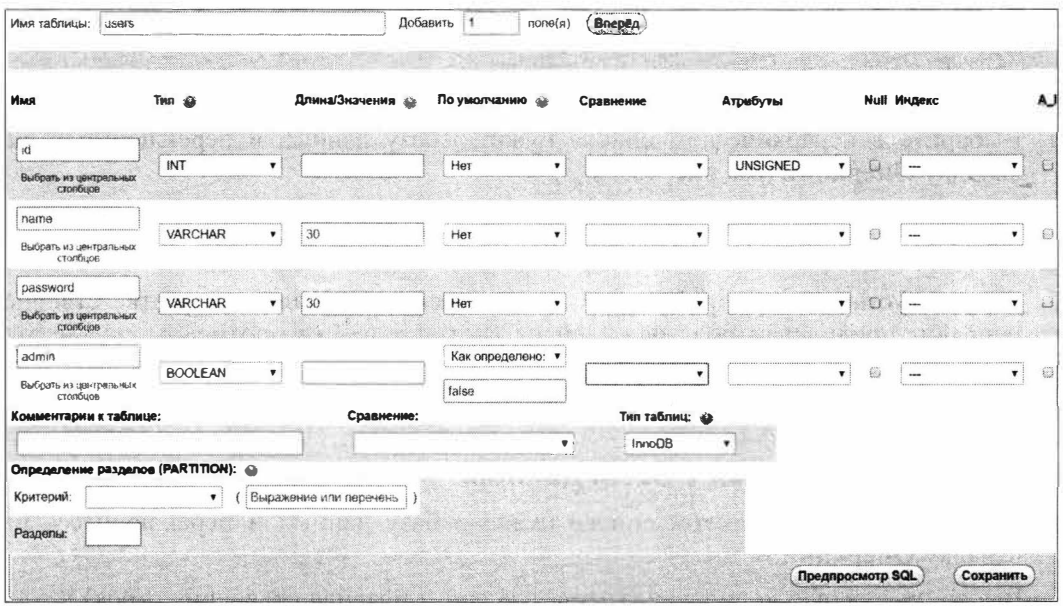


Рис. 13.9. Веб-форма для создания полей таблицы

4. Для каждого создаваемого поля укажите:
   - имя — в поле ввода в столбце **Имя**;
   - тип данных — в раскрывающемся списке **Тип**;
   - максимальную длину значения для полей типа `VARCHAR` или величину вида `<общее количество цифр>,<количество цифр после запятой>` для полей типа `DECIMAL` — в поле ввода **Длина/Значения**;

- наличие или отсутствие значения по умолчанию — посредством раскрывающегося списка **По умолчанию**:
  - если значения по умолчанию у поля нет — выберите в списке пункт **Нет**;
  - для указания конкретного значения по умолчанию — выберите пункт **Как определено** и занесите нужное значение в появившееся ниже поле ввода;
  - для указания `NULL` в качестве значения по умолчанию (доступно только для полей, способных хранить значение `NULL`) — пункт **NULL**;
  - для указания текущей временно́й отметки в качестве значения по умолчанию (только для полей типа `TIMESTAMP`) — пункт **CURRENT_TIMESTAMP**;
- является ли поле беззнаковым (только у полей целочисленных типов) — с помощью раскрывающегося списка **Атрибуты**:
  - если поле знаковое — «пустой» пункт;
  - если поле беззнаковое — пункт **UNSIGNED**;
- если поле должно иметь возможность хранить `NULL` — установите флажок **NULL**;
- если поле должно быть автоинкрементным — установите флажок **A_I**.

5. Создайте необходимые индексы.
   - Выберите тип индекса — в раскрывающемся списке **Индекс**:
     - ключевой индекс — пункт **PRIMARY**;
     - уникальный индекс — пункт **UNIQUE**;
     - обычный индекс — пункт **INDEX**;
     - чтобы указать отсутствие индекса, выберите пункт **---** (три дефиса).

   На экране появится небольшая форма для указания параметров создаваемого индекса (рис. 13.10).

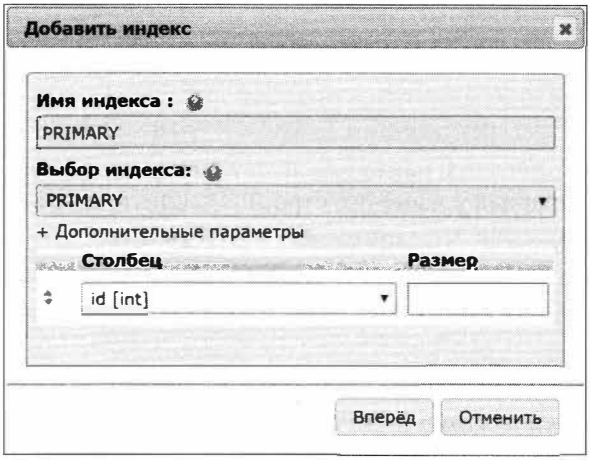


Рис. 13.10. Веб-форма для указания параметров индекса, создаваемого в новой таблице

- Введите уникальное имя индекса в поле **Имя индекса** (если создается ключевой индекс, это поле уже заполнено).
- Нажмите в окне параметров индекса кнопку **Вперед** — после чего вы вновь окажетесь на странице для указания параметров полей.

Добавление строк в веб-форму

Введите в поле **Добавить**, расположенное в верхней части страницы для указания параметров полей (см. рис. 13.9), количество добавляемых строк и нажмите кнопку **Вперед**.

Если в веб-форме остались лишние строки

Тогда их просто не следует заполнять.

6. Нажмите кнопку **Сохранить** — появится страница со сведениями о структуре вновь созданной таблицы, включая перечень имеющихся в ней полей и индексов (рис. 13.11).

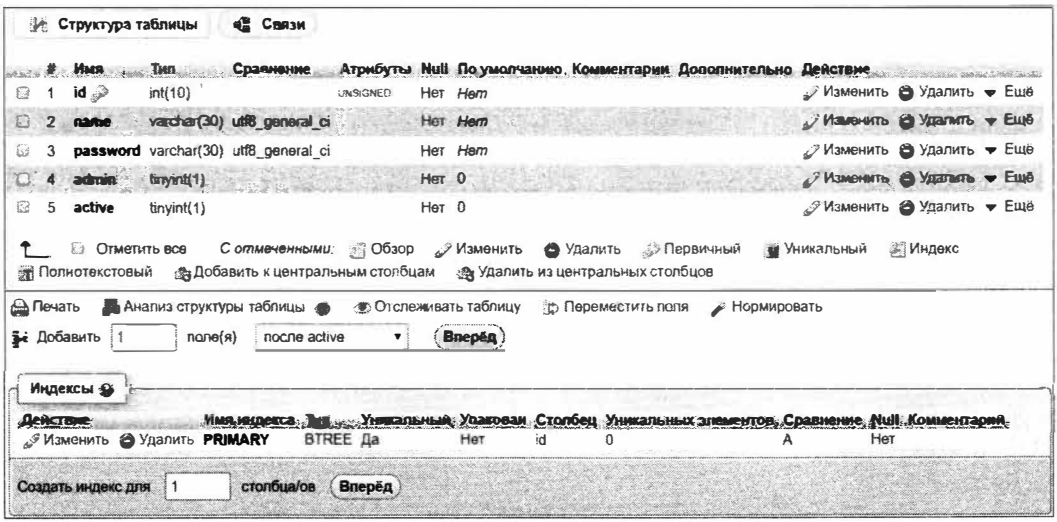


Рис. 13.11. Сведения о структуре таблицы

Полезно знать...

- Раскрывающийся список **Сравнение** позволяет указать для поля другую кодировку (и заодно правила сравнения строк), отличную от заданной для всей базы данных. Однако нужда в этом возникает очень редко.
- Остальные элементы управления в форме задают параметры для крайне специфических случаев, которые в этой книге не рассматриваются.

Переименование таблицы

1. Выберите в иерархическом списке (см. рис. 13.6) таблицу, которую хотите переименовать, и переключитесь на вкладку **Операции** — откроется страница с формой **Параметры таблицы**.

2. Занесите новое имя таблицы в поле ввода **Переименовать таблицу в**.
3. Установите флажок **Настроить привилегии**, чтобы phpMyAdmin соответственно исправил привилегии пользователей на доступ к этой таблице.
4. Нажмите кнопку **Вперед**.

Полезно знать...

- Раскрывающийся список **Сравнение** позволяет указать другую кодировку для таблицы.
- Остальные элементы управления в форме задают низкоуровневые параметры таблицы. Их следует менять лишь в специфических случаях.

Удаление таблицы

1. Выберите в иерархическом списке нужную базу данных.
2. Найдите в списке таблиц нужную и щелкните на гиперссылке **Удалить**, расположенной в колонке **Действие**.
3. Нажмите кнопку **ОК** в открывшемся на экране предупреждении.

13.3.5. Работа с полями

Добавление поля в таблицу

1. Выберите в иерархическом списке таблицу, в которую хотите добавить поле.
2. Переключитесь на вкладку **Структура** — появится страница со сведениями о структуре таблицы (см. рис. 13.11).
3. Введите количество добавляемых полей в поле **Добавить**, расположенное под списком уже имеющихся в таблице полей.
4. Укажите в раскрывающемся списке, находящемся правее, где должны располагаться добавляемые поля:
   - в начале таблицы — выберите одноименный пункт;
   - после поля с заданным *именем* — пункт **после <*имя поля*>**.
5. Нажмите кнопку **Вперед**.
6. Задайте параметры добавляемых полей в форме для добавления полей (см. рис. 13.9).
7. Нажмите кнопку **Сохранить**.

Правка поля

1. Выберите в иерархическом списке поле, которое хотите исправить, — откроется страница с формой для правки поля, аналогичной форме, показанной на рис. 13.9, но состоящей из одной строки.
2. Исправьте требуемые параметры поля (имя, тип, длина значения и др.).
3. Нажмите кнопку **Сохранить**.

Удаление поля

1. Выберите в иерархическом списке таблицу, из которой хотите удалить поле, и переключитесь на вкладку **Структура**.
2. Выполните следующее:
   - если нужно удалить одно поле — найдите его в списке полей, щелкните на гиперссылке **Удалить**, находящейся в колонке **Действия**, и нажмите кнопку **ОК** в открывшемся на экране предупреждении;
   - для удаления нескольких полей — пометьте их, установив находящиеся слева флажки, щелкните на гиперссылке **Удалить**, расположенной под списком полей, и нажмите кнопку **Да** на странице подтверждения, которая появится далее.

13.3.6. Работа с индексами

Добавление индекса в таблицу

1. Выберите в иерархическом списке таблицу, в которую хотите добавить индекс, и переключитесь на вкладку **Структура** — появится страница со сведениями о структуре таблицы (см. рис. 13.11). Список уже созданных индексов находится под списком полей.
2. Введите количество полей, на основе которых нужно создать индекс, в поле **Создать индекс для**, находящееся под списком индексов.
3. Нажмите кнопку **Вперед** — на экране появится форма для задания параметров индекса (рис. 13.12).

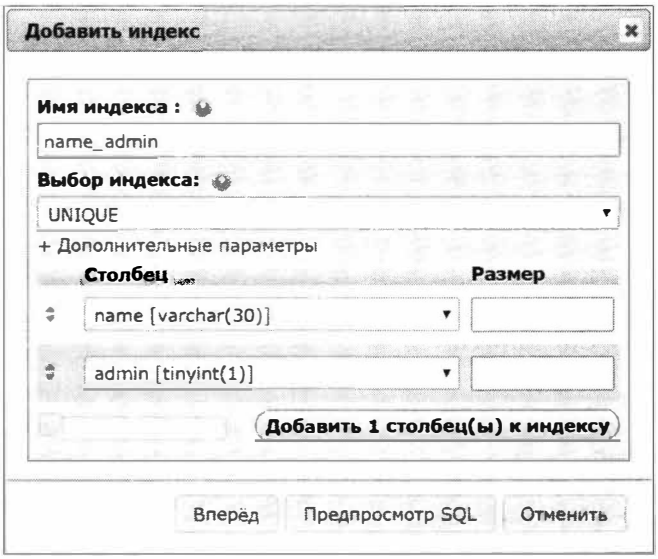


Рис. 13.12. Веб-форма для указания параметров индекса

4. Введите в поле **Имя индекса** имя создаваемого индекса, уникальное в пределах таблицы.
5. Выберите тип создаваемого индекса в раскрывающемся списке **Выбор индекса**:
   - обычный индекс — пункт **INDEX**;
   - уникальный индекс — пункт **UNIQUE**;
   - ключевой индекс — пункт **PRIMARY**.

   Ниже находится организованный в виде таблицы список для указания полей, по которым будет создан новый индекс.
6. Укажите имена полей, формирующих индекс, в раскрывающихся списках в колонке **Столбец**.

   Добавление строк в список

   Задайте их *количество* посредством регулятора, находящегося под левым нижним углом списка полей, и нажмите кнопку **Добавить <*количество*> столбец(ы) к индексу** (под словом «столбцы» здесь подразумеваются поля таблицы).

   Если в списке полей остались лишние строки

   Выберите в списках пункт **-- Игнорировать --**.
7. Нажмите кнопку **Вперед** — вы вернетесь на страницу со структурой таблицы.

Полезно знать...

Колонка **Размер** в списке полей (см. рис. 13.12) служит для указания рекомендуемого размера ячейки индекса, хранящей сведения об одной записи. Этот параметр поддерживался в базах данных старого формата MyISAM. Современный формат InnoDB, в котором MySQL сохраняет базы данных в настоящее время, его игнорирует.

Правка индекса

1. Выберите в иерархическом списке индекс, который хотите исправить.
2. Исправьте параметры индекса в появившейся форме (см. рис. 13.12).
3. Нажмите кнопку **Вперед**.

Удаление индекса

1. Выберите в иерархическом списке таблицу, чей индекс хотите удалить, и переключитесь на вкладку **Структура**.
2. Найдите нужный индекс в списке и щелкните на гиперссылке **Удалить**, находящейся в колонке **Действия**.
3. Нажмите кнопку **ОК** в открывшемся на экране предупреждении.

13.3.7. Работа со связями

Создание связи

1. Выберите в иерархическом списке *вторичную* таблицу, которая будет участвовать в создаваемой связи.

   Связь создается на стороне вторичной таблицы!
   Но никак не первичной!

2. Переключитесь на вкладку **Структура**.
3. Нажмите кнопку **Связи**, находящуюся под корешками вкладок, — phpMyAdmin выведет страницу с перечнем уже созданных связей (рис. 13.13). В нем присутствует одна «пустая» позиция, в которую и вводятся параметры создаваемой связи. Если же ни одной связи пока не создано, вы увидите лишь одну «пустую» позицию (как показано на рис. 13.13).

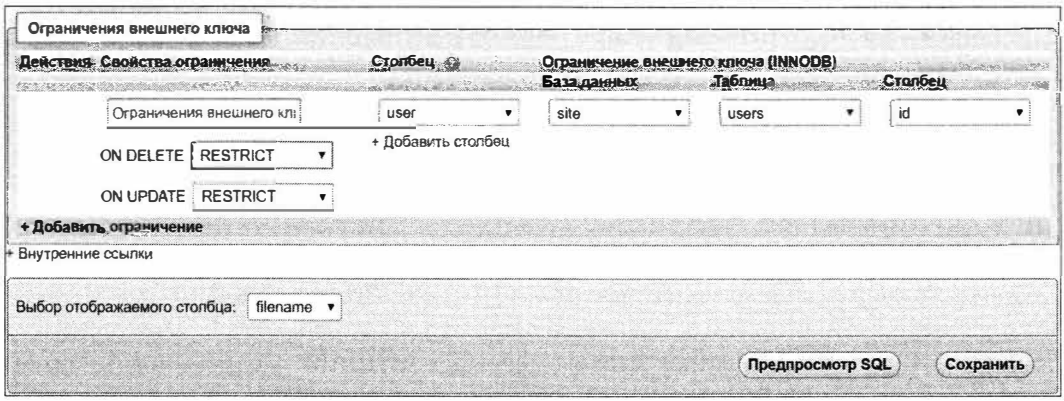


Рис. 13.13. Перечень связей

4. Занесите в «пустую» позицию сведения о новой связи:
   - поле внешнего ключа (которое уже должно быть создано) — в левый раскрывающийся список **Столбец**;
   - имя базы данных, в которой находится связываемая первичная таблица, — в раскрывающийся список **База данных**;
   - имя первичной таблицы — в раскрывающийся список **Таблица**;
   - ключевое поле первичной таблицы — phpMyAdmin сам подставит в правом раскрывающемся списке **Столбец**. Если программа подставила не то поле, выберите нужное вручную;
   - действие при удалении записи первичной таблицы — в раскрывающемся списке **ON DELETE**:
     - запрет удаления, если имеются связанные записи вторичной таблицы, — пункт **RESTRICT**;

- каскадное удаление связанных записей вторичной таблицы — пункт **CASCADE**;
- занесение в поле внешнего ключа связанных записей вторичной таблицы значения `NULL` — пункт **SET NULL**;

- действие при изменении ключевого поля в записи первичной таблицы — в раскрывающемся списке **ON UPDATE**:
  - запрет изменения, если имеются связанные записи вторичной таблицы, — пункт **RESTRICT**;
  - каскадное изменение полей внешнего ключа в связанных записях вторичной таблицы — пункт **CASCADE**;
  - занесение в поле внешнего ключа связанных записей вторичной таблицы значения `NULL` — пункт **SET NULL**.

Добавление позиций в перечень связей

Щелкните на гиперссылке **Добавить ограничение** (так в phpMyAdmin называется связь).

Если в перечне связей остались лишние «пустые» позиции

Тогда их просто не следует заполнять.

5. Нажмите кнопку **Сохранить**.

Правка связи

1. Выберите в иерархическом списке *вторичную* таблицу, которая будет участвовать в создаваемой связи, и переключитесь на вкладку **Структура**.
2. Нажмите кнопку **Связи**, находящуюся под корешками вкладок.
3. Исправьте параметры связи непосредственно в перечне связей (см. рис. 13.13).
4. Нажмите кнопку **Сохранить**.

Удаление связи

1. Выберите в иерархическом списке *вторичную* таблицу, к которой относится удаляемая связь, и переключитесь на вкладку **Структура**.
2. Нажмите кнопку **Связи**, находящуюся под корешками вкладок.
3. Щелкните на гиперссылке **Удалить**, находящейся в колонке **Действия** нужной позиции списка.
4. Нажмите кнопку **ОК** в открывшемся на экране предупреждении.

13.3.8. Работа с пользователями

Создание пользователя

1. Выберите пункт **Создать БД** иерархического списка и переключитесь на вкладку **Учетные записи пользователей** — phpMyAdmin выведет страницу со списком имеющихся на текущий момент пользователей (рис. 13.14).

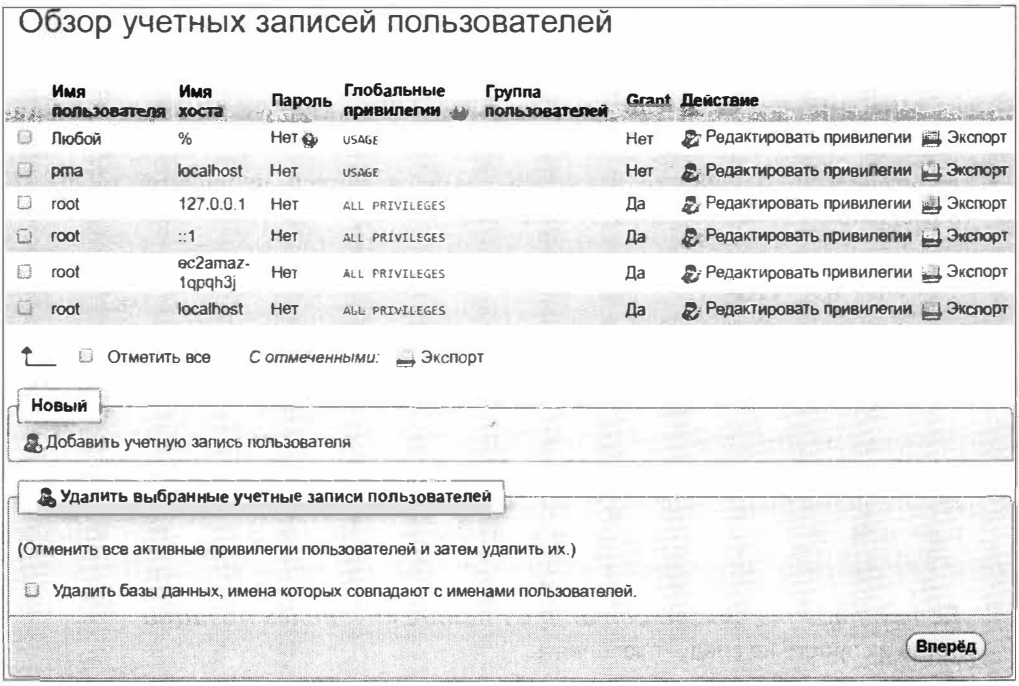


Рис. 13.14. Список пользователей

2. Щелкните на гиперссылке **Добавить учетную запись пользователя** в группе **Новый** — появится страница с формой **Информация учетной записи**, куда вводятся основные сведения о пользователе (рис. 13.15).

Рис. 13.15. Веб-форма **Информация учетной записи**

3. Введите основные сведения о пользователе:
   - имя пользователя — в поле ввода **Имя пользователя**;
   - интернет-адрес, с которого имеет право подключаться пользователь, — в раскрывающемся списке **Имя хоста**:

- любой интернет-адрес — выберите пункт **Любой хост**;
- локальный хост — выберите пункт **Локальный**;
- конкретный интернет-адрес — выберите пункт **Использовать текстовое поле** и занесите интернет-адрес в расположенное правее поле ввода;

- наличие или отсутствие пароля — в раскрывающемся списке **Пароль**:
  - отсутствие пароля — выберите пункт **Без пароля**;
  - наличие пароля — выберите пункт **Использовать текстовое поле** и занесите пароль дважды: в поле ввода, расположенное правее списка, и в поле **Подтверждение**.

Стойкость пароля

Индикатор **Strength**, находящийся правее поля ввода **Пароль**, показывает *стойкость пароля* — его устойчивость к подбору.

phpMyAdmin может сгенерировать стойкий пароль самостоятельно. Нажмите кнопку **Генерировать**, после чего сгенерированный пароль будет подставлен в поля ввода **Пароль**, **Подтверждение** и **Создать пароль** (где он будет выведен в открытом виде).

Выбор пароля

На момент разработки сайта лучше указать несложный пароль, чтобы впоследствии не забыть его, или даже вовсе отсутствие пароля. Но после публикации сайта пароль безусловно следует сменить на более стойкий.

Ниже на той же странице находится веб-форма **Глобальные привилегии**, в которой указываются привилегии на доступ ко всем базам данных, для которых привилегии доступа не были заданы явно (рис. 13.16).

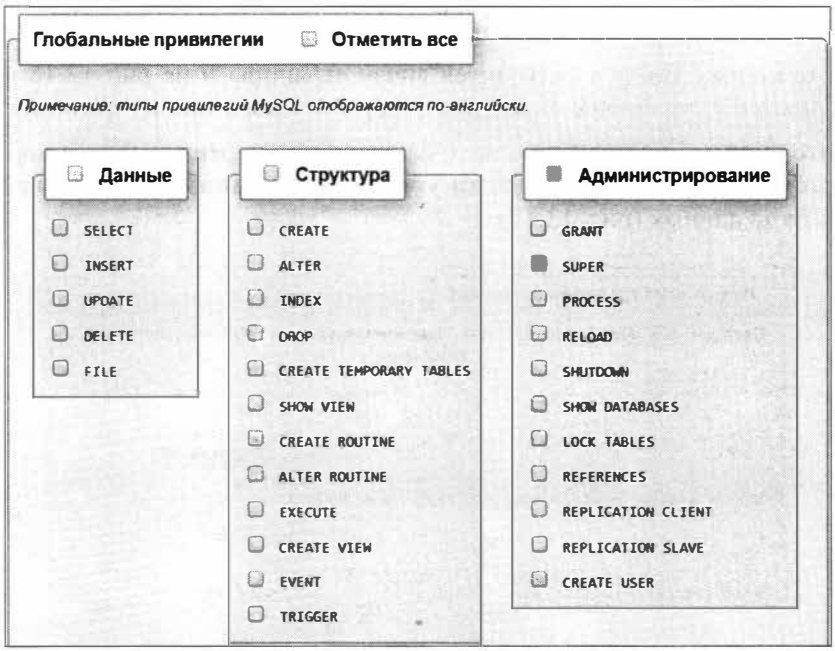


Рис. 13.16. Веб-форма **Глобальные привилегии**

Она включает три раздела, каждый из которых, в свою очередь, содержит флажки, устанавливающие привилегию на выполнение определенной операции:

- **Данные** — операции с содержимым таблиц:
  - **SELECT** — выборка записей;
  - **INSERT** — добавление записей;
  - **UPDATE** — правка записей;
  - **DELETE** — удаление записей;
  - **FILE** — импорт и экспорт данных;
- **Структура** — создание, правка и удаление таблиц, полей, индексов и связей;
- **Администрирование** — административные действия (управление пользователями, назначение им привилегий, просмотр списка баз данных и пр.).

4. Укажите глобальные привилегии пользователя:
   - полные привилегии (на выполнение всех доступных действий) — установите флажок в заголовке формы **Глобальные привилегии**;
   - привилегии на выполнение всех операций из определенного раздела — установите флажок в названии раздела **Данные**, **Структура** или **Администрирование**;
   - привилегии на выполнение определенных операций — установите флажок, соответствующий этой операции (например, **SELECT**);
   - если пользователь не должен иметь глобальных привилегий (например, вы собираетесь разрешить ему доступ лишь к одной базе данных) — сбросьте все флажки в форме **Глобальные привилегии**.
5. Нажмите кнопку **Вперед** (находится внизу страницы и на рис. 13.16 не показана) — появится страница с одной лишь формой **Глобальные привилегии**.
6. Нажмите кнопку **База данных**, находящуюся под корешками вкладок. Откроется страница с формой **Привилегии уровня базы данных**, подготовленной для выбора базы данных (рис. 13.17).

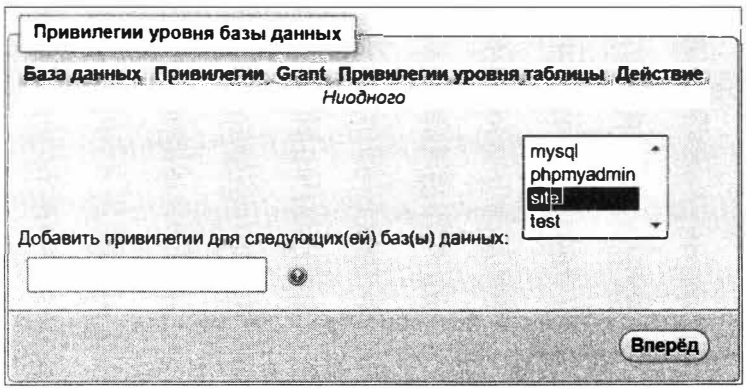


Рис. 13.17. Веб-форма **Привилегии уровня базы данных** на этапе выбора базы данных

7. Выберите в списке **Добавить привилегии для следующих(ей) баз(ы) данных** базу, привилегии на доступ к которой хотите дать пользователю.
8. Нажмите кнопку **Вперед** — откроется страница с формой **Привилегии уровня базы данных**, подготовленной для указания привилегий. Она похожа на форму **Глобальные привилегии** (см. рис. 13.16) и используется аналогичным образом.
9. Укажите привилегии на доступ к выбранной ранее базе данных, установив соответствующие флажки.
10. Нажмите кнопку **Вперед**.

Полезно знать...

- phpMyAdmin также позволяет указать привилегии на доступ к отдельной таблице базы данных, что может пригодиться в некоторых случаях.
- Изначально в MySQL уже имеется несколько зарегистрированных пользователей (они показаны в списке на рис. 13.14). Все они являются администраторами, и все имеют пустые пароли.

Правка привилегий пользователя

1. Выберите пункт **Создать БД** иерархического списка и переключитесь на вкладку **Учетные записи пользователей**.
2. Найдите нужного пользователя в списке и щелкните на гиперссылке **Редактировать привилегии** в колонке **Действие**.
3. Исправьте привилегии:
   - глобальные — непосредственно в форме **Глобальные привилегии**, после чего нажмите кнопку **Вперед**;
   - уровня баз данных — для этого выполните шаги, представленные далее.
     - Нажмите кнопку **База данных**, расположенную под корешками вкладок — появится страница с формой **Привилегии уровня базы данных**, подготовленная для правки привилегий (рис. 13.18).

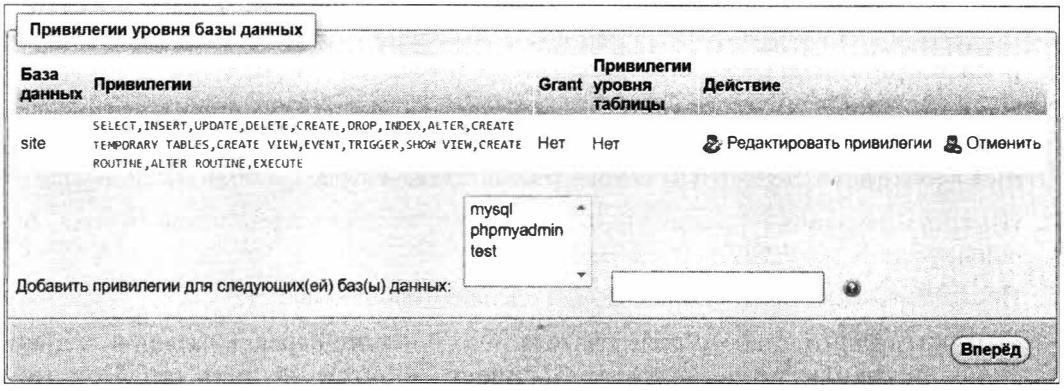


Рис. 13.18. Веб-форма Привилегии уровня базы данных

- Чтобы исправить заданные ранее привилегии, найдите нужную базу данных в списке вверху формы и щелкните на гиперссылке **Редактировать привилегии** в колонке **Действие**.
- Чтобы задать привилегии для другой базы данных, выберите ее в списке **Добавить привилегии для следующих(ей) баз(ы) данных** и нажмите кнопку **Вперед**.
- Исправьте привилегии в форме **Привилегии уровня базы данных**.
- Нажмите кнопку **Вперед**.

Изменение пароля пользователя

1. Выберите пункт **Создать БД** иерархического списка и переключитесь на вкладку **Учетные записи пользователей**.
2. Найдите нужного пользователя в списке и щелкните на гиперссылке **Редактировать привилегии** в колонке **Действие**.
3. Нажмите кнопку **Изменить пароль**, расположенную под корешками вкладок, — откроется страница с формой **Изменить пароль**.
4. Сделайте следующее:
   - чтобы дать пользователю пустой пароль — установите переключатель **Без пароля**;
   - чтобы задать или изменить пароль — установите переключатель **Пароль** и занесите пароль дважды: в поля ввода **Введите** и **Подтверждение**.
5. Нажмите кнопку **Вперед**.

Правка основных сведений о пользователе

> ***phpMyAdmin не позволяет переименовать пользователя***
> Вместо этого программа создаст нового пользователя с заданным именем. Старого пользователя можно как оставить, так и удалить.

1. Выберите пункт **Создать БД** иерархического списка и переключитесь на вкладку **Учетные записи пользователей**.
2. Найдите нужного пользователя в списке и щелкните на гиперссылке **Редактировать привилегии** в колонке **Действие**.
3. Нажмите кнопку **Информация учетной записи**, расположенную под корешками вкладок, — откроется страница с формой **Изменить регистрационные данные / Копировать учетную запись пользователя** (рис. 13.19).
4. Измените сведения о пользователе: имя, допустимый интернет-адрес и (или) пароль.
5. После изменения имени пользователя сделайте следующее:
   - чтобы удалить старого пользователя — установите переключатель **отменить все активные привилегии предыдущего и затем удалить его**. При этом phpMyAdmin вместе с пользователем удалит данные ему привилегии;

- чтобы сохранить старого пользователя — установите переключатель **сохранить предыдущего**.

6. Нажмите кнопку **Вперед**.

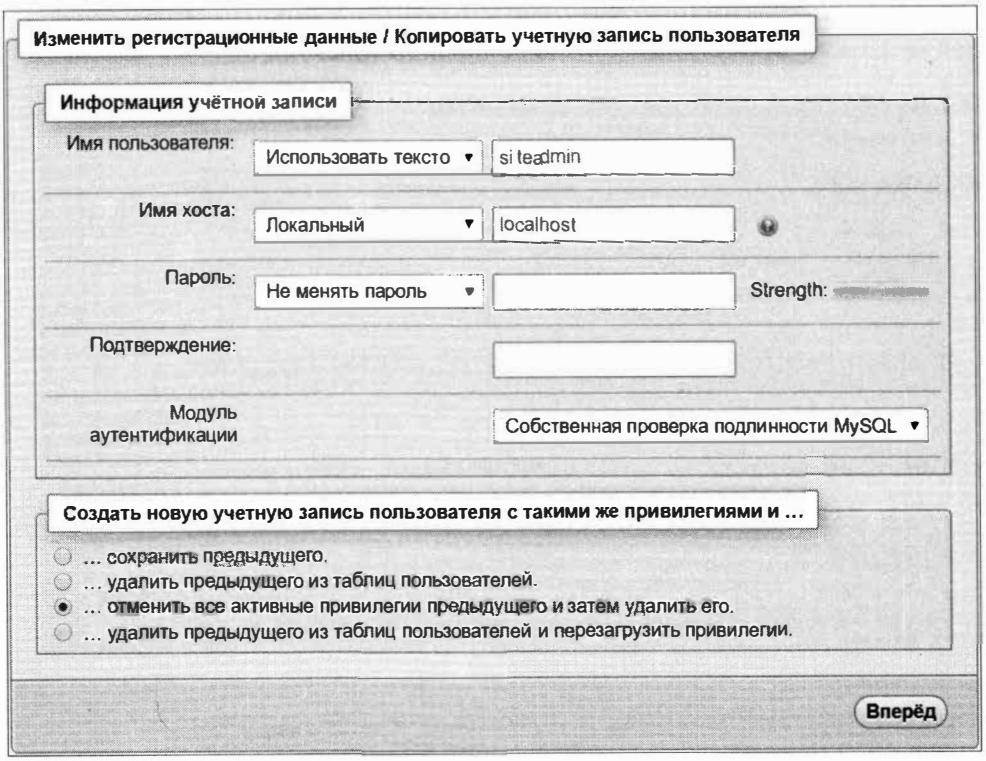


Рис. 13.19. Веб-форма
Изменить регистрационные данные / Копировать учетную запись пользователя

Полезно знать...

Остальные переключатели в форме **Изменить регистрационные данные / Копировать учетную запись пользователя** применяются в весьма специфических случаях и обычно малополезны.

Удаление пользователя

1. Выберите пункт **Создать БД** иерархического списка и переключитесь на вкладку **Учетные записи пользователей**.
2. Найдите нужного пользователя в списке и пометьте его, установив расположенный слева флажок.
3. Нажмите кнопку **Вперед** в форме **Удалить выбранные учетные записи пользователей**.
4. Нажмите кнопку **ОК** в открывшемся на экране предупреждении.

13.3.9. Работа с записями

Добавление записей

1. Выберите в иерархическом списке таблицу, в которую хотите добавить записи.
2. Переключитесь на вкладку **Вставка** — появится страница с формой для добавления записей, организованной в виде таблицы (рис. 13.20).

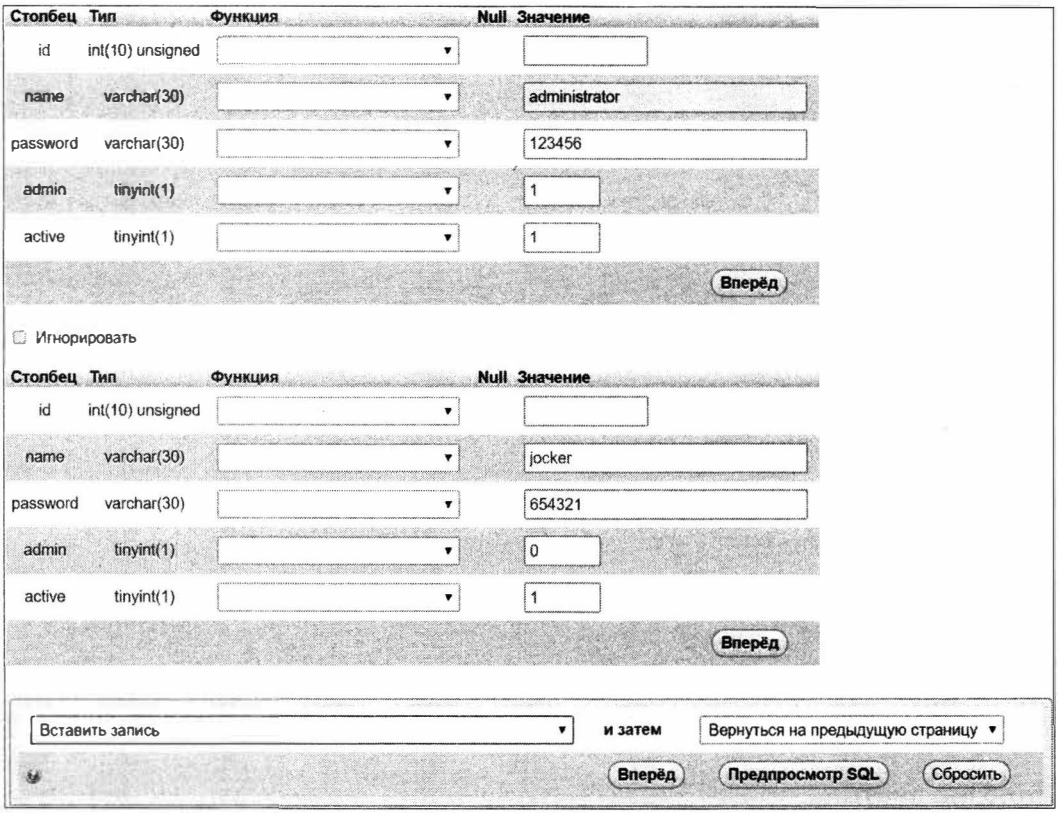


Рис. 13.20. Веб-форма для добавления записей в таблицу

В самой левой колонке выводятся имена полей, а в самой правой — поля ввода для занесения значений в эти поля. Поскольку эта таблица имеет две строки, одновременно можно добавить не более двух записей.

Значения вводятся в следующих форматах:

- в логические поля — в виде чисел `1` (`TRUE`) и `0` (`FALSE`);
- в поля внешнего ключа — выбираются в раскрывающихся списках, содержащих все записи из связанной первичной таблицы;

По поводу этих списков...

В них присутствуют два набора пунктов: формата ***<значение выбранного поля> - <значение ключевого поля>*** и ***<значение ключевого поля> - <значение выбранного поля>***. Можно использовать любой набор.

По умолчанию phpMyAdmin использует в качестве *выбранного* первое строковое поле первичной таблицы. Указать другое *выбранное поле* можно на странице с формой задания связей (см. рис. 13.13) *первичной таблицы*, в раскрывающемся списке **Выбор отображаемого столбца**, после чего нажать кнопку **Сохранить**. Эта настройка сохраняется в собственной базе `phpmyadmin` программы phpMyAdmin.

- автоинкрементные поля следует оставить пустыми — MySQL занесет в них значения самостоятельно;
- в поля остальных типов — вводятся как есть.

3. Сделайте следующее:
   - если нужно добавить одну запись — введите значения ее полей в первую строку таблицы и нажмите находящуюся ниже этой строки кнопку **Вперед**;
   - если нужно добавить две записи — заполните обе строки, проверьте, сброшен ли флажок **Игнорировать** над второй строкой, и нажмите кнопку **Вперед**, находящуюся в самом низу формы.

После добавления записей откроется вкладка **SQL** (мы познакомимся с ней на *уроке 14*). Переключитесь на вкладку **Обзор**, чтобы увидеть содержимое таблицы — имеющиеся в ней записи (рис. 13.21).

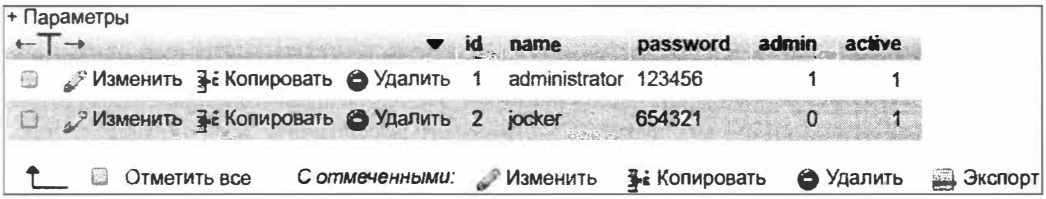


Рис. 13.21. Содержимое таблицы

Если нужно добавить больше двух записей...

В форме для добавления записей (см. рис. 13.20) перед нажатием кнопки **Вперед** в раскрывающемся списке **и затем** выберите пункт **Добавить новую запись**. После сохранения добавленных записей в базе снова появится форма добавления записей.

Правка записей

1. Выберите в иерархическом списке таблицу, записи в которой хотите исправить.
2. Сделайте следующее:
   - если нужно исправить одну запись — найдите ее в списке записей и щелкните на гиперссылке **Изменить** в левой колонке;
   - для правки нескольких записей — пометьте их, установив находящиеся слева флажки, и щелкните на гиперссылке **Изменить**, расположенной под списком записей.
3. Исправьте записи в веб-форме, аналогичной форме, приведенной на рис. 13.20.
4. Нажмите кнопку **Вперед**.

Полезно знать...

phpMyAdmin позволяет исправить также и значения автоинкрементных полей (правда, это может пригодиться в очень редких случаях).

Удаление записей

1. Выберите в иерархическом списке таблицу, из которой хотите удалить записи.
2. Сделайте следующее:
   - если нужно удалить одну запись — найдите ее в списке записей, щелкните на гиперссылке **Удалить**, находящейся в левой колонке, и нажмите кнопку **ОК** в открывшемся на экране предупреждении;
   - для удаления нескольких записей — пометьте их, установив находящиеся слева флажки, щелкните на гиперссылке **Удалить**, расположенной под списком записей, и нажмите кнопку **Да** на странице подтверждения, которая появится далее.

13.3.10. Экспорт и импорт баз данных

Экспорт — сохранение базы данных MySQL в файле дампа с целью создать резервную копию или перенести базу на другой компьютер.

Дамп — текстовый файл, содержащий набор команд на языке SQL (им мы займемся на *уроке 14*), которые полностью воссоздают базу данных.

В дамп включаются SQL-команды, создающие базу данных, таблицы со всеми нужными полями, индексами и связями и добавляющие в них записи.

Импорт — восстановление базы данных из дампа.

Экспорт базы данных

1. Выберите в иерархическом списке базу данных, которую хотите экспортировать, и переключитесь на вкладку **Экспорт** — появится страница с большой веб-формой для указания параметров экспорта.
2. Установите переключатель **Обычный** в группе **Метод экспорта**, чтобы показать все возможные настройки экспорта.
3. Укажите в группе **Таблицы**, какие таблицы вы хотите экспортировать, и что именно подлежит экспорту: только структура или структура вместе с данными. Для этого установите или сбросьте соответствующие флажки.
4. Установите в группе **Параметры создания объектов** флажок **Добавить выражение CREATE DATABASE / USE**. В результате в дамп будет добавлена SQL-команда, создающая базу данных, если таковая отсутствует (в противном случае перед импортом придется создавать базу вручную).
5. Нажмите кнопку **Вперед**.

6. Сохраните загруженный файл с дампом, имеющий имя формата *<имя базы данных>*.sql, на локальном диске.

Импорт базы данных

1. Выберите пункт **Создать БД** иерархического списка, проверьте, существует ли уже такая база данных, и, если существует, удалите ее.
2. Переключитесь на вкладку **Импорт**.

   Появится страница с веб-формой для указания параметров импорта.
3. Нажмите кнопку **Выберите файл** в группе **Импортируемый файл** и выберите нужный файл дампа в открывшемся на экране диалоговом окне открытия файла.
4. Нажмите кнопку **Вперед** — появится страница со сведениями о выполненном импорте.

13.4. Упражнение. Создаем базу данных для веб-сайта фотогалереи

Настала пора начать разработку новой фотогалереи, хранящей данные в информационной базе. База данных обновленной фотогалереи будет хранить следующие перечни:

- *изображений* — каждое из изображений будет относиться к определенной категории (растения, архитектура, скульптура и др.) и принадлежать выгрузившему его пользователю;
- *комментариев* — каждый комментарий будет связан с определенным изображением;
- *категорий*;
- *пользователей*.

Для работы воспользуемся программой phpMyAdmin, описанной в *разд. 13.3*.

1. Создадим базу данных `photogallery`, не забыв указать у нее кодировку UTF-8.
2. Список зарегистрированных пользователей фотогалереи будем хранить в таблице `users`. На момент создания учетные записи, хранящиеся в этой таблице, будут содержать лишь уникальные имена пользователей (поля для хранения остальных сведений — в частности, паролей — мы добавим на *уроке 21*).

   > ***В каждую таблицу включим поле `id`!***
   >
   > Это поле будет хранить уникальные номера изображений, комментариев, категорий и пользователей. По этому номеру мы сможем извлечь из таблицы нужную запись.

   Создадим в базе `photogallery` таблицу `users` с полями, приведенными в табл. 13.2.
3. Добавим в таблицу `users` пользователей с именами `admin`, `jocker` и `basil`.
4. Таблицу для хранения списка категорий назовем `categories`. Каждая категория будет иметь уникальные имя и слаг.

Таблица 13.2. *Поля таблицы* `users`

Имя	Тип	Описание	Параметры
`id`	`INT`, беззнаковое	Номер	Автоинкрементное, ключевой индекс
`name`	`VARCHAR`, 20 символов	Имя пользователя	Уникальный индекс с именем `name`

> *Слаг* — слово или фраза, однозначно идентифицирующая определенную запись в таблице. Более информативен, чем числовой номер, но может быть весьма громоздким.

Слаг чаще всего представляет собой имя какой-либо сущности, набранное латиницей. Для разделения отдельных слов в слаге применяются подчеркивания.

Создадим в базе `photogallery` таблицу `categories` с полями, приведенными в табл. 13.3.

Таблица 13.3. *Поля таблицы* `categories`

Имя	Тип	Описание	Параметры
`id`	`INT`, беззнаковое	Номер	Автоинкрементное, ключевой индекс
`name`	`VARCHAR`, 20 символов	Название	Уникальный индекс с именем `name`
`slug`	`VARCHAR`, 20 символов	Слаг	Уникальный индекс с именем `slug`

5. Добавим в таблицу `categories` записи с параметрами из табл. 13.4.

Таблица 13.4. *Записи, добавляемые в таблицу* `categories`

name	slug
Растения	`plants`
Архитектура	`architecture`
Пейзаж	`landscape`
Скульптура	`sculpture`

6. Сведения об изображениях сохраним в таблице `pictures`. Сведения о каждом изображении включат имя файла, название, необязательное примечание, номера категории и пользователя, с которыми оно связано, и временну́ю отметку его публикации в галерее (будет заполняться автоматически).

   Создадим в базе данных таблицу `pictures` с полями согласно табл. 13.5, а также необходимые связи с таблицами `categories` и `users`.

7. Добавим в таблицу `pictures` записи из табл. 13.6.

Таблица 13.5. *Поля таблицы* `pictures`

Имя	Тип	Описание	Параметры
`id`	`INT`, беззнаковое	Номер	Автоинкрементное, ключевой индекс
`filename`	`VARCHAR`, 24 символа	Имя файла с расширением	
`title`	`TINYTEXT`	Название	
`description`	`TEXT`	Описание	
`category`	`INT`, беззнаковое	Номер категории	Связь с таблицей `categories`
`user`	`INT`, беззнаковое	Номер пользователя	Связь с таблицей `users`
`uploaded`	`TIMESTAMP`	Временна́я отметка публикации	Значение по умолчанию — текущая временна́я отметка, обычный индекс с именем `uploaded`

Таблица 13.6. *Записи, добавляемые в таблицу* `pictures`

filename	title	description	category	user
200804282020.jpg	Цветы кактуса	Снято в Волжском гуманитарном институте	Растения	basil
201802131844.jpg	Блинная скульптура	Снято в кафе «Закусочная "Ахтуба"»	Скульптура	admin
201506152037.jpg	Памятник сусликам		Скульптура	basil
200901092053.jpg	Зимний закат		Пейзаж	jocker
201709252026.jpg	Кот ученый	Поставлен по инициативе Волжского гуманитарного института	Скульптура	basil
201710242002.jpg	Памятник собаке-поводырю	Стоит на автобусной остановке	Скульптура	admin
200807192032.jpg	Пустынная аллея	В запущенном парке на окраине	Пейзаж	jocker
201410131614.jpg	Роза		Растения	basil
201709180821.jpg	Бегемотики		Скульптура	basil

Пояснения к табл. 13.6:

- в столбце `category` приведены названия категорий — так их удобнее выбрать в раскрывающихся списках;
- в столбце `user` здесь и далее приведены имена пользователей.

8. Комментарии сохраним в таблице `comments`. Сведения о комментарии будут содержать номер пользователя, который его оставил, текстовое содержание, номер изображения, к которому относится комментарий, и временну́ю отметку добавления (будет заполняться автоматически).

Создадим в базе данных таблицу comments с полями и связями с таблицами users и pictures согласно табл. 13.7.

***Таблица 13.7.** Поля таблицы comments*

Имя	Тип	Описание	Параметры
id	INT, беззнаковое	Номер	Автоинкрементное, ключевой индекс
user	INT, беззнаковое	Номер пользователя — автора комментария	Связь с таблицей users
contents	TEXT	Содержание	
picture	INT, беззнаковое	Номер изображения, к которому относится комментарий	Связь с таблицей pictures, каскадные удаление и обновление
uploaded	TIMESTAMP	Временна́я отметка публикации	Значение по умолчанию — текущая временна́я отметка, обычный индекс с именем uploaded

9. Добавим в таблицу comments записи из табл. 13.8.

***Таблица 13.8.** Записи, добавляемые в таблицу comments*

user	contents	picture
admin	Забавные зверушки	201506152037.jpg
basil	Какое пустынное место...	200807192032.jpg
jocker	Весьма аппетитно выглядит эта скульптура	201802131844.jpg

Пояснение к табл. 13.8: в столбце picture приведены имена файлов с изображениями.

Внимание!

Актуальные дампы базы данных будут храниться в файлах *photogallery.sql*, находящихся в папках электронного архива (см. *приложение 3*). Так, файл с дампом базы, созданной в этом упражнении, находится в файле *13\13.4\ex\photogallery.sql*.

Чтобы восстановить базу данных, достаточно выполнить импорт этого файла в программе phpMyAdmin.

Урок 14
Написание запросов к базам данных. Язык SQL

Запросы к СУБД на извлечение или изменение информации, хранящейся в базах данных, пишутся на языке SQL.

> *SQL* (Structured Query Language, язык структурированных запросов) — язык программирования, предназначенный для создания запросов к реляционным базам данных.

Посредством SQL-запросов выполняются выборка, добавление, правка и удаление записей, создание, правка и удаление баз данных, таблиц, полей, индексов и связей, создание, правка, удаление пользователей и управление их привилегиями, а также административные задачи (наподобие получения списка всех баз данных, обрабатываемых текущей СУБД).

На этом уроке будут рассмотрены только команды для выборки, добавления, правки и удаления записей, поскольку именно они наиболее востребованы в веб-программировании.

Команды SQL нечувствительны к регистру символов

Но традиционно их набирают в верхнем регистре.

Каждый SQL-запрос должен завершаться точкой с запятой

Но при выполнении запросов в программе phpMyAdmin (см. *разд. 13.3*) этого делать необязательно — точка с запятой ставится автоматически.

14.1. Выполнение SQL-запросов в phpMyAdmin

1. Выберите в иерархическом списке слева базу данных, к которой хотите выполнить запрос.
2. Переключитесь на вкладку **SQL** — появится страница с формой для ввода SQL-запроса (рис. 14.1).
3. Введите код SQL-запроса в большую область редактирования.
4. Установите флажок **Оставить поле запроса**, чтобы phpMyAdmin не скрывал форму после выполнения запроса.

5. Нажмите кнопку **Вперед** — программа выведет результат выполнения запроса в виде таблицы под формой для ввода запроса (рис. 14.2).

После этого в форму можно ввести и тут же выполнить другой запрос.

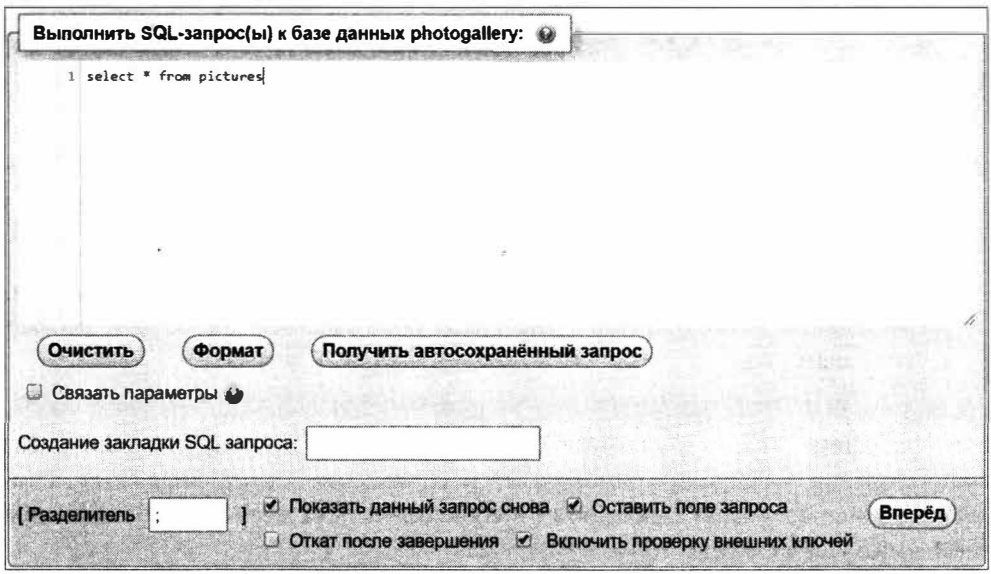


Рис. 14.1. Веб-форма для ввода SQL-запроса

Рис. 14.2. Результат выполнения SQL-запроса

14.2. Выборка записей

14.2.1. Простая выборка записей

Выборка записей из указанной *таблицы* выполняется командой SELECT:

```
SELECT [DISTINCT] <список полей> FROM <таблица>
```

> ***Примеры SQL-запросов и выдаваемых результатов***
>
> В приводимых далее примерах SQL-запросов сначала записывается SQL-запрос, а за ним, через пустую строку, показан результат его выполнения в виде таблицы.

В качестве *списка полей* можно указать:

- * (звездочку) — извлечь все поля:

```
SELECT * FROM comments;
```

id	user	contents	picture	uploaded
1	1	Забавные зверушки	3	2019-09-06 14:18:39
2	3	Какое пустынное место...	7	2019-09-06 14:20:02
3	2	Весьма аппетитно выглядит эта скульптура	2	2019-09-06 14:21:38

- перечень имен полей через запятую — для извлечения только избранных полей:

```
SELECT contents, uploaded FROM comments;
```

contents	uploaded
Забавные зверушки	2019-09-06 14:18:39
Какое пустынное место...	2019-09-06 14:20:02
Весьма аппетитно выглядит эта скульптура	2019-09-06 14:21:38

> ***По возможности выбирайте только нужные вам поля***
>
> Тем самым вы ускорите выполнение запроса и уменьшите объем памяти, занимаемый результатом его выполнения.

Любому полю можно дать псевдоним, записав в *списке полей* конструкцию вида *<имя поля>* AS *<псевдоним>*:

```
SELECT contents AS cnt, uploaded AS up FROM comments;
```

cnt	up
Забавные зверушки	2019-09-06 14:18:39
Какое пустынное место...	2019-09-06 14:20:02
Весьма аппетитно выглядит эта скульптура	2019-09-06 14:21:38

Если в команде SELECT указать языковую конструкцию DISTINCT, будут извлечены только уникальные строки. Для примера извлечем из таблицы pictures значения поля user — номера пользователей, выгрузивших картинки:

```
SELECT user FROM pictures;
```

```
user
1
1
2
2
3
3
3
3
3
```

Мы получили девять значений (по числу записей), из которых шесть повторяются. Теперь извлечем только уникальные записи:

```
SELECT DISTINCT user FROM pictures;
```

```
user
1
2
3
```

14.2.2. Связывание таблиц и выборка полей из связанных записей

Если нужно связать две таблицы и выбрать поля из связанных записей обеих таблиц, команду SELECT следует дополнить конструкцией, создающей межтабличную связь:

```
SELECT . . . FROM <первая таблица> INNER JOIN <вторая таблица> ↵
ON <первая таблица>.<связываемое поле первой таблицы> = ↵
   <вторая таблица>.<связываемое поле второй таблицы>
```

Как правило, в качестве *первой таблицы* указывают вторичную, в качестве *второй* — первичную, *связываемым полем первой таблицы* делают ее поле внешнего ключа, а *связываемым полем второй таблицы* — ее ключевое поле.

Список извлекаемых из записей полей указывается, как обычно, после языковой конструкции SELECT.

> ***Обе таблицы содержат одноименные поля?***
>
> При попытке извлечения такого поля СУБД выдаст ошибку, поскольку «не знает», поле какой таблицы имеется в виду. В таких случаях нужно явно указать таблицу, которой принадлежит поле, записав его имя в формате: *<имя таблицы>.<имя поля>*.
>
> Если нужно извлечь одноименные поля из обеих таблиц, следует дать псевдоним одному из полей или разные псевдонимы — обоим.

В результате выполнения запроса с языковой конструкцией INNER JOIN из таблиц будут извлечены только связанные друг с другом записи.

Пример извлечения перечня названий изображений и названий категорий, к которым относятся эти изображения:

```
SELECT pictures.title, categories.name FROM pictures ↳
INNER JOIN categories ON pictures.category = categories.id;
```

title	name
Цветы кактуса	Растения
Блинная скульптура	Скульптура
Памятник сусликам	Скульптура
Зимний закат	Пейзаж
Кот ученый	Скульптура
Памятник собаке-поводырю	Скульптура
Пустынная аллея	Пейзаж
Роза	Растения
Бегемотики	Скульптура

Можно, наоборот, указать в качестве *первой таблицы* первичную, а в качестве *второй* — вторичную:

```
SELECT categories.name, pictures.title FROM categories ↳
INNER JOIN pictures ON categories.id = pictures.category;
```

name	title
Растения	Цветы кактуса
Скульптура	Блинная скульптура
. . .	
Скульптура	Бегемотики

Поскольку запрос с языковой конструкцией `INNER JOIN` извлекает только связанные друг с другом записи обеих таблиц, в приведенном ранее перечне отсутствует категория «Архитектура», не имеющая ни одного связанного изображения.

Чтобы извлечь *все* записи *первой таблицы* и связанные с ними записи *второй таблицы*, в запросе вместо конструкции `INNER JOIN` нужно использовать `LEFT JOIN`. Если какая-либо запись *первой таблицы* не имеет связанных записей *второй таблицы*, при попытке обращения к полю второй таблицы возвращается `NULL`.

Пример извлечения *всех* категорий и названий связанных с ними изображений:

```
SELECT categories.name, pictures.title FROM categories ↳
LEFT JOIN pictures ON categories.id = pictures.category;
```

name	title
Архитектура	NULL
Пейзаж	Зимний закат
Пейзаж	Пустынная аллея
Растения	Цветы кактуса
Растения	Роза
Скульптура	Блинная скульптура

```
Скульптура      Памятник сусликам
Скульптура      Кот ученый
Скульптура      Памятник собаке-поводырю
Скульптура      Бегемотики
```

Аналогичная языковая конструкция `RIGHT JOIN` извлекает *все* записи *второй таблицы* и связанные с ними записи *первой таблицы*:

```
SELECT pictures.title, categories.name FROM pictures ↳
RIGHT JOIN categories ON pictures.category = categories.id;
```

title	name
NULL	Архитектура
Зимний закат	Пейзаж
. . .	
Бегемотики	Скульптура

В рассмотренных нами запросах связи устанавливались между двумя таблицами: *первой* и *второй*. Но SQL позволяет связать *первую таблицу* с несколькими *вторыми*, для чего достаточно записать конструкции, создающие связи, друг за другом через пробел.

Далее приведен пример извлечения названий изображений, связанных с ними категорий и имен пользователей, опубликовавших эти изображения. Поскольку из таблиц `categories` и `users` требуется извлечь поля с одинаковым именем `name`, обоим полям были даны разные псевдонимы.

```
SELECT title, categories.name AS cat_name, users.name AS user_name ↳
FROM pictures ↳
INNER JOIN categories ON pictures.category = categories.id ↳
INNER JOIN users ON pictures.user = users.id;
```

title	cat_name	user_name
Цветы кактуса	Растения	jocker
Блинная скульптура	Скульптура	admin
Памятник сусликам	Скульптура	jocker
Зимний закат	Пейзаж	basil
Кот ученый	Скульптура	jocker
Памятник собаке-поводырю	Скульптура	admin
Пустынная аллея	Пейзаж	basil
Роза	Растения	jocker
Бегемотики	Скульптура	jocker

14.2.3. Фильтрация записей

> *Фильтрация* — извлечение произвольного количества записей по идентифицирующему их значению или комбинации значений.

Фильтрация выполняется языковой конструкцией `WHERE`:

```
SELECT . . . [<создание связей>] WHERE <условия>
```

Условия записываются так же, как и в PHP, за тем исключением, что их не нужно брать в круглые скобки.

Обычные значения записываются в *условиях* в следующих форматах:

- числа — указываются как есть;
- строки — берутся в одинарные или двойные кавычки;
- логические величины — в виде чисел `1` (`TRUE`) или `0` (`FALSE`);
- временны́е отметки — в формате:

  <год>-<номер месяца от 1 до 12>-<число> <часы>:<минуты>:<секунды>

  И также берутся в одинарные или двойные кавычки.

В *условиях* можно использовать следующие операторы:

- `=` — аналогичен оператору `==` PHP. Пример выборки категории с номером 3:

```
SELECT name FROM categories WHERE id = 3;
```

name
Пейзаж

Выбираем все комментарии, оставленные пользователем `jocker`:

```
SELECT contents, users.name AS user_name FROM comments ↳
INNER JOIN users ON comments.user = users.id ↳
WHERE users.name = 'jocker';
```

contents	user_name
Какое пустынное место...	jocker

- `!=`, `<`, `<=`, `>`, `>=` — аналогичны одноименным операторам PHP. Пример выборки всех изображений с номерами больше 5:

```
SELECT id, title FROM pictures WHERE id > 5;
```

id	title
6	Памятник собаке-поводырю
7	Пустынная аллея
8	Роза
9	Бегемотики

Выборка изображений, опубликованных не позднее 13:45 6 сентября 2019 года:

```
SELECT title, uploaded FROM pictures ↳
WHERE uploaded < '2019-09-06 13:45:00';
```

title	uploaded
Цветы кактуса	2019-09-06 13:38:41
Блинная скульптура	2019-09-06 13:41:19
Памятник сусликам	2019-09-06 13:43:51

- `&&`, `AND`, `||`, `OR`, `XOR`, `!` — аналогичны одноименным операторам PHP. Пример выборки изображений, опубликованных после 13:55 6 сентября 2019 года пользователем с номером 1:

```
SELECT title, uploaded, user FROM pictures ↳
WHERE uploaded > '2019-09-06 13:55:00' AND user = 1;
```

title	uploaded	user
Памятник собаке-поводырю	2019-09-06 13:55:46	1

Выборка изображений, опубликованных после 13:55 6 сентября 2019 года ИЛИ пользователем с номером 2:

```
SELECT title, uploaded, user FROM pictures ↳
WHERE uploaded > '2019-09-06 13:55:00' OR user = 2;
```

title	uploaded	user
Зимний закат	2019-09-06 13:50:01	2
Памятник собаке-поводырю	2019-09-06 13:55:46	1
Пустынная аллея	2019-09-06 13:58:40	2
Роза	2019-09-06 14:00:01	3
Бегемотики	2019-09-06 14:01:14	3

Выборка всех изображений, не относящихся к растениям и скульптуре:

```
SELECT title, categories.name AS cat_name FROM pictures ↳
INNER JOIN categories ON pictures.category = categories.id ↳
WHERE !(categories.name = 'Растения' OR ↳
categories.name = 'Скульптура');
```

title	cat_name
Зимний закат	Пейзаж
Пустынная аллея	Пейзаж

- `NOT` — то же, что и оператор `!` PHP;
- *<поле>* `IN (`*<список величин через запятую>*`)` — возвращает `TRUE`, если значение *поля* равно одной из величин, приведенных в *списке*. Выбираем все изображения категорий с номерами 1 и 3:

```
SELECT title, category FROM pictures WHERE category IN (1, 3);
```

title	category
Цветы кактуса	1
Зимний закат	3
Пустынная аллея	3
Роза	1

- *<поле>* `NOT IN (`*<список величин через запятую>*`)` — возвращает `TRUE`, если значение *поля не* равно не одной из величин, приведенных в *списке*. Выбираем все изображения, не относящиеся к категориям с номерами 1 и 3:

```
SELECT title, category FROM pictures WHERE category NOT IN (1, 3);
```

title	category
Блинная скульптура	4
Памятник сусликам	4
Кот ученый	4
Памятник собаке-поводырю	4
Бегемотики	4

- *<поле>* BETWEEN *<минимум>* AND *<максимум>* — возвращает TRUE, если значение *поля* находится в диапазоне между указанными *минимумом* и *максимумом* включительно. Выводим изображения с номерами от 3 до 6;

```
SELECT id, title FROM pictures WHERE id BETWEEN 3 AND 6;
```

id	title
3	Памятник сусликам
4	Зимний закат
5	Кот ученый
6	Памятник собаке-поводырю

- *<поле>* NOT BETWEEN *<минимум>* AND *<максимум>* — возвращает TRUE, если значение *поля* ***не*** находится в диапазоне между указанными *минимумом* и *максимумом* включительно;
- *<поле>* IS NULL — возвращает TRUE, если *поле* хранит NULL;
- *<поле>* IS NOT NULL — возвращает TRUE, если *поле* хранит значение, отличное от NULL;
- *<поле>* LIKE *<шаблон>* — возвращает TRUE, если *поле* хранит значение, совпадающее с указанным *шаблоном*. В *шаблоне* можно использовать следующие специализированные литералы:
  - % — совпадает с любым количеством (включая 0) любых символов;
  - _ (подчеркивание) — совпадает с любым одиночным символом;
  - \% — символ процента;
  - _ — символ подчеркивания.

  Выбираем всех пользователей, чьи имена состоят из пяти символов;

```
SELECT name FROM users WHERE name LIKE '_____';
```

name
admin
basil

Выбираем все изображения, у которых в названии или описании присутствует подстрока «институт»:

```
SELECT title, description FROM pictures ↳
WHERE title LIKE '%институт%' OR description LIKE '%институт%';
```

title	description
Цветы кактуса	Снято в Волжском гуманитарном институте
Кот ученый	Поставлен по инициативе Волжского гуманитарного института

- *<поле>* NOT LIKE *<шаблон>* — возвращает TRUE, если *поле* хранит значение, *не* совпадающее с указанным *шаблоном*.

14.2.4. Сортировка записей

> *Сортировка* выполняется указанием языковой конструкции ORDER BY:
> SELECT . . . [WHERE . . .] ORDER BY *<список полей через запятую>*

Сначала сортировка выполняется по первому полю из приведенных в *списке*, потом — по второму и т. д.

По умолчанию сортировка выполняется по возрастанию. Чтобы указать сортировку по убыванию значений поля, следует после имени этого поля поставить конструкцию DESC.

Сортируем категории по их названиям:

```
SELECT * FROM categories ORDER BY name;
```

id	name	slug
2	Архитектура	architecture
3	Пейзаж	landscape
1	Растения	plants
4	Скульптура	sculpture

Сортируем изображения сначала по именам опубликовавших их пользователей, а потом — по убыванию даты публикации:

```
SELECT title, users.name AS user_name, uploaded FROM pictures ↳
INNER JOIN users ON pictures.user = users.id ↳
ORDER BY users.name, uploaded DESC;
```

title	user_name	uploaded
Памятник собаке-поводырю	admin	2019-09-06 13:55:46
Блинная скульптура	admin	2019-09-06 13:41:19
Пустынная аллея	basil	2019-09-06 13:58:40
Зимний закат	basil	2019-09-06 13:50:01
Бегемотики	jocker	2019-09-06 14:01:14
Роза	jocker	2019-09-06 14:00:01
Кот ученый	jocker	2019-09-06 13:51:37
Памятник сусликам	jocker	2019-09-06 13:43:51
Цветы кактуса	jocker	2019-09-06 13:38:41

14.2.5. Выборка заданного количества записей

Для выборки указанного количества записей к запросу следует добавить языковую конструкцию LIMIT:

```
SELECT . . . [WHERE . . .] [ORDER BY . . .] ↵
LIMIT [<номер первой выбираемой записи>, ]<количество выбираемых записей>
```

Нумерация записей в таблице начинается с 0. Если *номер первой выбираемой записи* не указан, выборка выполняется с начала таблицы — с первой записи.

Выбираем первые три изображения:

```
SELECT id, title FROM pictures LIMIT 3;
```

id	title
1	Цветы кактуса
2	Блинная скульптура
3	Памятник сусликам

Выбираем четыре записи, начиная с четвертой:

```
SELECT id, title FROM pictures LIMIT 3, 4;
```

id	title
4	Зимний закат
5	Кот ученый
6	Памятник собаке-поводырю
7	Пустынная аллея

14.3. Вычисления над группами записей

14.3.1. Агрегатные функции

> *Агрегатная функция* — выполняет вычисления над группой записей.

Агрегатные функции записываются в списке полей после языковой конструкции SELECT с обязательным указанием псевдонима.

По умолчанию, если не задана группировка записей (о ней разговор пойдет в *разд. 14.3.2*), агрегатные функции обрабатывают все записи в таблице (с учетом заданных условий фильтрации).

Посредством агрегатных функций можно получить:

- количество записей — функция COUNT(*). Пример подсчета общего количества изображений:

```
SELECT COUNT(*) AS pic_cnt FROM pictures;
```

pic_cnt
9

Считаем, у какого количества изображений не заполнено описание:

```
SELECT COUNT(*) AS pic_cnt FROM pictures WHERE description IS NULL;
```

pic_cnt
4

- количество уникальных значений заданного *поля* — функция `COUNT(DISTINCT <поле>)`. Подсчитываем количество категорий, к которым относятся опубликованные изображения:

```
SELECT COUNT(DISTINCT category) AS cat_cnt FROM pictures;
```

cat_cnt
3

- сумма значений *поля* — функция `SUM([DISTINCT] <поле>)`. Если указана конструкция `DISTINCT`, суммируются только уникальные значения *поля*;
- минимальное из значений *поля* — функция `MIN(<поле>)`;
- максимальное из значений *поля* — функция `MAX(<поле>)`;
- среднее арифметическое от значений *поля* — функция `AVG([DISTINCT] <поле>)`. Если указана конструкция `DISTINCT`, среднее подсчитывается на основе только уникальных значений *поля*.

14.3.2. Группировка записей

> *Группировка записей* выполняется по значению какого-либо поля (полей). Каждая из получившихся в результате групп объединяет записи, содержащие в этом поле одинаковые значения (или одинаковые комбинации значений в указанных полях).

Если задана группировка записей, агрегатные функции обрабатывают каждую из получившихся групп отдельно и для каждой из них возвращают отдельный результат.

Группировка задается указанием языковой конструкции `GROUP BY`, которая ставится между конструкциями `WHERE` и `ORDER BY`:

```
SELECT . . . [WHERE . . .] GROUP BY <список полей через запятую> ↳
[ORDER BY . . .] [LIMIT . . .]
```

Вывод списка категорий с указанием количества относящихся к ним изображений:

```
SELECT categories.name AS cat_name, COUNT(*) AS pic_cnt FROM categories ↳
INNER JOIN pictures ON categories.id = pictures.category ↳
GROUP BY categories.name;
```

cat_name	pic_cnt
Пейзаж	2
Растения	2
Скульптура	5

Обратим внимание, что в полученном результате выводятся только категории, к которым относится хотя бы одно опубликованное изображение. Чтобы вывести все категории, придется применить вложенный запрос (о них мы поговорим в *разд. 14.4*).

Подсчет количества изображений, выгруженных отдельными пользователями, с разбивкой по категориям:

```
SELECT users.name AS user_name, categories.name AS cat_name, ↳
COUNT(*) AS pic_cnt FROM pictures ↳
INNER JOIN users ON pictures.user = users.id ↳
INNER JOIN categories ON pictures.category = categories.id ↳
GROUP BY users.name, categories.name;
```

user_name	cat_name	pic_cnt
admin	Скульптура	2
basil	Пейзаж	2
jocker	Растения	2
jocker	Скульптура	3

14.3.3. Фильтрация групп

Отфильтровать группы по какому-либо критерию можно, дописав в запрос сразу после конструкции `GROUP BY` языковую конструкцию `HAVING`:

```
SELECT . . . [WHERE . . .] GROUP BY . . . HAVING <условия> ↳
[ORDER BY . . .] [LIMIT . . .]
```

Условия записываются так же, как и в языковой конструкции `WHERE`. В них можно использовать агрегатные функции.

Выводим только пользователей, опубликовавших не менее трех изображений:

```
SELECT users.name AS user_name, COUNT(*) AS pic_cnt FROM users ↳
INNER JOIN pictures ON users.id = pictures.user GROUP BY users.name ↳
HAVING COUNT(*) >= 3;
```

user_name	pic_cnt
jocker	5

14.4. Вложенные запросы

> *Вложенный запрос* является частью другого SQL-запроса. Всегда берется в круглые скобки.

Вложенные запросы обычно используются в двух случаях:

- вычисление какого-либо значения, которое будет подставлено в результат, возвращаемый «внешним» запросом. Такой вложенный запрос записывается в конструкции `SELECT` наряду с обычными полями.

Выводим список *всех* категорий с указанием количества относящихся к ним изображений:

```
SELECT categories.name AS cat_name, ↳
(SELECT COUNT(*) FROM pictures WHERE pictures.category = ↳
categories.id) AS pic_cnt FROM categories;
```

cat_name	pic_cnt
Архитектура	0
Пейзаж	2
Растения	2
Скульптура	5

- вычисление значений в условиях фильтрации, задаваемых конструкцией `WHERE` или `HAVING`.

Выводим имя пользователя, опубликовавшего наибольшее количество изображений:

```
SELECT users.name AS user_name FROM users WHERE users.id = ↳
(SELECT pictures.user FROM pictures GROUP BY pictures.user ↳
ORDER BY COUNT(*) DESC LIMIT 1);
```

user_name
jocker

Вложенный запрос может возвращать не одно значение, а целый набор, который также можно использовать в условиях фильтрации. В таком случае пригодятся следующие операторы:

- `ANY` — выполняет сравнение с любым значением из набора, выдаваемого *вложенным запросом*. Записывается в формате:

  <поле> <оператор сравнения> ANY *<вложенный запрос>*

  В качестве *оператора сравнения* можно применить любой из операторов, поддерживаемых SQL, — например: `=` или `!=`.

  Выводим список изображений, к которым были оставлены комментарии:

```
SELECT title FROM pictures WHERE id = ANY ↳
(SELECT picture FROM comments);
```

title
Блинная скульптура
Памятник сусликам
Пустынная аллея

- `ALL` — выполняет сравнение со всеми значениями выдаваемого *вложенным запросом* набора. Записывается в формате:

  <поле> <оператор сравнения> ALL *<вложенный запрос>*

Вывести список изображений без комментариев:

```
SELECT title FROM pictures WHERE id != ALL ↳
(SELECT picture FROM comments);
```

title
Цветы кактуса
Зимний закат
Кот ученый
Памятник собаке-поводырю
Роза
Бегемотики

- *<поле>* IN *<вложенный запрос>* — возвращает TRUE, если значение *поля* равно одной из величин, возвращенных *вложенным запросом*.

  Другой способ вывести список изображений без комментариев:

```
SELECT title FROM pictures WHERE NOT id IN ↳
(SELECT picture FROM comments);
```

- EXISTS *<вложенный запрос>* — возвращает TRUE, если *вложенный запрос* выдал хотя бы одну запись.

  Выводим список категорий, к которым относится хотя бы одно опубликованное изображение:

```
SELECT name FROM categories WHERE EXISTS ↳
(SELECT pictures.id FROM pictures WHERE category = ↳
categories.id);
```

name
Пейзаж
Растения
Скульптура

- NOT EXISTS *<вложенный запрос>* — возвращает TRUE, если *вложенный запрос* не выдал ни одной записи.

14.5. Добавление, правка и удаление записей

14.5.1. Добавление записей

Для добавления записей в таблицу применяется SQL-команда INSERT:

```
INSERT INTO <таблица> (<список полей через запятую>) ↳
VALUES (<список значений, заносимых в поля, через запятую>);
```

Добавляем новую категорию:

```
INSERT INTO categories (name, slug) VALUES ('Люди', 'people');
```

Можно добавить сразу несколько записей, записав после конструкции VALUES наборы значений, взятые в круглые скобки, через запятую.

Добавляем сразу три новых категории:

```
INSERT INTO categories (name, slug) VALUES ('Животные', 'animals'), ↳
('Машины', 'cars'), ('Небо', 'sky');
```

14.5.2. Правка записей

Исправить запись позволяет команда UPDATE:

```
UPDATE <таблица> SET <поле 1> = <новое значение 1>,
                     <поле 2> = <новое значение 2>,
                     . . .
                     <поле n> = <новое значение n>
WHERE <условие для поиска исправляемой записи>;
```

Обычно исправляемую запись ищут по ее уникальному номеру или слагу.

Исправляем изображение № 4, заменив название на «Закат зимой», а описание — на «Снято около 16:00»:

```
UPDATE pictures SET title = 'Закат зимой', ↳
description = 'Снято около 16:00' WHERE id = 4;
```

14.5.3. Удаление записей

Удаляет запись команда DELETE:

```
DELETE FROM <таблица> WHERE <условие для поиска удаляемой записи>;
```

Удаляем категорию «Небо», выполнив поиск по ее номеру — 8:

```
DELETE FROM categories WHERE id = 8;
```

Урок 15
Выполнение запросов к базам данных MySQL средствами PHP

PHP предоставляет удобные средства для взаимодействия с MySQL: отправку SQL-запросов на обработку данных и получение результатов их выполнения.

15.1. Соединение с базой данных

Чтобы установить соединение с базой данных, следует создать объект встроенного класса `PDO`. Его конструктор имеет следующий формат вызова:

```
PDO(<строка соединения>, <имя пользователя>, <пароль>)
```

Значения всех параметров указываются в виде строк.

Строка соединения содержит все необходимые параметры для подключения к базе и записывается в формате:

```
mysql:<список параметров и их значений через точку с запятой>
```

Сами параметры и их значения записываются в формате *<имя параметра>*=*<значение>*. С их помощью можно указать:

- интернет-адрес компьютера, на котором работает MySQL, — в параметре `host`;
- TCP-порт, через который работает MySQL, — в параметре `port`. Если не указан, будет использоваться порт по умолчанию 3306;
- имя базы данных — `dbname`;
- кодировка — `charset`. Чтобы указать кодировку UTF-8, следует дать этому параметру значение `utf8`.

Полученный объект класса `PDO` предоставляет соединение с базой данных. С его помощью можно отправлять к ней запросы.

Пример соединения с базой `photogallery`, обслуживаемой СУБД MySQL, которая работает на локальном хосте, от имени пользователя `root` с пустым паролем:

```
$conn = new PDO('mysql:host=localhost;dbname=photogallery;charset=utf8',
    'root', '');
```

По окончании работы следует закрыть соединение с базой данных. Для этого достаточно уничтожить полученный ранее объект-соединение, присвоив переменной, где он хранится, любое другое значение — обычно NULL.

Разрываем установленное ранее соединение с базой данных photogallery:

```
$conn = NULL;
```

Полезно знать...

Любое значение ссылочного типа (объект), на которое не ссылается ни одна переменная, считается ненужным и поступает в распоряжение *сборщика мусора* — подсистемы PHP, уничтожающей неиспользуемые данные.

Перед уничтожением объекта будет вызван его деструктор (если он объявлен в классе). Код, разрывающий соединение с базой данных, записан в деструкторе класса PDO и, таким образом, выполнится непосредственно перед уничтожением объекта-соединения.

15.2. Выполнение простых запросов

Простые запросы можно выполнить средствами объекта класса PDO, представляющего соединение с базой данных.

15.2.1. Простые запросы на выборку записей

Для выполнения запросов на выборку записей применяется метод query объекта-соединения:

```
query(<код выполняемого SQL-запроса>[, <режим выборки>[,
      <дополнительный параметр>]])
```

В качестве результата метод вернет объект класса PDOStatement, представляющий выполненный запрос и обладающий функциональностью итератора. При каждой итерации этот объект возвращает массив или объект с содержимым очередной записи, полученной в результате выполнения SQL-запроса. Что именно будет возвращаться методом, зависит от заданного *режима выборки*:

- PDO::FETCH_ASSOC — ассоциативный массив, в котором каждый элемент представляет одно поле и доступен по ключу, совпадающему с именем поля:

```
$result = $conn->query('SELECT name, slug FROM categories;',
    PDO::FETCH_ASSOC);
foreach ($result as $row)                    // Растения - plants
    echo $row['name'], ' - ',                // Архитектура - architecture
        $row['slug'], "\r\n";                // Пейзаж - landscape
                                             // Скульптура - sculpture
```

- PDO::FETCH_NUM — индексированный массив, элементы которого представляют отдельные поля возвращенного результата и располагаются в том же порядке, в котором располагаются поля:

```
$result = $conn->query('SELECT name, slug FROM categories;',
    PDO::FETCH_NUM);
foreach ($result as $row)
    echo $row[0], ' - ', $row[1],
        "\r\n";                          // Результат тот же, что и ранее
```

- PDO::FETCH_BOTH — комбинированный массив, содержащий элементы-поля как с ключами, так и с индексами (поведение по умолчанию, если *режим выборки* не указан):

```
$result = $conn->query('SELECT name, slug FROM categories;');
foreach ($result as $row)
    echo $row[0], ' - ',
        $row['slug'], "\r\n";            // Результат тот же, что и ранее
```

- PDO::FETCH_COLUMN — значение поля, порядковый номер которого указан в *дополнительном параметре* (нумерация полей начинается с 0):

```
$result = $conn->query('SELECT * FROM categories ORDER BY slug;',
    PDO::FETCH_COLUMN, 2);
foreach ($result as $slug)
    echo $slug, ' ';          // architecture landscape plants sculpture
```

- PDO::FETCH_CLASS — новый объект класса, имя которого в виде строки указано в *дополнительном параметре*. Значения полей результата заносятся в одноименные свойства объекта:

```
class User {
    public $id;
    public $name;
}
$result = $conn->query('SELECT id, name FROM users;',
    PDO::FETCH_CLASS, 'User');
foreach ($result as $user)
    echo $user->id, ': ',
        $user->name, '   ';          // 1: admin   2: basil   3: jocker
```

  При использовании этого режима для каждой записи из возвращенного результата будет создан отдельный объект указанного класса, что не всегда оправдано и может привести к перерасходу памяти;

- PDO::FETCH_INTO — созданный заранее объект, указанный в *дополнительном параметре*. Значения полей результата заносятся в одноименные свойства этого объекта;

```
$user = new User();
$result = $conn->query('SELECT * FROM users;', PDO::FETCH_INTO,
    $user);
foreach ($result as $user)
    echo $user->id, ': ',
        $user->name, '   ';          // 1: admin   2: basil   3: jocker
```

В этом случае для извлечения всех записей будет использован один объект, который следует создать заранее.

15.2.2. Простые запросы на изменение данных

Запросы на изменение данных, не возвращающие результат в виде набора записей, выполняются вызовом метода `exec` объекта-соединения:

```
exec(<код выполняемого SQL-запроса>)
```

Метод возвращает количество записей, обработанных запросом: добавленных, исправленных или удаленных.

Добавляем новую категорию:

```
$n = $conn->exec("INSERT INTO categories (name, slug) VALUES " .
    "('Люди', 'people');");
// Сколько записей было добавлено?
echo $n;                                                    // 1
```

> ***Значения, подставляемые в SQL-запросы***
>
> Если значения, подставляемые в SQL-запросы (например, предназначенные для занесения в поля добавляемой записи), вычисляются в процессе работы программы или, тем более, получаются из веб-форм, для их вставки в запрос лучше использовать средства по выполнению параметризованных запросов (см. *разд. 15.3*). Эти средства перед вставкой в запрос преобразуют значения к нужному формату и очищают их от потенциально опасных данных, которые злоумышленник может ввести в веб-форму.

15.3. Параметризованные запросы

> *Параметризованный запрос* — запрос с параметрами, в которые перед его выполнением подставляются конкретные значения.

15.3.1. Подготовка параметризованного запроса

Для подготовки параметризованного запроса к выполнению нужно вызвать метод `prepare` объекта-соединения:

```
prepare(<код подготавливаемого SQL-запроса>)
```

Возвращаемый результат — объект класса `PDOStatement`, представляющий подготовленный запрос.

Параметры в *коде запроса* можно записать как:

- *неименованные* — обозначаемые вопросительным знаком `?`. Пример запроса на выборку изображений, опубликованных в указанный промежуток времени, с двумя параметрами, задающими начальную и конечную дату этого промежутка:

```
$qryl = $conn->prepare('SELECT * FROM pictures ' .
    'WHERE uploaded >= ? AND uploaded <= ?;');
```

- *именованные* — записываемые в формате :<*имя параметра*>. Пример запроса на выборку комментариев с параметрами picture и user, задающими номера изображения и пользователя:

```
$qry2 = $conn->prepare('SELECT * FROM comments ' .
    'WHERE picture = :picture AND user = :user;');
```

15.3.2. Задание значений параметров в параметризованном запросе

Значения параметров у параметризованного запроса можно как задать непосредственно, так и указать брать их из заданных переменных.

- Непосредственное задание значений для параметров запроса выполняется методом bindValue, вызываемого у объекта-запроса, относящегося к классу PDOStatement:

```
bindValue(<порядковый номер или имя параметра>, <значение>[,
        <тип значения>])
```

Нумерация параметров в запросах начинается с 1. *Имена параметров* записываются с начальным символом двоеточия.

Типы полей, поддерживаемых MySQL, и обозначения *типов значений*, которые следует указать в вызовах метода bindValue, приведены в табл. 15.1. Если *тип значения* не указан, используется тип PDO::PARAM_STR.

***Таблица 15.1.** Типы полей MySQL и соответствующие им типы значений, указываемые в вызовах метода bindValue*

Тип поля MySQL	Тип значения для метода bindValue
TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT	PDO::PARAM_INT
FLOAT, DOUBLE, DECIMAL	PDO::PARAM_STR
VARCHAR	
TINYTEXT, MEDIUMTEXT, TEXT, LONGTEXT	
BOOLEAN	PDO::PARAM_BOOL
TIMESTAMP	PDO::PARAM_STR

Если необходимо занести в какое-либо поле значение NULL, в качестве типа следует поставить PDO::PARAM_NULL.

Метод возвращает TRUE при успешном задании значения у параметра и FALSE в противном случае.

Пример указания значений для неименованных параметров запроса $qry1:

```
$qry1->bindValue(1, '2019-09-06 13:50:00');
$qry1->bindValue(2, '2019-09-06 14:00:00');
```

- Связывание параметра запроса с указанной *переменной*, после чего параметр получит свое значение из этой *переменной*. Выполнить это позволяет метод `bindParam`, аналогичный рассмотренному ранее методу `bindValue`:

```
bindParam(<порядковый номер или имя параметра>, <переменная>[,
          <тип значения>])
```

  Пример указания значений для именованных параметров запроса `$qry2`:

```
$picture_id = 7;
$user_id = 3;
$qry2->bindParam(':picture', $picture_id, PDO::PARAM_INT);
$qry2->bindParam(':user', $user_id, PDO::PARAM_INT);
```

Кроме того, значения для параметров можно задать непосредственно при выполнении запроса (см. *разд. 15.3.3*).

15.3.3. Выполнение запроса

Выполнение подготовленного заранее запроса производится вызовом метода `execute` объекта-запроса:

```
execute([<массив со значениями параметров>])
```

Если параметры запроса были заданы ранее вызовами методов `bindValue` и `bindParam`, этот метод вызывается без параметров. В противном случае значения для параметров запроса можно указать в массиве: индексированном — для неименованных параметров, или ассоциативном — для именованных параметров (в качестве ключей у элементов нужно задать имена параметров с начальными символами двоеточия). Всем значениям параметров дается тип `PDO::PARAM_STR`, и изменить его нельзя.

> ***О типах параметров***
>
> Как показывает практика, у целых и вещественных чисел, заносимых в параметры запроса, можно указывать тип `PDO::PARAM_STR` — и запрос будет успешно выполнен.

Примеры:

```
$qry1->execute(['2019-09-06 13:50:00', '2019-09-06 14:00:00']);
$qry2->execute([':picture' => 7, ':user' => 3]);
```

15.3.4. Получение результатов

> ***Результат возвращают лишь запросы на выборку данных***
>
> Запросы на изменение данных результата не возвращают.

Чтобы получить текущую запись из результата запроса, следует вызвать метод `fetch` объекта-запроса:

```
fetch([<режим выборки>])
```

В качестве *режима выборки* можно указать константу PDO::FETCH_ASSOC, PDO::FETCH_NUM или PDO::FETCH_BOTH. Все они, равно как и обозначаемые ими режимы выборки, описаны в *разд. 15.1*.

Метод возвращает, в зависимости от заданного *режима выборки*, массив или объект со значениями из очередной записи и готовит для выборки следующую. Таким образом, последовательно вызывая метод fetch, можно перебрать все записи из результата запроса. Когда записи закончатся, метод вернет FALSE.

В качестве примера выбираем и выводим все изображения, опубликованные между 13:50 и 14:00 6 сентября 2019 года:

```
$qry1 = $conn->prepare('SELECT title, uploaded FROM pictures ' .
    'WHERE uploaded >= ? AND uploaded <= ? ORDER BY uploaded DESC;');
$qry1->execute(['2019-09-06 13:50:00', '2019-09-06 14:00:00']);
while ($row = $qry1->fetch(PDO::FETCH_ASSOC))
    echo $row['title'], ' - ', $row['uploaded'], "\r\n";
                        // Пустынная аллея - 2019-09-06 13:58:40
                        // Памятник собаке-поводырю - 2019-09-06 13:55:46
                        // Кот ученый - 2019-09-06 13:51:37
                        // Зимний закат - 2019-09-06 13:50:01
```

Чтобы получить запись в виде объекта класса с заданным *именем*, вызовите метод fetchObject объекта-запроса:

```
fetchObject([<имя класса в виде строки>])
```

При каждом вызове метод возвращает новый объект, хранящий сведения об очередной записи, или FALSE, если записи кончились.

Выбираем изображения, опубликованные с 13:30 по 13:45 6 сентября 2019 года:

```
class Picture {
    public $title;
    public $uploaded;
}
$qry1->execute(['2019-09-06 13:30:00', '2019-09-06 13:45:00']);
while ($obj = $qry1->fetchObject('Picture'))
    echo $obj->title, ' - ', $obj->uploaded, "\r\n";
                              // Памятник сусликам - 2019-09-06 13:43:51
                              // Блинная скульптура - 2019-09-06 13:41:19
                              // Цветы кактуса - 2019-09-06 13:38:41
```

Чтобы получить значение указанного поля, нужно вызвать метод fetchColumn:

```
fetchColumn([<номер поля, чье значение нужно получить>])
```

Нумерация полей начинается с 0. Если *номер поля* не указан, возвращается значение первого поля (с номером 0).

После возврата значения поля также выполняется перемещение на следующую запись. Если записей больше нет, возвращается FALSE.

Выбираем все комментарии к изображению № 7, оставленные пользователем № 3:

```
$qry2 = $conn->prepare('SELECT contents FROM comments ' .
    'WHERE picture = :picture AND user = :user;');
$qry2->execute([':picture' => 7, ':user' => 3]);
while ($cnt = $qry2->fetchColumn())
    echo $cnt, "\r\n";                          // Какое пустынное место...
```

Получить все записи можно вызовом метода `fetchAll` объекта-курсора:

```
fetchAll([<режим выборки>[, <дополнительный параметр>]])
```

В качестве *режима выборки* можно указать константу: `PDO::FETCH_ASSOC`, `PDO::FETCH_NUM`, `PDO::FETCH_BOTH`, `PDO::FETCH_COLUMN` или `PDO::FETCH_CLASS` (см. *разд. 15.1*).

Метод возвращает массив массивов или объектов (в зависимости от указанного режима выборки), каждый из которых представляет одну запись из результата выполненного запроса.

Выбираем все комментарии к изображению № 3, оставленные пользователем № 1:

```
$qry2->execute([':picture' => 3, ':user' => 1]);
$result = $qry2->fetchAll(PDO::FETCH_COLUMN, 0);
foreach ($result as $cnt)
    echo $cnt, "\r\n";                          // Забавные зверушки
```

15.4. Дополнительные инструменты

- Получение номера последней добавленной записи — метод `lastInsertId()` объекта-соединения:

```
$conn->exec("INSERT INTO categories (name, slug) VALUES " .
    "('Люди', 'people');");
echo $conn->lastInsertId();                     // 5
```

- получение количества записей, добавленных, исправленных или удаленных последним параметризованным запросом, — метод `rowCount()` объекта-запроса:

```
$qry3 = $conn->prepare('DELETE FROM categories WHERE id = :cat_id;');
$qry3->execute([':cat_id' => 5]);
echo $qry3->rowCount();                         // 1
```

- освобождение памяти, занимаемой результатом выполнения параметризованного запроса, перед выполнением того же запроса с другим набором параметров — метод `closeCursor()` объекта-запроса:

```
$qry = $conn->prepare( . . . );
// Выполняем запрос с одним набором параметров
$qry->execute( . . . );
$qry-fetchAll();
// Очищаем занятую запросом память
$qry->closeCursor();
```

```
// Выполняем запрос с другим набором параметров
$qry->execute( . . . );
. . .
```

15.5. Обработка ошибок при работе с базами данных

При возникновении ошибки в коде SQL-запроса или нештатной ситуации при работе с базой данных возникает исключение `PDOException`.

Пример обработки этого исключения:

```
try {
    $conn = new PDO('mysql:host=localhost;dbname=photogallery;' .
        'charset=utf8', 'root', '');
} catch(PDOException $e) {
    // Ошибка? Предпринимаем какие-либо действия по ее устранению.
}
```

15.6. Упражнение. Пишем базовый класс модели, работающей с базой данных

Для обновленной фотогалереи, работающей с базой данных, напишем базовый класс `Models\Model`, который включит функциональность, типовую для всех моделей.

Этот класс будет содержать следующие общедоступные и защищенные элементы:

- защищенные константы:
  - `TABLE_NAME` — имя таблицы, с которой работает модель;
  - `DEFAULT_ORDER` — порядок сортировки по умолчанию;
  - `RELATIONS` — описание связей с первичными таблицами. Представляет собой ассоциативный массив, каждый элемент которого имеет ключ, совпадающий с именем связываемой таблицы, и хранит вложенный ассоциативный массив с элементами: `external` (имя поля внешнего ключа), `primary` (имя ключевого поля в связываемой таблице) и необязательный `type` (тип связи: `INNER`, `LEFT` или `RIGHT`). Если он не указан, будет создана связь типа `INNER`.

  В классе `Models\Model` все эти константы получат в качестве значений пустые строки и пустой массив. В производных классах моделей, которые будут работать с конкретными таблицами, мы переопределим их;
- общедоступные методы:
  - `run` — выполняет заданный *запрос* с указанными *параметрами*:

    `run(<SQL-запрос>[, <массив с параметрами>])`

Запрос передается в виде строки с его SQL-кодом. Если *запрос* непараметризованный, *массив с параметрами* указывать не нужно;

- `select` — формирует запрос из переданных в качестве параметров составных частей, значений констант: `TABLE_NAME`, `DEFAULT_ORDER` и `RELATIONS`, после чего выполняет его:

```
select([<список извлекаемых полей (SELECT)>[,
        <массив с именами связываемых первичных таблиц>[,
        <условия фильтрации (WHERE)>[, <массив с параметрами>[,
        <порядок сортировки (ORDER)>[,
        <номер первой выбираемой записи>,
        <количество выбираемых записей (LIMIT)>[,
        <условия группировки (GROUP)>[,
        <условия фильтрации групп (HAVING)>]]]]]]]])
```

Если *список извлекаемых полей* не указан, будут извлечены все поля. Если не указан *порядок сортировки*, будет использован порядок, заданный в константе класса `DEFAULT_ORDER`;

- методы `current`, `key`, `next`, `rewind` и `valid`, объявленные в интерфейсе `Iterator` (см. *разд. 10.2.1*), — в них будет выполняться выборка записей из результата выполнения запроса;

- `get_record` — выбирает согласно заданным *условиям фильтрации* единственную запись и возвращает ее в качестве результата:

```
get_record([<список извлекаемых полей (SELECT)>[,
            <массив с именами связываемых первичных таблиц>[,
            <условия фильтрации (WHERE)>[,
            <массив с параметрами>]]]])
```

- `get` — выбирает согласно заданному *значению* указанного *поля* конкретную запись и возвращает ее в качестве результата:

```
get(<искомое значение>[, <поле, в котором ищется значение>[,
    <список извлекаемых полей>[,
    <массив с именами связываемых первичных таблиц>]]])
```

Если *поле* не указано, поиск *значения* будет выполняться в поле `id`;

- `get_or_404` — то же самое, что и `get`, но в случае, если подходящая запись не будет найдена, генерирует исключение `Page404Exception` (см. *разд. 12.5*).

Файлы сайта фотогалереи хранятся в папке 12\12.5\ex, а файл photogallery.sql с дампом базы данных `photogallery` — в папке 13\13.4\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Сначала добавим в пространство имен `Settings` константы, задающие параметры подключения к базе данных: `DB_HOST` (интернет-адрес СУБД), `DB_NAME` (имя базы данных), `DB_USERNAME` (имя пользователя) и `DB_PASSWORD` (его пароль).

   Откроем конфигурационный модуль modules\settings.php в текстовом редакторе и добавим эти константы (здесь и далее добавления и правки в ранее написанном коде выделены полужирным шрифтом):

```
<?php
namespace Settings {
    . . .
    const DB_HOST = 'localhost';
    const DB_NAME = 'photogallery';
    const DB_USERNAME = 'root';
    const DB_PASSWORD = '';
}
```

2. Нам понадобится функция, которая создаст соединение с базой данных и вернет представляющий его объект класса `PDO`. Назовем эту функцию `connect_to_db` и поместим в пространство имен `Helpers`.

   Откроем модуль modules\helpers.php, в котором объявлено это пространство имен, и добавим объявление функции:

```
<?php
namespace Helpers {
    . . .
    function connect_to_db(){
        $conn_str = 'mysql:host=' . \Settings\DB_HOST . ';dbname=' .
            \Settings\DB_NAME . ';charset=utf8';
        return new \PDO($conn_str, \Settings\DB_USERNAME,
            \Settings\DB_PASSWORD);
    }
}
```

   Код базового класса модели `Models\Model` весьма объемен, и мы будем писать его по частям.

3. Создадим модуль modules\models\Model.php и запишем в него первую часть объявления класса `Models\Model`:

```
<?php
namespace Models;
require_once $base_path . 'modules\helpers.php';
class Model implements \Iterator {
    protected const TABLE_NAME = '';
    protected const DEFAULT_ORDER = '';
    protected const RELATIONS = [];

    static private $connection = NULL;
    static private $connection_count = 0;

    function __construct() {
        if (!self::$connection)
            self::$connection = \Helpers\connect_to_db();
        self::$connection_count++;
    }
```

```
    function __destruct() {
        self::$connection_count--;
        if (self::$connection_count == 0)
            self::$connection = NULL;
    }
}
```

Все модели, которые мы напишем, будут использовать одно соединение с базой данных. Мы объявим закрытые свойства `connection` и `connection_count`, которые будут хранить, соответственно, объект-соединение и количество использующих его моделей. Конструктор класса создаст соединение, если оно еще не создано, и увеличит количество моделей на единицу. Деструктор уменьшит это количество на единицу и, если соединение более не используется ни одной моделью, удалит его.

4. Добавим в класс `Models\Model` метод `run` и закрытое свойство `query`, в котором будет храниться объект класса `PDOStatement`, представляющий запрос:

```
class Model implements \Iterator {
    . . .
    private $query = NULL;

    function run($sql, $params = NULL) {
        if ($this->query)
            $this->query->closeCursor();
        $this->query = self::$connection->prepare($sql);
        if ($params) {
            foreach ($params as $key => $value) {
                $k = (is_integer($key)) ? $key + 1 : $key;
                switch (gettype($value)) {
                    case 'integer':
                        $t = \PDO::PARAM_INT;
                        break;
                    case 'boolean':
                        $t = \PDO::PARAM_BOOL;
                        break;
                    case 'NULL':
                        $t = \PDO::PARAM_NULL;
                        break;
                    default:
                        $t = \PDO::PARAM_STR;
                }
                $this->query->bindValue($k, $value, $t);
            }
        }
        $this->query->execute();
    }
}
```

Если в свойстве query уже хранится какой-то старый запрос, мы предварительно освобождаем занятую им память. Далее выполняем подготовку запроса и задаем значения его параметров, если таковые есть. Тип, который следует указать для параметра в вызове метода bindValue, определяем по типу значения этого параметра.

5. Добавим в класс Models\Model объявление метода select:

```
class Model implements \Iterator {
    . . .
    function select($fields = '*', $links = NULL, $where = '',
        $params = NULL, $order = '', $offset = NULL, $limit = NULL,
        $group = '', $having = '') {
        $s = 'SELECT ' . $fields . ' FROM ' . static::TABLE_NAME;
        if ($links)
            foreach ($links as $ext_table) {
                $rel = static::RELATIONS[$ext_table];
                $s .= ' ' . ((key_exists('type', $rel)) ?
                    $rel['type'] : 'INNER') . ' JOIN ' . $ext_table .
                    ' ON ' . static::TABLE_NAME . '.' .
                    $rel['external'] . ' = ' . $ext_table . '.' .
                    $rel['primary'];
            }
        if ($where)
            $s .= ' WHERE ' . $where;
        if ($group) {
            $s .= ' GROUP BY ' . $group;
            if ($having)
                $s .= ' HAVING ' . $having;
        }
        if ($order)
            $s .= ' ORDER BY ' . $order;
        else
            $s .= ' ORDER BY ' . static::DEFAULT_ORDER;
        if ($limit && $offset !== NULL)
            $s .= ' LIMIT ' . $offset . ', ' . $limit;
        $s .= ';';
        $this->run($s, $params);
    }
}
```

6. Добавим в класс Models\Model код, объявляющий методы интерфейса Iterator, и закрытое свойство record, в котором будет храниться массив со значениями очередной извлеченной записи:

```
class Model implements \Iterator {
    . . .
    private $record = FALSE;
```

```
    function current() {
        return $this->record;
    }

    function key() {
        return 0;
    }

    function next() {
        $this->record = $this->query->fetch(\PDO::FETCH_ASSOC);
    }

    function rewind() {
        $this->record = $this->query->fetch(\PDO::FETCH_ASSOC);
    }

    function valid() {
        return $this->record !== FALSE;
    }
}
```

Метод `key` интерфейса `Iterator` должен возвращать индекс или ключ очередного элемента последовательности. Но, поскольку индексы записей для нас интереса не представляют, у нас этот метод всегда возвращает `0`.

7. Добавим в класс `Models\Model` методы `get_record`, `get` и `get_or_404`:

```
class Model implements \Iterator {
    . . .
    function get_record($fields = '*', $links = NULL, $where = '',
        $params = NULL) {
        $this->record = FALSE;
        $this->select($fields, $links, $where, $params);
        return $this->query->fetch(\PDO::FETCH_ASSOC);
    }

    function get($value, $key_field = 'id', $fields = '*',
        $links = NULL) {
        return $this->get_record($fields, $links, $key_field . ' = ?',
            [$value]);
    }

    function get_or_404($value, $key_field = 'id', $fields = '*',
        $links = NULL) {
        $rec = $this->get($value, $key_field, $fields, $links);
        if ($rec)
            return $rec;
```

```
            else
                throw new \Page404Exception();
        }
    }
```

Метод `get_record`, помимо всего прочего, удаляет запись, ранее сохраненную в свойстве `record`, чтобы освободить занимаемую ей память.

На следующем упражнении мы сделаем новую главную страницу и страницу списка изображений, относящихся к выбранной категории (страницу категории).

15.7. Упражнение. Создаем главную веб-страницу и веб-страницу категории

Выведем на главной странице список категорий с гиперссылками, ведущими на страницы этих категорий, и три последние опубликованные изображения, при щелчке на которых будут выведены страницы для их просмотра. На странице категорий будет показываться список изображений, относящихся к выбранной категории.

Для этого необходимо:

- объявить классы моделей `Models\Category` и `Models\Picture`, работающие с таблицами `categories` и `pictures` базы данных;
- переписать метод `list` контроллера `Controllers\Images` и шаблон modules\templates\list.php, формирующие главную страницу;
- добавить в маршрутизатор код, который при получении интернет-адреса формата **/cats/<*слаг категории*>/** вызовет метод `by_cat` контроллера `Controllers\Images` и передаст ему извлеченный из адреса *слаг категории*;
- объявить в классе `Controllers\Images` метод `by_cat`, выводящий страницу категории;
- написать шаблон страницы категории modules\templates\by_cat.php;
- решить проблему корректного возврата.

  На страницу для просмотра изображения посетитель фотогалереи сможет перейти с главной страницы, со страницы категории или со страницы пользователя (мы напишем ее потом). Как выяснить, с какой страницы он пришел, и какой интернет-адрес нужно подставить в гиперссылку **Назад**?

  Проще всего передать странице просмотра изображения какой-либо идентификатор, обозначающий страницу, с которой был выполнен переход, и на основе этого идентификатора формировать интернет-адрес для гиперссылки возврата. Передачу можно выполнять через какой-либо GET-параметр.

  Дадим главной странице идентификатор `index`, странице категории — `cat`, а странице пользователя — `user`. При отсутствии идентификатора пусть выполняется возврат на страницу пользователя. А передавать его будем через GET-параметр `ref`.

Файлы сайта фотогалереи хранятся в папке 15\15.6\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Найдем в папке 15\!sources файл с новой таблицей стилей styles.css и скопируем его в корневую папку сайта, затерев имеющийся там старый файл.
2. Создадим модуль modules\models\category.php, в который запишем код модели `Models\Category` из листинга 15.1.

Листинг 15.1. Модуль modules\models\category.php

```
<?php
namespace Models;
class Category extends \Models\Model {
    protected const TABLE_NAME = 'categories';
    protected const DEFAULT_ORDER = 'name';
}
```

Объявление класса содержит лишь переопределенные константы: `TABLE_NAME` и `DEFAULT_ORDER`, задающие, соответственно, имя таблицы `categories` и порядок сортировки по полю `name`. Более ничего тут писать не нужно — всю основную функциональность мы реализовали в базовом классе `Models\Model` в *разд. 15.6*.

3. Создадим модуль modules\models\picture.php, в который запишем код модели `Models\Picture` из листинга 15.2.

Листинг 15.2. Модуль modules\models\picture.php

```
<?php
namespace Models;
class Picture extends \Models\Model {
    protected const TABLE_NAME = 'pictures';
    protected const DEFAULT_ORDER = 'uploaded DESC';
    protected const RELATIONS =
        ['categories' => ['external' => 'category',
                          'primary' => 'id'],
         'users' => ['external' => 'user', 'primary' => 'id']];
}
```

Эта модель будет работать с таблицей `pictures`, сортировать записи по убыванию временно́й отметки публикации и связываться с первичными таблицами `categories` и `users` (см. значение переопределенной константы `RELATIONS`).

4. Откроем модуль маршрутизатора modules\router.php и добавим в него код, распознающий интернет-адрес категории (здесь и далее добавления и правки в ранее написанном коде выделены полужирным шрифтом):

```
. . .
$result = [];
```

```
if (preg_match('/^cats\/(\w+)$/', $request_path, $result) == 1) {
    $ctr = new \Controllers\Images();
    $ctr->by_cat($result[1]);
} else if (preg_match('/^(\d+)$/', $request_path, $result) == 1) {
. . .
```

5. Откроем модуль modules\controllers\images.php, в котором хранится код контроллера `Controllers\Images`, и полностью перепишем метод `list`:

```
class Images extends BaseController {
    function list() {
        $cats = new \Models\Category();
        $cats->select();
        $picts = new \Models\Picture();
        $picts->select('pictures.id, title, filename, uploaded, ' .
            'users.name AS user_name, categories.name AS cat_name, ' .
            'categories.slug, (SELECT COUNT(*) FROM comments ' .
            'WHERE comments.picture = pictures.id) AS comment_count',
            ['users', 'categories'], '', NULL, '', 0, 3);
        $ctx = ['cats' => $cats, 'picts' => $picts];
        $this->render('list', $ctx);
    }
    . . .
}
```

Из таблицы `pictures` извлекаем только те поля, которые нужно вывести на странице: номер, название, имя файла, временну́ю отметку публикации, имя пользователя, название и слаг категории. Помимо этого, пользуясь вложенным запросом, вычисляем количество комментариев к каждому из выводимых изображений.

6. Добавим в класс `Controllers\Images` метод `by_cat`:

```
class Images extends BaseController {
    . . .
    function by_cat(string $slug) {
        $cats = new \Models\Category();
        $cat = $cats->get_or_404($slug, 'slug', 'id, name');
        $picts = new \Models\Picture();
        $picts->select('pictures.id, title, filename, uploaded, ' .
            'users.name AS user_name, ' .
            '(SELECT COUNT(*) FROM comments WHERE ' .
            'comments.picture = pictures.id) AS comment_count',
            ['users'], 'category = ?', [$cat['id']]);
        $ctx = ['cat' => $cat, 'picts' => $picts];
        $this->render('by_cat', $ctx);
    }
}
```

По извлеченному из пути слагу находим категорию, извлекаем ее номер и название, после чего выбираем все изображения, которые относятся к этой категории, по ее номеру.

7. Откроем модуль modules\templates\list.php с кодом шаблона главной страницы и заменим имеющийся в нем код приведенным в листинге 15.3.

Листинг 15.3. Модуль modules\templates\list.php

```
<?php require \Helpers\get_fragment_path('__header') ?>
<h2>Категории</h2>
<section id="categories">
    <?php foreach ($cats as $cat) { ?>
    <h3><a href="/cats/<?php echo $cat['slug'] ?>/">
        <?php echo $cat['name'] ?>
    </a></h3>
    <?php } ?>
</section>
<h2>Последние 3 изображения</h2>
<table id="gallery">
    <tr><th></th><th></th><th>Категория</th>
    <th>Опубликовано пользователем</th>
    <th>Дата и время публикации</th><th>Комментариев</th></tr>
    <?php foreach ($picts as $pict) { ?>
    <tr>
        <td><a href="/<?php echo $pict['id'] ?>/?ref=index">
            <img src="<?php echo \Settings\IMAGE_PATH . ↳
                $pict['filename'] ?>">
        </a></td>
        <td><a href="/<?php echo $pict['id'] ?>/?ref=index">
            <h3><?php echo $pict['title'] ?></h3>
        </a></td>
        <td><h4><a href="/cats/<?php echo $pict['slug'] ?>">
            <?php echo $pict['cat_name'] ?>
        </a></h4></td>
        <td><h4><a href="/users/<?php echo $pict['user_name'] ?>">
            <?php echo $pict['user_name'] ?>
        </a></h4></td>
        <td><?php echo $pict['uploaded'] ?></td>
        <td><?php echo $pict['comment_count'] ?></td>
    </tr>
    <?php } ?>
</table>
<?php require \Helpers\get_fragment_path('__footer') ?>
```

Под заголовком **Категории** выводим перечень категорий, каждая из которых представляет собой гиперссылку, ведущую на страницу соответствующей категории.

Под заголовком **Последние 3 изображения** выводим таблицу с тремя самыми «свежими» изображениями. Эта таблица содержит 6 колонок:

- миниатюра изображения — представляет собой гиперссылку на страницу просмотра изображения;
- название изображения — аналогичная гиперссылка;
- **Категория** — гиперссылка на страницу категории;
- **Опубликовано пользователем** с именем пользователя — гиперссылка на страницу пользователя;
- **Дата и время публикации**;
- **Комментариев** — с количеством оставленных комментариев.

Откроем в веб-обозревателе обновленную главную страницу нашей фотогалереи и проверим, правильно ли она выводится (рис. 15.1).

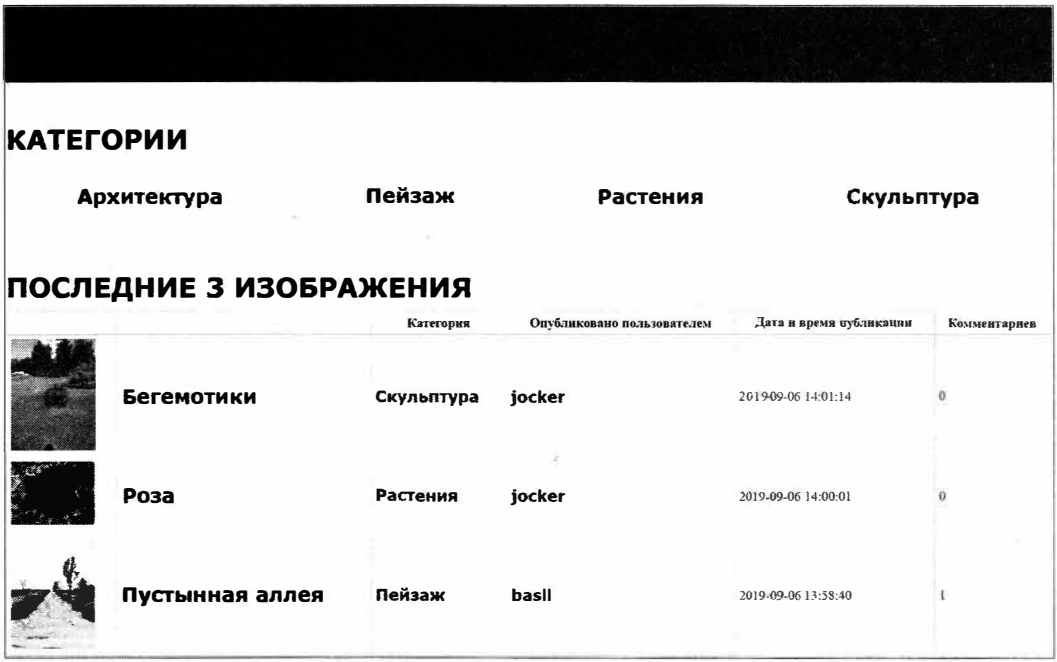


Рис. 15.1. Главная веб-страница новой фотогалереи

Сделайте сами

♦ Создайте шаблон страницы категории modules\templates\by_cat.php. Возьмите за основу шаблон главной страницы, удалите список категорий, добавьте заголовок второго уровня с названием выбранной категории, уберите из таблицы изображений колонку **Категория** и замените значение, передаваемое с GET-параметром `ref`, на `cat`.

- Откройте шаблон modules\templates__header.inc.php и превратите название сайта, выводящееся в виде заголовка первого уровня, в гиперссылку, указывающую на главную страницу.
- Откройте шаблон modules\templates__footer.inc.php и добавьте перед закрывающим тегом </body> абзац с якорем copyright со следующим текстом:

```
© Все права принадлежат читателям книги "PHP и MySQL"
```

15.8. Упражнение. Реализуем пагинацию

Если в какой-либо категории присутствует очень много, тысячи и десятки тысяч, изображений, выборка их перечня из базы данных и формирование на его основе веб-страницы займет много времени. Поэтому при выводе таких перечней применяют пагинацию.

> *Пагинация* — разбиение длинных перечней каких-либо позиций (например, опубликованных изображений) на отдельные пронумерованные части и вывод получившихся частей на отдельных веб-страницах.

> *Пагинатор* — программный инструмент (обычно класс) для выполнения пагинации.

Реализуем вывод изображений на страницах категорий с применением пагинации. Пока установим размер одной части равным двум записям — так мы сможем проверить работу пагинации, не занося в фотогалерею слишком много изображений.

Номер выводимой части будем передавать через GET-параметр с именем page, а нумеровать части — начиная с 1. Если номер части не указан, будет выведена первая часть.

Файлы сайта фотогалереи хранятся в папке 15\15.7\s сопровождающего книгу электронного архива (см. *приложение 3*).

1. Откроем конфигурационный модуль modules\settings.php в текстовом редакторе и добавим константу Settings\RECORDS_ON_PAGE, хранящую количество записей в части (здесь и далее добавления и правки в ранее написанном коде выделены полужирным шрифтом):

```
namespace Settings {
    . . .
    const RECORDS_ON_PAGE = 2;
}
```

2. Нам понадобится функция, формирующая набор GET-параметров. Назовем ее get_GET_params и поместим в пространство имен Helpers. Вот формат вызова этой функции:

```
get_GET_params(<массив с именами существующих GET-параметров>[,
                <массив с новыми GET-параметрами>])
```

В *первом массиве* приводятся имена GET-параметров, которые нужно извлечь из текущего интернет-адреса. Во *втором*, ассоциативном, *массиве* записываются

вновь добавляемые GET-параметры и их значения. Ключ элемента *второго массива* задаст имя GET-параметра, а значение элемента — значение параметра.

Откроем модуль modules\helpers.php в текстовом редакторе и добавим объявление функции `get_GET_params`:

```
namespace Helpers {
    . . .
    function get_GET_params(array $existing_param_names,
        array $new_params = []): string {
        $a = [];
        foreach ($existing_param_names as $name)
            if (!empty($_GET[$name]))
                $a[] = $name . '='. urlencode($_GET[$name]);
        foreach ($new_params as $name => $value)
            $a[] = $name . '='. urlencode($value);
        $s = implode('&', $a);
        if ($s)
            $s = '?' . $s;
        return $s;
    }
}
```

В массив `$a` помещаем строковые элементы формата *<имя GET-параметра>=<значение параметра>*, причем каждое *значение* предварительно кодируем в вид, пригодный для вставки в интернет-адрес, «пропустив» его через функцию `urlencode` (мы разберемся с ней на *уроке 16*). Далее сводим все элементы этого массива в строку, разделив их амперсандами, для чего используем знакомую по *разд. 2.2.3* функцию `implode`. После чего добавляем в начало получившейся строки вопросительный знак.

3. Пагинатор мы реализуем в виде класса `Paginator`. Вот формат вызова его конструктора:

```
Paginator(<общее количество записей>[,
          <массив с именами существующих GET-параметров>])
```

Указанный вторым параметром *массив* мы передадим функции `Helpers\get_GET_params`, которая будет использована для формирования интернет-адресов отдельных частей пагинатора.

Класс `Paginator` получит функциональность итератора. При его переборе в качестве индекса будет возвращаться номер очередной части, а в качестве значения — интернет-адрес этой части.

Помимо этого, класс пагинатора сможет поддерживать следующие общедоступные свойства:

- `current_page` — номер текущей части;
- `page_count` — общее количество частей;
- `first_record_num` — номер первой извлекаемой из таблицы записи.

Создадим модуль modules\Paginator.php и запишем в него код класса пагинатора `Paginator` из листинга 15.4.

Листинг 15.4. Модуль modules\Paginator.php

```
<?php
require_once $base_path . 'modules\helpers.php';
class Paginator implements \Iterator {
    public $current_page = 1;
    public $page_count = 1;
    public $first_record_num = 1;

    private $existing_params;
    private $cur = 1;

    function __construct(int $record_count,
        array $existing_params = []) {
        $this->page_count = ceil($record_count /
            \Settings\RECORDS_ON_PAGE);
        $page_num = (isset($_GET['page'])) ? (int)$_GET['page'] : 1;
        if ($page_num < 1)
            $page_num = 1;
        if ($page_num > $this->page_count)
            $page_num = $this->page_count;
        $this->current_page = $page_num;
        $this->first_record_num = ($page_num - 1) *
            \Settings\RECORDS_ON_PAGE;
        $this->existing_params = $existing_params;
    }

    function current() {
        return \Helpers\get_GET_params($this->existing_params,
            ['page' => $this->cur]);
    }

    function key() {
        return $this->cur;
    }

    function next() {
        $this->cur++;
    }

    function rewind() {
        $this->cur = 1;
    }
```

```
    function valid() {
        return $this->cur >= 1 && $this->cur <= $this->page_count;
    }
}
```

В конструкторе класса выполняются все необходимые расчеты: получение из GET-параметра `page` номера текущей части, подсчет общего количества частей и вычисление первой извлекаемой записи.

4. Откроем модуль modules\controllers\images.php с кодом контроллера `Controllers\Images.php` и внесем изменения в код метода `by_cat`:

```
class Images extends BaseController {
    . . .
    function by_cat(string $slug) {
        . . .
        $picts = new \Models\Picture();
        $pict_count_rec = $picts->get_record('COUNT(*) AS cnt', NULL,
            'category = ?', [$cat['id']]);
        $paginator = new \Paginator($pict_count_rec['cnt']);
        $picts->select('pictures.id, title, filename, uploaded, ' .
            . . .
            ['users'], 'category = ?', [$cat['id']], '',
            $paginator->first_record_num, \Settings\RECORDS_ON_PAGE);
        $ctx = ['cat' => $cat, 'picts' => $picts,
            'paginator' => $paginator];
        $this->render('by_cat', $ctx);
    }
}
```

Вычисляем количество изображений, относящихся к выбранной категории, на его основе создаем объект класса `Paginator`, извлекаем из его свойства `first_record_num` номер первой записи, входящей в состав текущей части, и получаем из таблицы `pictures` изображения, входящие в состав этой части.

Объект-пагинатор мы передаем шаблону под именем `paginator`. Условимся и в дальнейшем передавать его под этим же именем.

5. Интерфейс пагинатора — набор гиперссылок, указывающий на его отдельные части, — мы будем формировать с применением шаблона-фрагмента modules\templates__paginator.inc.php. Этот шаблон-фрагмент далее используем при формировании других страниц.

Создадим модуль modules\templates__paginator.inc.php и запишем в него код шаблона-фрагмента, создающего пагинатор (листинг 15.5).

Листинг 15.5. Модуль modules\templates__paginator.inc.php

```
<?php if ($paginator->page_count > 1) { ?>
<div id="paginator">
```

```
    <?php foreach ($paginator as $number => $url) {
        if ($paginator->current_page == $number) { ?>
            <span><?php echo $number ?></span>
        <?php } else { ?>
            <a href="<?php echo $url ?>"><?php echo $number ?></a>
        <?php } ?>
    <?php } ?>
</div>
<?php } ?>
```

Пагинатор выводится только в том случае, если количество частей превышает одну. Текущая часть представляется тегом `<span>` — создавать указывающую на нее гиперссылку нет смысла.

6. В шаблоне страницы категории, в гиперссылках, указывающих на страницу просмотра изображения, помимо идентификатора страницы, с которой был выполнен переход, следует передавать и номер текущей части пагинатора. Это нужно для выполнения возврата на ту часть, с которой был выполнен переход на страницу изображения.

   Для формирования набора GET-параметров `ref` и `page` мы применим написанную ранее функцию `Helpers\get_GET_params`.

   Откроем модуль modules\templates\by_cat.php с шаблоном страницы категории и внесем следующие правки (здесь и далее удаленный код зачеркнут):

```
<?php require \Helpers\get_fragment_path('__header') ?>
<?php $gets = \Helpers\get_GET_params(['page'], ['ref' => 'cat']) ?>
. . .
<table id="gallery">
    . . .
    <tr>
        <td><a href="/<?php echo $pict['id'], $gets ?>/?ref=cat">
            . . .
        </a></td>
        <td><a href="/<?php echo $pict['id'], $gets ?>/?ref=cat">
            . . .
        </a></td>
        . . .
    </tr>
    <?php } ?>
</table>
<?php require \Helpers\get_fragment_path('__paginator') ?>
<?php require \Helpers\get_fragment_path('__footer') ?>
```

Не забываем включить шаблон-фрагмент, создающий интерфейс пагинатора.

Проверим, как работают исправленные страницы категорий (рис. 15.2).

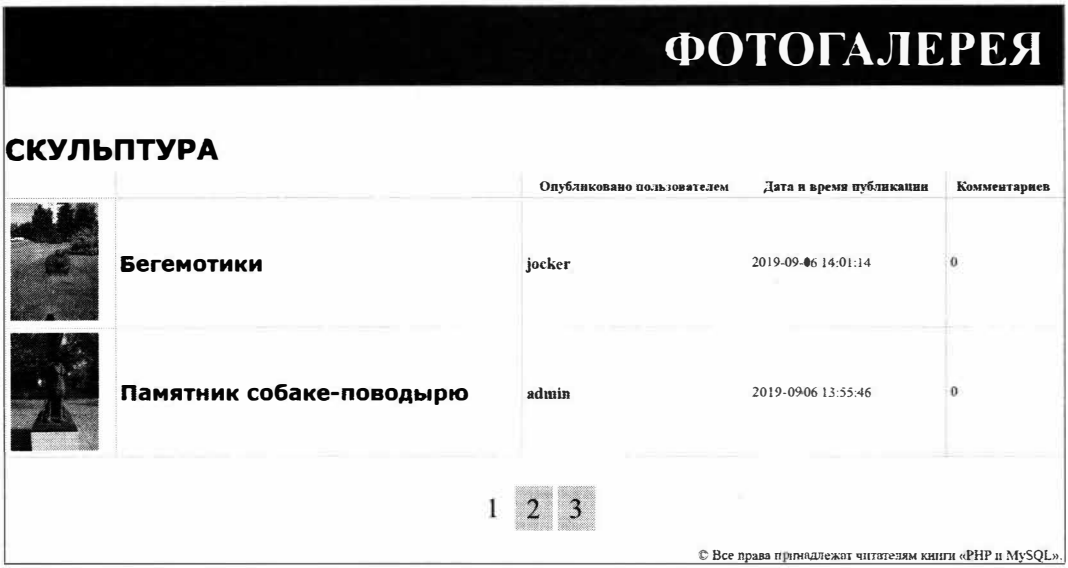


Рис. 15.2. Веб-страница категории с поддержкой пагинации

15.9. Упражнение. Реализуем фильтрацию изображений

Реализуем фильтрацию изображений по введенной посетителем подстроке. Пусть выводятся только изображения, в названии или описании которых она встретилась.

Подстрока будет передаваться через GET-параметр `filter`. Веб-форму для ее ввода вынесем в отдельный шаблон-фрагмент, чтобы ее потом можно было использовать и на странице пользователя.

Файлы сайта фотогалереи хранятся в папке 15\15.8\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Создадим модуль modules\templates__filter_form.inc.php — шаблон-фрагмент формы фильтрации, в который запишем код из листинга 15.6.

Листинг 15.6. Модуль modules\templates__filter_form.inc.php

```
<?php
    if (isset($_GET['filter']) && !empty($_GET['filter']))
        $f = $_GET['filter'];
    else
        $f = '';
?>
<form id="filter_form" method="get">
    <input type="text" name="filter" placeholder="Фильтрация"
    value="<?php echo $f ?>">
    <input type="submit" value="Вперед">
</form>
```

Если GET-параметр `filter` присутствует в интернет-адресе, извлекаем из него подстроку фильтрации и подставляем в поле ввода формы. Так посетитель сразу поймет, что выполняется фильтрация изображений.

2. Откроем модуль modules\controllers\images.php с контроллером `Controllers\Images` и исправим метод `by_cat` (здесь и далее дополнения и правки в ранее написанном коде выделены полужирным шрифтом, удаленный код зачеркнут):

```
class Images extends BaseController {
    . . .
    function by_cat(string $slug) {
        $cats = new \Models\Category();
        $cat = $cats->get_or_404($slug, 'slug', 'id, name');
        $w = 'category = ?';
        $p = [$cat['id']];
        if (isset($_GET['filter']) && !empty($_GET['filter'])) {
            $w .= ' AND (title LIKE ? OR description LIKE ?)';
            $f = '%' . $_GET['filter'] . '%';
            $p[] = $f;
            $p[] = $f;
        }
        $picts = new \Models\Picture();
        $pict_count_rec = $picts->get_record('COUNT(*) AS cnt', NULL,
            $w~~'category = ?'~~, $p~~[$cat['id']]~~);
        $paginator = new \Paginator($pict_count_rec['cnt'],
            ['filter']);
        $picts->select('pictures.id, title, filename, uploaded, ' .
            . . .
            ['users'], $w~~'category = ?'~~, $p~~[$cat['id']]~~, '',
            $paginator->first_record_num, \Settings\RECORDS_ON_PAGE);
        . . .
    }
}
```

3. Откроем модуль modules\templates\by_cat.php с шаблоном страницы категории и исправим его код следующим образом:

```
<?php require \Helpers\get_fragment_path('__header') ?>
<?php require \Helpers\get_fragment_path('__filter_form') ?>
<?php $gets = \Helpers\get_GET_params(['page', 'filter'],
    ['ref' => 'cat']) ?>
. . .
```

Мы включили шаблон-фрагмент формы фильтрации и добавили в список GET-параметров, передаваемых странице просмотра изображения, параметр `filter`.

Осталось проверить фильтрацию в действии. Перейдем на страницу какой-либо категории, занесем какое-либо слово в поле ввода, нажмем кнопку **Вперед** и посмотрим, что получилось.

Сделайте сами

- Создайте страницу для вывода изображений, опубликованных выбранным пользователем (страницу пользователя). Для этого:
  - объявите класс модели `Models\User`, работающей с таблицей `users` и по умолчанию сортирующей список пользователей по полю `name`;
  - добавьте в модуль маршрутизатора код, при поступлении интернет-адреса формата **/users/<*имя пользователя*>/** вызывающий метод `by_user` контроллера `Controllers\Images` и передающий ему в качестве параметра ***имя пользователя***, извлеченное из интернет-адреса;
  - добавьте в контроллер `Controllers\Images` метод `by_user`, выводящий страницу пользователя, взяв за образец метод `by_cat`;
  - создайте шаблон страницу пользователя by_user.php, взяв за образец шаблон modules\templates\by_cat.php. Вместо имен пользователей, опубликовавших изображения, выводите на ней названия категорий.
- Реализуйте вывод названий (название страницы задается в теге `<title>`) страниц категории и пользователя в форматах:

  <название категории> :: Категории :: <название веб-сайта>
  <имя пользователя> :: Пользователи :: <название веб-сайта>

  Подсказка: вам нужно лишь поместить текст названия в элемент `site_title` контекста шаблона.
- Перепишите страницу для просмотра выбранного изображения, чтобы она работала с базой данных. Выведите на ней название изображения, само изображение, описание к нему, название категории, имя пользователя, опубликовавшего изображение, дату и время публикации и гиперссылку **Назад**.

  Эта гиперссылка должна возвращать посетителя на страницу, с которой он выполнил переход: главную, страницу категории или пользователя, причем в последних двух случаях — на исходную часть и с сохранением исходной подстроки фильтрации. Подсказка: определить страницу, с которой был выполнен переход, можно по значению GET-параметра `ref`, а в интернет-адрес гиперссылки **Назад** следует включить GET-параметры `page` и `filter`.
- Реализуйте вывод комментариев на странице просмотра изображения. Объявите класс модели `Models\Comment`, добавьте необходимый код в метод `item` контроллера `Controllers\Images` и в шаблон страницы изображения. Поместите комментарии между сведениями об изображении и гиперссылкой **Назад**. Для каждого комментария выведите имя оставившего его пользователя, содержание и временну́ю отметку публикации.
- Удалите модуль modules\models\image.php, хранящий код написанной на предыдущих уроках модели `Models\Image`, поскольку она более не нужна.

ЧАСТЬ III

Практическое PHP-программирование

- *Урок 16*. Обработка клиентских запросов, генерирование ответов и включение модулей
- *Урок 17*. Обработка данных, введенных в веб-формы
- *Урок 18*. Работа с файлами и папками
- *Урок 19*. Работа с графикой
- *Урок 20*. Cookie и сессии
- *Урок 21*. Базовые средства безопасности. Разграничение доступа
- *Урок 22*. Отправка электронной почты
- *Урок 23*. Усиленные меры безопасности
- *Урок 24*. Веб-службы REST
- *Урок 25*. Настройки PHP

Урок 16
Обработка клиентских запросов, генерирование ответов и включение модулей

16.1. Средства для обработки клиентских запросов

16.1.1. Получение данных, отправленных из веб-форм

Значения GET- и POST-параметров, отправленных из веб-формы, извлекаются из ассоциативных массивов, хранящихся в следующих суперглобальных переменных:

- `$_GET` — значения GET-параметров;
- `$_POST` — значения POST-параметров;
- `$_FILES` — файлы, отправленные из веб-формы (подробно о работе с файлами будет рассказано на *уроке 18*);
- `$_REQUEST` — значения и GET-, и POST-параметров (а также данные cookie, о которых мы узнаем на *уроке 20*).

Ключ элемента такого массива совпадает с наименованием соответствующего элемента управления (которое задается атрибутом тега `name`).

16.1.2. Получение сведений о клиентском запросе и веб-сервере

Различные сведения о полученном клиентском запросе, равно как и о веб-сервере, под которым работает сайт, можно получить из ассоциативного массива, хранящегося в суперглобальной переменной `$_SERVER`. В табл. 16.1 представлены ключи элементов этого массива, хранящие наиболее полезные данные.

***Таблица 16.1.** Ключи наиболее полезных элементов ассоциативного массива `$_SERVER`*

Ключ элемента	Значение элемента
`REQUEST_URI`	Путь, извлеченный из интернет-адреса запроса, — например, при открытии страницы категории «Пейзаж» нашей фотогалереи: `'/cats/landscape/'`

***Таблица 16.1** (окончание)*

Ключ элемента	Значение элемента
REQUEST_METHOD	Наименование HTTP-метода, посредством которого был выполнен клиентский запрос, в верхнем регистре. Например: 'GET' или 'POST'
REMOTE_ADDR	IP-адрес клиента, выполнившего запрос
HTTP_REFERER	Интернет-адрес страницы, с которой был выполнен переход на текущую страницу. Должен устанавливаться веб-обозревателем, но не все из них делают это достаточно аккуратно, и доверять этому значению не стоит
HTTPS	Присутствует в массиве, если запрос был выполнен по протоколу HTTPS. Если запрос выполнялся по протоколу HTTP, отсутствует
SERVER_ADDR	IP-адрес компьютера, на котором работает веб-сервер
SERVER_NAME	Доменное имя компьютера, на котором работает веб-сервер
SERVER_PORT	Номер TCP-порта, используемого веб-сервером: 80 — при работе по протоколу HTTP, 433 — по протоколу HTTPS

16.2. Вывод данных

16.2.1. Простой вывод данных

Для простого, без всякого форматирования, вывода указанных *значений* лучше использовать языковую конструкцию echo:

```
echo <список выводимых значений через запятую>
```

Выводимые *значения* могут быть любого типа. *Значения* выводятся вплотную друг к другу, без каких бы то ни было разделителей между ними.

Конструкция print может вывести всего одно *значение* любого типа, зато работает чуть быстрее:

```
print <одно выводимое значение>
```

> Конструкции echo и print перед выводом преобразуют все значения, не являющиеся строками, в строки.

Это может вызвать проблемы при выводе таких величин, как TRUE, FALSE и NULL: TRUE будет преобразовано в строку '1', а FALSE и NULL — в пустые строки, которые вообще не будут выведены на экран:

```
echo TRUE;                    // Реально будет выведено: 1
echo FALSE;                   // Реально ничего не будет выведено
print NULL;                   // Реально ничего не будет выведено
```

Примечание

В примерах, приведенных в книге, величины TRUE, FALSE и NULL, если они должны быть результатами выполнения примеров, для наглядности показаны как есть.

16.2.2. Форматированный вывод данных

Форматированный вывод строк и чисел

Для форматированного вывода данных применяется функция `printf`:

```
printf(<строка, задающая формат>, <значение 1>, <значение 2>, . . .,
       <значение n-1>, <значение n>)
```

В *строке, задающей формат*, можно использовать как обычные символы, которые будут выведены как есть, так и особые литералы, обозначающие местоположение и формат выводимых *значений*. Первый встретившийся в *строке* литерал соответствует выводимому *значению 1*, второй — *значению 2*, и т. д.

Литералы функции `printf` записываются в формате:

```
%[<флаги>][<ширина вывода>][.[<точность>]]<обозначение формата>
```

Обозначение формата указывает, в каком формате будет выведено значение:

- как строка — обозначение формата `s`:

```
printf('<h1>%s</h1>', 'Фотогалерея');          // <h1>Фотогалерея</h1>
printf('Функция <strong>%s</strong>: %s', 'strlen',
    'получение длины строки');
            // Функция <strong>strlen</strong>: получение длины строки
```

- как целое число со знаком — `d`:

```
printf('Всего %d картинок', 11659);            // Всего 11659 картинок
printf('И %d комментариев', 18000000);      // И 18000000 комментариев
```

- как вещественное число — `f` или `F`. По умолчанию после десятичной точки выводится 6 цифр (их количество можно изменить, указав точность, — о чем см. далее в этом разделе);

```
printf('Средний радиус Луны: %F км.', 1737.1);
                              // Средний радиус Луны: 1737.100000 км.
```

- как целое или вещественное число в экспоненциальной нотации *<мантисса>*e*<порядок>* — `e`:

```
printf('Всего %e картинок', 11659);      // Всего 1.165900e+4 картинок
printf('Средний радиус Луны: %e км.', 1737.1);
                              // Средний радиус Луны: 1.737100e+3 км.
```

- то же самое, но с буквой «E» в верхнем регистре — `E`:

```
printf('Всего %E картинок', 11659);      // Всего 1.165900E+4 картинок
```

- как целое или вещественное число в наиболее компактной нотации — `g`. Выбирается та нотация, при которой количество значащих цифр в выведенном значении не превышает 6 (значение по умолчанию, которое можно изменить, указав точность, — о чем см. далее в этом разделе);

```
printf('Всего %g картинок', 11659);            // Всего 11659 картинок
printf('И %g комментариев', 18000000);       // И 1.8e+7 комментариев
```

- то же самое, но с буквой «E» в верхнем регистре при выводе в экспоненциальной нотации — G:

```
printf('И %G комментариев', 18000000);          // И 1.8E+7 комментариев
```

- как целое число в шестнадцатеричной системе счисления — x:

```
printf('%d шестнадцатеричное: %x', 234, 234);  // 234 шестнадцатеричное: ea
```

- то же самое, только с буквами в верхнем регистре — X:

```
printf('%d шестнадцатеричное: %X', 234, 234); // 234 шестнадцатеричное: EA
```

- как целое число в восьмеричной системе счисления — o:

```
printf('%d восьмеричное: %o', 234, 234);       // 234 восьмеричное: 352
```

- как целое число в двоичной системе счисления — b:

```
printf('%d двоичное: %b', 234, 234);          // 234 двоичное: 11101010
```

- как символ с ASCII-кодом, совпадающим со значением, — c:

```
printf('Символ с кодом %d: %c', 86, 86);       // Символ с кодом 86: V
```

  К сожалению, вывод символов по Unicode-кодам не поддерживается;

- вывод знака процента — %:

```
printf('Заполнено %d%% хранилища', 47);      // Заполнено 47% хранилища
```

Ширина вывода определяет минимальное количество символов, которое будет напечатано. Если она больше длины выводимого значения, последнее при выводе по умолчанию будет выровнено по правому краю (выравнивание можно изменить, указав флаг, — о чем см. далее в этом разделе):

```
printf('|%s|', 'PHP и MySQL');                  // |PHP и MySQL|
printf('|%15s|', 'PHP и MySQL');                // |    PHP и MySQL|
```

Если указанная *ширина вывода* меньше реальной длины значения, она будет проигнорирована, и значение выведено полностью:

```
printf('|%5s|', 'PHP и MySQL');                    // |PHP и MySQL|
```

Величина *точности* имеет разный смысл у разных типов выводимых значений:

- у типов e, E, f и F — количество цифр после десятичной точки:

```
printf('Всего %.3e картинок', 11659);        // Всего 1.166e+4 картинок
printf('Средний радиус Луны: %.2F км.', 1737.1);
                                  // Средний радиус Луны: 1737.10 км.
```

- у типов g и G — максимальное количество печатаемых значащих цифр:

```
printf('Всего %g картинок', 11659);            // Всего 11659 картинок
printf('Всего %.3g картинок', 11659);         // Всего 1.17e+4 картинок
printf('Всего %.1g картинок', 11659);          // Всего 1.0e+4 картинок
```

- у типа s — номер символа в выводимом значении, который, вместе с последующими, не будет выведен на экран. Нумерация символов начинается с 1:

```
printf('|%15s|', 'PHP и MySQL');          // |   PHP и MySQL|
printf('|%15.8s|', 'PHP и MySQL');        // |       PHP и M|
```

Флаги задают дополнительные параметры вывода:

- выравнивание значения по левому краю — флаг - (минус):

```
printf('|%15s|', 'PHP и MySQL');          // |   PHP и MySQL|
printf('|%-15s|', 'PHP и MySQL');         // |PHP и MySQL   |
```

- печать знаков «+» перед положительными числами — + (плюс):

```
printf('%d    %d', 45, -93);              // 45    -93
printf('%+d    %+d', 45, -93);            // +45    -93
```

- дополнение чисел слева нулями — 0:

```
printf('%10.3f', 25.90764);              //     25.908
printf('%010.3f', 25.90764);             // 000025.908
```

Функция vprintf выполняет то же самое действие, что и printf, но принимает все выводимые значения в виде *массива*:

```
vprintf(<строка, задающая формат>, <массив со значениями>)
```

Пример:

```
vprintf('Функция <strong>%s</strong>: %s',
    ['strlen', 'получение длины строки']);
```

Две функции не выводят отформатированную строку, а возвращают ее в качестве результата:

- sprint — аналогична printf;
- vsprintf — аналогична vprintf.

Форматированный вывод временны́х отметок

Функция strftime форматирует указанную *временну́ю отметку* (или текущие дату и время, если *временна́я отметка* не задана) и возвращает ее в виде строки:

```
strftime(<строка, задающая формат>[, <временна́я отметка>])
```

В *строке, задающей формат*, можно использовать специализированные литералы, представленные в табл. 16.2.

***Таблица 16.2.** Литералы, поддерживаемые функцией* `strftime`

Литерал	Описание
%d	Число от 01 до 31
%m	Номер месяца от 01 (январь) до 12 (декабрь)
%Y	Год из четырех цифр
%u	Порядковый номер дня недели от 1 (понедельник) до 7 (воскресенье)
%w	Порядковый номер дня недели от 0 (воскресенье) до 6 (суббота)

Таблица 16.2 (окончание)

Литерал	Описание
%H	Часы в 24-часовом формате от 00 до 23
%I	Часы в 12-часовом формате от 00 до 12
%M	Минуты от 00 до 59
%S	Секунды от 00 до 59
%p	'AM' или 'PM', в зависимости от указанного времени
%P	'am' или 'pm', в зависимости от указанного времени
%%	Символ «%»

Примеры:

```
echo strftime('%d.%m.%Y %H:%M:%S');                    // 20.09.2019 15:03:42
$t = strtotime('2019-09-06 14:18:39');
echo strftime('%I:%M:%S %p', $t);                      // 02:18:39 PM
```

16.2.3. Преобразование текста к виду, пригодному для вставки в HTML-код

- Преобразование знаков «<», «>», одинарных, двойных кавычек и амперсандов, присутствующих в заданной *строке*, в соответствующие им HTML-литералы — функция `htmlspecialchars(<строка>)`:

```
echo htmlspecialchars('Контора "Кук и сыновья"');
                                  // Контора &quot;Кук и сыновья&quot;
echo htmlspecialchars('Тег <p> создает абзац');
                                  // Тег &lt;p&gt; создает абзац
```

- вставка перед каждым переводом строки, присутствующим в заданной *строке*, HTML-тега `<br>` — функция `nl2br(<строка>)`:

```
echo nl2br("PHP\r\nMySQL");                            // PHP<br />
                                                       // MySQL
```

- удаление из *строки* HTML- и PHP-тегов — функция `strip_tags`:

```
strip_tags(<строка>[, <набор тегов, которые следует оставить>])
```

Набор тегов, которые следует оставить в строке, указывается в виде строки с тегами, записанными вплотную друг к другу. Если *набор тегов* не указан, будут удалены все теги.

Примеры:

```
echo strip_tags('<p><em>PHP</em> и <strong>MySQL</strong></p>');
                                                       // PHP и MySQL
echo strip_tags('<p><em>PHP</em> и <strong>MySQL</strong></p>',
    '<em><p>');                          // <p><em>PHP</em> и MySQL</p>
```

- кодирование *строки* в формат application/x-www-form-urlencoded, к виду, пригодному для помещения в состав интернет-адреса, — функция urlencode(<*строка*>):

```
$filter_phrase = 'кот';
$url = 'http://www.photogallery.ry/users/jocker/?filter=';
$url .= urlencode($filter_phrase);
echo $url;
 // http://www.photogallery.ru/users/jocker/?filter=%D0%BA%D0%BE%D1%82
```

16.3. Упражнение. Форматируем временны́е отметки

Сделаем так, чтобы временны́е отметки публикации изображений выводились в привычном формате: <*число*>.<*месяц*>.<*год*> <*часы*>:<*минуты*>:<*секунды*>.

Файлы сайта фотогалереи хранятся в папке 15\15.9\s, а файл photogallery.sql с дампом базы данных photogallery — в папке 13\13.4\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Сначала объявим функцию Helpers\get_formatted_timestamp, форматирующую *временну́ю отметку*, указанную в виде строки (именно в таком виде она извлекается из базы данных MySQL), и возвращающую ее в строковом виде:

   ```
   get_formatted_timestamp(<временна́я отметка>)
   ```

   Откроем модуль modules\helpers.php и добавим код, объявляющий эту функцию (здесь и далее добавления и правки в написанном ранее коде выделены полужирным шрифтом):

   ```
   namespace Helpers {
       . . .
       function get_formatted_timestamp(string $timestamp): string {
           return strftime('%d.%m.%Y %H:%M:%S', strtotime($timestamp));
       }
   }
   ```

2. Откроем шаблон главной страницы modules\templates\list.php и внесем следующие правки:

   ```
   . . .
   <table id="gallery">
       . . .
           <td><?php echo \Helpers\get_formatted_timestamp↵
           ($pict['uploaded']) ?></td>
       . . .
   </table>
   ```

   Перед выводом временно́й отметки публикации мы «прогнали» ее через функцию get_formatted_timestamp.

Проверим исправленную главную страницу в действии.

Сделайте сами

Соответственно исправьте остальные шаблоны, входящие в состав сайта.

16.4. Задание заголовков и состояния ответа

> *Заголовок ответа* — содержит различные характеристики веб-страницы (например, кодировку или временну́ю отметку последнего изменения) и указания веб-обозревателю: не кэшировать страницу, выполнить перенаправление на другой интернет-адрес и др. Содержит также набор параметров и их значений и находится в самом начале ответа.

> *Состояние ответа* — обозначает результат выполнения веб-сервером какой-либо операции: успешной отправки файла, отсутствия запрошенного файла, ошибки в серверном приложении и др. или действие, которое следует предпринять веб-обозревателю, — например, перенаправление на интернет-адрес, указанный в заголовке. Обозначается целочисленным кодом.

Часть параметров заголовков и состояние ответа задаются веб-сервером и платформой PHP. Мы можем добавить к заголовку свои параметры, изменить значения уже присутствующих, равно как и изменить состояние.

Единичный параметр, добавляемый в заголовок ответа, задается вызовом функции `header`:

```
header(<параметр и значение>[, <заменить существующий заголовок?>[,
       <код состояния>]])
```

Параметр и значение указываются в виде строки формата *<имя параметра>*: *<значение>*.

Если вторым параметром передать значение `TRUE` или вообще не указывать его, заданное *значение* параметра заменит значение, уже присутствующее в заголовке. Если передать `FALSE`, заданные *параметр и значения* будут добавлены к заголовку, даже если в нем уже присутствует тот же параметр с другим значением (что может понадобиться в специфических случаях).

Код состояния указывается в виде целого числа.

> Задавать заголовки следует до выполнения какого-либо вывода, формирующего HTML-код страницы (в противном случае мы получим ошибку).

Пример указания параметра, запрещающего кэшировать страницу:

```
header('Cache-Control: no-cache');
```

Для указания состояния ответа применяется функция `http_response_code`:

```
http_response_code(<код состояния>)
```

Пример указания кода состояния 404 (запрошенная страница не найдена):

```
http_response_code(404);
```

Получить текущий код состояния можно, вызвав функцию `http_response_code` без параметров.

> ***Где найти параметры заголовка и коды состояния ответа?***
>
> Список параметров заголовка можно найти в статье «Список заголовков HTTP» русской «Википедии». Форматы, в которых задаются их параметры, описаны там же в статье «Заголовки HTTP», а коды состояния — в статье «Список кодов состояния HTTP».

16.4.1. Перенаправление

> *Перенаправление* — принудительный переход по другому интернет-адресу, выполненный программно. Выполняется отправкой особого параметра в заголовке и указанием одного из состояний, обозначающих перенаправление.

Для указания перенаправления служат два кода состояния:

- 302 — временное перенаправление. Используется, если нужно перенаправить посетителя на другую страницу сайта в процессе работы с ним;
- 301 — постоянное перенаправление. Получив ответ с таком состоянием, веб-обозреватель изменяет старый интернет-адрес, записанный в истории посещения и в избранном, на новый, полученный в запросе. Применяется в случае, когда сайт был перенесен с одного интернет-адреса на другой.

Интернет-адрес перенаправления пересылается в составе заголовка ответа, в параметре с именем `Location`.

Пример временного перенаправления на другую страницу того же сайта:

```
header('Location: /users/admin', TRUE, 302);
```

Пример постоянного перенаправления:

```
header('Location: https://www.photogallery.ru/', TRUE, 301);
```

> Поскольку при перенаправлении изменяется заголовок ответа, его следует выполнять перед любым выводом. Более того, выполнять какой бы то ни было вывод при перенаправлении необязательно, поскольку веб-обозреватель все равно не отобразит его на экране.

16.5. Упражнение. Задаем состояния ответа у веб-страниц сообщений об ошибках 404 и 503

Желательно указать у страниц сообщений об ошибках 404 и 503 соответствующие состояния ответа, чтобы дать веб-обозревателю понять, что произошла ошибка.

Файлы сайта фотогалереи хранятся в папке 16\16.3\s сопровождающего книгу электронного архива (см. *приложение 3*).

1. Откроем шаблон modules\templates\404.php, хранящий шаблон страницы ошибки 404, и добавим в начало кода следующее выражение (выделено полужирным шрифтом):

```
<?php http_response_code(404) ?>
<?php require \Helpers\get_fragment_path('__header') ?>
. . .
```

2. Выполним в веб-обозревателе переход по заведомо не обрабатываемому интернет-адресу, например: **http://localhost/1234567/**, и проверим, выводится ли страница с сообщением об ошибке 404.

Сделайте сами

Соответственно исправьте шаблон страницы с сообщением об ошибке 503. Укажите у нее код состояния 503.

16.6. Включение модулей

При включении одного программного модуля в другой все переменные, константы, функции и классы, созданные во включаемом модуле, становятся доступными во включающем.

Для выполнения включения PHP предоставляет четыре языковые конструкции, имеющие одинаковый формат написания:

<языковая конструкция> <путь к включаемому модулю>

Путь к включаемому модулю указывается в виде строки.

Эти конструкции имеют следующие особенности:

- `require` — если *включаемый модуль* отсутствует, выдает фатальную ошибку, что вызывает аварийный останов приложения;
- `require_once` — то же самое, что `require`, но включает каждый модуль только единожды, повторные включения игнорируются;

> ***Конструкции `require` и `require_once` обрабатываются в момент компиляции***
>
> Поэтому, если включаемый модуль отсутствует, будет выдана фатальная ошибка компиляции, не приводящая к возникновению исключения.

- `include` — если *включаемый модуль* отсутствует, выдает предупреждение, выполнение приложения не прерывается;
- `include_once` — то же самое, что `include`, но включает каждый модуль только единожды, повторные включения игнорируются.

На практике автор рекомендует для включения модулей пользоваться конструкциями `require` и `require_once`. В случае отсутствия включаемого модуля они тут же выдадут фатальную ошибку, что позволит сразу же найти и устранить проблему.

Конструкцией `require_once` рекомендуется включать модули с объявлениями констант, функций и классов. Таким образом будет исключено повторное объявление константы, функции или класса, что в PHP приводит к фатальной ошибке.

Урок 17
Обработка данных, введенных в веб-формы

Получить данные, введенные посетителем в веб-форму и отправленные методом GET, можно из ассоциативного массива $_GET, данные, отправленные методом POST, — из ассоциативного массива $_POST. Об этом говорилось на *уроке 16*.

17.1. Элементы управления HTML и отправляемые ими данные

> Каждый элемент управления, присутствующий в веб-форме, отправляет серверному приложению GET- или POST-параметр, имя которого совпадает с наименованием элемента управления (задается атрибутом тега name).

Например, поле ввода:

```
<input type="text" name="username">
```

отправит серверному приложению GET- или POST-параметр username, значением которого является строка, введенная в это поле.

Далее приведены значения, отправляемые элементами управления разных типов:

- обычное поле ввода, поля ввода пароля, адреса электронной почты, интернет-адреса и область редактирования — отправляется набранная строка:

```
$user_name = $_POST['username'];
```

- поле ввода числа и регулятор — занесенное число в виде строки, которое следует преобразовать в целочисленный тип:

```
<input type="number" name="age">
. . .
$age = (int)$_POST['age'];
```

- поле ввода даты — указанная дата в виде строки формата: *<год>-<номер месяца>-<число>*, которую можно преобразовать во временну́ю отметку PHP вызовом функции strtotime;

```
<input type="date" name="uploaded_date">
. . .
$uploaded = strtotime($_POST['uploaded_date']);
```

- поле ввода времени — указанное время в виде строки формата: *<часы>*:*<минуты>*:*<секунды>*, которое можно добавить к дате, полученной из поля ввода даты:

```
<input type="time" name="uploaded_time">
. . .
$timestamp = $_POST['uploaded_date'] . ' ' . $_POST['uploaded_time'];
$uploaded = strtotime($timestamp);
```

- поле выбора цвета — выбранный цвет в виде строки формата: #*<R><G><B>*, где *R* — доля красного цвета, *G* — доля зеленого, *B* — доля синего (все доли указаны в виде шестнадцатеричных чисел);
- флажок — если установлен, будет передано значение, записанное в атрибуте тега `value`, если сброшен, ничего передано не будет:

```
<input type="checkbox" name="agree" value="yes">
. . .
if (empty($_POST['agree']))
    // Флажок сброшен
else
    // Флажок установлен
```

- набор переключателей — значение, записанное в атрибуте тега `value` у установленного переключателя:

```
<input type="radio" name="gender" value="male">
<input type="radio" name="gender" value="female">
. . .
switch($_POST['gender']) {
    case 'male':
        // Установлен первый переключатель
        break;
    case 'female':
        // Установлен второй переключатель
}
```

- список:
  - позволяющий выбирать лишь один пункт — значение, записанное в атрибуте тега `value` у выбранного пункта, или, если такого атрибута тега нет, текст выбранного пункта:

```
<select name="gender">
    <option value="male">Муж.</option>
    <option>Жен.</option>
</select>
. . .
switch($_POST['gender']) {
    case 'male':
        // Выбран первый пункт списка
        break;
```

```
    case 'Жен.':
        // Выбран второй пункт списка
}
```

- позволяющий выбрать произвольное количество пунктов — набор значений, представляющих все выбранные пункты.

  Если завершить наименование, записанное в атрибуте тега `name`, пустыми квадратными скобками, PHP получит *массив* значений, представляющих выбранные пункты:

```
<select name="categories[]" size="4" multiple>
    <option value="plants">Растения</option>
    <option value="architecture">Архитектура</option>
    <option value="landscape">Пейзаж</option>
    <option value="sculpture">Скульптура</option>
</select>
. . .
foreach ($_POST['categories'] as $category) {
    // Обрабатываем каждый из выбранных в списке пунктов
}
```

17.2. Валидация и нормализация данных

17.2.1. Валидация данных

> *Валидация* — проверка полученных из веб-формы данных на корректность (например, если требуется целое число — проверка, является ли занесенное в форму значение целым числом).

Валидация в PHP производится с применением *фильтров*, каждый из которых проверяет заданное значение на соответствие одному из поддерживаемых типов.

Для валидации заданного *значения* посредством фильтра с указанным *обозначением* служит функция `filter_var`:

```
filter_var(<значение>[, <обозначение фильтра>[, <массив с настройками>]])
```

Ассоциативный *массив с настройками* может содержать два элемента: `options`, хранящий ассоциативный массив с дополнительными параметрами, и `flags`, содержащий флаги.

> *Флаг* — дополнительный параметр, наличие которого включает какой-либо режим проверки. Представляет собой целое число. Флаги могут быть объединены оператором побитового ИЛИ | (см. *разд. 2.1.2*).

Функция `filter_var` возвращает само заданное *значение*, если валидация прошла успешно, и `FALSE` — в противном случае.

Далее приведены поддерживаемые функцией типы значений, соответствующие им константы-*обозначения фильтров*, флаги и дополнительные параметры, поддерживаемые разными фильтрами:

- если *обозначение типа* не указано — используется обозначение FILTER_DEFAULT, указывающее фактическое отсутствие валидации. В большинстве случаев бесполезно, но может пригодиться при валидации массивов (см. в конце этого раздела);
- целочисленный — FILTER_VALIDATE_INT. Флаг FILTER_FLAG_ALLOW_OCTAL разрешает задавать числа в восьмеричной системе счисления, а флаг FILTER_FLAG_ALLOW_HEX — в шестнадцатеричной. Если указаны дополнительные параметры min_range и max_range, также проводится проверка, попадает ли число в диапазон от min_range до max_range включительно. Примеры:

```
$val = '123';
echo filter_var($val, FILTER_VALIDATE_INT);        // 123
$val = 'cba123';
echo filter_var($val, FILTER_VALIDATE_INT);        // FALSE
$val = '0xfab';
echo filter_var($val, FILTER_VALIDATE_INT,
    ['flags' => FILTER_FLAG_ALLOW_OCTAL | FILTER_FLAG_ALLOW_HEX]);   // 4011
$val = '123';
echo filter_var($val, FILTER_VALIDATE_INT, ['options' =>
    ['min_range' => 10, 'max_range' => 100]]);      // FALSE
```

- вещественный — FILTER_VALIDATE_FLOAT. Если указан флаг FILTER_FLAG_ALLOW_THOUSAND, в числе допускаются запятые в качестве разделителей тысяч. Примеры:

```
$val = '123.456';
echo filter_var($val, FILTER_VALIDATE_FLOAT);      // 123.456
$val = '123%456';
echo filter_var($val, FILTER_VALIDATE_FLOAT);      // FALSE
$val = '1,234,567.89';
echo filter_var($val, FILTER_VALIDATE_FLOAT);      // FALSE
echo filter_var($val, FILTER_VALIDATE_FLOAT,
    ['flags' => FILTER_FLAG_ALLOW_THOUSAND]);      // 1234567.89
```

- интернет-адрес в виде строки — FILTER_VALIDATE_URL. Поддерживаются флаги:
  - FILTER_FLAG_SCHEME_REQUIRED — требуется указание протокола (http:// или https://);
  - FILTER_FLAG_HOST_REQUIRED — требуется указание интернет-адреса хоста;
  - FILTER_FLAG_PATH_REQUIRED — требуется указание пути;
  - FILTER_FLAG_QUERY_REQUIRED — требуется указание набора GET-параметров.

  Примеры:

```
$val = 'http://www.photogallery.ru';
echo filter_var($val, FILTER_VALIDATE_URL);
                                        // http://www.photogallery.ru
echo filter_var($val, FILTER_VALIDATE_URL,
    ['flags' => FILTER_FLAG_PATH_REQUIRED]);       // FALSE
```

- адрес электронной почты в виде строки — FILTER_VALIDATE_EMAIL. Если указан флаг FILTER_FLAG_EMAIL_UNICODE, в имени почтового ящика будут допускаться символы кириллицы. Примеры:

```
$val = 'admin@photogallery.ru';
echo filter_var($val, FILTER_VALIDATE_EMAIL); // admin@photogallery.ru
$val = 'admin[photogallery]ru';
echo filter_var($val, FILTER_VALIDATE_EMAIL);      // FALSE
$val = 'адмнн@photogallery.ru';
echo filter_var($val, FILTER_VALIDATE_EMAIL);      // FALSE
echo filter_var($val, FILTER_VALIDATE_EMAIL,
    ['flags' => FILTER_FLAG_EMAIL_UNICODE]);  // адмнн@photogallery.ru
```

- доменное имя в виде строки — FILTER_VALIDATE_DOMAIN. Удостоверяется, что длина каждой части доменного имени, выраженная в символах, больше нуля. Если указан флаг FILTER_FLAG_HOSTNAME, также будет проверяться корректность написания доменного имени. Примеры:

```
$val = 'www.photogallery.ru';
echo filter_var($val, FILTER_VALIDATE_DOMAIN);  // www.photogallery.ru
$val = '.photogallery.';
echo filter_var($val, FILTER_VALIDATE_DOMAIN);     // FALSE
$val = 'www+photogallery:ru';
echo filter_var($val, FILTER_VALIDATE_DOMAIN);  // www+photogallery:ru
echo filter_var($val, FILTER_VALIDATE_DOMAIN,
    ['flags' => FILTER_FLAG_HOSTNAME]);            // FALSE
```

- IP-адрес в виде строки — FILTER_VALIDATE_IP. Поддерживаются следующие флаги:
  - FILTER_FLAG_IPV4 — IP-адрес должен соответствовать стандарту IPv4, но не IPv6;
  - FILTER_FLAG_IPV6 — IP-адрес должен соответствовать стандарту IPv6, но не IPv4;
  - FILTER_FLAG_NO_PRIV_RANGE — IP-адрес не должен входить в частные диапазоны;
  - FILTER_FLAG_NO_RES_RANGE — IP-адрес не должен входить в зарезервированные диапазоны.

  Примеры:

```
$val = '127.0.0.1';
echo filter_var($val, FILTER_VALIDATE_IP);         // 127.0.0.1
$val = '555.a.b-d';
echo filter_var($val, FILTER_VALIDATE_IP);         // FALSE
$val = '::1';
echo filter_var($val, FILTER_VALIDATE_IP);         // ::1
echo filter_var($val, FILTER_VALIDATE_IP,
    ['flags' => FILTER_FLAG_IPV4]);                // FALSE
```

- строка, совпадающая с регулярным выражением, которое указывает дополнительный параметр regexp, — FILTER_VALIDATE_REGEXP:

```
$rex = '/^[a-zA-Z0-9_]{5,20}$/';
$val = 'admin';
echo filter_var($val, FILTER_VALIDATE_REGEXP,
    ['options' => ['regexp' => $rex]]);          // admin
$val = 'Вася Пупкин';
echo filter_var($val, FILTER_VALIDATE_REGEXP,
    ['options' => ['regexp' => $rex]]);          // FALSE
```

Чтобы удостовериться, является ли подвергаемое валидации значение массивом, следует указать флаг `FILTER_REQUIRE_ARRAY`:

```
$val = ['architecture', 'landscape'];
$arr = filter_var($val, FILTER_DEFAULT,
    ['flags' => FILTER_REQUIRE_ARRAY]);
echo implode(' ', $arr);                    // architecture landscape
$val = 'architecture';
echo filter_var($val, FILTER_DEFAULT,
    ['flags' => FILTER_REQUIRE_ARRAY]);          // FALSE
```

Удостовериться, что заданное значение не является массивом, позволяет флаг `FILTER_REQUIRE_SCALAR`:

```
$val = 'architecture';
echo filter_var($val, FILTER_DEFAULT,
    ['flags' => FILTER_REQUIRE_SCALAR]);         // architecture
$val = ['architecture', 'landscape'];
echo filter_var($val, FILTER_DEFAULT,
    ['flags' => FILTER_REQUIRE_SCALAR]);         // FALSE
```

Указать значение по умолчанию, возвращаемое, если заданное значение не пройдет валидацию, можно в дополнительном параметре `default`:

```
$val = 'cba123';
echo filter_var($val, FILTER_VALIDATE_INT);          // FALSE
echo filter_var($val, FILTER_VALIDATE_INT,
    ['options' => ['default' => 15]]);               // 15
```

Для валидации значения, полученного из веб-формы, удобно использовать функцию `filter_input`:

```
filter_input(<массив, из которого извлекается значение>,
             <имя параметра>[, <обозначение фильтра>[,
             <массив с настройками>]])
```

Суперглобальный *массив*, из которого извлекается подвергаемое валидации значение, обозначается константой `INPUT_GET` (`$_GET`) или `INPUT_POST` (`$_POST`). Вторым параметром указывается строка с именем GET- или POST-параметра, хранящего это значение. Остальные параметры аналогичны таковым у функции `filter_var`.

Функция `filter_input` возвращает само значение, если валидация прошла успешно, `FALSE` — в противном случае и `NULL`, если GET- или POST-параметр с указанным *именем* отсутствует.

Пример:

```
$age = filter_input(INPUT_POST, 'age', FILTER_VALIDATE_INT);
```

17.2.2. Нормализация данных

> *Нормализация* — в PHP очистка полученного из веб-формы значения от лишних символов и преобразование его в нужный тип.

Нормализация выполняется с помощью знакомых нам функций `filter_var` и `filter_input`. Типы значений, которые нужно получить в результате нормализации, и соответствующие им константы-*обозначения фильтров* приведены далее:

- исходная строка — `FILTER_DEFAULT` или `FILTER_UNSAFE_RAW` (эти константы имеют одинаковые значения и полностью взаимозаменяемы). При указании определенных флагов (см. табл. 17.1) может удалять или преобразовывать недопустимые символы. Нормализация в величину такого типа выполняется по умолчанию, если *обозначение типа* не задано:

```
$val = '123 сантиметра';
echo filter_var($val);                                  // 123 сантиметра
```

- целое число — `FILTER_SANITIZE_NUMBER_INT`:

```
$val = '123 сантиметра';
echo filter_var($val, FILTER_SANITIZE_NUMBER_INT); // 123
```

  Символы, не являющиеся цифрами и знаком «минус», в том числе и десятичная точка, будут удалены:

```
$val = '123.456 сантиметров';
echo filter_var($val, FILTER_SANITIZE_NUMBER_INT); // 123456
```

- вещественное число — `FILTER_SANITIZE_NUMBER_FLOAT` с флагом `FILTER_FLAG_ALLOW_FRACTION`:

```
$val = '123.456 сантиметров';
echo filter_var($val, FILTER_SANITIZE_NUMBER_FLOAT,
    ['flags' => FILTER_FLAG_ALLOW_FRACTION]);       // 123.456
```

  Без флага `FILTER_FLAG_ALLOW_FRACTION` выполняется нормализация в целочисленный тип;

- интернет-адрес в виде строки — `FILTER_SANITIZE_URL`:

```
$val = 'Наш адрес http://www.photogallery.ru';
echo filter_var($val, FILTER_SANITIZE_URL);
                                        // http://www.photogallery.ru
```

- адрес электронной почты в виде строки — `FILTER_SANITIZE_EMAIL`:

```
$val = '    support@photogallery.ru адрес поддержки';
echo filter_var($val, FILTER_SANITIZE_EMAIL);
                                            // support@photogallery.ru
```

- строка с недопустимыми символами, преобразованными в HTML-литералы, — FILTER_SANITIZE_SPECIAL_CHARS:

```
$val = 'Контора "Кук и сыновья"';
echo filter_var($val, FILTER_SANITIZE_SPECIAL_CHARS);
                                    // Контора &#34;Кук и сыновья&#34;
```

- строка, закодированная и подготовленная для помещения в состав интернет-адреса, — FILTER_SANITIZE_ENCODED:

```
$val = 'кот';
echo filter_var($val, FILTER_SANITIZE_ENCODED);  // %D0%BA%D0%BE%D1%82
```

- строка с удаленными HTML-тегами — FILTER_SANITIZE_STRING или FILTER_SANITIZE_STRIPPED (эти константы имеют одинаковые значения и полностью взаимозаменяемы):

```
$val = '<p><em>PHP</em> и <strong>MySQL</strong></p>';
echo filter_var($val, FILTER_SANITIZE_STRING);     // PHP и MySQL
```

Флаги, которые можно использовать при нормализации, представлены в табл. 17.1.

Таблица 17.1. *Флаги, используемые при нормализации*

Флаг	Обозначения совместимых типов	Описание
FILTER_FLAG_ENCODE_LOW	FILTER_SANITIZE_ENCODED, FILTER_SANITIZE_STRING, FILTER_SANITIZE_RAW	Преобразует символы с кодами от 0 до 31 в литералы
FILTER_FLAG_STRIP_LOW	FILTER_SANITIZE_ENCODED, FILTER_SANITIZE_SPECIAL_CHARS, FILTER_SANITIZE_STRING, FILTER_UNSAFE_RAW	Удаляет символы с кодами от 0 до 31
FILTER_FLAG_ENCODE_HIGH		Преобразует символы с кодами от 128 и выше в литералы
FILTER_FLAG_STRIP_HIGH		Удаляет символы с кодами от 128 и выше
FILTER_FLAG_STRIP_BACKTICK		Удаляет символы обратной косой черты \
FILTER_FLAG_NO_ENCODE_QUOTES	FILTER_SANITIZE_STRING	Не преобразует символы одинарной и двойной кавычки
FILTER_FLAG_ENCODE_AMP	FILTER_SANITIZE_STRING, FILTER_SANITIZE_RAW	Преобразует амперсанд & в литерал
FILTER_FLAG_ALLOW_FRACTION	FILTER_SANITIZE_NUMBER_FLOAT	Необходим для нормализации вещественных чисел с дробной частью
FILTER_FLAG_ALLOW_THOUSAND		Разрешает наличие запятой в качестве разделителя тысяч
FILTER_FLAG_ALLOW_SCIENTIFIC		Выполняет нормализацию чисел в экспоненциальной нотации

Таблица 17.1 (окончание)

Флаг	Обозначения совместимых типов	Описание
FILTER_FORCE_ARRAY	Все	Если указано обычное значение, преобразует его в массив из одного элемента

Также поддерживаются описанные ранее флаги FILTER_REQUIRE_ARRAY и FILTER_REQUIRE_SCALAR.

Примеры:

```
$val = '<p><em>PHP</em> и <strong>MySQL</strong></p>';
echo filter_var($val, FILTER_SANITIZE_STRING);        // PHP и MySQL
echo filter_var($val, FILTER_SANITIZE_STRING,
    ['flags' => FILTER_FLAG_ENCODE_HIGH]);      // PHP &#208;&#184; MySQL
$val = '1.2e5';
echo filter_var($val, FILTER_SANITIZE_NUMBER_FLOAT,
    ['flags' => FILTER_FLAG_ALLOW_FRACTION]);         // 1.25
echo filter_var($val, FILTER_SANITIZE_NUMBER_FLOAT, ['flags' =>
    FILTER_FLAG_ALLOW_FRACTION | FILTER_FLAG_ALLOW_SCIENTIFIC]); // 1.2e5
```

17.3. Упражнение. Реализуем в базовом классе модели добавление, правку и удаление записей

Добавим в базовый класс модели Models\Model следующие методы:

- insert(*<массив со значениями полей>*) — добавляет запись, занося в ее поля значения из переданного ему ассоциативного *массива*, и возвращает номер добавленной записи.

  Ключи элемента *массива* соответствуют именам полей таблицы, а значения элементов станут значениями этих полей;

- update — ищет запись по указанному *искомому значению* в заданном *поле* и исправляет ее, занося в поля значения из переданного ассоциативного *массива*:

  ```
  update(<массив со значениями полей>, <искомое значение>[,
        <поле, в котором ищется значение>])
  ```

- delete — ищет запись по указанному *искомому значению* в заданном *поле* и удаляет ее:

  ```
  delete(<искомое значение>[, <поле, в котором ищется значение>])
  ```

  Если поле не задано, поиск *искомого значения* будет проводиться в поле id.

Также мы объявим ряд пустых защищенных статических методов, вызываемых непосредственно перед выполнением каждой из этих операций, и позволяющих —

путем перекрытия в производных классах — выполнить в эти моменты времени какие-либо дополнительные действия.

Файлы сайта фотогалереи хранятся в папке 16\16.5\s, а файл photogallery.sql с дампом базы данных `photogallery` — в папке 13\13.4\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Откроем модуль models\model.php, хранящий базовый класс модели `Models\Model`, и добавим в него методы `insert` и `before_insert` (здесь и далее добавления в ранее написанный код выделены полужирным шрифтом):

```
class Model implements \Iterator {
    . . .
    protected function before_insert(&$fields) {}

    function insert($fields) {
        static::before_insert($fields);
        $s = 'INSERT INTO ' . static::TABLE_NAME;
        $s2 = $s1 = '';
        foreach ($fields as $n => $v) {
            if ($s1) {
                $s1 .= ', ';
                $s2 .= ', ';
            }
            $s1 .= $n;
            $s2 .= ':' . $n;
        }
        $s .= ' (' . $s1 . ') VALUES (' . $s2 . ');';
        $this->run($s, $fields);
        $id = self::$connection -> lastInsertId();
        return $id;
    }
}
```

Мы перебираем полученный *массив со значениями полей* и создаем две строки: первая — с именами полей, разделенных запятыми, вторая — с именами параметров запроса, имена которых совпадают с именами полей. Далее формируем SQL-запрос, подставив в нужные места только что сформированные строки. После чего выполняем запрос и возвращаем номер добавленной записи.

Метод `before_insert`, принимающий *массив со значениями полей* (обязательно передаваемый по ссылке, поскольку, возможно, он будет изменен в этом методе), вызывается перед добавлением записи. Перекрыв его в производном классе, мы можем выполнить перед добавлением записи какие-либо дополнительные действия.

2. Добавим в тот же класс методы `update` и `before_update`:

```
class Model implements \Iterator {
    . . .
```

```
    protected function before_update(&$fields, $value,
        $key_field = 'id') {}

    function update($fields, $value, $key_field = 'id') {
        static::before_update($fields, $value, $key_field);
        $s = 'UPDATE ' . static::TABLE_NAME . ' SET ';
        $s1 = '';
        foreach ($fields as $n => $v) {
            if ($s1)
                $s1 .= ', ';
            $s1 .= $n . ' = :' . $n;
        }
        $s .= $s1 . ' WHERE ' . $key_field . ' = :__key;';
        $fields['__key'] = $value;
        $this->run($s, $fields);
    }
}
```

Здесь мы также перебираем полученный *массив со значениями полей* и создаем строку с разделенными запятыми позициями вида: *<имя поля> = <имя параметра запроса>*, которую подставляем в формируемый SQL-запрос. Далее добавляем к запросу языковую конструкцию `WHERE` с условием, отфильтровывающим лишь ту запись, у которой указанное *поле* хранит *искомое значение*, представляемое параметром запроса `__key`. Напоследок добавляем *искомое значение* в *массив* под ключом `__key` и выполняем готовый запрос.

Метод `before_update` вызывается перед правкой записи, принимает те же параметры, что и метод `update`, и служит для выполнения в указанные моменты дополнительных действий.

3. Добавим методы `delete` и `before_delete`:

```
class Model implements \Iterator {
    . . .
    protected function before_delete($value, $key_field = 'id') {}

    function delete($value, $key_field = 'id') {
        static::before_delete($value, $key_field);
        $s = 'DELETE FROM ' . static::TABLE_NAME;
        $s .= ' WHERE ' . $key_field . ' = ?;';
        $this->run($s, [$value]);
    }
}
```

Метод `before_delete` вызывается перед удалением записи, принимает те же параметры, что и метод `delete`, и предназначен для выполнения дополнительных действий.

17.4. Упражнение. Создаем базовый класс формы

Напишем базовый класс формы `Forms\Form`, который будет готовить извлеченные из базы данные к выводу в веб-форме и, наоборот, преобразовывать полученные из веб-формы данные к сохранению в базе. Такая форма должна содержать набор *полей*, каждое из которых представляет одно значение, извлекаемое из базы данных и отображаемое в элементе управления. Имя поля будет совпадать с наименованием соответствующего элемента управления.

Класс формы получит защищенную статическую константу `FIELDS`, хранящую ассоциативный массив с параметрами полей формы. Ключи элементов этого массива будут совпадать с именами полей, а значениями станут ассоциативные массивы, содержащие элементы с ключами:

- `initial` — изначальное значение для поля, выводимое в пустой веб-форме. Если отсутствует, изначальным значением будет пустая строка;
- `type` — тип значения в виде строки `'integer'` (целое число), `'float'` (вещественное число), `'boolean'` (логическая величина), `'timestamp'` (временна́я отметка), `'email'` (адрес электронной почты) или `'string'` (обычная строка). Если отсутствует, поле будет иметь тип `'string'`;
- `optional` — если присутствует, поле необязательно к заполнению. В противном случае поле обязательно должно быть заполнено. Значение этого элемента может быть любым — важен лишь сам его факт присутствия;
- `nosave` — если присутствует, поле не предназначено к сохранению в базе данных. Значение элемента также может быть любым.

Еще класс `Forms\Form` получит три общедоступных статических метода:

- `get_initial_data([<`*ассоциативный массив с изначальными данными*`>])` — возвращает ассоциативный массив с данными, подготовленными для помещения в выводимую на экран веб-форму. В возвращаемом массиве ключи элементов совпадают с именами полей, а значения уже преобразованы в формат, подходящий для помещения в HTML-код, который формирует элементы управления.

  Изначальные данные, выводимые в веб-форме, указываются в передаваемом с параметром *массиве*. Если он не указан, будет возвращен массив для вывода пустой веб-формы, элементы управления которой хранят изначальные значения (записываются в константе `FIELDS`);

- `get_normalized_data(<`*ассоциативный массив с данными из веб-формы*`>)` — возвращает ассоциативный массив с нормализованными данными, занесенными в веб-форму. Возвращенный массив может содержать элемент с ключом `__errors`, хранящий ассоциативный массив с сообщениями об ошибках для каждого из полей формы;

- `get_prepared_data(<`*ассоциативный массив с нормализованными данными*`>)` — возвращает ассоциативный массив с данными из формы, подготовленными для сохранения в базе данных.

Объявим также три пустых защищенных статических метода, вызываемых при выполнении каждой из этих операций непосредственно перед возвратом результата, и позволяющих — путем перекрытия в производных классах — произвести над возвращаемыми данными какие-либо дополнительные действия.

Остальные технические вопросы решим по ходу программирования.

Файлы сайта фотогалереи хранятся в папке 17\17.3\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Создадим в папке modules папку forms.

   Код базового класса формы `Forms\Form` весьма объемен, и мы будем писать его по частям.

2. Создадим модуль modules\forms\form.php и запишем в него первую часть кода класса, содержащую объявления константы `FIELDS`, методов `get_initial_data`, `after_initialize_data` и служебного метода `get_initial_value`:

```
<?php
namespace Forms;
class Form {
    protected const FIELDS = [];

    private static function get_initial_value($fld_name, $fld_params,
        $initial = []) {
        if (isset($initial[$fld_name])}
            $val = $initial[$fld_name];
        else if (isset($fld_params['initial']))
            $val = $fld_params['initial'];
        else
            $val = '';
        if ($fld_params['type'] == 'timestamp') {
            if (gettype($val) == 'integer')
                $val = strftime('%Y-%m-%d %H:%M:%S', $val);
            $val = explode(' ', $val);
        }
        return $val;
    }

    protected static function after_initialize_data(&$data) {}

    public static function get_initial_data($initial = []) {
        $data = [];
        foreach (static::FIELDS as $fld_name => $fld_params)
            $data[$fld_name] = self::get_initial_value($fld_name,
                $fld_params, $initial);
        static:: after_initialize_data($data);
        return $data;
    }
}
```

Закрытый статический метод `get_initial_value` принимает в качестве параметров имя поля, массив с его параметрами, необязательный массив с изначальными данными и возвращает изначальное значение для указанного в первом параметре поля.

В этом методе проверяем, есть ли в массиве с изначальными данными искомое поле, и, если есть, берем значение оттуда. В противном случае извлекаем изначальное значение из элемента `initial` параметров поля (записанных в константе `FIELDS`), а если этого элемента нет, используем пустую строку.

Если поле принадлежит типу `'timestamp'` (временна́я отметка), мы проверяем, является ли изначальное значение целым числом, и, если так, преобразуем его в строку формата:

<год>-<номер месяца>-<число>[<часы>:<минуты>:<секунды>]

А ее, в свою очередь, разбиваем на дату и время, из которых формируем массив из двух элементов, чтобы потом по отдельности занести их в поля ввода даты и времени.

Метод `get_initial_data` перебирает поля, хранящиеся в константе `FIELDS`, и каждому полю присваивает изначальное значение, полученное вызовом метода `get_initial_value`. Эти значения он заносит в генерируемый ассоциативный массив `$data`, который потом возвращает в качестве результата.

Метод `after_initialize_data`, принимающий в качестве параметра ссылку на массив с изначальными данными, вызывается непосредственно перед его возвратом. Переопределив этот метод в производном классе, мы сможем добавить в генерируемый массив с данными новые значения или изменить уже существующие.

3. Добавим в класс `Forms\Form` объявление методов `get_normalized_data` и `after_normalize_data` (здесь и далее добавления и правки в ранее написанном коде выделены полужирным шрифтом):

```
class Form {
    . . .
    protected static function after_normalize_data(&$data, &$errors)
        {}

    public static function get_normalized_data($form_data) {
        $data = [];
        $errors = [];
        foreach (static::FIELDS as $fld_name => $fld_params) {
            $fld_type = (isset($fld_params['type'])) ?
                $fld_params['type'] : 'string';
            if ($fld_type == 'boolean')
                $data[$fld_name] = !empty($form_data[$fld_name]);
            else {
                if (empty($form_data[$fld_name])) {
                    $data[$fld_name] =
                      self::get_initial_value($fld_name, $fld_params);
```

```
        if (!isset($fld_params['optional']))
            $errors[$fld_name] = 'Это поле обязательно ' .
            'к заполнению';
    } else {
        $fld_value = $form_data[$fld_name];
        switch ($fld_type) {
            case 'integer':
                $v = filter_var($fld_value,
                    FILTER_SANITIZE_NUMBER_INT);
                if ($v)
                    $data[$fld_name] = $v;
                else
                    $errors[$fld_name] = 'Введите ' .
                        'целое число';
                break;
            case 'float':
                $v = filter_var($fld_value,
                FILTER_SANITIZE_NUMBER_FLOAT,
                ['flags' => FILTER_FLAG_ALLOW_FRACTION]);
                if ($v)
                    $data[$fld_name] = $v;
                else
                    $errors[$fld_name] = 'Введите ' .
                        'вещественное число';
                break;
            case 'timestamp':
                $v = strtotime(implode(' ', $fld_value));
                if ($v)
                    $data[$fld_name] = $fld_value;
                else
                    $errors[$fld_name] = 'Выберите ' .
                        'дату и время';
                break;
            case 'email':
                $v = filter_var($fld_value,
                    FILTER_SANITIZE_EMAIL);
                if ($v)
                    $data[$fld_name] = $v;
                else
                    $errors[$fld_name] = 'Введите ' .
                        'адрес электронной почты';
                break;
            default:
                $data[$fld_name] = filter_var($fld_value,
                    FILTER_SANITIZE_STRING);
        }
    }
```

```
            }
        }
        static::after_normalize_data($data, $errors);
        if ($errors)
            $data['__errors'] = $errors;
        return $data;
    }
}
```

Перебираем список полей из константы `FIELDS` и для каждого из полей:

- получаем из элемента `type` параметров поля его тип;
- если поле логического типа, заносим в генерируемый ассоциативный массив `$data` в качестве значения этого поля `TRUE`, если в массиве данных, полученных из веб-формы, есть элемент, одноименный с полем, и `FALSE` — в противном случае. После этого завершаем обработку текущего поля;
- если поле относится к другому типу, пытаемся извлечь значение из массива с данными, полученными из веб-формы. Если значения там нет, заносим в генерируемый ассоциативный массив `$data` изначальное значение этого поля (полученное вызовом метода `get_initial_value`). А если поле еще и не помечено как необязательное, добавляем в другой генерируемый массив `$errors`, хранящий список сообщений об ошибках, соответствующее сообщение. После чего прекращаем обработку текущего поля;
- выполняем нормализацию значения поля, если она увенчалась успехом, заносим результат в массив `$data`, в противном случае — текст сообщения об ошибке ввода в массив `$errors`.

  Если поле имеет тип `'string'` (обычная строка), в процессе нормализации мы очищаем ее от HTML- и PHP-тегов, чтобы не допустить внедрения злоумышленником в страницы фотогалереи нежелательного HTML-кода (например, тега `<script>`, загружающего вредоносный веб-сценарий).

Если массив с ошибками `$errors` не пуст, заносим его в массив с данными `$data` под ключом `__errors`, после чего возвращаем массив `$data`.

Метод `after_normalize_data`, принимающий в качестве параметров ссылки на массивы с данными и ошибками, вызывается непосредственно перед возвратом результата. Переопределив его, мы можем изменить уже созданные массивы.

4. Добавим методы `get_prepared_data` и `after_prepare_data`:

```
class Form {
    . . .
    protected static function after_prepare_data(&$data, &$norm_data)
        {}

    public static function get_prepared_data($norm_data) {
        $data = [];
```

```
            foreach (static::FIELDS as $fld_name => $fld_params) {
                if (!isset($fld_params['nosave']) &&
                    isset($norm_data[$fld_name])) {
                    $val = $norm_data[$fld_name];
                    if ($fld_params['type'] == 'timestamp')
                        $data[$fld_name] = implode(' ', $val);
                    else
                        $data[$fld_name] = $val;
                }
            }
            static::after_prepare_data($data, $norm_data);
            return $data;
        }
    }
```

Метод `after_prepare_data`, принимающий ссылки на массивы с данными, подготовленными для сохранения в базе, и с нормализованными данными, вызывается перед выдачей результата.

Далее реализуем в фотогалерее добавление, правку и удаление комментариев.

17.5. Упражнение. Реализуем добавление комментариев

Создадим на странице просмотра изображения — над списком комментариев — форму для добавления нового комментария. Поместим на нее раскрывающийся список для выбора пользователя, область редактирования, куда будет заноситься содержимое комментария, а также, в целях отладки, поля для указания даты и времени добавления комментария (потом, проверив код в работе, мы удалим их).

Файлы сайта фотогалереи хранятся в папке 17\17.4\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Найдем в папке 17\!sources файл с дополненной таблицей стилей styles.css и скопируем его в корневую папку сайта, затерев имеющийся там старый файл.
2. Объявим функцию `Helpers\redirect`, выполняющую перенаправление по указанному *интернет-адресу* с заданным *статусом* (если не указан — 302, то есть временное перенаправление):

   ```
   redirect(<интернет-адрес>[, <код статуса>])
   ```

   Откроем модуль modules\helpers.php в текстовом редакторе и добавим объявление функции перенаправления (здесь и далее дополнения и правки в уже написанном коде выделены полужирным шрифтом):

   ```
   namespace Helpers {
       . . .
       function redirect(string $url, int $status = 302) {
           header('Location: ' . $url, TRUE, $status);
       }
   }
   ```

3. Объявим функцию Helpers\show_errors, выводящую сообщения об ошибках ввода в элемент управления с указанным *наименованием*:

```
show_errors(<наименование элемента управления>,
            <массив данных для веб-формы>)
```

Массив данных для веб-формы — тот самый массив, что возвращается методами: get_initial_data и get_normalized_data класса формы.

Сообщения об ошибках будут выводиться в блоках (тегах) со стилевым классом error.

Добавим в модуль modules\helpers.php объявление этой функции:

```
namespace Helpers {
    . . .
    function show_errors(string $fld_name, array $form_data) {
        if (isset($form_data['__errors'][$fld_name]))
            echo '<div class="error">' .
                $form_data['__errors'][$fld_name] . '</div>';
    }
}
```

4. Нам понадобится класс формы — для обработки введенного комментария. Дадим ему имя Forms\Comment и сделаем производным от класса Forms\Form.

Создадим модуль modules\forms\comment.php и запишем в него код этого класса, показанный в листинге 17.1.

Листинг 17.1. Модуль modules\forms\comment.php

```
<?php
namespace Forms;
class Comment extends \Forms\Form {
    protected const FIELDS = [
        'user' => ['type' => 'integer'],
        'contents' => ['type' => 'string'],
        'uploaded' => ['type' => 'timestamp']
    ];
}
```

В массиве, присвоенном константе FIELDS, указываем все поля, которые должны присутствовать в веб-форме, и их типы: user (номер пользователя, целое число), contents (содержание комментария, строка) и uploaded (временна́я отметка публикации).

5. Страница с выбранным изображением отображается методом item контроллера Controllers\Images. Дополним его кодом, выводящим форму для ввода комментария и сохраняющим введенный комментарий в базе данных.

При первом обращении к этой странице, которое будет выполнено с применением HTTP-метода GET, на экране, помимо запрошенного изображения и коммен-

тариев к нему, будет присутствовать форма для ввода нового комментария. После заполнения формы и нажатия кнопки отправки данных введенные данные будут отправлены по тому же интернет-адресу, но уже с применением метода POST. В этот момент новый комментарий будет сохранен в базе данных, и выполнится перенаправление на тот же интернет-адрес. В результате появится та же страница, на которой, в числе уже существующих, будет присутствовать и только что введенный комментарий.

Откроем модуль modules\controllers\images.php с кодом контроллера Controllers\Images и внесем исправления в метод item:

```
class Images extends BaseController {
    . . .
    function item(int $index) {
        if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            $comment_form =
                \Forms\Comment::get_normalized_data($_POST);
            if (!isset($comment_form['__errors'])) {
                $comment_form =
                    \Forms\Comment::get_prepared_data($comment_form);
                $comment_form['picture'] = $index;
                $comments = new \Models\Comment();
                $comments->insert($comment_form);
                \Helpers\redirect('/' . $index .
                    \Helpers\get_GET_params(['page', 'filter',
                    'ref']));
            }
        } else
            $comment_form =
            \Forms\Comment::get_initial_data(['uploaded' => time()]);
        $users = new \Models\User();
        $users->select('*', NULL, '', NULL, 'name');
        . . .
        $ctx = ['pict' => $pict, 'site_title' => $pict['title'],
            'comments' => $comments, 'form' => $comment_form,
            'users' => $users];
        $this->render('item', $ctx);
    }
}
```

При запросе HTTP-методом GET дополнительно формируем, вызвав метод get_initial_data класса формы, массив с изначальными данными для формы ввода комментария, извлекаем из базы данных список пользователей и добавляем его в контекст шаблона.

При запросе методом POST нормализуем полученные из формы данные, вызвав метод get_normalized_data класса формы, и проверяем, есть ли в полученном массиве с этими данными элемент __errors, то есть были ли допущены при вво-

де комментария ошибки. Если данные корректны, вызовом метода `get_prepared_data` формы получаем массив с данными, подготовленными к сохранению в базе, добавляем к этому массиву элемент `picture` с номером изображения (он будет сохранен в одноименном поле таблицы `comments`), сохраняем комментарий и производим перенаправление на тот же интернет-адрес с сохранением всех GET-параметров, которые есть в текущем интернет-адресе. Если же данные некорректны, снова будет отображена та же страница с формой, в которой выведены описания допущенных при вводе ошибок.

> ***После сохранения данных всегда выполняйте перенаправление!***
>
> В нашем случае, если этого не сделать, будет выведена страница с изображением и списком комментариев, в числе которых окажется и только что добавленный. Однако если пользователь обновит страницу, запрос снова выполнится с применением HTTP-метода `POST`, и тот же комментарий будет добавлен еще раз.

6. Для вывода веб-формы ввода комментария мы создадим шаблон-фрагмент, который позже используем и на странице правки комментария.

   Создадим модуль modules\templates_comment_form.inc.php и сохраним в нем код шаблона-фрагмента, выводящего форму комментария (листинг 17.2).

Листинг 17.2. Модуль modules\templates_comment_form.inc.php

```
<form class="bigform" method="post">
    <label for="comment_user">Пользователь</label>
    <select id="comment_user" name="user">
        <?php foreach ($users as $user) { ?>
            <option value="<?php echo $user['id'] ?>"
            <?php if ($form['user'] == $user['id']) { ?>
            selected<?php } ?>>
                <?php echo $user['name'] ?>
            </option>
        <?php } ?>
    </select>
    <label for="comment_contents">Содержание</label>↵
    <textarea id="comment_contents" name="contents">
    <?php echo $form['contents'] ?></textarea>
    <?php \Helpers\show_errors('contents', $form) ?>
    <label for="comment_uploaded">Дата и время написания</label>
    <div class="field">
        <input type="date" id="comment_uploaded" name="uploaded[]"
        value="<?php echo $form['uploaded'][0] ?>">
        <input type="time" name="uploaded[]"
        value="<?php echo $form['uploaded'][1] ?>">
    </div>
    <input type="submit" value="Отправить">
</form>
```

Раскрывающийся список с наименованием `user`, в котором выбирается пользователь-автор, заполняем именами имеющихся в базе пользователей. Если номер очередного пользователя совпадает с номером, хранящимся в данных формы, делаем соответствующий пункт списка выбранным.

Для ввода даты и времени публикации комментария применяем отдельные поля ввода, которым даем одинаковые наименования `uploaded[]`. В результате в POST-параметре `uploaded` окажется массив из двух элементов, хранящих дату и время, который будет успешно обработан написанным нами классом формы `Forms\Form`.

Для вывода сообщений об ошибках применяем объявленную ранее функцию `Helpers\show_errors`. Выводить сообщения имеет смысл только у области редактирования с наименованием `contents`, в которое заносится содержание комментария, поскольку указать некорректные данные в раскрывающемся списке, полях ввода даты и времени невозможно в принципе.

7. Откроем модуль modules\templates\item.php с шаблоном страницы для просмотра изображения и добавим код, включающий шаблон-фрагмент формы:

```
. . .
<p>Дата и время публикации:
<?php echo \Helpers\get_formatted_timestamp($pict['uploaded']) ?></p>
<h3>Добавить комментарий</h3>
<?php require \Helpers\get_fragment_path('__comment_form') ?>
<h3>Комментарии</h3>
. . .
```

Перейдем на наш сайт, откроем какое-либо изображение и попробуем оставить к нему произвольный комментарий.

17.6. Упражнение. Реализуем правку и удаление комментариев

Переход на страницу правки комментария будет выполняться при обращении по интернет-адресу формата: **/<*номер изображения*>/comments/<*номер комментария*>/edit/**. Исправив комментарий в веб-форме на этой странице, пользователь нажмет кнопку отправки данных и после сохранения комментария вернется на страницу просмотра изображения.

На страницу удаления комментария можно будет перейти по интернет-адресу формата: **/<*номер изображения*>/comments/<*номер комментария*>/delete/**. Страница покажет сведения о комментарии (имя пользователя-автора, содержание и временна́я отметка публикации) и веб-форму, содержащую лишь кнопку отправки данных. При нажатии на нее и будет выполнено удаление комментария, после чего, опять же, произойдет перенаправление на страницу просмотра изображения.

Файлы сайта фотогалереи и файл photogallery.sql с дампом базы данных `photogallery` хранятся в папке 17\17.5\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Откроем модуль маршрутизатора modules\router.php и добавим код, распознающий интернет-адреса страниц правки и удаления комментария (здесь и далее добавления и правки в написанном ранее коде выделены полужирным шрифтом):

```
. . .
} else if (preg_match('/^(\d+)$/', $request_path, $result) === 1) {
    . . .
} else if (preg_match('/^(\d+)\/comments\/(\d+)\/edit$/',
    $request_path, $result) == 1) {
    $picture_index = (integer)$result[1];
    $comment_index = (integer)$result[2];
    $ctr = new \Controllers\Comments();
    $ctr->edit($picture_index, $comment_index);
} else if (preg_match('/^(\d+)\/comments\/(\d+)\/delete$/',
    $request_path, $result) == 1) {
    $picture_index = (integer)$result[1];
    $comment_index = (integer)$result[2];
    $ctr = new \Controllers\Comments();
    $ctr->delete($picture_index, $comment_index);
} else if ($request_path == '') {
. . .
```

2. Создадим модуль modules\controllers\comments.php и запишем в него код контроллера Controllers\Comments (листинг 17.3), который будет выполнять правку и удаление комментариев.

Листинг 17.3. Модуль modules\controllers\comments.php

```
<?php
namespace Controllers;
class Comments extends BaseController {
    function edit(int $picture_index, int $comment_index) {
        if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            $comment_form =
                \Forms\Comment::get_normalized_data($_POST);
            if (!isset($comment_form['__errors'])) {
                $comment_form =
                    \Forms\Comment::get_prepared_data($comment_form);
                $comments = new \Models\Comment();
                $comments->update($comment_form, $comment_index);
                \Helpers\redirect('/' . $picture_index .
                    \Helpers\get_GET_params(['page', 'filter',
                    'ref']));
            }
        } else {
            $comments = new \Models\Comment();
            $comment = $comments->get_or_404($comment_index);
```

```
            $comment_form =
                \Forms\Comment::get_initial_data($comment);
        }
        $users = new \Models\User();
        $users->select('*', NULL, '', NULL, 'name');
        $ctx = ['form' => $comment_form, 'users' => $users,
            'picture' => $picture_index,
            'site_title' => 'Правка комментария'];
        $this->render('comment_edit', $ctx);
    }

    function delete(int $picture_index, int $comment_index) {
        if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            $comments = new \Models\Comment();
            $comments->delete($comment_index);
            \Helpers\redirect('/' . $picture_index .
                \Helpers\get_GET_params(['page', 'filter',
                'ref']));
        } else {
            $comments = new \Models\Comment();
            $comment = $comments->get_or_404($comment_index,
                'comments.id',
                'users.name AS user_name, contents, uploaded',
                ['users']);
            $ctx = ['comment' => $comment,
                'picture' => $picture_index,
                'site_title' => 'Удаление комментария'];
            $this->render('comment_delete', $ctx);
        }
    }
}
```

При запросе методом GET выводится страница правки или удаления комментария, а при POST-запросе, собственно, выполнится правка или удаление. В контекст шаблона заносится, помимо всего прочего, еще и номер изображения — он понадобится для формирования интернет-адреса в гиперссылке **Назад**.

3. Откроем модуль modules\templates\item.php, в котором хранится шаблон страницы для просмотра изображения, и добавим код, выводящий в составе каждого комментария абзац с гиперссылками **Исправить** и **Удалить**:

```
. . .
<h3>Добавить комментарий</h3>
<?php require \Helpers\get_fragment_path('__comment_form') ?>
<?php $u2 = \Helpers\get_GET_params(['page', 'filter', 'ref']) ?>
<h3>Комментарии</h3>
<?php foreach ($comments as $comment) { ?>
    . . .
```

```
    <?php $u1 = '/' . $pict['id'] . '/comments/' . $comment['id'] ?>
    <p><a href="<?php echo $u1 . '/edit' . $u2 ?>">Исправить</a>
    <a href="<?php echo $u1 . '/delete' . $u2 ?>">Удалить</a></p>
    <p> </p>
<?php } ?>
. . .
```

В интернет-адреса этих гиперссылок добавляем все GET-параметры, присутствующие в текущем интернет-адресе.

4. Создадим модуль modules\templates\comment_edit.php и запишем в него код шаблона, создающего страницу для правки комментария (листинг 17.4).

Листинг 17.4. Модуль modules\templates\comment_edit.php

```
<?php require \Helpers\get_fragment_path('__header') ?>
<h2>Правка комментария</h2>
<?php require \Helpers\get_fragment_path('__comment_form') ?>
<?php $ret = '/' . $picture .
    \Helpers\get_GET_params(['page', 'filter', 'ref']) ?>
<p><a href="<?php echo $ret ?>">Назад</a></p>
<?php require \Helpers\get_fragment_path('__footer') ?>
```

Код получился очень компактным за счет того, что для вывода веб-формы, в которой выполняется правка комментария, мы использовали созданный ранее шаблон-фрагмент __comment_form.inc.php.

Перейдем на наш сайт, откроем страницу с изображением, к которому ранее были добавлены комментарии, и попробуем исправить один из них — его содержание, пользователя-автора и временну́ю отметку публикации.

Сделайте сами

- Создайте шаблон страницы для удаления комментария modules\templates\comment_delete.php. Выведите на этой странице имя пользователя-автора, содержание комментария, временну́ю отметку публикации и веб-форму, включающую лишь кнопку отправки данных, по нажатию на которую и будет выполнено удаление комментария.

  Добавьте какой-либо комментарий к одному из изображений и попробуйте удалить его.

- Удалите из веб-формы добавления и правки комментария те поля ввода даты и времени, посредством которых задается временна́я отметка публикации и которые были помещены в форму в целях отладки. Также не забудьте удалить объявление поля `uploaded` из класса формы `Forms\Comment` и код из метода `item` контроллера `Controllers\Images`, задающий для временно́й отметки начальное значение. После этого при добавлении комментария в базу данных в качестве временно́й отметки будут записываться текущие дата и время.

Урок 18
Работа с файлами и папками

18.1. Получение сведений о текущем программном модуле

- Полный путь к файлу текущего модуля — константа __FILE__:

```
// Модуль modules\controllers\images.php
echo __FILE__;        // C:\xampp\htdocs\modules\controllers\images.php
```

- Полный путь к папке, в которой хранится файл с текущим модулем, без завершающего слеша — константа __DIR__:

```
// Модуль modules\controllers\images.php
echo __DIR__;                    // C:\xampp\htdocs\modules\controllers
```

18.2. Работа с файлами

18.2.1. Получение сведений о файле

- Проверка существования файла с заданным *путем* — функция file_exists(<*путь к файлу*>). Возвращает TRUE, если файл существует, и FALSE — в противном случае:

```
echo file_exists('C:\xampp\readme_en.txt');        // TRUE
echo file_exists('C:\xampp\readme_ru.txt');        // FALSE
```

- Отдельные части пути к файлу — функция pathinfo:

```
pathinfo(<путь к файлу>[, <обозначение возвращаемой части>])
```

Если *обозначение возвращаемой части* не указано, будет возвращен ассоциативный массив со следующими элементами:

- dirname — путь к папке, где хранится файл, без конечного слеша;
- basename — имя файла с расширением;
- filename — имя файла без расширения;
- extension — расширение файла без начальной точки.

Пример:

```
$info = pathinfo('C:\xampp\xampp_control.exe');
echo $info['dirname'];                              // C:\xampp
```

```
echo $info['basename'];                          // xampp_control.exe
php > echo $info['filename'];                     // xampp_control
php > echo $info['extension'];                    // exe
```

Если *обозначение возвращаемой части* указано, возвращается только обозначенная часть пути. В качестве *обозначения* можно указать константу: PATHINFO_DIRNAME, PATHINFO_BASENAME, PATHINFO_FILENAME или PATHINFO_EXTENSION.

Пример:

```
echo pathinfo('C:\xampp\xampp-control.log',
    PATHINFO_EXTENSION);                          // log
```

- Размер файла — функция `filesize(<путь к файлу>)`. Размер возвращается в байтах. В случае ошибки чтения возвращается FALSE:

```
echo filesize('C:\xampp\readme_en.txt');          // 7367
```

- Время последнего изменения файла — функция `filemtime(<путь к файлу>)`. Время возвращается в виде временно́й отметки PHP. В случае ошибки чтения возвращается FALSE:

```
echo filemtime('C:\xampp\readme_en.txt');         // 1567439188
```

- Время последнего доступа к файлу — функция `fileatime(<путь к файлу>)`. Время возвращается в виде временно́й отметки PHP. В случае ошибки чтения возвращается FALSE:

```
echo fileatime('C:\xampp\readme_en.txt');         // 1567505206
```

- Проверка, является ли файл собственно файлом, — функция `is_file(<путь к файлу>)`:

```
echo is_file('C:\xampp\readme_en.txt');           // TRUE
// C:\xampp\htdocs — это папка, а не файл
echo is_file('C:\xampp\htdocs');                  // FALSE
```

- Проверка, является ли файл исполняемым, — функция `is_executable(<путь к файлу>)`:

```
echo is_executable('C:\xampp\xampp-control.exe'); // TRUE
echo is_executable('C:\xampp\readme_en.txt');     // FALSE
```

- Проверка, существует ли файл и доступен ли он для чтения, — функция `is_readable(<путь к файлу>)`:

```
echo is_readable('C:\xampp\readme_en.txt');       // TRUE
```

- Проверка, существует ли файл и доступен ли он для записи, — функция `is_writable(<путь к файлу>)`:

```
echo is_writable('C:\xampp\readme_en.txt');       // TRUE
// Файлы из папки c:\windows недоступны для записи
echo is_writable('C:\windows\win.ini');           // FALSE
```

Также можно использовать аналогичную функцию `writeable`.

- Абсолютный путь к файлу — функция `realpath(<путь к файлу>)`. В процессе формирования абсолютного пути раскрываются переходы \..\ и устраняются лишние слеши:

```
$fr1 = 'C:\xampp\phpMyAdmin';
$fr2 = '\..\\';
$fr3 = '\passwords.txt';
$path = $fr1 . $fr2 . $fr3;
echo $path;                    // C:\xampp\phpMyAdmin\..\\passwords.txt
echo realpath($path);                    // C:\xampp\passwords.txt
```

18.2.2. Манипуляции с файлами

- Копирование файла — функция `copy`:

```
copy(<путь к исходному файлу>, <путь к создаваемой копии>)
```

Возвращается `TRUE`, если копирование завершилось удачно, и `FALSE` — в противном случае.

Пример создания копии файла readme_en.txt, хранящегося в папке C:\xampp. Копии присваивается имя readme_en.bak:

```
copy('C:\xampp\readme_en.txt', 'C:\xampp\readme_en.bak');
```

- Переименование или перемещение файла — функция `rename`:

```
rename(<исходный путь к файлу>, <новый путь к файлу>)
```

Возвращается `TRUE`, если операция завершилась удачно, и `FALSE` — в противном случае.

Пример переименования файла readme_en.bak, хранящегося в папке C:\xampp, в readme_en.txt.bak:

```
rename('C:\xampp\readme_en.bak', 'C:\xampp\readme_en.txt.bak');
```

Пример перемещения файла readme_en.txt.bak в «корень» диска D::

```
rename('C:\xampp\readme_en.txt.bak', 'D:\readme_en.txt.bak');
```

- Удаление файла — функция `unlink(<путь к удаляемому файлу>)`. Возвращает `TRUE`, если файл был успешно удален, и `FALSE` — в противном случае:

```
unlink('C:\xampp\readme_en.bak');
```

18.2.3. Чтение и запись

- Чтение содержимого текстового файла — функция `file_get_contents(<путь к файлу>)`. Содержимое файла возвращается в виде строки. Если операция чтения не удалась, возвращается `FALSE`:

```
$file_contents = file_get_contents('C:\xampp\readme_en.txt');
```

- Запись строки в файл — функция `file_put_contents`:

```
file_put_contents(<путь к файлу>, <записываемая строка>[, <флаги>])
```

Если файл с указанным *путем* существует, он будет перезаписан (если не указан флаг FILE_APPEND).

Флаги указывают дополнительные параметры записи:

- FILE_APPEND — если файл с указанным *путем* существует, дописать *строку* в его конец, а не перезаписывать;
- LOCK_EX — заблокировать файл на момент записи, сделав его недоступным для других программ.

Флаги можно объединять с помощью оператора побитового ИЛИ |.

Функция возвращает количество записанных в файл байтов или FALSE в случае ошибки записи.

Пример:

```
file_put_contents('d:\sample.txt', 'PHP');
file_put_contents('d:\sample.txt', ' и MySQL', FILE_APPEND);
```

Полезно знать...

PHP поддерживает много других функций для работы с файлами, позволяющих получать и устанавливать права на доступ к файлам, работать с жесткими и символическими ссылками, производить побайтовые чтение и запись двоичных файлов и др. Эти функции используются относительно редко.

18.3. Работа с папками

18.3.1. Получение сведений о папке

- Проверка существования папки с заданным *путем* — функция file_exists(<*путь к папке*>):

```
echo file_exists('C:\xampp\htdocs');              // TRUE
echo file_exists('C:\xampp\special_programs');    // FALSE
```

- Проверка, является ли папка собственно папкой, — функция is_dir(<*путь к папке*>):

```
echo is_dir('C:\xampp\htdocs');                   // TRUE
// C:\xampp\readme_en.txt - это файл, а не папка
echo is_dir('C:\xampp\readme_en.txt');            // FALSE
```

18.3.2. Манипуляции с папками

- Создание папки — функция mkdir:

```
mkdir(<путь к создаваемой папке>[,
      <права доступа к создаваемой папке>[,
      <рекурсивное создание папок?>]])
```

Права на доступ к папке имеет смысл устанавливать лишь в очень специфических случаях. Если они не указаны, созданная папка получит максимально широкие права, обозначаемые восьмеричным числом 0777. В системе Windows этот параметр игнорируется.

Если третьим параметром передать значение FALSE или вообще не указывать его, функция не будет создавать вложенные папки, указанные в *пути*. Чтобы она создала вложенные папки, следует передать третьим параметром значение TRUE.

В случае успешного создания папки возвращается TRUE, в случае неуспеха — FALSE.

Примеры:

```
// Создаем на диске D: папку htdocs
mkdir('D:\htdocs');
// Создаем в папке D:\htdocs вложенные друг в друга папки
// modules\controllers, для чего указываем третьим параметром TRUE
mkdir('D:\htdocs\modules\controllers', 0777, TRUE);
```

- Переименование или перемещение папки — функция rename:

```
rename(<исходный путь к папке>, <новый путь к папке>)
```

Пример переименования папки controllers, хранящейся в папке D:\htdocs\modules, в models:

```
rename('D:\htdocs\modules\controllers', 'D:\htdocs\modules\models');
```

Пример перемещения папки models в «корень» диска D::

```
rename('D:\htdocs\modules\models', 'D:\models');
```

- Удаление папки — функция rmdir(<*путь к удаляемой папке*>). Возвращает TRUE, если папка была успешно удалена, и FALSE — в противном случае:

```
rmdir('D:\models');
rmdir('D:\htdocs\modules');
rmdir('D:\htdocs');
```

> ***Удалять можно только пустые папки!***
>
> Попытка удаления папки, содержащей файлы и (или) другие папки, завершится неудачей.

Полезно знать...

PHP поддерживает ряд других функций для работы с папками, в частности, позволяющих получить список файлов и папок, хранящихся в указанной папке.

18.4. Получение сведений о дисках

- Объем свободного пространства на диске с указанным *обозначением* — функция disk_free_space(<*обозначение диска*>). Буквенное *обозначение диска* указывается в виде строки и должно завершаться двоеточием.

Объем свободного пространства возвращается в байтах. Если его получить не удалось, возвращается FALSE.

Пример:

```
echo disk_free_space('c:');                    // 182959235072
```

Также можно использовать функцию diskfreespace, работающую аналогично.

- Общий объем диска — функция disk_total_space(<*обозначение диска*>). Буквенное *обозначение диска* указывается в виде строки и должно завершаться двоеточием.

  Объем диска возвращается в байтах. Если его получить не удалось, возвращается FALSE.

  Пример:

```
echo disk_total_space('c:');                   // 238463488000
```

18.5. Сохранение файлов, отправленных из веб-формы

> Для успешной отправки файла серверному приложению необходимо указать у веб-формы:
>
> - метод отправки данных POST (отправка файлов методом GET не поддерживается);
> - метод кодирования данных multipart/form-data (при использовании других методов файлы не будут отправлены).

Пример HTML-кода, создающего веб-форму с полем выбора файла:

```
<form method="post" enctype="multipart/form-data">
    Выберите файл: <input type="file" name="picture">
</form>
```

Поле выбора файла отправляет серверному приложению POST-параметр, чьим значением является содержимое выбранного файла.

В веб-форму можно поместить скрытое поле с наименованием MAX_FILE_SIZE, а в качестве значения указать максимально допустимый размер отправляемого файла в байтах (при попытке отправить файл большего размера возникнет ошибка):

```
<form method="post" enctype="multipart/form-data">
    <input type="hidden" name="MAX_FILE_SIZE" value="2097152" />
    Выберите файл: <input type="file" name="picture">
</form>
```

> PHP временно сохраняет полученные файлы в особой папке, из которой их следует переместить на место постоянного хранения. Если этого не сделать, полученные файлы будут удалены, как только посетитель, отправивший файлы, перейдет на другую страницу.

Полученные файлы можно получить из ассоциативного массива, хранящегося в суперглобальной переменной `$_FILES`, и аналогичного таковым из переменных `$_GET` и `$_POST`. Пример получения файла, отправленного из веб-формы, HTML-код которого был приведен ранее:

```
$picture_file = $_FILES['picture'];
```

Каждый элемент массива `$_FILES` представляет собой вложенный ассоциативный массив со следующими элементами:

- `name` — оригинальное имя файла, под которым он хранился на компьютере посетителя, без пути;
- `type` — MIME-тип файла, если веб-обозреватель предоставил его;
- `size` — размер в байтах;
- `tmp_name` — имя, под которым файл был временно сохранен в папке полученных файлов;
- `error` — числовой код, обозначающий успех или неудачу при получении файла:
  - `UPLOAD_ERR_OK` — файл успешно получен;
  - `UPLOAD_ERR_INI_SIZE` — размер файла превысил максимально допустимый, записанный в конфигурации PHP;
  - `UPLOAD_ERR_FORM_SIZE` — размер файла превысил максимально допустимый, записанный в скрытом поле `MAX_FILE_SIZE` веб-формы;
  - `UPLOAD_ERR_PARTIAL` — файл получен частично (например, в результате прерывания загрузки);
  - `UPLOAD_ERR_NO_FILE` — файл вообще не был получен (обычно вследствие того, что посетитель не указал его в веб-форме);
  - `UPLOAD_ERR_NO_TMP_DIR` — отсутствует папка для временного сохранения полученных файлов;
  - `UPLOAD_ERR_CANT_WRITE` — не удалось сохранить полученный файл (например, исчерпано свободное место на диске).

Если файл вообще *не* был отправлен, соответствующий элемент в массиве `$_FILES` все равно будет присутствовать. В таком случае элементы `name`, `type` и `tmp_name` вложенного массива будут хранить пустые строки, элемент `size` — число `0`, а элемент `error` — числовой код `UPLOAD_ERR_NO_FILE`. По этим признакам можно проверить, был ли файл вообще отправлен:

```
if ($_FILES['picture']['error'] == UPLOAD_ERR_NO_FILE)
    // Файл не был отправлен
```

Перемещение полученного файла из временной папки по месту постоянного хранения с указанным *путем* выполняет функция `move_uploaded_file`:

```
move_uploaded_file(<временное имя файла>,
                   <путь до места постоянного хранения>)
```

Временное имя файла можно получить из элемента `tmp_name` ассоциативного массива, хранящего сведения о файле.

Функция возвращает `TRUE` в случае успешного перемещения файла и `FALSE`, если операцию не удалось выполнить.

Пример перемещения полученного из веб-формы файла по месту постоянного хранения под изначальным именем:

```
if ($_FILES['picture']['error'] == UPLOAD_ERR_OK) {
    $dest_path = 'c:\xampp\htdocs\images\\' . $_FILES['picture']['name'];
    move_uploaded_file($_FILES['picture']['tmp_name'], $dest_path);
}
```

Если серверному приложению нужно отправить сразу несколько файлов, укажите после наименования поля выбора файлов пустые квадратные скобки:

```
<form method="post" enctype="multipart/form-data">
    Выберите файл: <input type="file" name="pictures[]" multiple>
</form>
```

После этого каждый элемент ассоциативного массива, представляющего файл: `name`, `type`, `size`, `tmp_name` и `error` — будет хранить индексированный массив с соответствующими сведениями о каждом из полученных файлов:

```
$count = count($_FILES['pictures']['error']);
for ($i = 0; $i < $count; $i++) {
    if ($_FILES['picture']['error'][$i] == UPLOAD_ERR_OK) {
        $dest_path = 'c:\xampp\htdocs\images\\' .
            $_FILES['picture']['name'][$i];
        move_uploaded_file($_FILES['picture']['tmp_name'][$i],
            $dest_path);
    }
}
```

К сожалению, в именах сохраняемых файлов все пробелы и символы, выходящие за пределы кодировки ASCII, преобразуются в нечитаемые последовательности. Эта ошибка существует с самых первых версий PHP и до сих пор не исправлена.

Поэтому рекомендуется сохранять файлы под именами, включающими только символы из кодировки ASCII и не содержащими пробелов. Наиболее часто такие имена генерируются искусственно по какому-либо алгоритму.

Полезно знать...

Максимальный размер получаемого файла, путь к папке временного хранения и другие параметры, касающиеся отправки файлов, хранятся в настройках PHP. Наиболее полезные из этих настроек описаны на *уроке 25*.

18.6. Упражнение. Реализуем добавление, правку и удаление изображений

Гиперссылку на страницу добавления изображения поместим на странице списка изображений, опубликованных выбранным пользователем. Интернет-адрес в этой гиперссылке будет иметь формат: **/users/<*имя пользователя*>/pictures/add/**, причем пользователь с записанным в интернет-адресе *именем* станет публикатором этого изображения.

После добавления изображения будет выполнено перенаправление на страницу с изображениями, опубликованными тем же пользователем, — чтобы пользователь сразу смог увидеть добавленную им картинку.

Файлы с картинками будем сохранять под именами формата: *<год><номер месяца><число><часы><минуты>*[_*<порядковый номер>*] с сохранением оригинальных расширений. Значения, составляющие имя, будут браться из даты и времени сохранения файла, а если в один момент времени сохраняется сразу несколько файлов, к именам будут добавляться последовательно увеличивающиеся *порядковые номера.*

Файлы сайта фотогалереи и файл photogallery.sql с дампом базы данных `photogallery` хранятся в папке 17\17.6\s сопровождающего книгу электронного архива (см. *приложение 3*).

1. Создадим константу `Settings\IMAGE_FILE_PATH`, хранящую путь к папке с файлами картинок относительно корневой папки сайта.

   Откроем модуль с настройками modules\settings.php и добавим объявление этой константы (здесь и далее дополнения и правки в написанном ранее коде выделены полужирным шрифтом):

   ```
   namespace Settings {
       . . .
       const IMAGE_FILE_PATH = 'images\\';
   }
   ```

   Как и ранее, вынесем все функции, выполняющие типовые и низкоуровневые операции, в модуль modules\helpers.php.

2. Функция `Helpers\get_filename` станет генерировать и возвращать имя, под которым будет сохранен отправленный посетителем файл:

   `get_filename(`*<массив сведений об отправленном файле>*`)`

   Массив сведений о файле берется из ассоциативного массива `$_FILES`.

   Откроем модуль modules\helpers.php в текстовом редакторе и добавим объявление этой функции:

   ```
   namespace Helpers {
       . . .
       function get_filename(array $file): string {
           global $base_path;
           $image_file_path = $base_path . \Settings\IMAGE_FILE_PATH;
   ```

```
        $ext = pathinfo($file['name'], PATHINFO_EXTENSION);
        $name = strftime('%Y%m%d%H%M');
        $postfix = '';
        $number = 0;
        while (file_exists($image_file_path . $name . $postfix .
            '.' . $ext))
            $postfix = '_' . ++$number;
        return $name . $postfix . '.' . $ext;
    }
}
```

Из полученного массива сведений о выгруженном файле извлекаем изначальное имя файла, а из него — расширение. Формируем имя из составных частей текущих даты и времени. Проверяем, существует ли такой файл в папке картинок, и, если существует, добавляем к имени последовательно увеличивающиеся порядковые номера, пока PHP не сообщит, что очередное имя файла «свободно». Это имя и возвращаем в качестве результата.

3. Функция `\Helpers\save_file` сохранит отправленный посетителем файл и вернет в качестве результата его имя:

   `save_file(<`*массив сведений об отправленном файле*`>)`

   Функция `Helpers\delete_file(<`*имя файла*`>)` удалит файл с заданным *именем*.

   Добавим в модуль modules\helpers.php объявление обеих функций:

```
namespace Helpers {
    . . .
    function save_file(array $file): string {
        global $base_path;
        $image_file_path = $base_path . \Settings\IMAGE_FILE_PATH;
        $filename = get_filename($file);
        move_uploaded_file($file['tmp_name'], $image_file_path .
            $filename);
        return $filename;
    }

    function delete_file(string $filename) {
        global $base_path;
        $file_path = $base_path . \Settings\IMAGE_FILE_PATH .
            $filename;
        if (file_exists($file_path))
            unlink($file_path);
    }
}
```

4. При добавлении изображения следует сохранить полученный файл с этим изображением в папке картинок images. При удалении изображения соответствующий файл надо удалить, поскольку он более не нужен. А при правке изображе-

ния, если посетитель указал другой файл с изображением, старый файл следует удалить, а новый — сохранить. Код, выполняющий все эти действия, добавим в модель Models\Picture.

Откроем модуль modules\models\Picture.php в текстовом редакторе и добавим необходимый код:

```
class Picture extends \Models\Model {
    . . .
    protected function before_insert(&$fields) {
        $filename = \Helpers\save_file($_FILES['picture']);
        $fields['filename'] = $filename;
    }

    protected function before_update(&$fields, $value,
        $key_field = 'id') {
        if ($_FILES['picture']['error'] != UPLOAD_ERR_NO_FILE) {
            $this->before_delete($value, $key_field);
            $this->before_insert($fields);
        }
    }

    protected function before_delete($value, $key_field = 'id') {
        $rec = $this->get_or_404($value, $key_field, 'filename');
        \Helpers\delete_file($rec['filename']);
    }
}
```

На *уроке 15* мы объявили в базовом классе модели Models\Model методы before_insert, before_update и before_delete, вызываемые перед, соответственно, добавлением, правкой и удалением записи и предназначенные для выполнения дополнительных действий. В модели изображения мы перекрыли их, записав код, который в нужные моменты времени сохраняет, заменяет или удаляет файл с изображением.

Метод before_insert сохраняет полученный файл и заносит его имя в поле filename таблицы pictures. Метод before_delete ищет в таблице pictures изображение по заданному номеру, извлекает имя файла и удаляет его. Метод before_update проверяет, отправил ли посетитель новый файл, и, если то так, удаляет старый файл и сохраняет новый, вызывая для этого методы before_delete и before_insert.

5. Во вновь созданном классе формы Forms\Picture будем выполнять дополнительные проверки: отправил ли посетитель файл, нужного ли он формата и не слишком ли велик по размеру. Здесь возникнет сложность: при добавлении изображения файл указать необходимо, а при правке его задавать необязательно. Поэтому мы напишем универсальный класс формы, учитывающий эту особенность.

Создадим модуль modules\forms\picture.php, в который запишем код класса формы `Forms\Picture` из листинга 18.1.

Листинг 18.1. Модуль modules\forms\picture.php

```
<?php
namespace Forms;
class Picture extends \Forms\Form {
    protected const FIELDS = [
        'title' => ['type' => 'string'],
        'description' => ['type' => 'string', 'optional' => TRUE],
        'category' => ['type' => 'integer']
    ];

    protected const IS_PICTURE_OPTIONAL = FALSE;

    private const EXTENSIONS = ['gif', 'jpg', 'jpeg', 'jpe', 'png',
        'svg'];

    protected static function after_normalize_data(&$data, &$errors) {
        $file = $_FILES['picture'];
        $error = $file['error'];
        if ($error == UPLOAD_ERR_NO_FILE) {
            if (!static::IS_PICTURE_OPTIONAL)
                $errors['picture'] = 'Укажите файл с изображением';
        } else if (!in_array(pathinfo($file['name'],
            PATHINFO_EXTENSION), self::EXTENSIONS))
            $errors['picture'] = 'Укажите файл с изображением ' .
                'в формате GIF, JPEG, PNG или SVG';
        else if ($error == UPLOAD_ERR_OK) {}
        else if ($error == UPLOAD_ERR_INI_SIZE ||
            $error == UPLOAD_ERR_FORM_SIZE)
            $errors['picture'] = 'Укажите файл размером не ' .
                'более 2 Мб';
        else
            $errors['picture'] = 'Файл не был отправлен';
    }
}
```

Мы объявили в классе защищенную константу `IS_PICTURE_OPTIONAL` со значением `FALSE`, сообщающим о том, что посетитель обязан указать файл с изображением. Позже, создавая форму, не требующую обязательного указания файла, мы объявим производный класс, в котором дадим этой константе значение `TRUE`.

На *уроке 15*, объявляя базовый класс формы, мы создали в нем метод `after_normalize_data`, вызываемый после формирования массива с нормализованными данными. Перекрыв этот метод в производном классе, мы зададим дополнительные проверки отправленного файла:

- если файл не отправлен, и его отправка обязательна, сообщаем, чтобы посетитель указал файл;
- если отправленный файл имеет недопустимое расширение (допустимые расширения приведены в массиве, присвоенном закрытой константе `EXTENSIONS`), сигнализируем об этом;
- если при выгрузке файла не возникло ошибок, ничего не делаем (для этого мы указали после соответствующей конструкции `else if` пустой блок);
- если файл слишком велик, сообщаем об этом;
- во всех остальных случаях сообщаем, что файл не был получен.

6. Откроем модуль маршрутизатора modules\router.php и запишем код, распознающий интернет-адрес формата: **/users/<*имя пользователя*>/pictures/add/** и вызывающий пока не объявленный метод `add` контроллера `Controllers\Images`:

```
. . .
} else if (preg_match('/^(\d+)$/', $request_path, $result) === 1) {
    . . .
} else if (preg_match('/^users\/(\w+)\/pictures\/add$/',
    $request_path, $result) == 1) {
    $ctr = new \Controllers\Images();
    $ctr->add($result[1]);
} else if (preg_match('/^(\d+)\/comments\/(\d+)\/edit$/',
. . .
```

7. Откроем модуль modules\controllers\images.php, хранящий код контроллера `Controllers\Images`, и объявим в нем метод `add`:

```
class Images extends BaseController {
    . . .
    function add(string $username) {
        if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            $picture_form =
                \Forms\Picture::get_normalized_data($_POST);
            if (!isset($picture_form['__errors'])) {
                $picture_form =
                    \Forms\Picture::get_prepared_data($picture_form);
                $users = new \Models\User();
                $user = $users->get_or_404($username, 'name', 'id');
                $picture_form['user'] = $user['id'];
                $pictures = new \Models\Picture();
                $pictures->insert($picture_form);
                \Helpers\redirect('/users/' . $username);
            }
        } else
            $picture_form =
                \Forms\Picture::get_initial_data();
```

```
        $categories = new \Models\Category();
        $categories->select();
        $ctx = ['site_title' => 'Добавление изображения',
            'username' => $username, 'form' => $picture_form,
            'categories' => $categories];
        $this->render('picture_add', $ctx);
    }
}
```

При выводе страницы добавления изображения не забываем сформировать перечень категорий — чтобы вывести в веб-форме раскрывающийся список категорий. А при сохранении изображения не забываем занести в поле `user` номер пользователя, чье имя присутствует в текущем интернет-адресе.

После добавления изображения будет выполнено перенаправление на первую часть неотфильтрованного списка. Так посетитель гарантированно увидит только что добавленное им изображение.

8. HTML-код веб-формы добавления изображения вынесем в отдельный шаблон-фрагмент — чтобы потом использовать его на странице правки изображения.

   Создадим модуль modules\templates__picture_form.inc.php, в который запишем код шаблона-фрагмента веб-формы изображения (листинг 18.2).

Листинг 18.2. Модуль modules\templates__picture_form.inc.php

```
<form class="bigform" method="post" enctype="multipart/form-data">
    <label for="picture_category">Категория</label>
    <select id="picture_category" name="category">
        <?php foreach ($categories as $category) { ?>
            <option value="<?php echo $category['id'] ?>"
            <?php if ($form['category'] == $category['id']) { ?>
                selected<?php } ?>>
                <?php echo $category['name'] ?>
            </option>
        <?php } ?>
    </select>
    <label for="picture_title">Название</label>
    <input type="text" id="picture_title" name="title"
    value="<?php echo $form['title'] ?>">
    <?php \Helpers\show_errors('title', $form) ?>
    <label for="picture_description">Описание</label>
    <textarea id="picture_description" name="description">
    <?php echo $form['description'] ?></textarea>
    <label for="picture_file">Файл с изображением</label>
    <input type="file" id="picture_file" name="picture">
    <?php \Helpers\show_errors('picture', $form) ?>
    <input type="submit" value="Отправить">
</form>
```

Не забываем указать у формы метод кодирования данных `multipart/form-data`, иначе файл с изображением не будет отправлен.

9. Создадим модуль modules\templates\picture_add.php и запишем в него код шаблона страницы для добавления изображения (листинг 18.3).

Листинг 18.3. Модуль modules\templates\picture_add.php

```
<?php require \Helpers\get_fragment_path('__header') ?>
<h2>Добавление изображения</h2>
<?php require \Helpers\get_fragment_path('__picture_form') ?>
<?php $ret = '/users/' . $username .
    \Helpers\get_GET_params(['page', 'filter']) ?>
<p><a href="<?php echo $ret ?>">Назад</a></p>
<?php require \Helpers\get_fragment_path('__footer') ?>
```

10. Откроем модуль modules\templates\by_user.php, хранящий шаблон страницы пользователя, и добавим в него код, создающий гиперссылку на страницу добавления изображения:

```
. . .
<h2><?php echo $user['name'] ?></h2>
<p><a href="<?php echo '/users/' . $user['name'] .
    '/pictures/add' . $gets ?>">Добавить изображение</a></p>
<table id="gallery">
    . . .
</table>
. . .
```

В созданной ранее переменной `$gets` хранится строка с набором GET-параметров.

Перейдем на страницу, например, пользователя `basil`, набрав интернет-адрес: **http://localhost/users/basil/**. В папке 18\!sources найдем графические файлы Октябрьский мотив.jpg и Сумрачный пейзаж.jpg и добавим их в фотогалерею, указав категории «Архитектура» и «Пейзаж» соответственно, названия, совпадающие с именами файлов, и произвольные описания.

Сделайте сами

Реализуйте правку и удаление изображений:

- переход на страницу правки изображения будет выполняться при обращении по интернет-адресу формата: **/users/<*имя пользователя*>/pictures/<*номер изображения*>/edit/**, а переход на страницу удаления изображения — при обращении по адресу: **/users/<*имя пользователя*>/pictures/<*номер изображения*>/delete/**;
- для правки изображения объявите класс формы `Forms\Picture2`. В нем переопределите константу `IS_PICTURE_OPTIONAL`, дав ей значение `TRUE`, — поскольку ука-

зывать файл с изображением в этой форме необязательно (только если нужно заменить само изображение);

- в контроллере `Controllers\Images` объявите методы `edit` и `delete`, выполняющие правку и удаление выбранного изображения.
- напишите шаблоны picture_edit.php и picture_delete.php, создающие страницы, соответственно, правки и удаления изображения. На странице удаления изображения выведите название изображения, имя опубликовавшего его пользователя, дату и время его публикации;
- на странице пользователя, в таблице изображений, выведите две дополнительных колонки, в которых поместите гиперссылки на страницы правки и удаления изображения.

Урок 19
Работа с графикой

PHP предоставляет весьма мощные средства для создания и правки растровых изображений.

> ***Мы будем рассматривать работу***
> ***только с файлами форматов GIF, JPEG и PNG.***
> То есть растровых форматов, применяемых в Интернете.

19.1. Создание и открытие изображений

19.1.1. Создание нового изображения

Новое пустое полноцветное изображение создается вызовом функции `imagecreatetruecolor`:

```
imagecreatetruecolor(<ширина>, <высота>)
```

Ширина и *высота* создаваемого изображения указываются в пикселах.

В качестве результата возвращается идентификатор созданного изображения или `FALSE`, если создать изображение почему-то не получилось.

Пример:

```
$image = imagecreatetruecolor(400, 300);
```

Новое пустое изображение имеет черный цвет. Закрасить его другим цветом, например белым, можно, вызвав функцию заливки `imagefill` (будет рассмотрена в *разд. 19.2.8*):

```
// Создаем с помощью функции imagecolorallocate (также будет рассмотрена
// далее) белый цвет
$white = imagecolorallocate($image, 255, 255, 255);
// Закрашиваем созданное путое изображение белым цветом
imagefill($image, 0, 0, $white);
```

19.1.2. Открытие существующего изображения

Для открытия изображения, хранящегося в уже существующем файле с указанным *путем*, служат три функции, имеющие одинаковый формат вызова:

```
<функция для открытия изображения>(<путь к файлу>)
```

Каждая из *функций* открывает файл строго определенного формата:

- GIF — `imagecreatefromgif`:
- JPEG — `imagecreatefromjpeg`;
- PNG — `imagecreatefrompng`.

В качестве результата возвращается идентификатор открытого изображения или `FALSE`, если открыть файл почему-то не получилось.

19.2. Рисование фигур

> Координаты рисуемых фигур измеряются в пикселах и отсчитываются от левого верхнего угла изображения. Горизонтальная ось координат направлена вправо, вертикальная — вниз.

19.2.1. Создание цветов для фигур

- Создание цвета на основе указанных составляющих: *R* (красный), *G* (зеленый) и *B* (синий) — функция `imagecolorallocate`:

```
imagecolorallocate(<идентификатор изображения>, <R>, <G>, <B>)
```

Составляющие цвета *R*, *G* и *B* указываются в виде чисел от `0` до `255`.

Возвращается целочисленный идентификатор созданного цвета или `FALSE`, если цвет создать не удалось.

Пример:

```
$red = imagecolorallocate($image, 255, 0, 0);
```

- Создание полупрозрачного цвета — функция `imagecolorallocatealpha`:

```
imagecolorallocatealpha(<идентификатор изображения>, <R>, <G>, <B>,
                        <степень полупрозрачности>)
```

Степень полупрозрачности задается в виде числа от `0` (полная непрозрачность) до `127` (полная прозрачность). В остальном функция аналогична описанной ранее функции `imagecolorallocate`.

Пример:

```
$stgreen = imagecolorallocatealpha($image, 0, 255, 0, 63);
```

- Превращение указанного цвета в прозрачный — функция `imagecolortransparent`:

```
imagecolortransparent(<идентификатор изображения>,
                      <идентификатор цвета>)
```

В качестве примера создадим новое пустое изображение и сделаем черный фон, который оно имеет по умолчанию, прозрачным:

```
$image = imagecreatetruecolor(400, 300);
$black = imagecolorallocate($image, 0, 0, 0);
imagecolortransparent($image, $black);
```

- Уничтожение созданного ранее *цвета* — функция `imagecolordeallocate`:

```
imagecolordeallocate(<идентификатор изображения>,
                     <идентификатор цвета>)
```

Возвращается `TRUE` в случае успешного уничтожения цвета и `FALSE` — в противном случае.

Каждый из созданных функциями `imagecolorallocate` и `imagecolorallocatealpha` цветов отнимает системные ресурсы. Поэтому, как только цвет станет ненужным, его рекомендуется уничтожить.

Примеры:

```
imagecolordeallocate($image, $red);
imagecolordeallocate($image, $stgreen);
```

> Все функции рисования фигур, рассмотренные далее, возвращают `TRUE` при успешном выполнении операции и `FALSE` — в противном случае. Обычно возвращаемый этими функциями результат игнорируется.

19.2.2. Рисование точек

Рисование точки с координатами [*X*, *Y*] созданным ранее *цветом* выполняет функция `imagesetpixel`:

```
imagesetpixel(<идентификатор изображения>, <X>, <Y>,
              <идентификатор цвета>)
```

Пример рисования черного квадрата с белой точкой посередине:

```
$img = imagecreatetruecolor(100, 100);
$white = imagecolorallocate($image, 255, 255, 255);
imagesetpixel($img, 50, 50, $white);
// Сохраняем изображение в формате JPEG по пути c:\sample.jpg
imagejpeg($img, 'c:\sample.jpg');
// Удаляем не нужное более изображение из памяти
imagedestroy($img);
```

19.2.3. Рисование линий

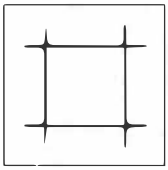
- Рисование сплошной линии — функция `imageline`:

```
imageline(<идентификатор изображения>,
          <координата X начала>, <координата Y начала>,
          <координата X конца>, <координата Y конца>,
          <идентификатор цвета>)
```

Пример:

```
$img = imagecreatetruecolor(80, 80);
$white = imagecolorallocate($img, 255, 255, 255);
imagefill($img, 0, 0, $white);
$black = imagecolorallocate($img, 0, 0, 0);
imageline($img, 10, 20, 70, 20, $black);
```

Результат:



```
imageline($img, 20, 10, 20, 70, $black);
imageline($img, 10, 60, 70, 60, $black);
imageline($img, 60, 10, 60, 70, $black);
imagejpeg($img, 'c:\sample.jpg');
imagedestroy($img);
```

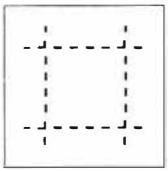
- Рисование пунктирной линии — функция `imagedashedline`:

```
imagedashedline(<идентификатор изображения>,
                <координата X начала>, <координата Y начала>,
                <координата X конца>, <координата Y конца>,
                <идентификатор цвета>)
```

Пример:

```
$img = imagecreatetruecolor(80, 80);
$white = imagecolorallocate($img, 255, 255, 255);
imagefill($img, 0, 0, $white);
$black = imagecolorallocate($img, 0, 0, 0);
imagedashedline($img, 10, 20, 70, 20, $black);
imagedashedline($img, 20, 10, 20, 70, $black);
imagedashedline($img, 10, 60, 70, 60, $black);
imagedashedline($img, 60, 10, 60, 70, $black);
imagejpeg($img, 'c:\sample.jpg');
imagedestroy($img);
```

Результат:



19.2.4. Рисование прямоугольников

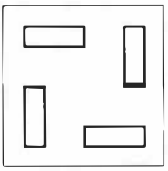
- Рисование прямоугольника — функция `imagerectangle`:

```
imagerectangle(<идентификатор изображения>,
               <координата X левого верхнего угла>,
               <координата Y левого верхнего угла>,
               <координата X правого нижнего угла>,
               <координата Y правого нижнего угла>,
               <идентификатор цвета>)
```

Пример:

```
$img = imagecreatetruecolor(80, 80);
$white = imagecolorallocate($img, 255, 255, 255);
imagefill($img, 0, 0, $white);
$black = imagecolorallocate($img, 0, 0, 0);
imagerectangle($img, 10, 10, 40, 20, $black);
imagerectangle($img, 10, 40, 20, 70, $black);
imagerectangle($img, 40, 60, 70, 70, $black);
imagerectangle($img, 60, 10, 70, 40, $black);
imagejpeg($img, 'c:\sample.jpg');
imagedestroy($img);
```

Результат:



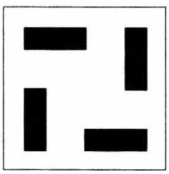
- Рисование закрашенного прямоугольника — функция `imagefilledrectangle`:

```
imagefilledrectangle(<идентификатор изображения>,
                     <координата X левого верхнего угла>,
                     <координата Y левого верхнего угла>,
                     <координата X правого нижнего угла>,
                     <координата Y правого нижнего угла>,
                     <идентификатор цвета>)
```

Пример:

```
$img = imagecreatetruecolor(80, 80);
$white = imagecolorallocate($img, 255, 255, 255);
imagefill($img, 0, 0, $white);
$black = imagecolorallocate($img, 0, 0, 0);
imagefilledrectangle($img, 10, 10, 40, 20, $black);
imagefilledrectangle($img, 10, 40, 20, 70, $black);
imagefilledrectangle($img, 40, 60, 70, 70, $black);
imagefilledrectangle($img, 60, 10, 70, 40, $black);
imagejpeg($img, 'c:\sample.jpg');
imagedestroy($img);
```

Результат:



19.2.5. Рисование эллипсов

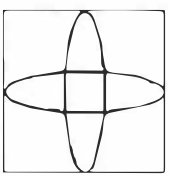
- Рисование эллипса — функция `imageellipse`:

```
imageellipse(<идентификатор изображения>,
             <координата X центра>, <координата Y центра>,
             <ширина>, <высота>, <идентификатор цвета>)
```

Пример:

```
$img = imagecreatetruecolor(80, 80);
$white = imagecolorallocate($img, 255, 255, 255);
imagefill($img, 0, 0, $white);
$black = imagecolorallocate($img, 0, 0, 0);
imageellipse($img, 40, 40, 20, 80, $black);
imageellipse($img, 40, 40, 80, 20, $black);
imagejpeg($img, 'c:\sample.jpg');
imagedestroy($img);
```

Результат:



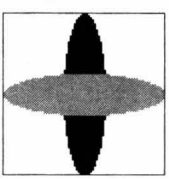
- Рисование закрашенного эллипса — функция `imagefilledellipse`:

```
imagefilledellipse(<идентификатор изображения>,
                   <координата X центра>, <координата Y центра>,
                   <ширина>, <высота>, <идентификатор цвета>)
```

Пример:

```
$img = imagecreatetruecolor(80, 80);
$white = imagecolorallocate($img, 255, 255, 255);
imagefill($img, 0, 0, $white);
$black = imagecolorallocate($img, 0, 0, 0);
imagefilledellipse($img, 40, 40, 20, 80, $black);
```

Результат:



```
$grey = imagecolorallocate($img, 190, 190, 190);
imagefilledellipse($img, 40, 40, 80, 20, $grey);
imagejpeg($img, 'c:\sample.jpg');
imagedestroy($img);
```

19.2.6. Рисование полигонов

- Рисование незамкнутого полигона — функция `imageopenpolygon`:

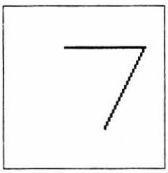
```
imageopenpolygon(<идентификатор изображения>,
                 <массив координат точек>, <количество точек>,
                 <идентификатор цвета>)
```

Количество точек должно быть не менее трех. Первый элемент *массива координат* должен задавать координату `X` первой точки, второй — координату `Y` первой точки, третий — координату `X` второй точки, и т. д.

Пример:

```
$img = imagecreatetruecolor(80, 80);
$white = imagecolorallocate($img, 255, 255, 255);
imagefill($img, 0, 0, $white);
$black = imagecolorallocate($img, 0, 0, 0);
$points = [30, 20, 70, 20, 50, 60];
imageopenpolygon($img, $points, 3, $black);
imagejpeg($img, 'c:\sample.jpg');
imagedestroy($img);
```

Результат:



- Рисование замкнутого полигона с автоматическим замыканием контура — функция `imagepolygon`:

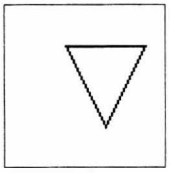
```
imagepolygon(<идентификатор изображения>,
             <массив координат точек>, <количество точек>,
             <идентификатор цвета>)
```

Задаваемые параметры те же, что и у функции `imageopenpolygon`.

Пример:

```
$img = imagecreatetruecolor(80, 80);
$white = imagecolorallocate($img, 255, 255, 255);
imagefill($img, 0, 0, $white);
$black = imagecolorallocate($img, 0, 0, 0);
$points = [30, 20, 70, 20, 50, 60];
imagepolygon($img, $points, 3, $black);
imagejpeg($img, 'c:\sample.jpg');
imagedestroy($img);
```

Результат:



- Рисование закрашенного полигона — функция `imagefilledpolygon`:

```
imagefilledpolygon(<идентификатор изображения>,
                   <массив координат точек>, <количество точек>,
                   <идентификатор цвета>)
```

Задаваемые параметры те же, что и у функции `imageopenpolygon`.

Пример:

```
$img = imagecreatetruecolor(80, 80);
$white = imagecolorallocate($img, 255, 255, 255);
imagefill($img, 0, 0, $white);
$black = imagecolorallocate($img, 0, 0, 0);
$points = [30, 20, 70, 20, 50, 60, 10, 60];
imagefilledpolygon($img, $points, 4, $black);
imagejpeg($img, 'c:\sample.jpg');
imagedestroy($img);
```

Результат:

19.2.7. Рисование кривых

- Рисование дуги — функция `imagearc`:

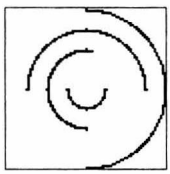
```
imagearc(<идентификатор изображения>,
         <координата X центра>, <координата Y центра>,
         <ширина>, <высота>, <угол начала>, <угол конца>,
         <идентификатор цвета>)
```

Углы начала и *конца* указываются в градусах и отсчитываются от горизонтальной оси по часовой стрелке.

Пример:

```
$img = imagecreatetruecolor(80, 80);
$white = imagecolorallocate($img, 255, 255, 255);
imagefill($img, 0, 0, $white);
$black = imagecolorallocate($img, 0, 0, 0);
imagearc($img, 40, 40, 20, 20, 0, 180, $black);
imagearc($img, 40, 40, 40, 40, 90, 270, $black);
imagearc($img, 40, 40, 60, 60, 180, 0, $black);
imagearc($img, 40, 40, 80, 80, 270, 90, $black);
imagejpeg($img, 'c:\sample.jpg');
imagedestroy($img);
```

Результат:



- Рисование закрашенной дуги — функция `imagefilledarc`:

```
imagefilledarc(<идентификатор изображения>,
               <координата X центра>, <координата Y центра>,
               <ширина>, <высота>, <угол начала>, <угол конца>,
               <идентификатор цвета>, <стиль закраски>)
```

Углы начала и *конца* указываются в градусах и отсчитываются от горизонтальной оси по часовой стрелке. *Стиль закраски* задается в виде одной из приведенных далее констант или их объединения посредством оператора побитового ИЛИ `|`:

- `IMG_ARC_PIE` — провести между концами дуги часть окружности;
- `IMG_ARC_CHORD` — провести между концами дуги прямую линию.

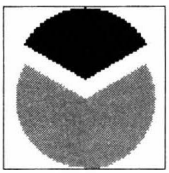
Значения `IMG_ARC_PIE` и `IMG_ARC_CHORD` объединять друг с другом нельзя;

- IMG_ARC_NOFILL — не рисовать заливку;
- IMG_ARC_EDGED — при использовании вместе с IMG_ARC_NOFILL — соединить концы дуги с центром.

Пример:

```
$img = imagecreatetruecolor(80, 80);
$white = imagecolorallocate($img, 255, 255, 255);
imagefill($img, 0, 0, $white);
$grey = imagecolorallocate($img, 190, 190, 190);
imagefilledarc($img, 40, 45, 70, 70, 330, 210, $grey,
    IMG_ARC_PIE);
$black = imagecolorallocate($img, 0, 0, 0);
imagefilledarc($img, 40, 35, 70, 70, 210, 330, $black,
    IMG_ARC_PIE);
imagejpeg($img, 'c:\sample.jpg');
imagedestroy($img);
```

Результат:



19.2.8. Закраска областей изображения

♦ Закраска замкнутой области заданным *цветом* — функция imagefill:

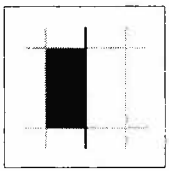
```
imagefill(<идентификатор изображения>,
          <координата X точки, в которой начнется закраска>,
          <координата Y точки, в которой начнется закраска>,
          <идентификатор цвета>)
```

Выполняет закраску области, имеющей тот цвет, что присутствует в *точке, из которой начинается закраска*.

Пример:

```
$img = imagecreatetruecolor(80, 80);
$white = imagecolorallocate($img, 255, 255, 255);
imagefill($img, 0, 0, $white);
$grey = imagecolorallocate($img, 190, 190, 190);
imageline($img, 10, 20, 70, 20, $grey);
imageline($img, 20, 10, 20, 70, $grey);
imageline($img, 10, 60, 70, 60, $grey);
imageline($img, 60, 10, 60, 70, $grey);
$black = imagecolorallocate($img, 0, 0, 0);
imageline($img, 40, 10, 40, 70, $black);
imagefill($img, 30, 40, $black);
imagejpeg($img, 'c:\sample.jpg');
imagedestroy($img);
```

Результат:



♦ Закраска области, ограниченной контуром с заданным *цветом*, — функция imagefilltoborder:

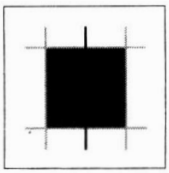
```
imagefilltoborder(<идентификатор изображения>,
                  <координата X точки, в которой начнется закраска>,
```

```
<координата Y точки, в которой начнется закраска>,
<идентификатор цвета ограничивающего контура>,
<идентификатор цвета закраски>)
```

Пример:

```
$img = imagecreatetruecolor(80, 80);
$white = imagecolorallocate($img, 255, 255, 255);
imagefill($img, 0, 0, $white);
$grey = imagecolorallocate($img, 190, 190, 190);
imageline($img, 10, 20, 70, 20, $grey);
imageline($img, 20, 10, 20, 70, $grey);
imageline($img, 10, 60, 70, 60, $grey);
imageline($img, 60, 10, 60, 70, $grey);
$black = imagecolorallocate($img, 0, 0, 0);
imageline($img, 40, 10, 40, 70, $black);
imagefilltoborder($img, 30, 40, $grey, $black);
imagejpeg($img, 'c:\sample.jpg');
imagedestroy($img);
```

Результат:



19.2.9. Задание толщины и стиля линий

- Задание толщины линий — функция `imagesetthickness`:

```
imagesetthickness(<идентификатор изображения>,
                  <толщина линий в пикселах>)
```

Пример:

```
$img = imagecreatetruecolor(80, 80);
$white = imagecolorallocate($img, 255, 255, 255);
imagefill($img, 0, 0, $white);
$black = imagecolorallocate($img, 0, 0, 0);
imageline($img, 10, 20, 70, 20, $black);
imagesetthickness($img, 5);
imageline($img, 10, 40, 70, 40, $black);
imagesetthickness($img, 10);
imageline($img, 10, 60, 70, 60, $black);
imagejpeg($img, 'c:\sample.jpg');
imagedestroy($img);
```

Результат:

- Задание стиля линий — функция `imagesetstyle`:

```
imagesetstyle(<идентификатор изображения>,
              <массив с цветами пикселов>)
```

Массив должен содержать цвета отдельных пикселов, образующих фрагменты, из которых будет нарисована линия. Цвет может быть указан:

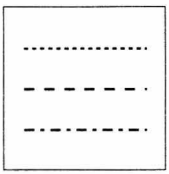
- идентификатором, возвращенным функцией `imagecolorallocate` или `imagecolorallocatealpha`;
- константой `IMG_COLOR_TRANSPARENT`, обозначающей прозрачный цвет.

Для рисования линиями заданного стиля в описанных ранее функциях следует указать цвет, обозначаемый константой `IMG_COLOR_STYLED` или `IMG_COLOR_STYLEDBRUSHED`.

Пример:

```
$img = imagecreatetruecolor(80, 80);
$white = imagecolorallocate($img, 255, 255, 255);
imagefill($img, 0, 0, $white);
$black = imagecolorallocate($img, 0, 0, 0);
$style = [$black, $black, $white, $white];
imagesetstyle($img, $style);
imageline($img, 10, 20, 70, 20, IMG_COLOR_STYLED);
$style = [$black, $black, $black, $black, $black,
    $white, $white, $white, $white, $white];
imagesetstyle($img, $style);
imageline($img, 10, 40, 70, 40, IMG_COLOR_STYLED);
$style = [$black, $black, $black, $black, $black,
    $white, $white, $white, $black, $black, $white,
    $white, $white];
imagesetstyle($img, $style);
imageline($img, 10, 60, 70, 60, IMG_COLOR_STYLED);
imagejpeg($img, 'c:\sample.jpg');
imagedestroy($img);
```

Результат:



19.2.10. Вывод текста

- Получение параметров рамки, которая обрамляет *строку*, выведенную шрифтом формата TrueType, — функция `imagettfbbox`:

```
imagettfbbox(<кегль шрифта>, <угол наклона строки>,
             <путь к файлу шрифта>, <строка>)
```

Кегль шрифта указывается в типографских пунктах. *Угол наклона строки* задается в градусах и отсчитывается от горизонтальной оси: положительные значения — против часовой стрелки, отрицательные — по часовой стрелке.

В качестве результата возвращается индексированный массив с элементами, хранящими:

- координату `X` нижнего левого угла;
- координату `Y` нижнего левого угла;
- координату `X` нижнего правого угла;
- координату `Y` нижнего правого угла;
- координату `X` верхнего правого угла;
- координату `Y` верхнего правого угла;
- координату `X` верхнего левого угла;
- координату `Y` верхнего левого угла.

Пример определения размеров строки, выведенной шрифтом Arial обычного начертания и кегля в 20 пунктов, и строки, выведенной тем же шрифтом с кеглем 80 пунктов с наклоном 45°:

```
$str = 'PHP и MySQL';
$font_file_path = 'c:\windows\fonts\arial.ttf';
$bbox = imagettfbbox(20, 0, $font_file_path, $str);
echo 'Ширина: ', $bbox[2] - $bbox[0];              // Ширина: 175
echo 'Высота: ', $bbox[1] - $bbox[5];              // Высота: 25
$bbox = imagettfbbox(80, 45, $font_file_path, $str);
echo 'Ширина: ', $bbox[2] - $bbox[0];              // Ширина: 483
echo 'Высота: ', $bbox[1] - $bbox[5];              // Высота: 554
```

- Вывод *строки* шрифтом формата TrueType — функция `imagettftext`:

```
imagettftext(<идентификатор изображения>, <кегль шрифта>,
             <угол наклона строки>,
             <координата X левого нижнего угла>,
             <координата Y левого нижнего угла>,
             <идентификатор цвета>, <путь к файлу шрифта>, <строка>)
```

Кегль шрифта указывается в типографских пунктах. *Угол наклона строки* задается в градусах и отсчитывается от горизонтальной оси: положительные значения — против часовой стрелки, отрицательные — по часовой стрелке.

В качестве результата возвращается индексированный массив, аналогичный возвращаемому функцией `imagettfbbox`.

Пример:

```
$img = imagecreatetruecolor(80, 80);
$white = imagecolorallocate($img, 255, 255, 255);
imagefill($img, 0, 0, $white);
$black = imagecolorallocate($img, 0, 0, 0);
$font_file_path = 'c:\windows\fonts\impact.ttf';
imagettftext($img, 16, 10, 22, 30, $black,
    $font_file_path, 'PHP');
imagettftext($img, 16, 10, 10, 70, $black,
    $font_file_path, 'и MySQL');
imagejpeg($img, 'c:\sample.jpg');
imagedestroy($img);
```

Результат:

Полезно знать...

PHP также поддерживает вывод текста шрифтами форматов FreeType 2 и PostScript Type 1.

19.3. Изменение размеров и копирование изображений

19.3.1. Изменение размеров изображения

Изменить размеры *изображения* можно вызовом функции `imagescale`:

```
imagescale(<идентификатор изображения>, <новая ширина>[, <новая высота>])
```

Если *новая высота* не указана или задана любым отрицательным числом, платформа PHP сама подберет новую высоту так, чтобы сохранить пропорции *изображения*.

В процессе работы функция создает новое изображение и возвращает его идентификатор. Если операцию выполнить не удалось, возвращается FALSE.

Пример:

```
$src_img = imagecreatefrompng('d:\home.png');
$dst_img = imagescale($src_img, 80);
imagejpeg($dst_img, 'c:\sample.jpg');
imagedestroy($dst_img);
imagedestroy($src_img);
```

Результат:

Полезно знать...

Также поддерживаются поворот, искривление изображения и применение к нему аффинных преобразований.

19.3.2. Копирование одного изображения в другое

Скопировать исходное изображение (*ИИ*) целиком или частично в целевое (*ЦИ*) с изменением размеров без потери качества можно вызовом функции imagecopyresampled:

```
imagecopyresampled(<идентификатор ЦИ>, <идентификатор ИИ>, <X_ЦИ>, <Y_ЦИ>,
                   <X_ИИ>, <Y_ИИ>, <W_ЦИ>, <H_ЦИ>, <W_ИИ>, <H_ИИ>)
```

Из *ИИ* будет скопирован прямоугольный фрагмент шириной $W_{ИИ}$ и высотой $H_{ИИ}$, левый верхний угол которого находится по координатам [$X_{ИИ}$, $Y_{ИИ}$]. Он будет помещен в *ЦИ* по координатам [$X_{ЦИ}$, $Y_{ЦИ}$] и получит ширину $W_{ЦИ}$ и высоту $H_{ЦИ}$.

Возвращается TRUE при успешном выполнении операции, FALSE — при неуспешном.

Пример:

```
$dst_img = imagecreatetruecolor(80, 80);
$white = imagecolorallocate($dst_img, 255, 255, 255);
imagefill($img, 0, 0, $white);
$src_img = imagecreatefrompng('c:\home.png');
imagecopyresampled($dst_img, $src_img, 40, 40, 0, 0, 40,
    40, 240, 179);
imagecopyresampled($dst_img, $src_img, 0, 0, 0, 90, 40,
    40, 120, 90);
imagejpeg($dst_img, 'c:\sample.jpg');
imagedestroy($dst_img);
imagedestroy($src_img);
```

Результат:

Полезно знать...

Также поддерживаются копирование изображения в градациях серого и быстрое копирование с возможной потерей качества.

19.4. Сохранение изображения

Для сохранения готового *изображения* в *файле* применяются три функции:

- в формате GIF — `imagegif`:

  ```
  imagegif(<идентификатор изображения>, <путь к файлу>)
  ```

- формате JPEG — `imagejpeg`:

  ```
  imagejpeg(<идентификатор изображения>, <путь к файлу>[, <качество>])
  ```

  Качество указывается в виде целого числа от `0` (очень низкое) до `100` (максимальное). Чем выше качество, тем больше размер получаемого файла. Если *качество* не задано или задано любым отрицательным числом, используется значение `75`;

- в формате PNG — `imagepng`:

  ```
  imagepng(<идентификатор изображения>, <путь к файлу>[,
          <степень сжатия>])
  ```

  Степень сжатия указывается в виде целого числа от `0` (отсутствует) до `9` (максимальное). Чем выше степень сжатия, тем меньший размер имеет получившийся файл и тем больше времени уходит на его создание. Если *степень сжатия* не задана или указана в виде отрицательного числа, PHP выбирает эту величину самостоятельно.

19.5. Удаление изображения

> Не нужное более изображение следует удалить, чтобы освободить системные ресурсы.

Удаление изображения выполняет функция `imagedestroy`:

```
imagedestroy(<идентификатор изображения>)
```

Возвращается `TRUE` в случае успешного удаления изображения или `FALSE` — в противном случае.

При удалении изображения также уничтожаются заданные для него цвета. Предварительно удалять их вызовом функции `imagecolordeallocate` необязательно.

19.6. Получение сведений об изображении

- Получение сведений об изображении, хранящемся в файле с указанным *путем*, — функция `getimagesize(<путь к файлу>)`.

  Результат возвращается в виде комбинированного массива, элементы которого имеют следующие индексы и ключи:

  - `0` — ширина изображения в пикселах;
  - `1` — высота изображения в пикселах;

- `2` — обозначение типа изображения в виде значения константы: `IMG_GIF`, `IMG_JPEG` или `IMG_PNG`;
- `3` — строка формата:

  `width="<ширина>" height="<высота>"`

  Эту строку можно вставить в HTML-тег `<img>`;
- `mime` — MIME-тип файла;
- `bits` — глубина цвета в виде количества битов на пиксел;
- `channels` — 3 у изображений RGB, 4 у изображений CMYK.

Пример:

```
$size = getimagesize('c:\xampp\htdocs\images\200804282020.jpg');
echo $size[0], ' x ', $size[1];                    // 480 x 360
echo $size[2] == IMG_JPEG;                         // TRUE
echo $size['mime'];                                // image/jpeg
```

- Получение ширины изображения с указанным *идентификатором* — функция `imagesx`:

  `imagesx(<идентификатор изображения>)`

  Если ширину получить не удалось, возвращается `FALSE`.
- Получение высоты изображения с указанным *идентификатором* — функция `imagesy`:

  `imagesy(<идентификатор изображения>)`

  Если высоту получить не удалось, возвращается `FALSE`.

19.7. Упражнение. Реализуем создание и вывод миниатюр

> *Миниатюра* — уменьшенная копия опубликованного на сайте графического изображения, которая выводится на страницах списков (в то время как на странице просмотра выбранной картинки выводится полное изображение). Файлы миниатюр имеют небольшой размер и быстрее загружаются по сети.

На главной странице, страницах категорий и пользователей будем выводить миниатюры изображений. Каждой миниатюре зададим ширину 100 пикселов, такую высоту, чтобы сохранить изначальные пропорции, и полупрозрачные белые полосы шириной 10 пикселов сверху и снизу — пусть это станет фирменным знаком нашей фотогалереи (рис. 19.1).

Рис. 19.1. Пример миниатюры

Миниатюры будем генерировать на лету, сохраняя их в папке thumbnails корневой папки сайта. Для простоты все миниатюры станем сохранять в формате JPEG.

Файлы сайта фотогалереи и файл photogallery.sql с дампом базы данных photogallery хранятся в папке 18\18.6\s сопровождающего книгу электронного архива (см. *приложение 3*).

1. Создадим в корневой папке сайта папку thumbnails для хранения миниатюр.
2. Создадим две константы: Settings\THUMBNAIL_PATH — с путем к папке миниатюр, вставляемым в интернет-адрес, и Settings\THUMBNAIL_FILE_PATH — с тем же путем, но уже в формате локальной файловой системы.

   Откроем модуль настроек modules\settings.php в текстовом редакторе и добавим в пространство имен Settings обе константы (здесь и далее добавления и правки в написанном ранее коде выделены полужирным шрифтом):

```
namespace Settings {
    . . .
    const THUMBNAIL_PATH = '/thumbnails/';
    const THUMBNAIL_FILE_PATH = 'thumbnails\\';
}
```

3. Объявим функцию Helpers\get_thumbnail(<*имя файла*>), которая вернет интернет-адрес файла миниатюры изображения, хранящегося в файле с заданным *именем*. Если миниатюры еще нет, она будет сгенерирована.

   Откроем модуль modules\helpers.php и добавим объявление этой функции:

```
namespace Helpers {
    . . .
    function get_thumbnail(string $filename) {
        global $base_path;
        $thumb_file_name = pathinfo($filename, PATHINFO_FILENAME) .
            '.jpg';
        $thumb_path = $base_path . \Settings\THUMBNAIL_FILE_PATH .
            $thumb_file_name;
        if (!file_exists($thumb_path)) {
            $file_path = $base_path . \Settings\IMAGE_FILE_PATH .
                $filename;
            $file_ext = pathinfo($filename, PATHINFO_EXTENSION);
            switch ($file_ext) {
                case 'gif':
                    $src_img = imagecreatefromgif($file_path);
                    break;
                case 'jpg':
                case 'jpeg':
                case 'jpe':
                    $src_img = imagecreatefromjpeg($file_path);
                    break;
                case 'png':
                    $src_img = imagecreatefrompng($file_path);
            }
```

```
            $img = imagescale($src_img, 100);
            $stwhite = imagecolorallocatealpha($img, 255, 255, 255,
                32);
            imagefilledrectangle($img, 0, 0, 100, 10, $stwhite);
            $height = imagesy($img);
            imagefilledrectangle($img, 0, $height - 10, 100, $height,
                $stwhite);
            imagejpeg($img, $thumb_path);
            imagedestroy($img);
            imagedestroy($src_img);
        }
        return \Settings\THUMBNAIL_PATH . $thumb_file_name;
    }
}
```

4. Откроем модуль modules\templates\list.php с шаблоном главной страницы и исправим его таким образом, чтобы он выводил миниатюры, сгенерированные функцией `Helpers\get_thumbnail` (удаленный код зачеркнут):

```
. . .
<table id="gallery">
    . . .
            <img src="<?php echo \Settings\IMAGE_PATH .
                $pict['filename'] ?>">
            <img src="<?php echo
                \Helpers\get_thumbnail($pict['filename']) ?>">
    . . .
</table>
. . .
```

5. Внесем аналогичные правки в модули modules\templates\by_cat.php (шаблон страницы категории) и modules\templates\by_user.php (шаблон страницы пользователя).

   Теперь нам нужно позаботиться о том, чтобы при удалении изображения стирался не только файл с самим изображением, но и файл с его миниатюрой.

6. Откроем модуль modules\helpers.php, если уже закрыли его, и добавим функцию `Helpers\delete_thumbnail(<имя файла>)`, удаляющую файл с миниатюрой:

```
namespace Helpers {
    . . .
    function delete_thumbnail(string $filename) {
        global $base_path;
        $thumb_path = $base_path . \Settings\THUMBNAIL_FILE_PATH .
            pathinfo($filename, PATHINFO_FILENAME) . '.jpg';
        if (file_exists($thumb_path))
            unlink($thumb_path);
    }
}
```

7. Откроем модуль modules\models\Picture.php, в котором хранится модель \Models\Picture, и добавим в метод before_delete выражение, удаляющее миниатюру вызовом функции Helpers\delete_thumbnail:

```
class Picture extends \Models\Model {
    . . .
    protected function before_delete($value, $key_field = 'id') {
        $rec = $this->get_or_404($value, $key_field, 'filename');
        \Helpers\delete_file($rec['filename']);
        \Helpers\delete_thumbnail($rec['filename']);
    }
}
```

Последовательно зайдем на главную страницу фотогалереи, страницу какой-либо категории и страницу одного из пользователей, чтобы проверить, выводятся ли миниатюры. После чего откроем папку thumbnails — в ней должны храниться файлы сгенерированных миниатюр.

Урок 20
Cookie и сессии

Часто бывает нужно сохранить какие-либо данные, чтобы их можно было использовать в нескольких веб-приложениях, составляющих сайт. PHP предоставляет для этого два механизма.

20.1. Cookie

> *Cookie* — небольшой фрагмент данных, отправляемый веб-обозревателю и сохраняемый на компьютере посетителя (на стороне клиента).

Cookie создается автоматически, как только веб-приложение записывает в него первое из сохраняемых значений. Каждое из сохраняемых значений должно иметь уникальное имя.

В создаваемый cookie также заносятся текущий домен, путь, по которому находится веб-приложение, и срок хранения. Готовый cookie пересылается веб-обозревателю, который сохраняет его на компьютере клиента.

При последующем обращении к тому же или любому другому веб-приложению, находящемуся в том же домене (или в его поддомене) и папке с тем же путем (или в одной из вложенных папок), сохраненные в cookie данные автоматически пересылаются на сервер в составе клиентского запроса. Веб-приложение извлекает их и использует в работе.

В cookie обычно сохраняются идентификаторы сессий (о них разговор пойдет в *разд. 20.2*), настройки сайта, не являющиеся конфиденциальными (например, обозначение выбранной цветовой темы, порядок сортировки позиций), и др.

> Cookie следует создавать перед выполнением любого вывода — в противном случае мы получим ошибку.

20.1.1. Запись данных в cookie

- Сохранение значения в закодированном cookie — функция `setcookie`:

```
setcookie(<имя значения>[, <значение>[, <срок действия>[, <путь>[,
          <доменное имя>[, <только по HTTPS?>[,
          <cookie будет доступен веб-сценариям?>]]]]]])
```

  Если *значение* не указано, под указанным *именем* будет записана пустая строка.

  Срок действия указывается в виде временнóй отметки PHP. Если его не задать или указать `0`, cookie будет храниться, пока посетитель находится на текущем сайте.

Путь указывается в виде строки. Если его не задать, cookie будет связан с путем, по которому расположено сохранившее его веб-приложение. Если в качестве *пути* указать прямой слеш `'/'`, cookie будет доступен любому веб-приложению сайта.

Домен указывается в виде строки. Если его не задать, cookie будет связан с текущим доменом.

Если шестым параметром передать значение `FALSE` или вообще опустить его, cookie будет пересылаться серверу по протоколам HTTP и HTTPS. Если же задать значение `TRUE`, cookie станет пересылаться только в случае использования протокола HTTPS.

Если седьмым параметром передать `FALSE` или не указать его, cookie может быть обработан веб-сценарием, написанным на языке JavaScript. Если задать `TRUE`, ни один веб-сценарий не сможет получить доступ к cookie.

Последние два параметра введены для повышения безопасности хранения конфиденциальных данных.

В качестве результата возвращается `TRUE`, если cookie был успешно сохранен и отправлен клиенту, и `FALSE` в противном случае.

Примеры:

```
// Сохраняем cookie с параметрами по умолчанию
setcookie('sort', 'updated');
// Сохраняем cookie, действительный до полуночи 1 января 2030 года
// и доступный по любому пути
setcookie('sort', 'updated', strtotime('2030-01-01 00:00:00'), '/');
// Сохраняем cookie с теми же параметрами, но максимально защищенный
setcookie('sort', 'updated', strtotime('2030-01-01 00:00:00'), '/',
    'localhost', TRUE, TRUE);
```

> Функция `setcookie` кодирует сохраняемые значения, поэтому для создания cookie рекомендуется использовать именно ее.

- Сохранение незакодированного cookie — функция `setrawcookie`, аналогичная `setcookie`.

> Функция `setrawcookie` не кодирует сохраняемые значения. Ее следует применять только в тех случаях, если созданные cookie нужно обрабатывать в веб-сценариях.

20.1.2. Получение данных, сохраненных в cookie

Данные, записанные в cookie, можно получить из ассоциативного массива, хранящегося в суперглобальной переменной `$_COOKIE`. Ключи элементов этого массива совпадают с именами сохраненных значений.

Пример:

```
echo $_COOKIE['sort'];                                    // updated
```

20.2. Сессии

Сессия — фрагмент данных, хранящийся на стороне сервера.

Веб-приложение запускает сессию, в результате чего либо создается новая сессия, либо открывается созданная ранее, записывает в него необходимые значения под уникальными именами и (или) считывает занесенные ранее данные.

Сессия вместе с записанными в ней значениями сохраняется, пока посетитель находится на текущем сайте.

Сессии используются для хранения рабочих данных веб-приложений — например, номера пользователя, выполнившего вход на сайт (разграничению доступа к сайту посвящен *урок 21*).

Полезно знать...

- PHP сохраняет сессии вместе с записанными в них данными в файлах в особой папке, путь к которой указывается в настройках исполняющей среды.
- Каждая сессия получает уникальный идентификатор, который пересылается веб-обозревателю в специальном cookie. При запуске сессии PHP извлекает идентификатор из полученного cookie, находит по нему созданную ранее сессию и открывает ее. Все это выполняется автоматически.

20.2.1. Запуск, проверка состояния и использование сессии

Запуск сессии выполняется вызовом функции `session_start()`. Она возвращает `TRUE` в случае успешного запуска и `FALSE`, если возникли проблемы.

Проверить состояние сессии можно вызовом функции `session_status()`. Она возвращает одно из следующих значений:

- `PHP_SESSION_ACTIVE` — подсистема сессий PHP работает, и сессия запущена;
- `PHP_SESSION_NONE` — подсистема сессий PHP работает, но сессия еще не запущена;
- `PHP_SESSION_DISABLED` — подсистема сессий PHP не работает (например, отключена в настройках платформы).

Получить данные, ранее сохраненные в сессии, равно как и записать в нее новые данные, можно, воспользовавшись ассоциативным массивом из суперглобальной переменной `$_SESSION`. Ключи элементов этого массива совпадают с именами сохраненных значений.

Пример:

```
if (session_status() != PHP_SESSION_ACTIVE)
    session_start();
$user_id = $_SESSION['user_id'];
$_SESSION['user_name'] = 'admin';
```

20.2.2. Удаление сессии и сохраненных в ней данных

Удаление сессии выполняется в два этапа:

1. Удаляются данные, сохраненные в сессии.
2. Удаляется сама сессия, для чего достаточно вызвать функцию `session_destroy()`. Функция возвращает `TRUE` в случае успешного удаления и `FALSE` в случае неудачи.

Пример удаления сессии:

```
unset ($_SESSION['logged_on']);
session_destroy();
```

Урок 21
Базовые средства безопасности. Разграничение доступа

Сделаем так, чтобы публиковать изображения и оставлять комментарии в нашей фотогалерее могли лишь зарегистрированные пользователи.

21.1. Реализация разграничения доступа

Принципы разграничения доступа применительно к базам данных были описаны в *разд. 13.1.4*. Разграничение доступа к сайтам реализуется аналогично.

В базе с данными сайта хранится список зарегистрированных пользователей. Сведения о каждом пользователе включают, как минимум, его имя, пароль и, возможно, сведения о привилегиях (например, обычные пользователи могут править и удалять лишь опубликованные ими же изображения, а администратор сайта имеет право делать это с любыми изображениями).

Посетитель вводит свои имя и пароль в веб-форме на странице входа и тем самым выполняет вход на сайт, получая доступ ко всем инструментам для работы с данными сайта. Закончив, он выполняет выход.

Отдельные этапы разграничения доступа реализуются следующим образом:

- вход на сайт:
  - посетитель открывает страницу входа и вводит в веб-форму имя и пароль;
  - веб-приложение проверяет, хранится ли в базе данных пользователь с такими именем и паролем;
  - если такой пользователь есть, его номер (иногда и другие данные — например, сведения о привилегиях) сохраняется в сессии;
- допуск посетителя на страницу, доступную только зарегистрированным пользователям:
  - выполняется проверка, хранится ли в сессии номер пользователя, выполнившего вход (то есть вошел ли пользователь на сайт);
  - если этот номер есть в сессии, пользователь допускается на страницу;
- допуск посетителя на страницу, доступную лишь пользователям, в отношении которых выполняется определенное условие (например, имеет ли пользователь необходимые привилегии):
  - из сессии извлекается номер пользователя, выполнившего вход;

- в базе данных ищется пользователь с этим номером;
- проверяется выполнение условия на основании сведений о пользователе, полученных из базы, и, если условие выполнилось, пользователь допускается на страницу;

♦ выход с сайта — из сессии удаляется сохраненный номер пользователя, после чего удаляется сама сессия.

21.2. Безопасное хранение паролей. Хэши

> *Хэш* — строка установленной длины, сгенерированная на основе заданного значения (например, пароля) и однозначно идентифицирующая это значение. При использовании достаточно сложного алгоритма вычисления хэша, по полученному хэшу невозможно получить исходное значение.

Для повышения безопасности пароли в таблице списка пользователей хранят в виде хэшей. Пароль, введенный посетителем в веб-форму входа, сравнивается с хранящимся в списке хэшем посредством специальной функции.

Для преобразования *пароля* в хэш PHP предлагает функцию `password_hash`:

```
password_hash(<пароль>, <обозначение используемого алгоритма>[,
              <массив с дополнительными параметрами>])
```

Пароль задается в виде строки, *обозначение используемого алгоритма* — в виде одной из следующих констант:

♦ `PASSWORD_DEFAULT` — алгоритм по умолчанию. В настоящее время это bcrypt, выдающий хэш длиной 60 символов, обеспечивающий достаточную стойкость хэша к взлому и весьма быстрый.

Нужно иметь в виду, что в последующих версиях PHP по умолчанию может быть использован другой алгоритм вычисления хэша, выдающий хэш большей длины. Поэтому в списке пользователей под хранение хэша пароля следует предусмотреть поле достаточной длины, наилучший вариант — 255 символов;

♦ `PASSWORD_BCRYPT` — алгоритм bcrypt;

♦ `PASSWORD_ARGON2I` — алгоритм Argon2i, вычисляющий хэш длиной 96 символов, стойкий к взлому, но весьма медленный;

♦ `PASSWORD_ARGON2ID` — алгоритм Argon2id, вычисляющий хэш длиной 97 символов, очень стойкий к взлому, но самый медленный из поддерживаемых.

Дополнительные параметры, указываемые в ассоциативном *массиве*, различаются у разных алгоритмов:

♦ `PASSWORD_BCRYPT`:

- `salt` — соль в виде строки. Должна иметь в длину не менее 22 символов.

> *Соль* — строковое значение, подмешиваемое в вычисляемый хэш и повышающее его стойкость к взлому.

Если соль не указана, PHP при вычислении генерирует ее самостоятельно;

- `cost` — степень сложности алгоритма в виде целого числа. Чем больше число, тем более стойким получается хэш и тем дольше он вычисляется. Если не указана, используется значение `10`;

♦ `PASSWORD_ARGON2I` и `PASSWORD_ARGON2ID`:

- `memory_cost` — максимальный объем памяти, используемый для вычисления хэша, в килобайтах. Если не указан, используется значение константы `PASSWORD_ARGON2_DEFAULT_MEMORY_COST` (65 536 Кбайт или 64 Мбайт);
- `time_cost` — максимальное время, затрачиваемое на вычисление хэша, в секундах. Если не указано, используется значение константы `PASSWORD_ARGON2_DEFAULT_TIME_COST` (4 с);
- `threads` — максимальное количество потоков, используемое для вычисления хэша. Если не указано, используется значение константы `PASSWORD_ARGON2_DEFAULT_THREADS` (1 поток).

Функция `password_hash` возвращает вычисленный хэш или `FALSE`, если вычислить его не удалось.

Примеры:

```
echo password_hash('superuser', PASSWORD_DEFAULT);
          // $2y$10$tjsgpTQ1q0V84FIzBWc/POvNPtylUuv/toJgDGNizbHZ57oo4G4m6
echo password_hash('superuser', PASSWORD_BCRYPT);
          // $2y$10$xK8LHL.LKCXln4z3uEor7.u98Fy5hW7DiZyOWDErqqRMtFknC2Bia
echo password_hash('superuser', PASSWORD_BCRYPT,
    ['salt' => 'abcdefghijklmnoprstuvwxyz', 'cost' => 5]);
          // $2y$05$abcdefghijklmnoprstuvu3WnxxloHXKzuKz0BKidgNX.XoZKwv3W
echo password_hash('superuser', PASSWORD_ARGON2I);
// $argon2i$v=19$m=65536,t=4,p=1$cmx0eHh5Q2psbFVWdU81QQ$0kirHKkgrkFI6ybR↵
// SvpMUWZu/S+D/g74ZI1q2ThAyS4
echo password_hash('superuser', PASSWORD_ARGON2I,
    ['memory_cost' => 16384, 'time_cost' => 2, 'threads' => 2]);
// $argon2i$v=19$m=16384,t=2,p=2$alhaeEpjRS4xdDBhcFpaMA$4+48AoM+eDITVHWC↵
// sXEHoRtVe71LLRmTSChvARvPkwo
echo password_hash('superuser', PASSWORD_ARGON2ID);
// $argon2id$v=19$m=65536,t=4,p=1$NTd6S21iTkJiUmk4b3dibQ$V6x+hRosx/qPDwR↵
// Rrlvi6r5YCJ4WLpWG9BFSI2PglFU
```

> Функция `password_hash` при последовательных вызовах с указанием одних и тех же параметров выдает разные хэши.

Проверить введенный посетителем *пароль* на соответствие *хэшу* позволяет функция `password_verify(<пароль>, <хэш>)`. Хэш может быть вычислен по любому из приведенных ранее алгоритмов:

```
$hash = '$2y$10$xK8LHL.LKCXln4z3uEor7.u98Fy5hW7DiZyOWDErqqRMtFknC2Bia';
echo password_verify('superuser', $hash);          // TRUE
echo password_verify('usersuper', $hash);          // FALSE
```

Полезно знать...

Вообще, PHP предлагает весьма много функций для вычисления хэша. Однако бóльшую их часть применять не рекомендуется, поскольку лежащие в их основе алгоритмы устарели, и генерируемые ими хэши слабо противостоят взлому.

21.3. Упражнение. Реализуем вход и выход

Прежде чем защищать данные сайта от незарегистрированных пользователей, следует реализовать вход на сайт и выход с него.

Дополним список зарегистрированных пользователей полями для хранения хэша пароля (будем вычислять его по алгоритму bcrypt), адреса электронной почты, настоящих имени и фамилии пользователя (сделаем их необязательными), признака, является ли пользователь активным (то есть может ли он входить на сайт), признака, хочет ли он получать по электронной почте уведомления о новых комментариях, и признака, является ли пользователь администратором.

Пусть страница входа открывается при переходе по интернет-адресу: **/login/**, а страница выхода — по интернет-адресу: **/logout/**.

Файлы сайта фотогалереи хранятся в папке 19\19.7\ex, а файл photogallery.sql с дампом базы данных `photogallery` — в папке 18\18.6\s сопровождающего книгу электронного архива (см. *приложение 3*).

1. Добавим в таблицу `users` базы `photogallery` поля, приведенные в табл. 21.1.

***Таблица 21.1.** Поля, добавляемые в таблицу* `users`

Имя	Тип	Описание	Параметры
password	VARCHAR, 60 символов	Хэш пароля, вычисленный по алгоритму bcrypt	
email	VARCHAR, 127 символов	Адрес электронной почты пользователя	
name1	VARCHAR, 30 символов	Настоящее имя пользователя	
name2		Настоящая фамилия пользователя	
active	BOOLEAN	Является ли пользователь активным	Значение по умолчанию — FALSE
emailme		Хочет ли пользователь получать оповещения	Значение по умолчанию — TRUE
admin		Является ли пользователь администратором	Значение по умолчанию — FALSE

Все вновь добавляемые в список пользователи изначально будут неактивными. На *уроке 23* мы создадим подсистему активизации нового пользователя при переходе на определенный интернет-адрес, высылаемый ему электронным письмом.

2. Зададим для поля `active` у всех трех имеющихся в списке пользователей значение `1` (то есть `TRUE`), чтобы сделать их активными.
3. Зададим для поля `admin` пользователя `admin` значение `1`, чтобы сделать его администратором.
4. Вычислим три хэша одного и того же пароля `123456`, трижды набрав в консоли PHP команду:

```
echo password_hash('123456', PASSWORD_BCRYPT);
```

Полученные хэши занесем в поле `password` всех пользователей.

> ***О паролях***
>
> На момент разработки лучше не задавать сложных паролей, чтобы не забыть их.

Как только пользователь выполнит вход, сохраним его номер в сессии под именем `current_user`.

При генерировании каждой страницы мы станем проверять, выполнил ли пользователь вход, и, если выполнил, извлекать сведения о нем из базы данных. Код, извлекающий эти данные, поместим в конструктор базового контроллера `Controllers\BaseController`. Для хранения этих сведений объявим в том же классе общедоступное свойство `current_user`.

Помимо того, мы станем добавлять сведения о текущем пользователе в контекст каждого шаблона под именем `__current_user`.

5. Откроем модуль modules\controllers\basecontroller.php, хранящий код базового контроллера `Controllers\BaseController`, и соответственно доработаем его (здесь и далее добавления и правки в написанном ранее коде выделены полужирным шрифтом):

```
class BaseController {
    public $current_user = NULL;

    protected function context_append(array &$context) {
        $context['__current_user'] = $this->current_user;
    }

    . . .

    function __construct() {
        if (session_status() != PHP_SESSION_ACTIVE)
            session_start();
        if (isset($_SESSION['current_user'])) {
            $users = new \Models\User();
            $this->current_user =
                $users->get_or_404($_SESSION['current_user']);
        } else
            session_destroy();
    }
}
```

В конструкторе запускаем сессию, если она еще не запущена, и проверяем, хранится ли в ней элемент `current_user`. Если он есть, извлекаем из него номер вошедшего на сайт пользователя, ищем его в базе данных и присваиваем свойству `current_user`. Если вход не был выполнен, удаляем ненужную пустую сессию, чтобы не занимать память и дисковое пространство.

Чтобы добавить текущего пользователя в контекст данных каждого шаблона, перекрываем метод `context_append`.

6. Откроем модуль маршрутизатора modules\router.php и внесем код, распознающий интернет-адреса **/login/** и **/logout/** и вызывающий в ответ методы `login` и `logout` пока не объявленного контроллера `Controllers\Login`:

```
. . .
} else if (preg_match('/^(\d+)\/comments\/(\d+)\/delete$/',
    . . .
} else if ($request_path == 'login') {
    $ctr = new \Controllers\Login();
    $ctr->login();
} else if ($request_path == 'logout') {
    $ctr = new \Controllers\Login();
    $ctr->logout();
} else if ($request_path == '') {
. . .
```

7. Объявим класс формы входа `Forms\Login`. В ней создадим строковые поля `name` и `password` для ввода, соответственно, имени пользователя и пароля, и статический метод `verify_user`:

```
verify_user(<данные из веб-формы входа>)
```

Этот метод проверит, существует ли в списке пользователь с заданным именем, активен ли он и соответствует ли заданный пароль хранящемуся в списке хэшу, и, если это не так, задаст соответствующие сообщения об ошибках ввода. В качестве результата метод вернет номер пользователя, в противном случае — `FALSE`.

Создадим модуль modules\forms\login.php и запишем в него код класса формы `Forms\Login` из листинга 21.1.

Листинг 21.1. Модуль modules\forms\login.php

```
<?php
namespace Forms;
class Login extends \Forms\Form {
    protected const FIELDS = [
        'name' => ['type' => 'string'],
        'password' => ['type' => 'string']
    ];
```

```
    static function verify_user(&$data) {
        $errors = [];
        $users = new \Models\User();
        $user = $users->get($data['name'], 'name',
            'id, password, active');
        if (!$user)
            $errors['name'] = 'Неверное имя пользователя';
        else {
            if (!$user['active'])
                $errors['name'] = 'Этот пользователь неактивен';
            else {
                if (!password_verify($data['password'],
                    $user['password']))
                    $errors['password'] = 'Неверный пароль';
                else
                    return $user['id'];
            }
        }
        $data['__errors'] = $errors;
        return FALSE;
    }
}
```

Отметим, что метод `verify_user` представляет посетителю детальные сообщения об ошибках ввода. Так, если посетитель ввел неправильное имя, метод сообщит ему об ошибке в имени пользователя, а если занес не тот пароль — сообщение о неправильном пароле.

8. Объявим контроллер `Controllers\Login`, обрабатывающий вход и выход. Вход на сайт реализует метод `login` этого контроллера, выход с сайта — метод `logout`.

   Создадим модуль **modules\controllers\login.php** и запишем в него код контроллера `Controllers\Login` из листинга 21.2.

Листинг 21.2. Модуль modules\controllers\login.php

```
<?php
namespace Controllers;
class Login extends BaseController {
    function login() {
        if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            $login_form = \Forms\Login::get_normalized_data($_POST);
            if (!isset($login_form['__errors'])) {
                $login_form =
                    \Forms\Login::get_prepared_data($login_form);
                $user_id = \Forms\Login::verify_user($login_form);
                if ($user_id) {
                    session_start();
```

```
                    $_SESSION['current_user'] = $user_id;
                    \Helpers\redirect('/users/' .
                        $login_form['name']);
                }
            }
        } else
            $login_form = \Forms\Login::get_initial_data();
        $ctx = ['form' => $login_form, 'site_title' => 'Вход'];
        $this->render('login', $ctx);
    }

    function logout() {
        if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            unset ($_SESSION['current_user']);
            session_destroy();
            \Helpers\redirect('/');
        } else {
            $ctx = ['site_title' => 'Выход'];
            $this->render('logout', $ctx);
        }
    }
}
```

В методе `login` извлекаем данные из веб-формы входа, проверяем их на корректность, после чего с помощью статического метода `verify_user` формы `Forms\Login` выясняем, существует ли зарегистрированный пользователь с указанным именем, активен ли он и соответствует ли заданный пароль хэшу, хранящемуся в базе данных. Если все это так, запускаем сессию, сохраняем в ней номер пользователя, сведения о котором были занесены в форму входа, — тем самым производя вход на сайт, — и выполняем перенаправление на страницу вошедшего на сайт пользователя.

В методе `logout` при запросе HTTP-методом `GET` выводим страницу выхода с веб-формой, содержащей только кнопку. По нажатию этой кнопки будет выполнен запрос по тому же интернет-адресу, но уже методом `POST`, и будет вызван тот же метод `logout`. В нем, при запросе HTTP-методом `POST`, удаляем из сессии сохраненный ранее номер пользователя, стираем саму сессию, тем самым выполняя выход, и производим перенаправление на главную страницу.

9. Создадим модуль **modules\templates\login.php**, в котором сохраним код шаблона страницы входа (листинг 21.3).

Листинг 21.3. Модуль modules\templates\login.php

```
<?php require \Helpers\get_fragment_path('__header') ?>
<h2>Вход</h2>
<form class="bigform" method="post">
```

```
    <label for="user_name">Имя</label>
    <input type="text" id="user_name" name="name"
    value="<?php echo $form['name'] ?>">
    <?php \Helpers\show_errors('name', $form) ?>
    <label for="user_password">Пароль</label>
    <input type="password" id="user_password" name="password">
    <?php \Helpers\show_errors('password', $form) ?>
    <input type="submit" value="Отправить">
</form>
<?php require \Helpers\get_fragment_path('__footer') ?>
```

10. Создадим модуль modules\templates\logout.php и запишем в него шаблон страницы выхода (листинг 21.4).

Листинг 21.4. Модуль modules\templates\logout.php

```
<?php require \Helpers\get_fragment_path('__header') ?>
<h2>Выход</h2>
<form method="post">
    <input type="submit" value="Выйти">
</form>
<?php require \Helpers\get_fragment_path('__footer') ?>
```

Если посетитель не выполнил вход на сайт, на все страницы следует вывести гиперссылку на страницу входа. Если же вход на сайт был выполнен, нужно вывести гиперссылки на страницу текущего пользователя и страницу выхода. Поместим обе гиперссылки непосредственно под шапкой сайта.

11. Откроем модуль modules\templates__header.inc.php, где хранится шаблон-фрагмент шапки сайта, и добавим в его конец код, выводящий упомянутые ранее гиперссылки:

```
. . .
<h1><a href="/"><?php echo \Settings\SITE_NAME ?></a></h1>
<?php
    if ($__current_user['name1'] && $__current_user['name2'])
        $user_name = $__current_user['name1'] . ' ' .
            $__current_user['name2'];
    else if ($__current_user['name1'])
        $user_name = $__current_user['name1'];
    else
        $user_name = $__current_user['name'];
?>
<section id="logged">
    <?php if ($__current_user) {?>
        <a href="/users/<?php echo $__current_user['name'] ?>">
        <?php echo $user_name ?></a>
        <a href="/logout">Выйти</a>
```

```
    <?php } else { ?>
        <a href="/login">Войти</a>
    <?php } ?>
</section>
```

Если в составе данных пользователя хранятся его настоящие имя и фамилия (заполнены поля `name1` и `name2` таблицы `users`), текстом гиперссылки, ведущей на страницу этого пользователя, станет строка, составленная из имени и фамилии. Если пользователь ввел лишь настоящее имя, будет выведено оно, а если не ввел ни настоящего имени, ни фамилии, отобразится лишь его имя из поля `name`.

Ранее мы добавили в базовый контроллер код, заносящий в контекст каждого шаблона под именем `__current_user` сведения о пользователе, выполнившем вход, или `NULL`, если вход не был выполнен (это значение берется из свойства `current_user` контроллера). Стало быть, мы можем выяснить, был ли выполнен вход, просто удостоверившись, что доступная в шаблоне переменная `$__current_user` хранит значение, отличное от `NULL`.

12. Найдем в папке 21\!sources файл с дополненной таблицей стилей styles.css и скопируем его в корневую папку сайта, затерев имеющийся там старый файл.

Откроем сайт, перейдем на страницу входа и попытаемся войти на сайт от имени пользователя — например, `admin`, введя его имя и общий для всех пользователей пароль `123456`. Далее выйдем с сайта и попробуем войти снова, но уже от имени другого пользователя — скажем, `basil`. Напоследок наберем либо имя несуществующего пользователя, либо неверный пароль, попытаемся выполнить вход и посмотрим, какое сообщение об ошибке будет выведено на экран.

21.4. Упражнение. Защищаем веб-сайт

Сделаем так, чтобы:

- добавлять новые изображения могли лишь зарегистрированные пользователи;
- править и удалять изображения и комментарии могли лишь их авторы и администраторы.

Необходимые проверки вставим как в контроллеры, так и в шаблоны. Если проверка завершилась неудачей, будем выдавать страницу с сообщением об ошибке 403 (доступ запрещен).

Файлы сайта фотогалереи и файл photogallery.sql с дампом базы данных `photogallery` хранятся в папке 21\21.3\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Объявим класс исключения `Page403Exception` и внесем исправления в код обработчика исключений по умолчанию, чтобы он при получении исключения упомянутого ранее класса вызывал метод `403` контроллера `Controllers\Error` (этот метод мы напишем позже).

Откроем модуль единой точки входа index.php и внесем в код нужные правки (здесь и далее добавления и правки в ранее написанном коде выделены полужирным шрифтом):

```
. . .
class Page404Exception extends Exception {}
class Page403Exception extends Exception {}

function exception_handler($e) {
    $ctr = new \Controllers\Error();
    if ($e instanceof Page404Exception)
        $ctr->page404();
    else if ($e instanceof Page403Exception)
        $ctr->page403();
    else
        $ctr->page503($e);
}
set_exception_handler('exception_handler');
. . .
```

2. Откроем модуль modules\controllers\error.php с контроллером `Controllers\Error` и добавим объявление метода `403`:

```
class Error extends BaseController {
    . . .
    function page403() {
        $this->render('403', []);
    }
}
```

3. Создадим модуль modules\templates\403.php и запишем в него код шаблона страницы с сообщением об ошибке 403 (листинг 21.5).

Листинг 21.5. Модуль modules\templates\403.php

```
<?php http_response_code(403) ?>
<?php require \Helpers\get_fragment_path('__header') ?>
<h2>Доступ запрещен</h2>
<p>У вас нет прав на доступ к этой странице или
выполнения этой операции.</p>
<p><a href="/">На главную</a></p>
<?php require \Helpers\get_fragment_path('__footer') ?>
```

При добавлении комментария будем записывать в поле `user` таблицы `comments` номер текущего пользователя — автора комментария. Так что раскрывающийся список пользователей в форме добавления комментария нам больше не нужен.

4. Откроем модуль modules\forms\comment.php с кодом формы `Forms\Comment` и удалим поле пользователя `user` (здесь и далее удаленный код зачеркнут):

```
class Comment extends \Forms\Form {
    protected const FIELDS = [
        'user' => ['type' => 'integer'],
        'contents' => ['type' => 'string']
    ];
}
```

5. Откроем модуль modules\templates__comment_form.inc.php с шаблоном-фрагментом веб-формы комментария и удалим HTML-код, выводящий раскрывающийся список пользователей:

```
<form class="bigform" method="post">
    <label for="comment_user">Пользователь</label>
    <select id="comment_user" name="user">
        . . .
    </select>
    <label for="comment_contents">Содержание</label>
    . . .
</form>
```

6. Исправим метод `item` контроллера `Controllers\Images`: уберем код, выбирающий из базы перечень пользователей, добавим в перечень комментариев поле с номером пользователя-автора и добавим код, записывающий в поле `user` сохраняемого нового комментария номер текущего пользователя.

   Откроем модуль modules\controllers\images.php и внесем необходимые правки:

```
class Images extends BaseController {
    . . .
    function item(int $index) {
        if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            . . .
            if (!isset($comment_form['__errors'])) {
                . . .
                $comment_form['picture'] = $index;
                $comment_form['user'] = $this->current_user['id'];
                $comments = new \Models\Comment();
                . . .
            }
        } else
            $comment_form =
                \Forms\Comment::get_initial_data();
        $users = new \Models\User();
        $users->select('*', NULL, '', NULL, 'name');
        $picts = new \Models\Picture();
        . . .
        $comments = new \Models\Comment();
        $comments->select('comments.id, contents, ' .
            'users.name AS user_name, uploaded, ' .
```

```
            'users.id AS user_id', ['users'], 'picture = ?',
            [$index]);
        $ctx = ['pict' => $pict, 'site_title' => $pict['title'],
            'comments' => $comments, 'form' => $comment_form,
            'users' => $users];
        $this->render('item', $ctx);
    }
    . . .
}
```

7. Исправим метод by_user того же контроллера, добавив в перечень изображения поле с номером пользователя, опубликовавшего это изображение:

```
class Images extends BaseController {
    . . .
    function by_user(string $username) {
        . . .
        $picts->select('pictures.id, title, filename, uploaded, ' .
            'categories.name AS cat_name, categories.slug, ' .
            '(SELECT COUNT(*) FROM comments WHERE ' .
            'comments.picture = pictures.id) AS comment_count, ' .
            'pictures.user', ['categories'], $w, $p, '',
            $paginator->first_record_num, \Settings\RECORDS_ON_PAGE);
            . . .
    }
    . . .
}
```

На странице изображения будем показывать форму добавления комментария, только если пользователь выполнил вход. А гиперссылки правки и удаления станем выводить лишь у комментариев, автором которых является пользователь, выполнивший вход, или если этот пользователь является администратором.

8. Откроем модуль modules\templates\item.php с шаблоном страницы изображения и внесем необходимые правки:

```
. . .
<p>Дата и время публикации:
<?php echo \Helpers\get_formatted_timestamp($pict['uploaded']) ?></p>
<?php if ($__current_user) { ?>
<h3>Добавить комментарий</h3>
<?php require \Helpers\get_fragment_path('__comment_form') ?>
<?php } ?>
. . .
<?php foreach ($comments as $comment) { ?>
    . . .
    <?php if ($__current_user &&
        ($__current_user['id'] == $comment['user_id'] ||
        $__current_user['admin'])) { ?>
```

```
    <?php $u1 = '/' . $pict['id'] . '/comments/' . $comment['id'] ?>
    <p><a href="<?php echo $u1 . '/edit' . $u2 ?>">Исправить</a>
    <a href="<?php echo $u1 . '/delete' . $u2 ?>">Удалить</a></p>
    <?php } ?>
    <p> </p>
<?php } ?>
. . .
```

Ранее в коде метода `item` контроллера `Controllers\Images` мы включили в состав перечня комментариев номера их авторов. Имея эти номера, без проблем выясним, является ли текущий пользователь автором очередного комментария.

На странице пользователя станем выводить гиперссылку на страницу добавления изображения, только если текущий пользователь является тем, кому принадлежит эта страница. А гиперссылки правки и удаления изображения выведем только пользователю, опубликовавшему изображение, и администратору.

9. Откроем модуль modules\templates\by_user.php, где хранится шаблон страницы пользователя, и внесем правки:

```
. . .
<h2><?php echo $user['name'] ?></h2>
<?php if ($__current_user &&
    $__current_user['id'] == $user['id']) { ?>
<p><a href="<?php echo '/users/' . $user['name'] .
    '/pictures/add'. $gets ?>">Добавить изображение</a></p>
<?php } ?>
<table id="gallery">
    . . .
    <tr>
        . . .
        <?php if ($__current_user &&
            ($__current_user['id'] == $pict['user'] ||
            $__current_user['admin'])) { ?>
        <td><a href="<?php echo '/users/', $user['name'],
            '/pictures/', $pict['id'], '/edit',
            $gets ?>">Исправить</a></td>
        <td><a href="<?php echo '/users/', $user['name'],
            '/pictures/', $pict['id'], '/delete',
            $gets ?>">Удалить</a></td>
        <?php } else { ?>
            <td></td><td></td>
        <?php } ?>
    </tr>
    <?php } ?>
</table>
. . .
```

Переменная `$user` хранит пользователя, которому принадлежит страница. Из этих сведений мы можем получить номер пользователя.

Теперь нежелательный посетитель не сможет ни добавить новое изображение, ни изменить, ни удалить не принадлежащее ему изображение или комментарий. По крайней мере, перейдя по гиперссылке. Однако он все-таки может войти на закрытую от него страницу, набрав ее интернет-адрес (например, он легко удалит изображение № 10, принадлежащее пользователю `basil`, перейдя по интернет-адресу: **http://localhost/users/basil/pictures/10/delete**).

Поэтому разграничение доступа нужно реализовывать на уровне не только шаблонов, но и контроллеров. В самом начале кода каждого метода, генерирующего закрытую от посторонних страницу, следует проверять, выполнил ли пользователь вход, и есть ли у него привилегии на доступ к этой странице, и, если это не так, генерировать исключение класса `Page403Exception`, объявленного в начале этого упражнения.

10. Добавим в метод `item` контроллера `Controllers\Images` проверку, выполняющуюся перед сохранением нового комментария и выясняющую, был ли выполнен вход.

    Откроем модуль modules\controllers\images.php с кодом этого контроллера (если уже закрыли его) и добавим нужный код:

```
class Images extends BaseController {
    . . .
    function item(int $index) {
        if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            if (!$this->current_user)
                throw new \Page403Exception();
            . . .
        } else
            . . .
        . . .
    }
    . . .
}
```

11. Добавим аналогичные проверки в методы `edit` и `delete` контроллера `Controllers\Comments`.

    Откроем модуль modules\controllers\comments.php, где хранится код этого контроллера, и добавим нужный код:

```
class Comments extends BaseController {
    private function check_user(int $comment_index) {
        $comments = new \Models\Comment();
        $comment = $comments->get_or_404($comment_index, 'id',
            'user');
        if (!($this->current_user &&
            ($this->current_user['id'] == $comment['user'] ||
            $this->current_user['admin'])))
            throw new \Page403Exception();
    }
```

```
    function edit(int $picture_index, int $comment_index) {
        $this->check_user($comment_index);
        . . .
    }

    function delete(int $picture_index, int $comment_index) {
        $this->check_user($comment_index);
        . . .
    }
}
```

Мы объявили закрытый метод `check_user`, принимающий номер комментария и выясняющий, имеет ли текущий пользователь привилегии на его правку или удаление. После этого нам осталось лишь вставить в начало методов `edit` и `delete` вызов метода `check_user`.

12. Таким же образом защитим контроллер изображений `Controllers\Images`. Объявим закрытый метод `check_user`, подобный одноименному методу контроллера комментариев, и вставим его вызовы в методы `edit` и `delete` контроллера изображений. В метод `add` вставим проверку, был ли выполнен вход.

Откроем сайт, посмотрим, выводятся ли гиперссылки добавления, правки, удаления изображений, форма для ввода комментария, гиперссылки правки и удаления комментариев. Войдем на сайт от имени пользователя, не являющегося администратором (`basil` или `jocker`), и проверим, какие гиперссылки и в каких случаях выводятся. Проведем аналогичную проверку, войдя от имени пользователя-администратора `admin`. Наконец, попробуем добавить комментарий с произвольным содержанием.

21.5. Упражнение. Создаем веб-страницу регистрации

Добавим в наш сайт страницу для регистрации новых пользователей, на которой новичок, желающий зарегистрироваться, введет имя, пароль, подтверждение пароля (тот же самый пароль еще раз — для надежности), адрес электронной почты, настоящие имя и фамилию (необязательно) и укажет, хочет ли он получать по электронной почте оповещения о новых комментариях.

Попасть на эту страницу будет можно, перейдя по интернет-адресу: **/register/**.

Файлы сайта фотогалереи и файл photogallery.sql с дампом базы данных `photogallery` хранятся в папке 21\21.4\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Откроем модуль маршрутизатора modules\router.php и добавим код, распознающий интернет-адрес **/register/** и запускающий в ответ метод `register` контроллера `Controllers\Login`, который мы вскоре напишем (здесь и далее дополнения и правки в написанном ранее коде выделены полужирным шрифтом):

```
. . .
} else if ($request_path == 'logout') {
    . . .
} else if ($request_path == 'register') {
    $ctr = new \Controllers\Login();
    $ctr->register();
} else if ($request_path == '') {
. . .
```

2. Объявим класс формы `Forms\Register`, обрабатывающий данные, которые заносятся в веб-форму регистрации нового пользователя.

   Создадим модуль modules\forms\register.php и занесем в него код формы регистрации из листинга 21.6.

Листинг 21.6. Модуль modules\forms\register.php

```
<?php
namespace Forms;
class Register extends \Forms\Form {
    protected const FIELDS = [
        'name' => ['type' => 'string'],
        'password1' => ['type' => 'string', 'nosave' => TRUE],
        'password2' => ['type' => 'string', 'nosave' => TRUE],
        'email' => ['type' => 'email'],
        'name1' => ['type' => 'string', 'optional' => TRUE],
        'name2' => ['type' => 'string', 'optional' => TRUE],
        'emailme' => ['type' => 'boolean', 'initial' => TRUE]
    ];

    protected static function after_normalize_data(&$data, &$errors) {
        if ($data['name'] &&
            !preg_match('/[a-zA-Z0-9_-]{5,20}/', $data['name']))
            $errors['name'] = 'Имя пользователя должно включать ' .
                'лишь латинские буквы, цифры, дефисы, символы ' .
                'подчеркивания и содержать от 5 до 20 знаков';
        $users = new \Models\User();
        if ($users->get($data['name'], 'name', 'id'))
            $errors['name'] = 'Пользователь с таким именем уже ' .
                'существует';
        if ($data['password1'] != $data['password2'])
            $errors['password2'] = 'Введите в эти поля один и тот же пароль';
    }

    protected static function after_prepare_data(&$data, &$norm_data)
    {
        $data['password'] = password_hash($norm_data['password1'],
            PASSWORD_BCRYPT);
    }
}
```

Поля password1 и password2, в которые заносятся пароль и его подтверждение, помечаем как не подлежащие сохранению в базе. У поля emailme (признак, желает ли пользователь получать оповещения) указываем изначальное значение TRUE.

В методе after_normalize_data выполняем дополнительную проверку данных на корректность. А в методе after_prepare_data вычисляем хэш пароля, который будет сохранен в базе.

3. Откроем модуль modules\controllers\login.php, где хранится код контроллера Controllers\Login, и добавим метод register, выполняющий регистрацию нового пользователя:

```
class Login extends BaseController {
    . . .
    function register() {
        if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            $reg_form =
                \Forms\Register::get_normalized_data($_POST);
            if (!isset($reg_form['__errors'])) {
                $reg_form =
                    \Forms\Register::get_prepared_data($reg_form);
                $reg_form['active'] = TRUE;
                $users = new \Models\User();
                $users->insert($reg_form);
                \Helpers\redirect('/login');
            }
        } else
            $reg_form = \Forms\Register::get_initial_data();
        $ctx = ['form' => $reg_form, 'site_title' => 'Регистрация'];
        $this->render('register', $ctx);
    }
}
```

В поле active таблицы users заносим значение TRUE, чтобы пометить пользователя как активного. Это временное решение, пока мы не сделали подтверждение регистрации по электронной почте (чем займемся на *уроке 23*).

4. Найдем в папке 21\!sources модуль register.php, в котором хранится шаблон страницы регистрации, и скопируем его в папку modules\templates сайта. Код этого шаблона, с одной стороны, довольно велик, а с другой, весьма тривиален, чтобы приводить его здесь полностью.

5. Поместим на все страницы сайта, непосредственно под шапкой, гиперссылку, ведущую на страницу регистрации. Сделаем ее доступной только гостям (поскольку зарегистрированным пользователям регистрироваться еще раз ни к чему).

Откроем модуль modules\templates__header.inc.php, где находится шаблон-фрагмент шапки, и вставим HTML-код, выводящий эту гиперссылку:

```
. . .
<section id="logged">
    <?php if ($__current_user) {?>
        . . .
    <?php } else { ?>
        <a href="/register">Зарегистрироваться</a>
        <a href="/login">Войти</a>
    <?php } ?>
</section>
```

Откроем сайт и зарегистрируемся под произвольным именем, например `timid`, и паролем `123456`, остальные данные могут быть произвольными. Попытаемся выполнить вход от имени пользователя `timid`.

Далее зарегистрируемся еще раз, указав имя `timid2`, пароль `654321` и выполним вход от имени этого пользователя. Найдем в папке 21\!sources изображение Сгоревший светодиод.jpg, опубликуем на сайте в любой категории и оставим к нему пару комментариев от имени разных пользователей. Этот пользователь и это изображение позже понадобятся, чтобы проверить, как работает удаление пользователей.

21.6. Упражнение. Реализуем правку и удаление пользователей

На тот случай, если зарегистрированный пользователь захочет покинуть нашу фотогалерею, реализуем удаление пользователей. Страница удаления пользователя будет доступна по интернет-адресу: **/users/<*имя пользователя*>/account/delete/** только после выполнения входа.

Правку данных пользователя — его адреса электронной почты и пароля — вы, уважаемые читатели, сделаете самостоятельно.

Файлы сайта фотогалереи и файл photogallery.sql с дампом базы данных `photogallery` хранятся в папке 21\21.5\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Исправим модель `\Models\User`, чтобы она при удалении пользователя также удаляла все опубликованные им картинки и их миниатюры.

   Откроем модуль modules\models\User.php и внесем правки (здесь и далее добавления и правки в уже написанном коде выделены полужирным шрифтом):

```
class User extends \Models\Model {
    . . .
    protected function before_delete($value, $key_field = 'id') {
        if ($key_field != 'id') {
            $users = new \Models\User();
            $user = $users->get($value, $key_field, 'id');
            $value = $user['id'];
        }
```

```
            $pictures = new \MODELS\Picture();
            $pictures2 = new \MODELS\Picture();
            $pictures->select('id', NULL, 'user = ?', [$value]);
            foreach ($pictures as $picture)
                $pictures2->delete($picture['id']);
        }
    }
```

Если поиск удаляемого пользователя выполняется не по номеру, а по значению другого поля, извлекаем пользователя и получаем его номер. Далее ищем все изображения, опубликованные пользователем с указанным номером, и удаляем их вызовом метода `delete` модели `\Models\Picture` — при этом также будут удалены файлы с самими изображениями и их миниатюрами.

2. Откроем модуль маршрутизатора modules\router.php и вставим код, распознающий интернет-адрес страницы удаления пользователя и вызывающий метод `delete` контроллера `Controllers\Account`, который мы скоро напишем:

```
. . .
} else if (preg_match('/^(\d+)\/comments\/(\d+)\/delete$/',
    . . .
} else if (preg_match('/^users\/(\w+)\/account\/delete$/',
    $request_path, $result) == 1) {
    $ctr = new \Controllers\Account();
    $ctr->delete($result[1]);
} else if ($request_path == 'login') {
. . .
```

3. Создадим модуль modules\controllers\account.php и запишем в него код контроллера `Controllers\Account`, обрабатывающего операцию удаления пользователя (листинг 21.7).

Листинг 21.7. Модуль modules\controllers\account.php

```
<?php
namespace Controllers;
class Account extends BaseController {
    private function check_user(string $username) {
        $users = new \Models\User();
        $user = $users->get_or_404($username, 'name', 'id');
        if (!($this->current_user &&
            ($this->current_user['id'] == $user['id'] ||
            $this->current_user['admin'])))
            throw new Page403Exception();
    }

    function delete(string $username) {
        $this->check_user($username);
```

```
        if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            $users = new \Models\User();
            $users->delete($username, 'name');
            unset ($_SESSION['current_user']);
            session_destroy();
            \Helpers\redirect('/');
        } else {
            $ctx = ['site_title' => 'Удаление пользователя'];
            $this->render('user_delete', $ctx);
        }
    }
}
```

Страницу удаления пользователя мы сделали доступной только для пользователя, чье имя указано в интернет-адресе, и администратора. Код, выполняющий эту проверку, мы вынесли в закрытый метод `check_user`.

Метод `delete`, выполнив проверку вызовом метода `check_user`, удалит пользователя из базы данных, номер пользователя, выполнившего вход, — из сессии и саму сессию, тем самым произведя выход с сайта. После чего будет выполнено перенаправление на главную страницу.

4. Создадим модуль modules\templates\user_delete.php и сохраним в нем шаблон страницы удаления пользователя, код которой приведен в листинге 21.8.

Листинг 21.8. Модуль modules\templates\user_delete.php

```
<?php require \Helpers\get_fragment_path('__header') ?>
<h2>Удаление пользователя</h2>
<form method="post">
    <input type="submit" value="Удалить">
</form>
<?php require \Helpers\get_fragment_path('__footer') ?>
```

5. Откроем модуль modules\templates__header.inc.php с шаблоном-фрагментом шапки сайта и добавим код, выводящий гиперссылку на страницу удаления пользователя, которая будет доступна лишь после выполнения входа:

```
. . .
<section id="logged">
    <?php if ($__current_user) {?>
        . . .
        <a href="/users/<?php echo $__current_user['name'] ?>↵
        /account/delete">Удалить пользователя</a>
    <?php } else { ?>
        . . .
    <?php } ?>
</section>
```

Выполним вход на сайт под именем ранее добавленного пользователя `timid2` по его паролю `654321` и удалим его. Проверим, чтобы после этого были удалены также файлы опубликованной им картинки и ее миниатюры.

Сделайте сами

- Создайте страницу правки сведений о пользователе: его адреса электронной почты, настоящих имени и фамилии, признака, хочет ли он получать оповещения о новых комментариях. Пусть эта страница будет доступна по интернет-адресу формата: **/users/<*имя пользователя*>/account/edit/**.
- Создайте страницу для смены пароля с веб-формой, содержащей поля для ввода пароля и его подтверждения. Пусть она будет доступна по интернет-адресу формата: **/users/<*имя пользователя*>/account/editpassword/**.

Код, выполняющий эти действия, поместите в контроллер `Controllers\Account`.

Урок 22
Отправка электронной почты

22.1. Класс *SendMailSmtpClass*

Для отправки электронной почты в серверных приложениях PHP удобно задействовать класс `SendMailSmtpClass`. Он написан независимым разработчиком Евгением Ипатовым, очень прост в использовании, позволяет посылать письма с вложениями, не требует внесения изменений в конфигурацию PHP и, самое главное, может отправлять почту SMTP-серверам, работающим по защищенным протоколам и требующим авторизации (а в настоящее время почти все почтовые службы используют такие SMTP-серверы).

> Архив с классом `SendMailSmtpClass` доступен по интернет-адресу: **https://github.com/Ipatov/SendMailSmtpClass**. Чтобы загрузить его, следует щелкнуть на кнопке **Clone or download**, а потом — на гиперссылке **Download ZIP** в появившейся под кнопкой панели (рис. 22.1).

В полученном архиве находится папка SendMailSmtpClass-master с несколькими файлами, из которых нам нужен лишь PHP-модуль SendMailSmtpClass.php.

Новое электронное письмо создается вызовом конструктора этого класса:

```
new SendMailSmtpClass(<имя учетной записи>, <пароль>,
                      <интернет-адрес SMTP-сервера>[,
                      <номер TCP-порта SMTP-сервера>[,
                      <кодировка письма>]])
```

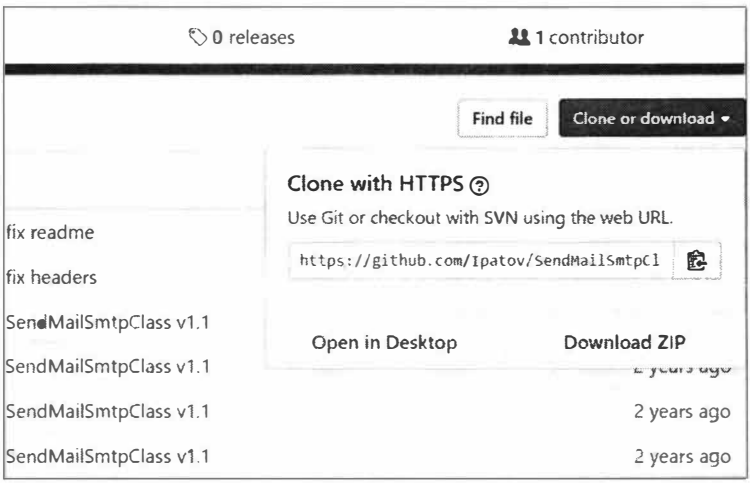


Рис. 22.1. Кнопка **Clone or download** и всплывающая панель с гиперссылкой **Download ZIP**

Имя учетной записи в почтовой службе, *пароль* и *интернет-адрес SMTP-сервера* указываются в виде строк, *номер TCP-порта* — в виде целого числа, *кодировка* — также в виде строки. Если не указан *номер TCP-порта*, будет использован стандартный порт 25, если не указана *кодировка* — UTF-8.

Отправка письма выполняется вызовом метода send:

```
send(<адрес получателя>, <тема письма>, <тело письма>,
    <сведения об отправителе>)
```

Сведения об отправителе указываются в виде индексированного массива из двух строковых элементов: имени отправителя и его адреса электронной почты. Остальные параметры должны быть заданы строками.

Метод возвращает TRUE в случае успешной отправки письма или строку с сообщением о возникшей ошибке.

Пример отправки письма через почтовую службу Mail.ru:

```
require_once 'SendMailSmtpClass.php';
$letter = new SendMailSmtpClass('photoadmin@mail.ru', 'AdM1Np-h-0-T-0',
    'ssl://smtp.mail.ru', 465);
$from = ['Администратор фотогалереи', 'photoadmin@mail.ru'];
$to = 'basil@mail.server.ru';
$subject = 'У вас новые комментарии';
$body = 'И довольно много.';
$letter->send($to, $subject, $body, $from);
```

Добавление *файла* к письму выполняется вызовом метода addFile(*<путь к файлу>*). Этот метод следует вызывать до отправки письма.

Пример отправки письма с двумя вложенными файлами:

```
$letter = new SendMailSmtpClass('photoadmin@mail.ru', 'AdM1Np-h-0-T-0',
    'ssl://smtp.mail.ru', 465);
$from = ['Администратор фотогалереи', 'photoadmin@mail.ru'];
$to = 'timid2@servermail.company.ru';
$subject = 'Картинки';
$body = 'Ваши горелые детали мне не нужны. Возвращаю их вам.';
$letter->addFile('c:\trash\Сгоревший светодиод.jpg');
$letter->addFile('c:\trash\Вспухший конденсатор.jpg');
$letter->send($to, $subject, $body, $from);
```

Сведения для подключения к SMTP-серверу...

...можно найти в справочном разделе сайта почтовой службы.

В настоящее время многие почтовые службы (в частности, Mail.ru) требуют, чтобы адрес электронной почты отправителя, указываемый в сведениях об отправляемом письме, совпадал с именем учетной записи. В противном случае письмо будет отвергнуто.

Полезно знать...

- PHP имеет встроенные средства для отправки электронной почты. Однако они не работают через защищенные протоколы и не обеспечивают авторизацию при подключении к SMTP-серверу, поэтому в настоящее время практически бесполезны.
- Класс `SendMailSmtpClass` работает с почтовыми службами на низком уровне — через TCP-сокеты, в обход встроенных в PHP инструментов отправки почты.

22.2. Упражнение. Реализуем отправку оповещений о новых комментариях

Сделаем так, чтобы при добавлении нового комментария к изображению, опубликовавшему его пользователю отправлялось электронное письмо с оповещением. При этом будем иметь в виду, что пользователь может отказаться от получения таких оповещений.

Адрес электронной почты пользователя хранится в поле `email`, а признак, хочет ли он получать оповещения, в поле `emailme` таблицы `users`.

Тему и тело электронных писем сформируем на основе шаблонов. Они будут храниться в папке modules\templates в обычных текстовых файлах, в содержимое которых мы вставим особые литералы формата `%<`*имя литерала*`>%`. При генерировании окончательного текста темы или тела на место каждого из этих литералов будет подставляться заданное значение.

Файлы сайта фотогалереи и файл photogallery.sql с дампом базы данных `photogallery` хранятся в папке 21\21.6\s сопровождающего книгу электронного архива (см. *приложение 3*).

1. Загрузим со страницы: **https://github.com/Ipatov/SendMailSmtpClass** архив с классом `SendMailSmtpClass` и поместим содержащийся в этом архиве модуль SendMailSmtpClass.php в папку modules.
2. Создадим константы с параметрами используемого SMTP-сервера: `Settings\SMTP_HOST` (интернет-адрес), `Settings\SMTP_PORT` (номер TCP-порта), `Settings\SMTP_LOGIN` (имя учетной записи), `Settings\SMTP_PASSWORD` (пароль), `Settings\MAIL_FROM` (сведения об отправителе) и `Settings\MAIL_SEND` (если `TRUE`, письма будут отправляться, если `FALSE`, не будут). Последняя константа позволит нам отменить отправку любых электронных писем, чтобы в дальнейшем при отладке сайта не отвлекаться на постоянно приходящие оповещения.

   Откроем модуль настроек modules\settings.php и добавим упомянутые ранее константы (здесь и далее добавления и правки в уже написанном коде выделены полужирным шрифтом):

```
namespace Settings {
    . . .
    const SMTP_HOST = '---';
    const SMTP_PORT = 25;
```

```
    const SMTP_LOGIN = '---';
    const SMTP_PASSWORD = '---';
    const MAIL_FROM = ['Photogallery', '---'];
    const MAIL_SEND = TRUE;
}
```

В константы: `Settings\SMTP_HOST`, `Settings\SMTP_PORT`, `Settings\SMTP_LOGIN` и `Settings\SMTP_PASSWORD` занесите реальные параметры SMTP-сервера вашей почтовой службы, а во второй элемент массива из константы `Settings\MAIL_FROM` — реальный адрес отправителя.

3. Напишем функцию `Helpers\render_mail`, заменяющую все присутствующие в текстовом *шаблоне* литералы формата `%<имя литерала>%` значениями из заданного *массива* и возвращающую текст с выполненными заменами:

   `render_text(<шаблон>, <массив со значениями>)`

   Ключи элементов ассоциативного *массива со значениями* должны совпадать с *именами литералов* без знаков процента.

   Откроем модуль modules\helpers.php и добавим объявление этой функции:

```
namespace Helpers {
    . . .
    function render_text(string $template, array $values): string {
        $literals = [];
        $vals = [];
        foreach ($values as $key => $value) {
            $literals[] = '/%' . $key . '%/iu';
            $vals[] = $value;
        }
        return preg_replace($literals, $vals, $template);
    }
}
```

   Мы выносим имена литералов и их значения в отдельные массивы, попутно добавляя к именам литералов знаки процента и преобразуя их в регулярные выражения. Это позволит выполнить все замены одним вызовом функции `preg_replace` (см. *разд. 11.3.4*).

4. Напишем функцию `Helpers\send_mail`, которая сгенерирует письмо на основе указанных *шаблонов* и отправит его по заданному *адресу*:

   `send_mail(<адрес электронной почты>, <шаблон темы>, <шаблон тела>, <массив со значениями>)`

   Добавим в модуль modules\helpers.php объявление этой функции:

```
namespace Helpers {
    . . .
    function send_mail(string $to, string $subject, string $body,
        array $values) {
        global $base_path;
```

```
            require_once $base_path . 'modules\SendMailSmtpClass.php';
            if (\Settings\MAIL_SEND) {
                $letter = new \SendMailSmtpClass(\Settings\SMTP_LOGIN,
                    \Settings\SMTP_PASSWORD, \Settings\SMTP_HOST,
                    \Settings\SMTP_PORT);
                $s_path = $base_path . '\modules\templates\\' . $subject .
                    '.txt';
                $s = render_text(file_get_contents($s_path), $values);
                $b_path = $base_path . '\modules\templates\\' . $body .
                    '.txt';
                $b = render_text(file_get_contents($b_path), $values);
                $letter->send($to, $s, $b, \Settings\MAIL_FROM);
            }
        }
    }
```

5. Добавим в метод `item` контроллера `Controllers\Images` код, отправляющий оповещение после сохранения нового комментария.

   Откроем модуль modules\controllers\images.php с кодом этого контроллера и внесем соответствующие правки:

```
class Images extends BaseController {
    . . .
    function item(int $index) {
        if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            . . .
            if (!isset($comment_form['__errors'])) {
                . . .
                $comments = new \Models\Comment();
                $comments->insert($comment_form);
                $pictures = new \Models\Picture();
                $user = $pictures->get($index, 'pictures.id',
                    'users.name, users.email, users.emailme',
                    ['users']);
                if ($user['emailme']) {
                    $values = ['title' => \Settings\SITE_NAME,
                        'name' => $user['name'],
                        'url' => 'http://' . $_SERVER['SERVER_NAME'] .
                        '/' . $index];
                    \Helpers\send_mail($user['email'],
                        'notification_subject', 'notification_body',
                        $values);
                }
                \Helpers\redirect('/' . $index .
                    \Helpers\get_GET_params(['page', 'filter',
                    'ref']));
            }
```

```
            } else
                . . .
            . . .
        }
        . . .
}
```

Для выяснения имени пользователя, опубликовавшего изображение, его адреса электронной почты и признака, хочет ли пользователь получать оповещения, используем модель `Models\Picture`. Выполняем поиск изображения по полученному с параметром номеру и извлекаем из связанной записи таблицы `users` нужные поля. Если пользователь желает получать оповещения, формируем массив из названия сайта, имени пользователя и полного интернет-адреса страницы с изображением и передаем его в функцию `Helpers\send_mail`. Доменное имя сервера получаем из элемента `SERVER_NAME` массива `$_SERVER` (был описан в *разд. 16.1.2*).

6. Создадим текстовый файл modules\templates\notification_subject.txt и сохраним в нем шаблон темы письма-оповещения (листинг 22.1).

Листинг 22.1. Файл modules\templates\notification_subject.txt

```
%title%. У вас новый комментарий
```

7. Создадим текстовый файл modules\templates\notification_body.txt и сохраним в нем шаблон тела письма-оповещения (листинг 22.2).

Листинг 22.2. Файл modules\templates\notification_body.txt

```
Здравствуйте, уважаемый %name%!

На сайте "%title%" вам только что оставили новый комментарий.

Чтобы просмотреть его, перейдите по интернет-адресу:
%url%

До свидания!

С уважением,
администрация сайта "%title%".
```

Войдем на сайт от имени какого-либо пользователя, перейдем на страницу правки сведений о пользователе, зададим наш адрес электронной почты и укажем, что пользователь хочет получать уведомления. Войдем на сайт под именем другого пользователя, выберем изображение, опубликованное первым пользова-

телем, оставим произвольный комментарий и проверим, пришло ли письмо с оповещением.

8. Откроем модуль с настройками modules\settings.php (если уже закрыли его) и исправим значение константы `Settings\MAIL_SEND` на `FALSE`. Так мы отключим отправку любой почты, чтобы приходящие письма не мешали дальнейшей отладке сайта.

   Если нам понадобится в будущем проверить отправку почты, мы вновь зададим у этой константы значение `TRUE`.

Урок 23
Усиленные меры безопасности

Мер безопасности, предпринятых нами на *уроке 21*, в настоящее время явно недостаточно. Необходимо повысить безопасность сайта, переведя его на защищенный протокол HTTPS, обеспечив защиту от атак типа CSRF и реализовав двухэтапную регистрацию новых пользователей.

23.1. Перевод веб-сайта на протокол HTTPS

Использовавшийся долгое время интернет-протокол HTTP не обеспечивает должную степень безопасности, поскольку пересылает данные в незашифрованном виде. Поэтому в настоящее время существующие сайты постепенно переводятся на шифрующий протокол HTTPS. Тогда при обращении к сайту по протоколу HTTP набором интернет-адреса вида:

http://*<остальная часть интернет-адреса>*

производится перенаправление на адрес вида:

https://*<остальная часть интернет-адреса>*

Фактически происходит переход на тот же адрес, но уже по протоколу HTTPS.

Выполнить такое перенаправление удобнее средствами веб-сервера, для чего достаточно вставить в конфигурационный файл .htaccess две строки (выделены полужирным шрифтом):

```
RewriteEngine On
RewriteCond %{HTTPS} !=on
RewriteRule ^/?(.*) https://%{SERVER_NAME}/$1 [R,L]
RewriteCond %{SCRIPT_FILENAME} !-f
RewriteRule ^(.*)$ ./index.php?route=$1 [L,QSA]
```

Первая строка проверяет, выполнилось ли обращение к сайту по протоколу, отличному от HTTPS. Вторая строка в таком случае выполняет описанное ранее перенаправление.

> ***Нужен цифровой сертификат!***
>
> При перенаправлении на протокол HTTPS веб-обозреватель сообщит, что сайт не защищен, и поначалу даже откажется открывать его (рис. 23.1). В таком случае для перехода на сайт нужно щелкнуть на гиперссылке **Перейти на сайт... (небезопасно)** (надписи в окне предупреждения различны у разных веб-обозревателей).

Чтобы избежать такого сообщения, необходимо получить для сайта цифровой сертификат и настроить веб-сервер таким образом, чтобы он использовал этот сертификат для шифрования данных. Описание получения сертификата и соответствующей настройки веб-сервера выходит за рамки этой книги.

Рис. 23.1. Сообщение о незащищенном веб-сайте

23.2. Защита от атак типа CSRF

> *Межсайтовая подделка запроса* (CSRF, Cross-Site Request Forgery) — хакерская атака, при которой данные, занесенные в веб-форму на одном сайте, тайно направляются другому сайту (например, номер банковской карты, введенный на сайте поддельного интернет-магазина, отправляется на сайт банка для тайного перевода денег на счет злоумышленника).

Алгоритм противодействия атаке такого рода следующий:

- При выводе каждой веб-формы генерируется электронный жетон, записываемый в скрытое поле, которое помещается в эту форму. Одновременно этот же жетон сохраняется в данных сессии.

  > *Жетон* — уникальное значение, идентифицирующее какие-либо данные. Может быть псевдослучайным числом, строкой или хэшем, вычисленным на основе какой-либо величины.

  Удобнее всего взять какое-либо неизменное значение — например, строку, и сохранить ее в сессии под ключом, совпадающим со сгенерированным жетоном. В таком случае защита будет работать даже в случае, если посетитель одновременно откроет несколько страниц с веб-формами.

- При получении данных из веб-формы из них извлекается жетон и выполняется его поиск в данных сессии. Если жетон найден, данные были отправлены с текущего сайта, и им можно доверять.

Жетон генерируется в два этапа:

- создание псевдослучайной последовательности байтов указанной *длины* — с помощью функции `random_bytes(<длина>)`. Результатом станет строка, содержащая двоичное представление этой последовательности;
- перевод полученной *последовательности* в шестнадцатеричное число, представленное в виде строки, — функцией `bin2hex(<последовательность>)`.

Пример:

```
// Перед выводом страницы с веб-формой генерируем жетон
$token = bin2hex(random_bytes(32));
// Создаем в сессии значение в виде обычной строки 'anti_csrf' с ключом,
// совпадающим с жетоном
$_SESSION[$token] = 'anti_csrf';
. . .
<!-- В веб-форме выводим скрытое поле и записываем в него жетон -->
<form . . .>
    <input type="hidden" name="token" value="<?php echo $token ?>">
    . . .
</form>
. . .
// Получив данные из веб-формы, проверяем наличие в сессии значения
// 'anti_csrf' с ключом, совпадающим с полученным из веб-формы жетоном
if (empty($__POST['token']))
    // Жетон в веб-форме отсутствует. Атака!
else {
    $token = $__POST['token'];
    if (empty($_SESSION[$token]))
        // Значения с ключом, совпадающим с жетоном, нет в сессии. Атака!
    else {
        // Извлекаем значение из сессии
        $val = $_SESSION[$token];
        // Удаляем значение из сессии, поскольку оно больше не нужно
        unset ($_SESSION[$token]);
        if ($val != 'anti_csrf')
            // Значение из сессии отлично от 'anti_csrf'. Атака!
        else
            // Данные из веб-формы безопасны
    }
}
```

23.3. Упражнение. Противодействуем атакам CSRF

Защитим веб-формы добавления, правки, удаления изображений, комментариев, правки сведений о пользователе, пароля и удаления пользователя — то есть все, доступные только после выполнения входа. При попытке отправить данные с другого сайта будем выводить страницу с ошибкой 403.

Файлы сайта фотогалереи и файл photogallery.sql с дампом базы данных `photogallery` хранятся в папке 22\22.2\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Объявим функцию `Helpers\generate_token()`, которая будет генерировать, сохранять в сессии и возвращать жетон.

   Откроем модуль modules\helpers.php и вставим код, объявляющий эту функцию (здесь и далее дополнения и правки в ранее написанном коде выделены полужирным шрифтом)

```
namespace Helpers {
    . . .
    function generate_token(): string {
        if (session_status() != PHP_SESSION_ACTIVE)
            session_start();
        $token = bin2hex(random_bytes(32));
        $_SESSION[$token] = 'anti_csrf';
        return $token;
    }
}
```

2. Объявим функцию `Helpers\check_token`, проверяющую, были ли данные, полученные из веб-формы, отправлены с текущего сайта:

   `check_token(<массив с данными из веб-формы>)`

   Добавим в модуль modules\helpers.php необходимый код:

```
namespace Helpers {
    . . .
    function check_token (array $form_data) {
        if (empty($form_data['__token']))
            throw new \Page403Exception();
        $token = $form_data['__token'];
        if (empty($_SESSION[$token]))
            throw new \Page403Exception();
        $val = $_SESSION[$token];
        unset ($_SESSION[$token]);
        if ($val != 'anti_csrf')
            throw new \Page403Exception();
    }
}
```

3. Добавим в метод `item` контроллера `Controllers\Images`, выводящий страницу изображения с формой для ввода нового комментария, код, который при выводе страницы добавит в данные формы сгенерированный жетон (под ключом `__token`), а при сохранении комментария — проверит, с текущего ли сайта были получены данные.

   Откроем модуль modules\controllers\images.php с кодом этого контроллера и добавим нужный код:

```
class Images extends BaseController {
    . . .
    function item(int $index) {
        if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            if (!$this->current_user)
                throw new \Page403Exception();
            \Helpers\check_token($_POST);
            . . .
            $comment_form =
                \Forms\Comment::get_normalized_data($_POST);
            . . .
        } else
            $comment_form =
                \Forms\Comment::get_initial_data();
        $comment_form['__token'] = \Helpers\generate_token();
        . . .
    }
    . . .
}
```

4. Добавим в веб-форму добавления и правки комментария скрытое поле для хранения жетона.

   Откроем модуль modules\templates__comment_form.inc.php с шаблоном-фрагментом формы комментария и добавим необходимый код:

```
<form class="bigform" method="post">
    <input type="hidden" name="__token"
    value="<?php echo $form['__token'] ?>">
    . . .
</form>
```

5. Доработаем методы `edit` и `delete` контроллера `Controllers\Comments`, добавив в них проверки на принадлежность формы, из которой были отправлены данные, текущему сайту.

   Откроем модуль modules\controllers\comments.php с кодом этого контроллера и исправим код:

```
class Comments extends BaseController {
    . . .
    function edit(int $picture_index, int $comment_index) {
        $this->check_user($comment_index);
```

```
        if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            \Helpers\check_token($_POST);
            $comment_form =
                \Forms\Comment::get_normalized_data($_POST);
            . . .
        } else {
            . . .
            $comment_form =
                \Forms\Comment::get_initial_data($comment);
        }
        $comment_form['__token'] = \Helpers\generate_token();
        . . .
    }

    function delete(int $picture_index, int $comment_index) {
        $this->check_user($comment_index);
        if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            \Helpers\check_token($_POST);
            . . .
        } else {
            . . .
            $ctx = ['comment' => $comment,
                'picture' => $picture_index,
                'site_title' => 'Удаление комментария',
                '__token' => \Helpers\generate_token()];
            $this->render('comment_delete', $ctx);
        }
    }
}
```

6. Добавим в веб-форму удаления комментария скрытое поле с жетоном.

   Откроем модуль modules\templates\comment_delete.php с шаблоном страницы удаления комментария и вставим код:

```
. . .
<form method="post">
    <input type="hidden" name="__token"
    value="<?php echo $__token ?>">
    <input type="submit" value="Удалить">
</form>
. . .
```

7. Внесем аналогичные правки в:

   - методы `add`, `edit` и `delete` контроллера `Controllers\Images`;
   - шаблон-фрагмент формы для ввода изображения;
   - шаблон страницы для удаления изображения;
   - методы `delete`, `edit` и `edit_password` контроллера `Controllers\Account`;

- шаблон страницы удаления пользователя;
- шаблон страницы для правки сведений о пользователе;
- шаблон страницы для правки пользовательского пароля.

Войдем на сайт под именем какого-либо пользователя, в тестовых целях оставим комментарий под любой картинкой, исправим этот комментарий и удалим. Опубликуем произвольное изображение (например, хранящееся в файле Сгоревший светодиод.jpg из папки 21\!sources), исправим его и удалим. Наконец, попробуем исправить сведения о пользователе и его пароль.

23.4. Двухэтапная регистрация пользователей

В настоящее время программы распространения спама способны самостоятельно регистрироваться на сайте и выполнять рассылки от имени зарегистрированного пользователя. Пресечь это можно, введя двухэтапную регистрацию.

> *Двухэтапная регистрация* — регистрация, при которой после непосредственно занесения сведений о себе в веб-форму (первый этап) пользователь обязан активироваться (второй этап). Активация выполняется переходом по особому интернет-адресу, который присылается пользователю в электронном письме.

Интернет-адрес, по которому выполняется переход на страницу активации, должен содержать в своем составе:

- значение, однозначно указывающее на зарегистрировавшегося пользователя и задаваемое в открытом, незашифрованном виде (обычно это имя пользователя);
- то же самое значение, но уже зашифрованное — представленное в виде хэша.

Функцию `password_hash` (см. *разд. 21.2*) для шифрования в этом случае использовать нельзя. Шифруемое значение уже известно программе, рассылающей спам (поскольку она только что занесла его в веб-форму регистрации), и программа без проблем сможет вычислить хэш этого значения, вызвав упомянутую ранее функцию.

Для двухэтапной регистрации хэш заданного *значения* следует вычислять функцией `hash_hmac`:

```
hash_hmac(<обозначение используемого алгоритма>, <значение>,
          <секретный ключ>)
```

Она вычисляет хэш с применением заданного в виде строки *секретного ключа*. Поскольку этот ключ хранится в тайне, никакая вредоносная программа не сможет получить его для расчета хэша и, таким образом, пройти второй этап регистрации.

Обозначение используемого алгоритма указывается в виде строки. PHP поддерживает более 40 алгоритмов, перечень которых в виде индексированного массива можно получить вызовом функции `hash_hmac_algos()`:

```
$algos = hash_hmac_algos();
echo implode(' ', $algos);
      // md2 md4 md5 sha1 sha224 sha256 sha384 sha512/224 sha512/256
      // sha512 sha3-224 sha3-256 sha3-384 sha3-512 ripemd128 ripemd160
      // ripemd256 ripemd320 whirlpool tiger128,3 tiger160,3 tiger192,3
      // tiger128,4 tiger160,4 tiger192,4 snefru snefru256 gost
      // gost-crypto haval128,3 haval160,3 haval192,3 haval224,3
      // haval256,3 haval128,4 haval160,4 haval192,4 haval224,4
      // haval256,4 haval128,5 haval160,5 haval192,5 haval224,5
      // haval256,5
```

Не используйте ненадежные алгоритмы!

К ним относятся самые первые из перечня, начиная с md2 и заканчивая sha3-512.

Хэш, вычисленный функцией `hash_hmac`, имеет длину 64 символа.

Пример:

```
echo hash_hmac('ripemd256', 'superadmin', 'seKreTT_KeY');
      // 61d7f6422e9d29a8178c1677e9454df47da53fe7d8ac28f4becef673d016a972
```

> Другая особенность функции `hash_hmac` — она при последовательных вызовах с указанием одних и тех же параметров выдает одинаковые хэши. Поэтому, чтобы проверить значение на соответствие хэшу, достаточно вычислить на основе этого значения хэш и сравнить его с имеющимся.

23.5. Упражнение. Делаем двухэтапную регистрацию

Еще на *уроке 21* мы подготовили для этого почву, создав в таблице `users` поле `active`, хранящее признак того, является ли пользователь активным. По умолчанию это поле получает значение `FALSE` — таким образом, любой вновь зарегистрированный пользователь автоматически станет неактивным (в тестовых целях нам даже пришлось принудительно активировать их).

При регистрации пользователя мы отправим на указанный при регистрации адрес электронной почты письмо с интернет-адресом, на который следует перейти для выполнения активации — второго этапа регистрации. Этот интернет-адрес будет иметь формат:

/users/<*имя пользователя*>/account/activation/<*хэш имени пользователя*>/

Далее выполним перенаправление по интернет-адресу: **/register/complete/**, в результате чего появится страница с сообщением об успешном выполнении первого этапа регистрации и отправке письма.

Когда новый пользователь перейдет по интернет-адресу активации, на основе извлеченного из него *имени пользователя* будет вычислен хэш, и, если он совпадет с *хэшем*, извлеченным из того же адреса, произойдет активация пользователя. А на экране появится страница с сообщением об успешной или неуспешной активации.

Файлы сайта фотогалереи хранятся в папке 23\23.3\ex, а файл photogallery.sql с дампом базы данных photogallery — в папке 22\22.2\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Объявим константу Settings\SECRET_KEY, хранящую секретный ключ, который будет использован в вызовах функции hash_hmac.

   Откроем модуль modules\settings.php и добавим объявление этой константы (здесь и далее дополнения и правки в написанном ранее коде выделены полужирным шрифтом):

```
namespace Settings {
    . . .
    const SECRET_KEY = 'pHoTo+GAlleRy';
}
```

2. Добавим в маршрутизатор код, распознающий интернет-адреса страницы с сообщением об отправке письма и страницы активации. В первом случае будет вызван метод register_complete контроллера Controllers\Login, во втором — метод activate контроллера Controllers\Account. Методу activate будут передаваться в качестве параметров имя пользователя и хэш.

   Откроем модуль маршрутизатора modules\router.php и вставим нужный код:

```
. . .
} else if ($request_path == 'register') {
    . . .
} else if ($request_path == 'register/complete') {
    $ctr = new \Controllers\Login();
    $ctr->register_complete();
} else if (preg_match('/^users\/(\w+)\/account\/activation\/(\w+)$/',
    $request_path, $result) == 1) {
    $ctr = new \Controllers\Account();
    $ctr->activate($result[1], $result[2]);
} else if ($request_path == '') {
. . .
```

3. Исправим метод register контроллера Controllers\Login таким образом, чтобы он после регистрации пользователя отсылал ему письмо с предложением активации и переходил на страницу с сообщением об успешной регистрации.

   Откроем модуль modules\controllers\login.php и исправим код:

```
class Login extends BaseController {
    . . .
    function register() {
        if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            . . .
            if (!isset($reg_form['__errors'])) {
                . . .
                $reg_form['active'] = TRUE;
```

```
                $users = new \Models\User();
                $users->insert($reg_form);
                $values = ['title' => \Settings\SITE_NAME,
                    'name' => $reg_form['name'],
                    'url' => 'http://' . $_SERVER['SERVER_NAME'] .
                    '/users/' . $reg_form['name'] .
                    '/account/activation/' .
                    hash_hmac('ripemd256', $reg_form['name'],
                    \Settings\SECRET_KEY) . '/'];
                \Helpers\send_mail($reg_form['email'],
                    'activation_subject', 'activation_body',
                    $values);
                \Helpers\redirect('/login');
                \Helpers\redirect('/register/complete/');
            }
        } else
            $reg_form = \Forms\Register::get_initial_data([]);
        . . .
    }
}
```

Не забудем удалить выражение, принудительно активирующее нового пользователя записью значения `TRUE` в поле `active` таблицы `users`.

Шаблоны темы (activation_subject.txt) и тела (activation_body.txt) письма об активации мы напишем позже.

4. Добавим в тот же контроллер `Controllers\Login` метод `register_complete`, выводящий страницу с сообщением об отправке письма:

```
class Login extends BaseController {
    . . .
    function register_complete() {
        $ctx = ['site_title' => 'Регистрация завершена'];
        $this->render('register_complete', $ctx);
    }
}
```

5. Создадим модуль modules\templates\register_complete.php и запишем в него код шаблона страницы с сообщением об успешной регистрации и отправке письма с предложением активации (листинг 23.1).

Листинг 23.1. Модуль modules\templates\register_complete.php

```
<?php require \Helpers\get_fragment_path('__header') ?>
<h2>Регистрация завершена</h2>
<p>На указанный Вами при регистрации адрес электронной
почты выслано письмо с предложением активации.</p>
<?php require \Helpers\get_fragment_path('__footer') ?>
```

6. Создадим текстовый файл modules\templates\activation_subject.txt и запишем в него шаблон темы письма с предложением активации (листинг 23.2).

Листинг 23.2. Файл modules\templates\activation_subject.txt

```
%title%. Пожалуйста, активируйтесь!
```

7. Создадим текстовый файл modules\templates\activation_body.txt и сохраним в нем шаблон тела письма с предложением активации (листинг 23.2).

Листинг 23.3. Файл modules\templates\activation_body.txt

```
Здравствуйте, уважаемый %name%!

Вам необходимо активироваться на сайте "%title%".

Для этого перейдите по интернет-адресу:
%url%

До свидания!

С уважением,
администрация сайта "%title%".
```

8. Дополним контроллер `Controllers\Account` методом `activate`, который выполнит активацию пользователя.

   Откроем модуль modules\controllers\account.php с кодом этого контроллера и внесем нужные дополнения.

```
class Account extends BaseController {
    . . .
    function activate(string $username, string $token) {
        $users = new \Models\user();
        $user = $users->get($username, 'name');
        if (!$user)
            $s = 'Пользователя нет в списке';
        else {
            if ($user['active'])
                $s = 'Пользователь уже активирован';
            else {
                $t = hash_hmac('ripemd256', $username,
                    \Settings\SECRET_KEY);
                if ($t != $token)
                    $s = 'Неверный интернет-адрес';
                else {
                    $users->update(['active' => TRUE], $user['id']);
```

```
                    $s = 'Активация прошла успешно';
                }
            }
        }
        $ctx = ['message' => $s,
            'site_title' => 'Активация пользователя'];
        $this->render('activate', $ctx);
    }
}
```

Последовательно проверяем, есть ли такой пользователь в списке зарегистрированных, не активирован ли он уже и совпадает ли хэш, вычисленный на основе его имени, с полученным в составе интернет-адреса хэшем. Если это так, записываем в поле `active` этого пользователя значение `TRUE`, делая его активным. В любом случае выводим страницу с сообщением об успешной или неуспешной активации пользователя.

9. Создадим модуль modules\templates\activate.php и сохраним в нем шаблон страницы с сообщением об успехе или неуспехе активации (листинг 23.4).

Листинг 23.4. Модуль modules\templates\activate.php

```
<?php require \Helpers\get_fragment_path('__header') ?>
<h2>Активация пользователя</h2>
<p><?php echo $message ?></p>
<?php require \Helpers\get_fragment_path('__footer') ?>
```

Откроем модуль настроек modules\settings.php, проверим, записаны ли в нем реальные параметры SMTP-сервера используемой почтовой службы, и изменим значение константы `Settings\MAIL_SEND` на `TRUE`, чтобы сделать возможной отправку писем.

Откроем сайт и зарегистрируем нового пользователя, указав произвольные имя, пароль и наш адрес электронной почты. Дождемся получения письма с предложением активации, скопируем присутствующий в нем интернет-адрес в буфер обмена, вставим в строку адреса веб-обозревателя и выполним переход. После активации пользователя попробуем войти на сайт от его имени. Напоследок удалим этого пользователя.

Урок 24
Веб-службы REST

> *Веб-служба* — серверное веб-приложение, выдающее в качестве результата работы данные, представленные в каком-либо компактном формате, — например, JSON. Эти данные далее обрабатываются клиентскими приложениями.

> *Бэкенд* — совокупность веб-служб, работающих в составе одного сайта и обслуживающих фронтенд.

> *Фронтенд* — клиентские приложения, взаимодействующие с бэкендом и представляющие пользователю полученные от бэкенда данные.

В качестве фронтенда может выступать обычная веб-страница с веб-сценариями, написанными на языке JavaScript, оконное Windows-приложение, мобильное приложение для операционной системы Google Android или Apple iOS и др.

Разделение веб-сайтов на фронтенд и бэкенд имеет следующие преимущества:

- одни и те же данные, выданные одним и тем же бэкендом, разными фронтендами могут быть представлены по-разному;
- уменьшение нагрузки на серверный компьютер, так как преобразование данных в формат, пригодный к выводу, теперь выполняется на клиентском компьютере фронтендом.

Недостатки:

- усложнение программирования, в некоторых случаях — значительное;
- бэкенд и фронтенд во многих случаях пишутся на разных языках.

24.1. Формат JSON и его поддержка в PHP

> *JSON* (JavaScript Object Notation, объектная нотация JavaScript) — текстовый формат обмена данными, основанный на объектной нотации JavaScript[1].

Данные в формате JSON очень похожи на JavaScript-код, объявляющий объект класса `Object`. Различия между JSON и JavaScript состоят в следующем:

[1] Описание языка JavaScript выходит за рамки этой книги.

- строки заключаются только в двойные кавычки (одинарные не допускаются);
- имена свойств также заключаются в двойные кавычки;
- поддерживаются лишь следующие типы данных:
  - строки;
  - числа;
  - логические величины;
  - массивы (заключаются в квадратные скобки);
  - служебные объекты класса `Object` (заключаются в фигурные скобки);
  - `null`.

  Временны́е отметки в виде объектов класса `Date`, функции и значение `undefined` не поддерживаются.

Пример записи сведений об отдельном изображении в формате JSON:

```
{"id":2,"filename":"201802131844.jpg","title":"Блинная скульптура"}
```

Пример записи списка изображений:

```
[{"id":1,"filename":"200804282020.jpg","title":"Цветы кактуса"},
{"id":2,"filename":"201802131844.jpg","title":"Блинная скульптура"},
{"id":3,"filename":"201506152037.jpg","title":"Памятник сусликам"}]
```

Для кодирования заданного *значения* в формат JSON применяется встроенная в PHP функция `json_encode`:

```
json_encode(<значение>[, <параметры кодирования>])
```

Значение может быть любого типа, кроме ресурса. В качестве *параметров кодирования* практически всегда указывается константа `JSON_UNESCAPED_UNICODE`, предписывающая не преобразовывать символы Unicode в коды, что значительно уменьшает объем результирующих JSON-данных.

Примеры:

```
$source_data = ['id' => 1, 'filename' => '200804282020.jpg',
    'title' => 'Цветы кактуса'];
echo json_encode($source_data);
                        // {"id":1,"filename":"200804282020.jpg",↳
                        // "title":"\u0426\u0432\u0435\u0442\u044b ↳
                        // \u043a\u0430\u043a\u0442\u0443\u0441\u0430"}
echo json_encode($source_data, JSON_UNESCAPED_UNICODE);
        // {"id":1,"filename":"200804282020.jpg","title":"Цветы кактуса"}
$source_data = [
    ['id' => 1, 'filename' => '200804282020.jpg',
    'title' => 'Цветы кактуса'],
    ['id' => 2, 'filename' => '201802131844.jpg',
    'title' => 'Блинная скульптура'],
```

```
    ['id' => 3, 'filename' => '201506152037.jpg',
    'title' => 'Памятник сусликам']
    ];
echo json_encode($source_data, JSON_UNESCAPED_UNICODE);
                              // [{"id":1,"filename":"200804282020.jpg",↵
                              // "title":"Цветы кактуса"},↵
                              // {"id":2,"filename":"201802131844.jpg",↵
                              // "title":"Блинная скульптура"},↵
                              // {"id":3,"filename":"201506152037.jpg",↵
                              // "title":"Памятник сусликам"}]
```

Перед отправкой сгенерированных данных в формате JSON следует добавить в серверный ответ следующие заголовки:

- `Content-Type: application/json; charset=UTF-8`

  Указывает, что пересылаются данные в формате JSON в кодировке UTF-8;

- `Cache-Control: no-cache, no-store, must-revalidate`

  Запрещает веб-обозревателю кэшировать полученные JSON-данные. Позволяет предотвратить так называемое *застревание в кэше*, когда веб-обозреватель продолжает использовать хранящуюся в кэше устаревшую редакцию файла, игнорируя поступившую от бэкенда более новую редакцию.

Пример:

```
// Добавляем в ответ необходимые заголовки
header('Content-Type: application/json; charset=UTF-8');
header('Cache-Control: no-cache, no-store, must-revalidate');
// Подготавливаем отправляемые данные
$source_data = . . .;
// Преобразуем их в формат JSON и отправляем
echo json_encode($source_data, JSON_UNESCAPED_UNICODE);
```

Полезно знать...

Функция `json_encode` поддерживает и другие параметры кодирования, задаваемые особыми встроенными константами. Однако они применяются крайне редко и в очень специфических случаях.

24.2. Стиль REST

> *REST* (Representational State Transfer, передача состояния представления) — стиль взаимодействия между фронтендом и бэкендом, особенности которого описаны далее.

- Для передачи данных используются протоколы HTTP и HTTPS.
- Любой фрагмент данных, который фронтенд может запросить у бэкенда, однозначно идентифицируется интернет-адресом.

Например, изображение с заданным *номером* идентифицируется интернет-адресом формата: **/api/images/<*номер изображения*>/**, а список комментариев к нему — интернет-адресом формата: **/api/images/<*номер изображения*>/comments/**.

- Действие, выполняемое над фрагментом данных, идентифицируется HTTP-методом, использованным при отправке запроса. Поддерживаются следующие методы:
  - `GET` — получение фрагмента данных с сервера;
  - `POST` — создание на сервере нового фрагмента данных;
  - `PUT` — изменение всего содержимого фрагмента данных, хранящегося на сервере;
  - `PATCH` — изменение части фрагмента данных на сервере.

    На практике в веб-службах обычно реализуют универсальную операцию изменения содержимого фрагмента данных, обозначаемую методом `PUT` или `PATCH`;
  - `DELETE` — удаление фрагмента данных с сервера.

  Например, для загрузки изображения № 7 отправим данные по интернет-адресу **/api/images/7/** методом `GET`, для изменения сведений о нем — методом `PUT` или `PATCH`, а для удаления — методом `DELETE`.
- Успех или неуспех выполнения какой-либо операции обозначается отправкой фронтенду ответа с установленным кодом состояния:
  - при успешной отправке данных — 200;
  - при успешном создании фрагмента данных — 201;
  - при успешной правке фрагмента данных — 200;
  - при успешном удалении фрагмента данных — 204;
  - при отсутствии запрашиваемого фрагмента данных — 404;
  - при отправке на сервер некорректных данных — 400;
  - при отсутствии у пользователя прав на доступ к запрашиваемым данным — 403;
  - если использованный при запросе HTTP-метод не поддерживается — 405;
  - в случае возникновения ошибки в самой программе — 503.

  Ответ может как содержать дополнительные сведения об ошибке (например, текстовое сообщение), так и быть пустым — в этом случае об успехе или неуспехе сообщит код состояния.
- Данные о клиенте хранятся исключительно на стороне клиента, но не на стороне сервера. Это касается и сведений о том, выполнил ли клиент вход на сайт.

Основное преимущество стиля REST состоит в том, что написанные в соответствии с ним бэкенды используют уже существующую инфраструктуру (протоколы HTTP

и HTTPS, службы DNS, обычные веб-серверы) и программные платформы, применяемые для разработки и эксплуатации обычных сайтов (например, PHP).

24.2.1. Решение проблемы с передачей данных HTTP-методами *PUT*, *PATCH* и *DELETE*. Подмена HTTP-метода

Значения любых GET-параметров, присутствующих в составе интернет-адреса, платформа PHP успешно извлечет, декодирует и поместит в ассоциативный массив `$_GET`. Это произойдет даже в случае, если запрос был выполнен с применением метода `POST`.

Однако значения POST-параметров декодируются и помещаются в массив `$_POST` только в том случае, если запрос был выполнен методом `POST`. При получении запроса, выполненного методом `PUT`, `PATCH` или `DELETE`, PHP проигнорирует эти параметры, и переданные с ними значения будут потеряны.

Решить эту проблему можно, реализовав трюк, называемый *подменой HTTP-метода*. Он заключается в следующем:

- веб-обозреватель, передавая занесенные в веб-форму данные веб-службе, добавляет в них POST-параметр, хранящий наименование нужного HTTP-метода: `PUT`, `PATCH` или `DELETE`;
- эти данные пересылаются методом `POST`, «понятным» платформе PHP;
- веб-служба, получив данные, сначала проверяет наименование использованного HTTP-метода, обратившись к элементу `REQUEST_METHOD` массива `$_SERVER`;
- если запрос был выполнен методом `POST`, проверяется значение упомянутого ранее POST-параметра:
  - если он присутствует — наименование метода извлекается из него;
  - в противном случае — данные были переданы методом `POST`.

Пример:

```
// Подразумевается, что наименование метода было передано
// в POST-параметре __method
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    if (isset($_POST['__method'])) {
        $method = $_POST['__method'];
        if ($method == 'PUT') {
            // Данные были переданы методом PUT
        } else if ($method == 'PATCH') {
            // Данные были переданы методом PATCH
        } else {
            // Данные были переданы методом DELETE
        }
```

```
    } else {
        // Данные были переданы методом POST
    }
} else {
    // Данные были переданы методом GET
}
```

24.3. Программирование фронтенда

В коде фронтенда для взаимодействия с бэкендом применяется технология AJAX, реализуемая объектом класса `XMLHttpRequest` языка JavaScript.

Далее приведены фрагменты JavaScript-кода, используемые для выполнения различных действий над фрагментами данных: получения, создания, правки и удаления.

24.3.1. Получение данных

В качестве примера рассмотрим получение от бэкенда списка изображений с интернет-адреса **/api/images/**. Предположим, что бэкенд находится на локальном хосте.

```
// Создаем объект класса XMLHttprequest
const ajax = new XMLHttpRequest();
// Задаем параметры отправки запроса: HTTP-метод GET, интернет-адрес
// /api/images/ и асинхронный характер запроса (значение true
// в третьем параметре)
ajax.open('GET', '/api/images/', true);
// Задаем обработчик события readystatechange, возникающего при изменении
// состояния загрузки данных
ajax.addEventListener('readystatechange', onResponse);
// Отправляем запрос бэкенду
ajax.send();

// Функция-обработчик события readystatechange, в которой будут выполнены
// обработка и вывод загруженных данных
function onResponse() {
    // Проверяем состояние загрузки данных. Если оно равно 4, данные
    // успешно загружены и могут быть извлечены для обработки
    if (this.readyState == 4) {
        // Проверяем код состояния ответа
        if (this.status == 200) {
            // Данные успешно загружены
            // Извлекаем их и преобразуем к виду, пригодному для
            // программной обработки
            jsonData = JSON.parse(this.responseText);
```

```
            // Выводим данные
        } else if (this.status == 404) {
            // Ошибка: запрошенный фрагмент данных не существует
        } else if (this.status == 403) {
            // Ошибка: у пользователя нет доступа к запрошенному
            // фрагменту данных
        } else {
            // Ошибка в коде веб-службы
        }
    }
}
```

Аналогичным образом можно получить, например, изображение № 7, написав соответствующий интернет-адрес в вызове метода `open`:

```
ajax.open('GET', '/api/images/7/', true);
```

24.3.2. Добавление фрагмента данных

Добавление нового фрагмента данных на сервер выполняется так же, только в вызове метода `open` нужно указать метод отправки данных `POST`, а в вызове метода `send` — объект класса `FormData`, хранящий добавляемые данные.

Далее приведен пример добавления нового комментария от имени пользователя № 2 к изображению № 5. Добавляемые данные отправляются по интернет-адресу формата: **/api/images/<*номер изображения*>/comments/**, идентифицирующему список комментариев к изображению с указанным *номером*.

```
ajax.open('POST', '/api/images/5/comments/', true);
ajax.addEventListener('readystatechange', onResponse);
// Создаем объект класса FormData
const fd = new FormData();
// Добавляем в него отправляемые данные: содержание комментария
// (поле content) и номер пользователя-автора (поле user)
fd.append('content', 'Отличная картинка!');
fd.append('user', 2);
// Отправляем запрос с добавляемыми данными
ajax.send(fd);

function onResponse() {
    if (this.readyState == 4) {
        if (this.status == 201) {
            // Новый комментарий успешно создан на сервере
            // Сигнализируем об этом пользователю
            . . .
        }
    }
}
```

24.3.3. Правка фрагмента данных

Правка фрагмента данных выполняется так же, как и добавление, только в качестве метода отправки данных нужно использовать PUT или PATCH. Поскольку эти методы не поддерживаются PHP, следует выполнить подмену HTTP-метода (см. *разд. 24.2.1*).

Далее показан пример кода, исправляющего текст комментария № 1 к изображению № 4. Данные отправляются по интернет-адресу формата: **/api/images/<*номер изображения*>/comments/<*номер комментария*>/**.

```
// Отправляем данные методом POST, поскольку методы PUT и PATCH
// платформа PHP не «понимает»
ajax.open('POST', '/api/images/4/comments/1/', true);
ajax.addEventListener('readystatechange', onResponse);
const fd = new FormData();
fd.append('content', 'Плохая картинка...');
// Выполняем подмену HTTP-метода, добавив в состав передаваемых данных
// POST-параметр __method с указанием нужного метода — PATCH (также можно
// указать метод PUT)
fd.append('__method', 'PATCH');
ajax.send(fd);

function onResponse() {
    if (this.readyState == 4) {
        if (this.status == 200) {
            // Комментарий успешно исправлен
            // Сигнализируем об этом пользователю
            . . .
        }
    }
}
```

24.3.4. Удаление фрагмента данных

Удаление фрагмента данных выполняется отправкой запроса методом DELETE (также посредством подмены метода). Вот так можно удалить комментарий № 10 к изображению № 2:

```
ajax.open('POST', '/api/images/2/comments/10/', true);
ajax.addEventListener('readystatechange', onResponse);
const fd = new FormData();
fd.append('__method', 'DELETE');
ajax.send(fd);

function onResponse() {
    if (this.readyState == 4) {
        if (this.status == 204) {
```

```
            // Комментарий успешно удален
            // Сообщаем об этом пользователю
            . . .
        }
    }
}
```

24.3.5. Реализация разграничения доступа

Веб-службы, работающие в стиле REST, не хранят на сервере никаких сведений о клиентах, в том числе признак того, выполнил ли пользователь вход на сайт. Следовательно, в коде веб-службы мы не сможем проверить, вошел ли пользователь на сайт, и есть ли у него привилегии на доступ к тому или иному фрагменту данных.

Чтобы веб-служба смогла выполнить такую проверку, фронтенд обязан пересылать с каждым запросом сведения о пользователе — его имя и пароль. На практике, чтобы сохранить пароль в тайне от возможных злоумышленников, реализуется следующий сценарий:

1. Пользователь, желающий войти на сайт, вводит имя и пароль в веб-форму.
2. Фронтенд пересылает эти данные бэкенду.
3. Бэкенд проверяет, есть ли такой пользователь в списке, и, если есть, создает жетон — вычисленный на основе имени хэш (применяя функцию `hash_hmac`) — и отправляет его фронтенду.
4. Фронтенд сохраняет полученный жетон вместе с именем пользователя на стороне клиента (лучше всего — в локальном или сессионном хранилище, входящим в состав программных средств HTML API).
5. При выполнении следующих запросов сохраненные имя и жетон вместе с другими данными (например, содержанием добавляемого комментария) пересылаются бэкенду, чтобы он смог проверить привилегии пользователя.

> *Жетонная авторизация* — проверка, имеет ли пользователь привилегии на доступ к данным, по имени пользователя и рассчитанному на его основе жетону.

Вот пример отправки первоначального запроса бэкенду с просьбой проверить, существует ли пользователь с заданными именем и паролем, и при положительном исходе проверки прислать идентифицирующий его жетон:

```
ajax.open('POST', '/api/login/', true);
ajax.addEventListener('readystatechange', onResponse);
const fd = new FormData();
// Отправляем имя и пароль пользователя
fd.append('name', 'admin');
fd.append('password', '123456');
ajax.send(fd);
```

```
function onResponse() {
    if (this.readyState == 4) {
        if (this.status == 200) {
            jsonData = JSON.parse(this.responseText);
            // Сохраняем в локальном хранилище полученные от бэкенда
            // имя пользователя и жетон
            localStorage.setItem('userName', jsonData.name);
            localStorage.setItem('userToken', jsonData.token);
            . . .
        }
    }
}
```

Пример отправки сохраненных имени и жетона пользователя при добавлении нового комментария:

```
ajax.open('POST', '/api/images/5/comments/', true);
. . .
const fd = new FormData();
fd.append('__user_name', localStorage.getItem('userName'));
fd.append('__user_token', localStorage.getItem('userToken'));
fd.append('content', 'Отличная картинка!');
fd.append('user', 2);
ajax.send(fd);
```

Процедура выхода будет выполняться исключительно на стороне клиента и заключаться лишь в удалении из локального хранилища сохраненных имени и жетона:

```
localStorage.removeItem('userName');
localStorage.removeItem('userToken');
```

> ***Локальное хранилище хранит данные в течение продолжительного времени***
>
> Для повышения безопасности можно хранить имя и пароль в сессионном хранилище, очищающемся при закрытии текущего окна веб-обозревателя:
>
> ```
> sessionStorage.setItem('userName', 'admin');
> sessionStorage.setItem('userToken', jsonData.token);
> ```

24.4. Упражнение. Пишем бэкенд

Он включит в себя три веб-службы:

- первая будет работать с изображениями.

  Она отправит фронтенду сведения об изображении с заданным *номером* — при обращении к интернет-адресу формат: **/api/images/<*номер изображения*>/** HTTP-методом `GET`. Для простоты будут отправляться лишь название, описание и полный интернет-адрес файла с изображением;

- вторая будет работать с комментариями к изображению с заданным *номером*:
  - выведет список комментариев — при обращении по интернет-адресу: **/api/images/<*номер изображения*>/comments/** HTTP-методом `GET`. Для простоты в этот список войдут лишь номера комментариев, их содержание и имена пользователей, оставивших их;
  - добавит новый комментарий — при обращении по тому же адресу методом `POST`;
  - исправит комментарий с указанным *номером* — **/api/images/<*номер изображения*>/comments/<*номер комментария*>/** методом `PUT` или `PATCH` (разумеется, с подменой метода);
  - исправит комментарий с указанным *номером* — при обращении по тому же адресу методом `DELETE`;
- третья станет работать с пользователями.

  Она проверит, существует ли зарегистрированный пользователь с полученными от фронтенда именем и паролем, и отправит вычисленный на основе имени электронный жетон — при обращении по адресу **/api/login/** методом `POST`.

Для тестирования написанного бэкенда применим готовый, написанный автором фронтенд, файлы которого хранятся в папке 24\!sources[1].

Файлы сайта фотогалереи хранятся в папке 23\23.5\ex, а файл photogallery.sql с дампом базы данных `photogallery` — в папке 22\22.2\ex сопровождающего книгу электронного архива (см. *приложение 3*).

1. Объявим в базовом классе модели `Models\Model` метод `get_all`, принимающий те же параметры, что и метод `select`, объявленный в *разд. 15.6*, и возвращающий массив всех записей, извлеченных из таблицы. Он понадобится нам для извлечения списка комментариев.

   Откроем модуль modules\models\model.php, хранящий код базовой модели, в текстовом редакторе и впишем объявление этого метода (здесь и далее добавления и правки в написанном ранее коде выделены полужирным шрифтом):

```
class Model implements \Iterator {
    . . .
    function get_all($fields = '*', $links = NULL, $where = '',
        $params = NULL, $order = '', $offset = NULL, $limit = NULL,
        $group = '', $having = '') {
        $this->select($fields, $links, $where, $params, $order,
        $offset, $limit, $group, $having);
        return $this->query->fetchAll(\PDO::FETCH_ASSOC);
        }
}
```

[1] В этой книге не описываются принципы, на основе которых писался фронтенд. Они достаточно тривиальны и приведены в любом руководстве по JavaScript.

2. Объявим функцию `Helpers\api_headers()`, задающую в серверном ответе нужные заголовки: обозначение формата и кодировки отправляемых данных и запрет веб-обозревателю их кэшировать.

   Откроем модуль modules\helpers.php и запишем в него нужный код:

```
namespace Helpers {
    . . .
    function api_headers() {
        header('Content-Type: application/json; charset=UTF-8');
        header('Cache-Control: no-cache, no-store, must-revalidate');
    }
}
```

3. Откроем модуль маршрутизатора modules\router.php и добавим код, распознающий описанные ранее интернет-адреса:

```
. . .
} else if (preg_match('/^users\/(\w+)\/account\/activation\/(\w+)$/',
    . . .
} else if (preg_match('/^api\/images\/(\d+)$/', $request_path,
    $result) == 1) {
    if ($_SERVER['REQUEST_METHOD'] == 'GET') {
        $ctr = new \Controllers\APIImages();
        $ctr->item($result[1]);
    } else
        http_response_code(405);
} else if (preg_match('/^api\/images\/(\d+)\/comments$/',
    $request_path, $result) == 1) {
    if ($_SERVER['REQUEST_METHOD'] == 'GET') {
        $ctr = new \Controllers\APIComments();
        $ctr->list($result[1]);
    } else if ($_SERVER['REQUEST_METHOD'] == 'POST') {
        $ctr = new \Controllers\APIComments();
        $ctr->add($result[1]);
    } else
        http_response_code(405);
} else if (preg_match('/^api\/images\/(\d+)\/comments\/(\d+)$/',
    $request_path, $result) == 1) {
    if ($_SERVER['REQUEST_METHOD'] == 'POST' &&
        ($_POST['__method'] == 'PUT' ||
        $_POST['__method'] == 'PATCH')) {
        $ctr = new \Controllers\APIComments();
        $ctr->edit($result[1], $result[2]);
    } else if ($_SERVER['REQUEST_METHOD'] == 'POST' &&
        $_POST['__method'] == 'DELETE') {
        $ctr = new \Controllers\APIComments();
        $ctr->delete($result[1], $result[2]);
```

```
    } else
        http_response_code(405);
} else if ($request_path == 'api/login') {
    if ($_SERVER['REQUEST_METHOD'] == 'POST') {
        $ctr = new \Controllers\APILogin();
        $ctr->check();
    } else
        http_response_code(405);
} else if ($request_path == '') {
. . .
```

Извлечение изображения с заданным номером будет выполнять метод `item` контроллера `Controllers\APIImages`, проверку существования пользователя с заданными именем и паролем и генерирование жетона на основе имени — метод `check` контроллера `Controllers\APILogin`, выдачу списка комментариев, добавление, правку и удаление комментария — методы: `list`, `add`, `edit` и `delete` контроллера `Controllers\APIComments`. Все эти контроллеры мы напишем очень скоро.

При попытке обратиться по какому-либо из интернет-адресов, поддерживаемых нашим бэкендом, с использованием недопустимого HTTP-метода (например, при попытке получения изображения методом `POST`) мы выдадим фронтенду пустой серверный ответ с кодом состояния 405 (такой HTTP-метод не поддерживается).

4. Создадим модуль modules\controllers\apiimages.php и сохраним в нем код класса контроллера `Controllers\APIImages` из листинга 24.1.

Листинг 24.1. Модуль modules\controllers\apiimages.php

```
<?php
namespace Controllers;
class APIImages extends BaseController {
    function item(int $index) {
        \Helpers\api_headers();
        $picts = new \Models\Picture();
        $pict = $picts->get_or_404($index, 'id',
            'title, description, filename');
        $pict['url'] = 'http://' . $_SERVER['SERVER_NAME'] .
            \Settings\IMAGE_PATH . $pict['filename'];
        echo json_encode($pict, JSON_UNESCAPED_UNICODE);
    }
}
```

Сведения об изображении, отправляемые фронтенду, представляют собой ассоциативный массив, который будет закодирован в формат JSON вызовом функции `json_encode`. Значения элементов `id`, `title` и `description` этого массива из-

влекаются из базы данных, а значение элемента `url` (интернет-адрес изображения) формируется программно.

Перед отправкой данных не забываем вызвать объявленную ранее функцию `Helpers\api_headers`, которая задаст в ответе нужные заголовки.

5. Создадим модуль **modules\controllers\apilogin.php** и сохраним в нем код класса контроллера `Controllers\APILogin` из листинга 24.2.

Листинг 24.2. Модуль modules\controllers\apilogin.php

```
<?php
namespace Controllers;
class APILogin extends BaseController {
    function check() {
        \Helpers\api_headers();
        $login_form = \Forms\Login::get_normalized_data($_POST);
        if (isset($login_form['__errors']) ||
            !\Forms\Login::verify_user($login_form)) {
            http_response_code(400);
            $user = ['errors' => $login_form['__errors']];
        } else {
            $user = ['name' => $login_form['name'],
                'token' => hash_hmac('ripemd256',
                $login_form['name'], \Settings\SECRET_KEY)];
        }
        echo json_encode($user, JSON_UNESCAPED_UNICODE);
    }
}
```

Для проверки существования в списке зарегистрированного пользователя с заданными именем и паролем мы используем статический метод `verify_user` формы `Forms\Login`, написанной в *разд. 21.3*.

Если фронтенд передал некорректные данные, или указанного пользователя в списке нет, мы устанавливаем код состояния 400 (заданы некорректные данные) и отсылаем фронтенду ассоциативный массив с элементом `errors`, хранящим массив с сообщениями об ошибках. Фронтенд может извлечь эти сообщения и показать их пользователю.

Если же заданный пользователь найден, генерируем на основе его имени электронный жетон и вместе с именем отсылаем фронтенду, чтобы он в дальнейшем мог выполнить операции, доступные лишь зарегистрированным пользователям.

Код контроллера `Controllers\APIComments` весьма велик, поэтому будем писать его по частям.

6. Создадим модуль **modules\controllers\apicomments.php** и запишем в него начальную часть кода контроллера `Controllers\APIComments`, содержащую только метод `list` (листинг 24.3).

Листинг 24.3. Модуль modules\controllers\apicomments.php

```
<?php
namespace Controllers;
class APIComments extends BaseController {
    function list(int $picture_index) {
        \Helpers\api_headers();
        $comments = new \Models\Comment();
        $coms = $comments->get_all('comments.id, contents, ' .
            'users.name AS user_name', ['users'], 'picture = ?',
            [$picture_index]);
        echo json_encode($coms, JSON_UNESCAPED_UNICODE);
    }
}
```

Для выборки всех комментариев к изображению с указанным номером применяем объявленный ранее метод `get_all` базовой модели `Models\Model`.

Чтобы выполнить операцию, разрешенную лишь зарегистрированным пользователям, фронтенд, наряду с прочими данными, пришлет имя и жетон пользователя. Бэкенду следует проверить, существует ли такой пользователь в списке.

7. Объявим в контроллере `Controllers\APIComments` частный метод `get_user()`, проверяющий корректность полученного жетона и выдающий пользователя по полученному имени. Если жетон некорректен или такого пользователя нет, метод возвращает `FALSE`.

   Добавим в контроллер `Controllers\APIComments` объявление метода `get_user`:

```
class APIComments extends BaseController {
    . . .
    private function get_user() {
        $username = $_POST['name'];
        if (hash_hmac('ripemd256', $username,
            \Settings\SECRET_KEY) != $_POST['token'])
            return FALSE;
        else {
            $users = new \Models\User();
            return $users->get($username, 'name');
        }
    }
}
```

8. Добавим в контроллер `Controllers\APIComments` объявление метода `add`:

```
class APIComments extends BaseController {
    . . .
    function add(int $picture_index) {
        \Helpers\api_headers();
```

```
        if (!($user = $this->get_user()))
            http_response_code(403);
        else {
            $comment_form =
                \Forms\Comment::get_normalized_data($_POST);
            if (isset($comment_form['__errors'])) {
                http_response_code(400);
                $comment = ['errors' => $comment_form['__errors']];
                echo json_encode($comment, JSON_UNESCAPED_UNICODE);
            } else {
                $comment_form['picture'] = $picture_index;
                $comment_form['user'] = $user['id'];
                $comments = new \Models\Comment();
                $comments->insert($comment_form);
                http_response_code(201);
            }
        }
    }
}
```

Вызовом метода `get_user` получаем пользователя, чьи имя и жетон передал фронтенд. Если такого пользователя в списке нет, возвращаем фронтенду пустой ответ с кодом состояния 403 (недостаточно прав для выполнения операции).

Если пользователь с заданным именем существует, проверяем корректность полученных от фронтенда данных (а именно, содержания комментария), пользуясь написанной ранее формой `Forms\Comment`. Если данные некорректны, отправляем фронтенду ответ со списком ошибок и кодом состояния 400. Если же данные прошли проверку, сохраняем новый комментарий в базе и возвращаем пустой ответ с кодом состояния 201 (фрагмент данных успешно создан).

9. Добавим в контроллер `Controllers\APIComments` методы `edit` и `delete`:

```
class APIComments extends BaseController {
    . . .
    function edit(int $picture_index, int $comment_index) {
        \Helpers\api_headers();
        if (!($user = $this->get_user()))
            http_response_code(403);
        else {
            $comments = new \Models\Comment();
            $comment = $comments->get($comment_index);
            if (!$comment)
                http_response_code(404);
            else if ($comment['user'] != $user['id'])
                http_response_code(403);
            else {
                $comment_form =
                    \Forms\Comment::get_normalized_data($_POST);
```

```
                if (isset($comment_form['__errors'])) {
                    http_response_code(400);
                    $comment =
                        ['errors' => $comment_form['__errors']];
                    echo json_encode($comment,
                        JSON_UNESCAPED_UNICODE);
                } else {
                    $comments = new \Models\Comment();
                    $comments->update($comment_form, $comment_index);
                    http_response_code(200);
                }
            }
        }
    }

    function delete(int $picture_index, int $comment_index) {
        \Helpers\api_headers();
        if (!($user = $this->get_user()))
            http_response_code(403);
        else {
            $comments = new \Models\Comment();
            $comment = $comments->get($comment_index);
            if (!$comment)
                http_response_code(404);
            else if ($comment['user'] != $user['id'])
                http_response_code(403);
            else {
                $comments = new \Models\Comment();
                $comments->delete($comment_index);
                http_response_code(204);
            }
        }
    }
}
```

В обоих методах, помимо описанных ранее проверок, также выясняем, является ли пользователь, чьи имя и жетон были получены от фронтенда, автором исправляемого или удаляемого комментария. Если это не так, возвращаем пустой ответ с кодом состояния 403.

После успешного сохранения исправленного комментария отправляем пустой ответ с кодом состояния 200, а после успешного удаления — с кодом состояния 204 (фрагмент данных успешно удален).

10. Найдем в папке 24\!sources папку frontend, содержащую код тестового фронтенда, и скопируем ее в корневую папку сайта.

Наберем в веб-обозревателе интернет-адрес **http://localhost/frontend/index.html**, чтобы открыть страницу фронтенда. Проверим, выводится ли изображение и комментарии к нему (рис. 24.1).

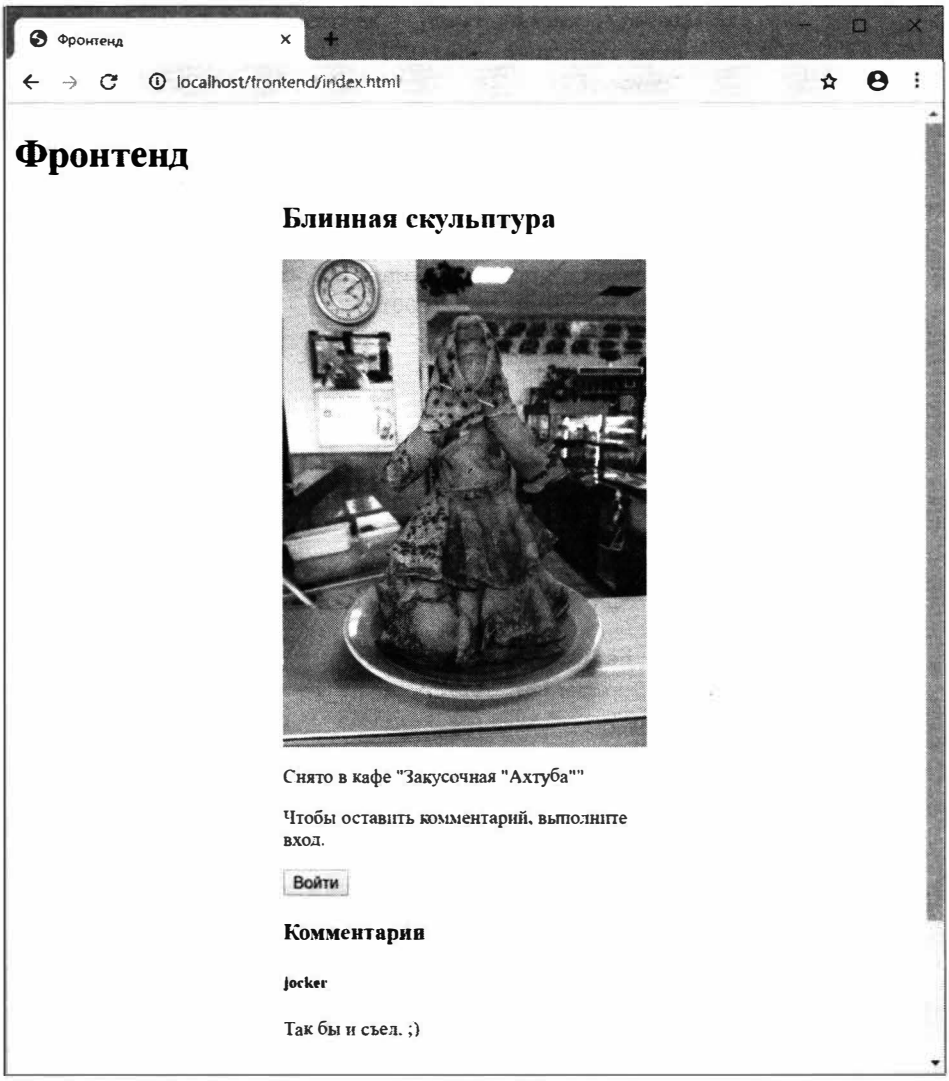


Рис. 24.1. Веб-страница тестового фронтенда

Выполним вход под именем пользователя, например, `jocker`. Для этого нажмем кнопку **Войти**, введем в появившуюся панель веб-формы (рис. 24.2) имя и пароль пользователя и нажмем кнопку **Войти**.

Нажмем появившуюся на странице кнопку **Добавить комментарий**, введем содержание нового комментария в появившуюся на экране панель веб-формы (подобную показанной на рис. 24.2) и нажмем кнопку **Добавить**.

Рис. 24.2. Панель с веб-формой входа

Проверим, появился ли добавленный комментарий в списке под изображением. Попробуем изменить этот комментарий (нажав кнопку **Изменить** под его содержанием), после чего удалим его (нажав кнопку **Удалить**). Напоследок выйдем с сайта, нажав кнопку **Выйти**.

О выводимом фронтендом изображении

Его номер задается в константе `imageID` файла *frontend\index.html* выражением:

```
const imageID = 2;
```

Изначально выводится изображение № 2.

Урок 25
Настройки PHP

Все настройки платформы PHP хранятся в текстовом файле php.ini, расположенном в той же папке, где хранятся сами файлы платформы (при установке в составе XAMPP это папка *<папка, в которой установлен XAMPP>*\php).

Настройки записываются в формате:

<название настройки>=<значение настройки>

Пример:

```
memory_limit=128M
```

В файле с настройками можно записывать строки с комментариями, поставив в их начале символ ; (точка с запятой):

```
; Максимальный объем памяти, который может занять под свои нужды
; выполняющаяся PHP-программа
memory_limit=128M
```

Вот правила указания значений у настроек различной природы:

- числовые величины — указываются в виде целых чисел (например, `30`);
- объем памяти и размеры файлов — в виде целых чисел с суффиксом, задающим размерность:
  - *<объем в байтах>*;
  - *<объем в килобайтах>*`K`;
  - *<объем в мегабайтах>*`M`;
  - *<объем в гигабайтах>*`G`.

  Например, `128M` — 128 Мбайт;
- предопределенные значения, наподобие `on` или `off`, — могут быть указаны в разном регистре (допустимы значения `on`, `On` и `ON`);
- пути к файлам и папкам — рекомендуется записывать в двойных кавычках (например: `"C:\xampp\tmp"`).

Чтобы установить для настройки значение по умолчанию, достаточно либо закомментировать ее, либо сделать «пустой», удалив ее значение (но оставив символ =).

На этом уроке мы рассмотрим некоторые наиболее полезные настройки платформы.

25.1. Включение и отключение расширений PHP

Функциональность PHP реализована частично в ее программном ядре, а частично — в различных расширениях.

> *Расширение* — исполняемый модуль (в редакции для Windows — библиотека DLL), загружающийся вместе с программным ядром PHP и дополняющий его функциональность. Расширения могут быть как включены, так и отключены, в последнем случае они не загружаются.

Отключение ненужных расширений может сэкономить системные ресурсы.

Настройки, указывающие включить или отключить то или иное расширение, записываются в формате:

```
extension=<имя файла расширения без файлового расширения>
```

Например, эта настройка указывает включить расширение `gd2`, реализующее инструменты для обработки графики (см. *урок 19*):

```
extension=gd2
```

Чтобы отключить ненужное расширение, достаточно закомментировать строку с соответствующей настройкой:

```
; Отключаем расширение для работы с данными формата EXIF
;extension=exif
```

В составе PHP поставляются порядка 20 расширений. Примерно половина из них изначально включена (табл. 25.1).

***Таблица 25.1.** Изначально включенные расширения, поставляемые в составе PHP*

Название	Реализуемые функции
`bz2`	Сжатие и распаковка данных по алгоритму bzip2
`curl`	Взаимодействие с другими интернет-службами, в частности, веб- и FTP-серверами: загрузка файлов, отправка данных и др.
`fileinfo`	Выяснение типа файла по его содержимому
`gd2`	Обработка графики (см. *урок 19*)
`gettext`	Упрощение программирования многоязычных веб-сайтов
`mbstring`	Функции для обработки строк в кодировке Unicode, в том числе и на русском языке. Имена эти функций начинаются с символов `mb_` (см. *разд. 2.2*)
`exif`	Чтение из графических файлов сведений об изображениях, записанных в формате EXIF. В работе использует расширение `mbstring` и должно быть указано после него
`mysqli`	Старые объектные инструменты для работы с базами данных MySQL
`pdo_mysql`	Новые объектные инструменты для работы с базами данных MySQL, описанные на *уроке 15*
`pdo_sqlite`	Новые объектные инструменты для работы с базами данных SQLite

В нашем случае можно отключить все приведенные в табл. 25.1 расширения, за исключением `gd2`, `mbstring` и `pdo_mysql`, — и сайт будет работать без проблем.

Полезно знать...

- В составе PHP также поставляются расширения для работы с базами данных форматов Interbase, PostgreSQL, интерфейсом доступа к базам данных ODBC, протоколами LDAP, SOAP и др. Все они изначально отключены.
- Имеется возможность загрузить, установить и использовать расширения PHP, написанные сторонними разработчиками.

25.2. Настройки отправки файлов на сервер

- Разрешение или запрет отправки файлов на сервер — `file_uploads`. Значение `on` разрешает отправку, значение `off` — запрещает. По умолчанию — `on`.
- Путь к папке для временного сохранения отправленных файлов — `upload_tmp_dir`. Значение по умолчанию различно в разных редакциях, так, в редакции, входящей в состав XAMPP, — `"C:\xampp\tmp"`.

  Если настройка не задана, используется системная папка для хранения временных файлов.
- Максимальный размер отправляемого файла — `upload_max_filesize`. Файлы большего размера будут отвергаться. Значение по умолчанию — `2M` (2 Мбайт).

  Если производится отправка нескольких файлов, учитывается их совокупный размер (не размер отдельного файла!).
- Максимальное количество одновременно отправляемых файлов — `max_file_uploads`. Избыточные файлы будут отвергаться. Значение по умолчанию — 20.

25.3. Настройки сессий

- Путь к папке для сохранения файлов с сессиями — `session.save_path`. Значение по умолчанию различно в разных редакциях, так, в редакции, входящей в состав XAMPP, — `"C:\xampp\tmp"`.
- Продолжительность существования cookie, хранящего идентификатор сессии, — `session.cookie_lifetime`. Указывается в секундах. Если задать значение `0`, cookie будет удален сразу же после закрытия веб-обозревателя. Значение по умолчанию — `0`.

  Как только будет удален cookie, хранящий идентификатор сессии, сессия с этим идентификатором станет «мусорной». «Мусорные» сессии удаляются самой платформой PHP спустя промежуток времени, заданный настройкой `session.gc_maxlifetime`.
- Время хранения «мусорной» сессии — `session.gc_maxlifetime`. По его истечении сессия будет удалена встроенным в PHP сборщиком мусора. Указывается в секундах. Значение по умолчанию — 1440 (24 минуты).

- Путь, сохраняемый в cookie с идентификатором сессии, — `session.cookie_path`. Если задать прямой слеш `'/'`, cookie будет доступен любому веб-приложению сайта. Значение по умолчанию — `'/'`.
- Домен, сохраняемый в cookie с идентификатором сессии, — `session.cookie_domain`. Если не указан, cookie будет связан с текущим доменом. По умолчанию настройка не задана.
- Должен ли cookie с идентификатором сессии передаваться только по протоколу HTTPS — `session.cookie_secure`. Значение `on` указывает передавать cookie только по протоколу HTTPS, значение `off` — по протоколам HTTP и HTTPS. Значение по умолчанию — `off`.
- Должен ли cookie с идентификатором сессии быть доступен клиентским веб-сценариям — `session.cookie_httponly`. Значение `on` запрещает веб-сценариям доступ к cookie, значение `off` — разрешает. Значение по умолчанию — `on`.

  Последние четыре настройки cookie описаны в *разд. 20.1*.

25.4. Настройки вывода сообщений об ошибках

- Отображать ли сообщения об ошибках времени выполнения непосредственно на веб-страницах — `display_errors`. Значение `on` указывает отображать сообщения, значение `off` — скрывать. Значение по умолчанию — `on`.
- Отображать ли сообщения об ошибках, возникающих при запуске исполняющей среды PHP, непосредственно на веб-страницах — `display_startup_errors`. Значение `on` указывает отображать их, значение `off` — скрывать. Значение по умолчанию — `on`.
- Записывать ли сообщения о возникших ошибках в файл журнала ошибок — `log_errors`. Значение `on` указывает записывать сообщения, значение `off` — не записывать их. Значение по умолчанию — `on`.
- Путь к файлу, в который записываются сообщения об ошибках, — `error_log`. Если задать значение `syslog`, сообщения будут записываться в системный журнал ошибок Windows. По умолчанию настройка не указана.

> ***Внимание!***
>
> При разработке сайта следует разрешить вывод сообщений об ошибках непосредственно на генерируемых веб-страницах.
>
> У готового сайта, опубликованного в Интернете, следует запретить вывод сообщений об ошибках на страницах и включить их запись в файл журнала.

25.5. Настройки ограничения ресурсов

- Максимальный объем памяти, который может занять под свои нужды PHP-программа, — `memory_limit`. Значение по умолчанию — `128M` (128 Мбайт).
- Максимальное время выполнения PHP-программы — `max_execution_time`. По истечении этого времени выполнение программы будет прервано. Указывается в секундах. Значение по умолчанию — 30.
- Максимальное количество принимаемых GET- или POST-параметров — `max_input_vars`. Избыточные параметры отвергаются. По умолчанию не ограничено.
- Максимальный объем данных, принимаемых методом POST, — `post_max_size`. Данные, превышающие этот объем, отвергаются. Значение по умолчанию — `8M` (8 Мбайт).

Заключение

Вот и закончилась книга о программировании веб-сайтов с применением платформы PHP и СУБД MySQL. Мы изучили необходимый минимум программных средств, применяемых для написания сайтов, познакомились с устройством баз данных, узнали много новых терминов и даже создали сайт фотогалереи. Его файлы можно найти в папке **24\24.4\ex**, а файл **photogallery.sql** с дампом базы данных `photogallery` — в папке **22\22.2\ex** сопровождающего книгу электронного архива (см. *приложение 3*).

Появившаяся в далеком уже 1995 году, постепенно избавившаяся от устаревших, неудачных и зачастую небезопасных программных инструментов, приобретя новые современные средства программирования, платформа PHP и поныне является одним из лидеров рынка веб-разработки. Также как и СУБД MySQL, «ровесница» PHP.

Так что время, потраченное вами на изучение этих программных продуктов, не будет потеряно зря, а полученные знания останутся актуальными еще очень долго.

А чтобы вы, уважаемые читатели, не блуждали по Интернету в поисках сайтов с документацией по PHP и MySQL, их список приведен в табл. 3.1.

Таблица 3.1. *Список интернет-ресурсов по теме книги*

Интернет-адрес	Описание
https://www.php.net/	Домашний сайт PHP. Дистрибутивы, новости, документация, справочные материалы
https://www.php.net/manual/ru/	Официальная документация по PHP на русском языке
https://www.mysql.com/	Домашний сайт MySQL. Дистрибутивы всех поддерживаемых программных продуктов, новости, документация, справочные материалы, учебные курсы
https://dev.mysql.com/doc/refman/5.7/en/	Документация по MySQL 5.7, полностью совместимой с текущей версией MariaDB
https://mariadb.org/	Домашний сайт MariaDB. Дистрибутивы, новости, но, к сожалению, нет документации — предлагается пользоваться руководством по MySQL 5.7
https://www.apachefriends.org/ru/index.html	Русскоязычный домашний сайт пакета XAMPP. Дистрибутивы, подборка расширений, справочная информация

На этом автор книги прощается с вами, уважаемые читатели. До свидания! И успехов вам на ниве веб-программирования!

Владимир Дронов

Приложения

- *Приложение 1*. Приоритет операторов
- *Приложение 2*. Пакет хостинга XAMPP
- *Приложение 3*. Описание электронного архива

Приложение 1
Приоритет операторов

Приоритет	Оператор	Описание	Порядок выполнения	Пример
22	new	Создание объекта		new Image()
21	**	Возведение в степень	Справа налево	5 ** 2
20	++	Инкремент		i++
	--	Декремент		a--
	(<*тип*>)	Приведение значения к заданному *типу*		$s = '123'; $i = (int)s;
19	instanceof	Проверка принадлежности объекта заданному классу		$img instanceof Image
18	!	Логическое НЕ	Справа налево	!session_start()
17	*	Умножение	Слева направо	2 * 3
	/	Деление		24 / 8
	%	Остаток от деления		24 / 10
16	+	Сложение		2 + 3
	-	Вычитание		20 - 1
	. (точка)	Конкатенация строк		'PHP' . '7'
15	<<	Сдвиг влево		0b1000 << 3
	>>	Сдвиг вправо		0b1000 >> 2
14	<	Меньше		2 < 3
	<=	Меньше или равно		2 >= 2
	>	Больше		3 > 2
	>=	Больше или равно		3 >= 2
13	==	Равно		2 == 2
	!=, <>	Не равно		2 != '3'
	===	Строго равно		2 === 2
	!==	Строго не равно		2 !== '2'
	<=>	«Космический корабль»		2 <=> 3

(окончание)

Приоритет	Оператор	Описание	Порядок выполнения	Пример
12	`&`	Побитовое И	Слева направо	`0b1010 & 0b1001`
11	`^`	Побитовое Исключающее ИЛИ		`0b1010 ^ 0b1001`
10	`|`	Побитовое ИЛИ		`0b1010 | 0b1001`
9	`&&`	Логическое И		`TRUE && FALSE`
8	`||`	Логическое ИЛИ		`TRUE || FALSE`
7	`??`	Сравнение с NULL	Справа налево	`$n ?? 0`
6	`? :`	Условный	Слева направо	`($n == 0) ? 'Ноль' : 'Не ноль'`
5	`=`	Простое присваивание	Справа налево	`$a = 4`
	`<знак>=`	Сложное присваивание	Справа налево	`$a = 1;` `$a += 2;`
4	`yield`	Приостановка выполнения генератора		`yield $i`
3	`and`	Логическое И	Слева направо	`TRUE and FALSE`
2	`xor`	Логическое Исключающее ИЛИ		`TRUE xor FALSE`
1	`or`	Логическое ИЛИ		`TRUE || FALSE`

Приложение 2
Пакет хостинга XAMPP

Для тестирования веб-приложений, написанных на PHP, нам понадобится работающий веб-сервер, СУБД MySQL (или совместимая с ней MariaDB) и исполняющая среда PHP. Все это входит в пакет хостинга XAMPP.

> *XAMPP* — программный пакет интернет-хостинга, включающий в свой состав все необходимое для запуска веб-сайта: веб-сервер Apache HTTP Server, программную платформу PHP, серверную СУБД MariaDB и программу для работы с базами данных phpMyAdmin. Все эти программы полностью работоспособны и не требуют ни специального сопряжения, ни дополнительной настройки.

П2.1. Установка пакета XAMPP

1. Перейдем в веб-обозревателе по интернет-адресу: **https://www.apachefriends.org/ru/index.html**.
2. На открывшейся странице найдем большую гиперссылку **XAMPP для Windows** и щелкнем на ней (рис. П2.1).

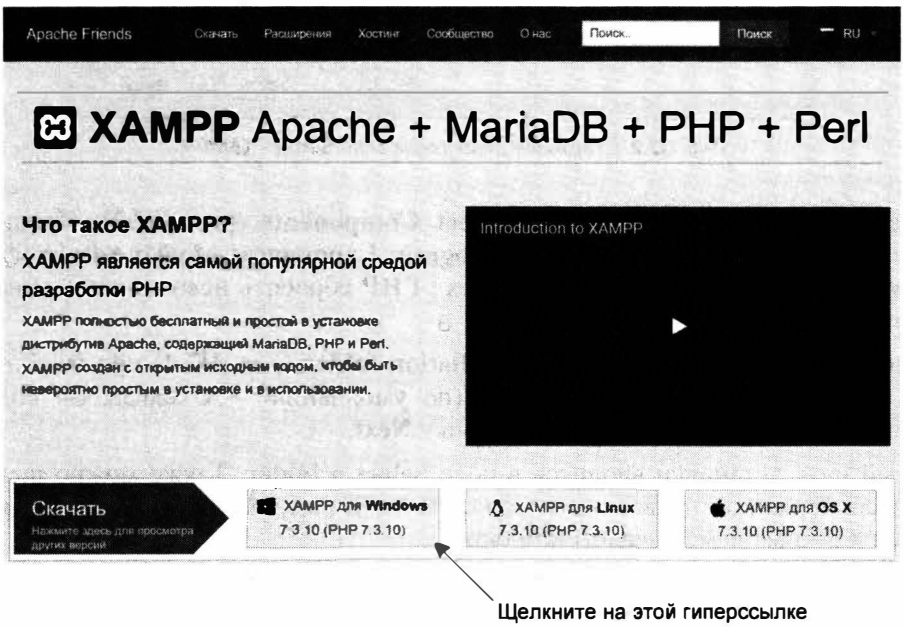


Рис. П2.1. Гиперссылка **XAMPP для Windows**

3. Сохраним загруженный файл с дистрибутивом XAMPP где-либо на локальном диске. Запустим его и положительно ответим на запрос подсистемы контроля доступа UAC.
4. В появившемся окне-сообщении, предостерегающем от установки XAMPP в папку C:\Program Files\ или в любую другую, защищенную UAC, нажмем кнопку **OK**.
5. В открывшемся диалоговом окне **Setup - XAMPP** (рис. П2.2) нажмем кнопку **Next**.

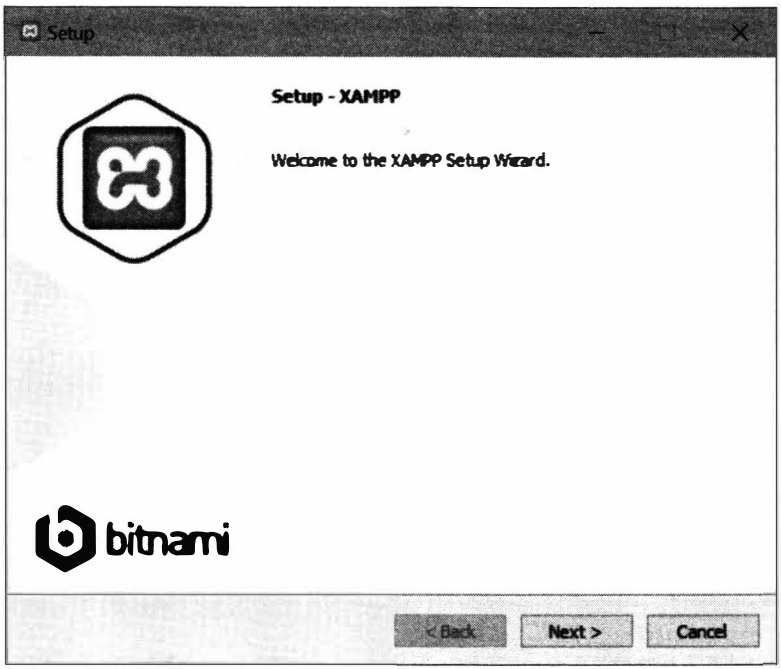


Рис. П2.2. Стартовое диалоговое окно **Setup - XAMPP**

6. В следующем диалоговом окне **Select Components** (рис. П2.3) сбросим все флажки, кроме **Server | MySQL** и **Program Languages | phpMyAdmin** (флажки **Server | Apache** и **Program Languages | PHP** сбросить невозможно), и нажмем кнопку **Next**.
7. В следующем диалоговом окне **Installation folder** (рис. П2.4) при необходимости укажем другой путь установки (по умолчанию — C:\xampp, он подойдет в большинстве случаев) и нажмем кнопку **Next**.

   Новый путь установки вводится в поле **Select a folder**. Также можно щелкнуть на расположенной правее этого поля кнопке и выбрать любую папку для установки в открывшемся диалоговом окне.
8. В следующем окне **Bitnami for XAMPP** (рис. П2.5) сбросим флажок **Learn more about Bitnami for XAMPP**, чтобы установщик не вывел веб-страницу с дополнительными сведениями (сейчас они нам не нужны), и нажмем кнопку **Next**.

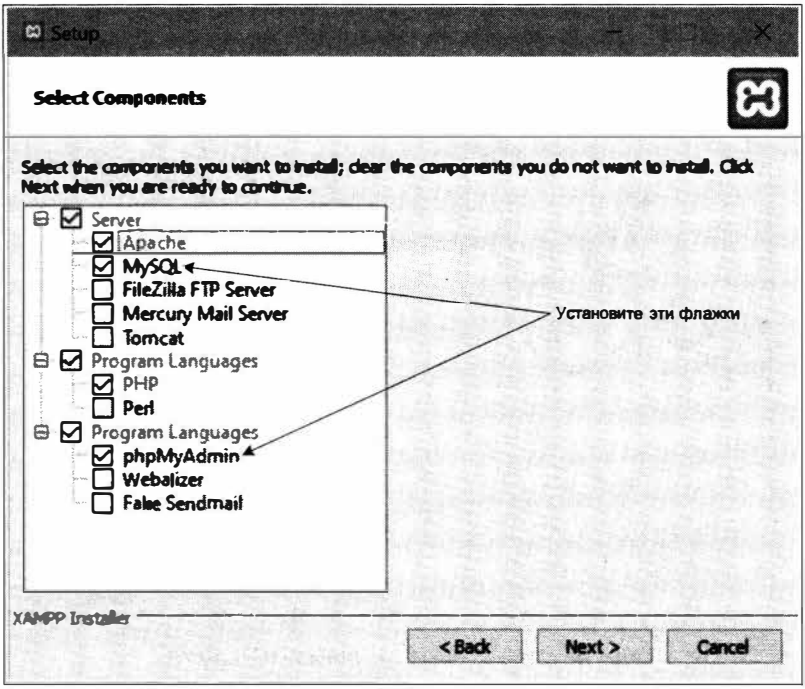


Рис. П2.3. Диалоговое окно **Select Components**

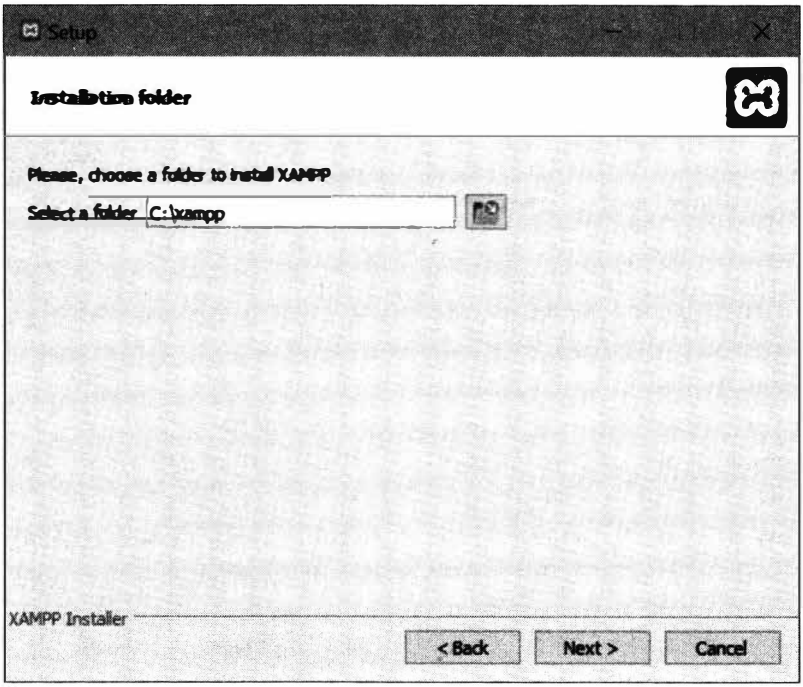


Рис. П2.4. Диалоговое окно **Installation folder**

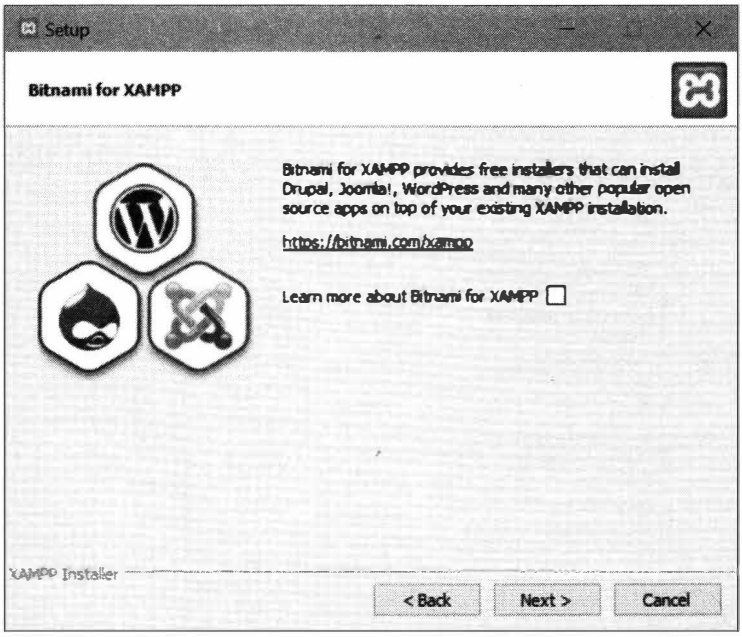


Рис. П2.5. Диалоговое окно **Bitnami for XAMPP**

9. В следующем окне **Ready to Install** (рис. П2.6) нажмем кнопку **Next**, чтобы запустить установку пакета.
10. Подождем окончания установки (рис. П2.7).

Рис. П2.6. Диалоговое окно **Ready to Install**

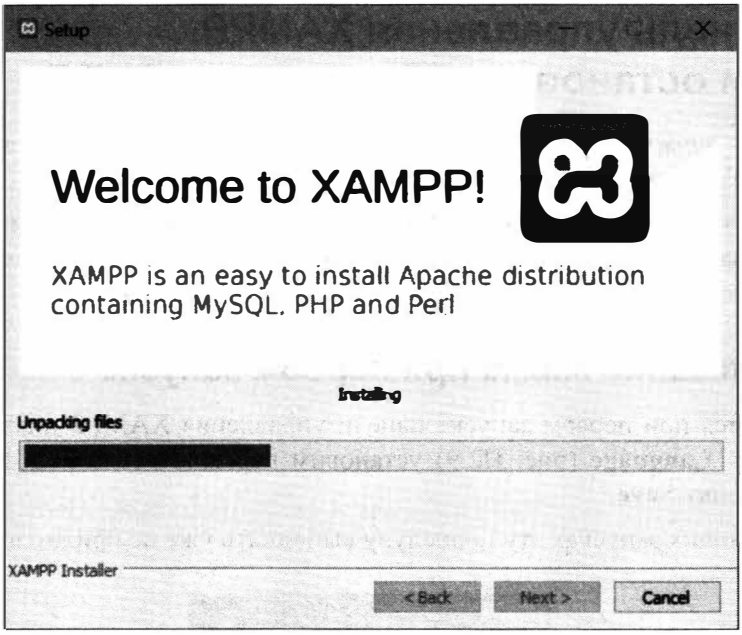


Рис. П2.7. Окно, отображающее ход процесса установки пакета

11. В последнем диалоговом окне **Completing the XAMPP Setup Wizard** (рис. П2.8) нажмем кнопку **Finish**, чтобы закрыть установщик. Если флажок **Do you want to start the Control Panel now?** не был сброшен, запустится панель управления XAMPP (см. далее).

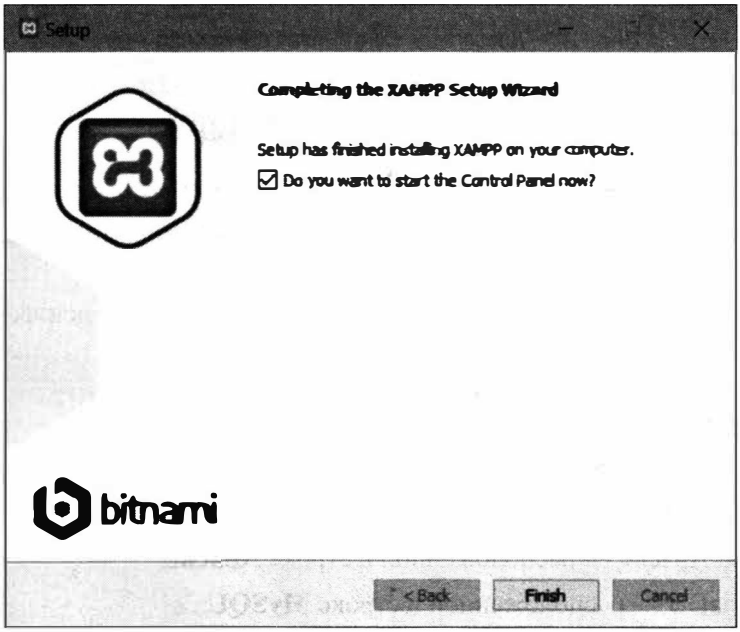


Рис. П2.8. Диалоговое окно **Completing the XAMPP Setup Wizard**

П2.2. Панель управления XAMPP. Запуск и остановка веб-сервера и СУБД

> *Панель управления XAMPP* — утилита для управления компонентами пакета XAMPP: их запуска, остановки и настройки.

Открывается панель управления XAMPP непосредственным запуском исполняемого файла xampp-control.exe, находящегося в папке, где установлен пакет. К сожалению, запустить ее из меню **Пуск** невозможно.

П2.2.1. Указание языка при первом запуске

В открывшемся при первом запуске панели управления XAMPP диалоговом окне выбора языка **Language** (рис. П2.9) установим переключатель под флагом США и нажмем кнопку **Save**.

При последующих запусках эту процедуру выполнять уже не придется.

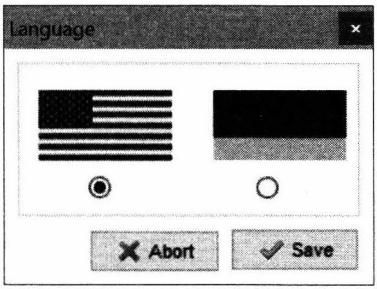


Рис. П2.9. Диалоговое окно выбора языка **Language**

П2.2.2. Окно панели управления XAMPP

Окно панели управления XAMPP по горизонтали делится на две части (рис. П2.10):

- *вверху* — таблица с перечнем всех установленных компонентов и наборы кнопок для управления ими (запуска, остановки и др.);
- *внизу* — журнал работы пакета, в котором сохраняются основные сообщения, выводимые его компонентами.

П2.2.3. Запуск веб-сервера и СУБД

Запуск программ производится нажатием соответствующей кнопки **Start** в панели управления XAMPP (рис. П2.11):

- веб-сервера Apache — расположенной в строке **Apache**;
- СУБД MariaDB — расположенной в строке **MySQL**.

Порядок запуска программ несущественен.

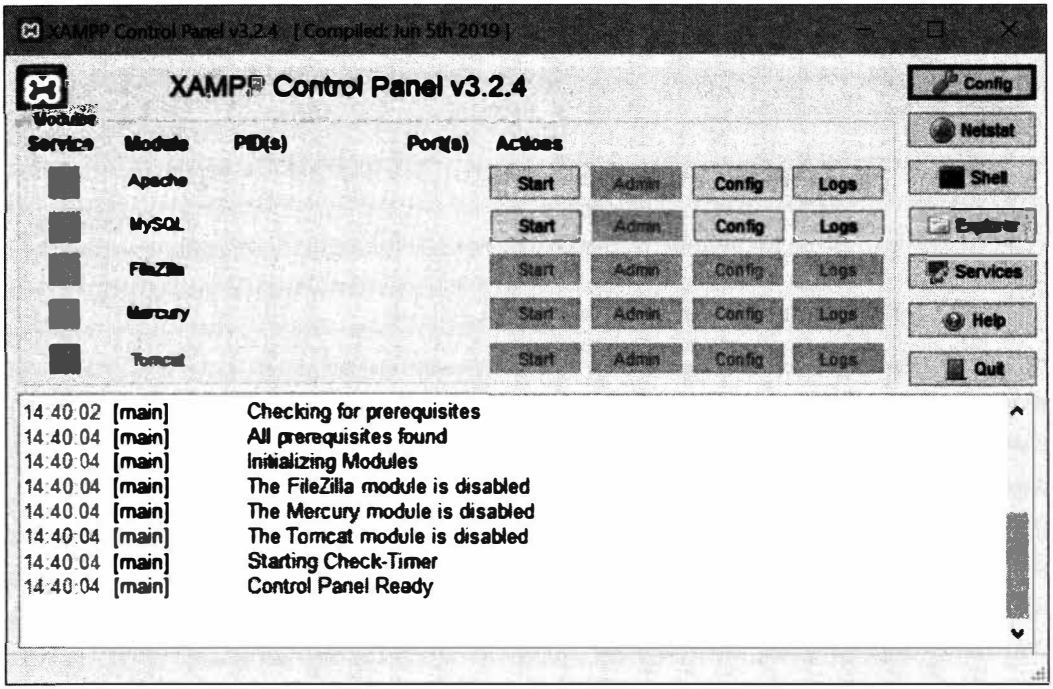


Рис. П2.10. Окно панели управления XAMPP

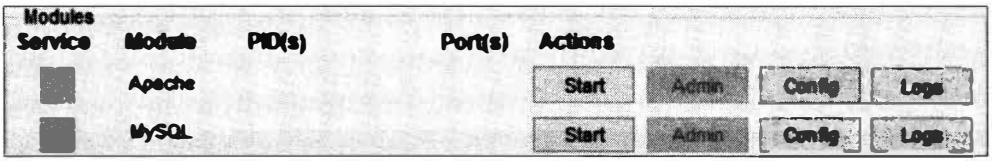


Рис. П2.11. Кнопки **Start** в строках **Apache** и **MySQL** перечня компонентов в панели управления XAMPP

Внимание!

В состав предыдущих версий XAMPP входила СУБД MySQL, но позже она, вследствие проблем с лицензией, была заменена на MariaDB. Однако ряд надписей в окнах утилиты все еще ссылаются на MySQL.

После этого надпись на соответствующей кнопке сменится на **Stop**.

П2.2.4. Проверка работоспособности веб-сервера и СУБД

- Для проверки работы веб-сервера нажмем кнопку **Admin** в строке **Apache** панели управления XAMPP (см. рис. П2.11). Также можно в веб-обозревателе перейти по интернет-адресу: **http://localhost/** или **http://localhost/dashboard/**.

  На экране появится единственная страница небольшого тестового сайта, поставляемого в составе XAMPP (рис. П2.12).

Рис. П2.12. Тестовый веб-сайт, поставляемый в составе XAMPP

- Для проверки работы СУБД нажмем кнопку **Admin** в строке **MySQL** панели управления XAMPP. Также можно в веб-обозревателе перейти по интернет-адресу: **http://localhost/phpmyadmin/**.

  На экране появится главная страница программы phpMyAdmin, предназначенной для работы с базами данных MySQL (см. *разд. 13.3*).

П2.2.5. Остановка веб-сервера и СУБД

Остановка программ выполняется нажатием одной из кнопок **Stop** в панели управления XAMPP (см. рис. П2.11):

- веб-сервера Apache — расположенной в строке **Apache**;
- СУБД MariaDB — расположенной в строке **MySQL**.

После этого надпись на соответствующей кнопке сменится на **Start**.

Порядок остановки программ несущественен.

П2.3. Использование веб-сервера

П2.3.1. Тестирование веб-сайта с применением XAMPP

1. Откроем папку по пути *<путь установки XAMPP>*\htdocs. Это корневая папка веб-сервера Apache HTTP Server из комплекта XAMPP.

   Изначально там хранится тестовый сайт, позволяющий проверить работоспособность пакета. Рекомендуется сохранить этот сайт на всякий случай.

2. Если в корневой папке хранятся файлы тестового сайта, переименуем ее, например, в _htdocs, и создадим в папке *<путь установки XAMPP>* новую корневую папку htdocs.

3. Скопируем файлы сайта, который хотим протестировать, в папку *<путь установки XAMPP>*\htdocs.

Не забываем запускать веб-сервер и СУБД перед тестированием сайта и останавливать перед крупными правками.

П2.3.2. Просмотр журналов работы веб-сервера и PHP

> *Журнал* (*лог*) — файл для сохранения сведений о работе программы-сервера: выполненных ею действиях, нештатных ситуациях и пр. Практически всегда имеет текстовый формат и расширение log.

1. Найдем в строке **Apache** перечня компонентов в панели управления XAMPP кнопку **Logs** (см. рис. П2.11) и нажмем ее.

2. Выберем в появившемся на экране меню (рис. П2.13) пункт:

   - **Apache (access.log)** — для просмотра журнала веб-сервера, содержащего сведения об отправленных им файлах;
   - **Apache (error.log)** — для просмотра журнала ошибок, возникших в работе веб-сервера. Также включает сообщения об ошибках в PHP-коде.

   Соответствующий журнал будет открыт в текстовом редакторе Блокнот.

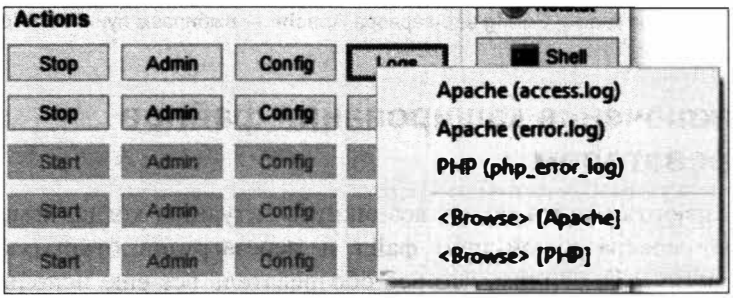


Рис. П2.13. Меню журналов работы веб-сервера и PHP

Внимание!

В меню также присутствует пункт **PHP (php_error_log)**, но при его выборе выводится сообщение о том, что файл с таким журналом отсутствует. Судя по всему, это программная ошибка в пакете XAMPP.

П2.4. Настройка PHP и решение проблем

П2.4.1. Настройка PHP

1. Запустим панель управления XAMPP.
2. Остановим веб-сервер, если он запущен (останавливать СУБД необязательно).
3. Найдем в строке **Apache** перечня компонентов кнопку **Config** (см. рис. П2.11) и нажмем ее.
4. Выберем в появившемся на экране меню (рис. П2.14) пункт **PHP (php.ini)** — конфигурационный файл php.ini, хранящий настройки PHP, будет открыт в Блокноте.

   Этот файл находится по пути *<папка, в которой установлен XAMPP>*\php — его можно открыть непосредственно оттуда.
5. Внесем необходимые правки, сохраним и закроем файл php.ini.

После этого можно запустить веб-сервер.

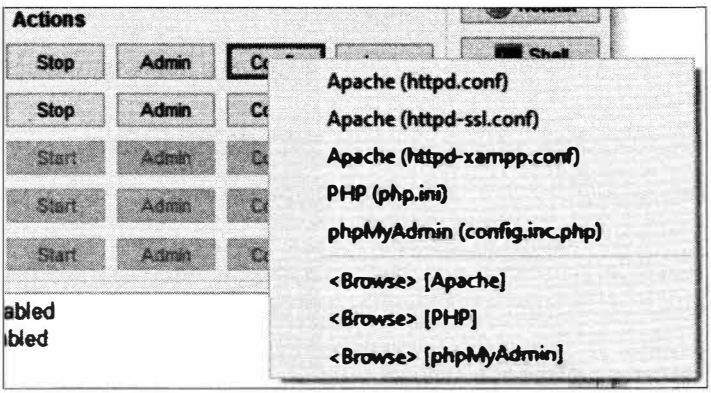


Рис. П2.14. Меню кнопки **Config** веб-сервера **Apache** — выбираем пункт **PHP (php.ini)**

П2.4.2. Отключение кэширования файлов веб-обозревателем

В процессе разработки сайта может возникнуть ситуация, когда, изменив в корневой папке веб-сервера какой-либо файл и перезагрузив открытую в веб-обозревателе страницу, мы увидим, что веб-обозреватель все еще использует устаревшую копию файла, взятую из кэша.

Проще всего решить проблему застревания файлов в кэше, отключив кэширование файлов веб-обозревателем. Процесс описывается на примере веб-обозревателя Google Chrome.

1. Выведем панель с инструментами разработчика, встроенными в веб-обозреватель, нажав клавишу <F12>.
2. Переключимся на вкладку **Network**, щелкнув на ее корешке в панели с инструментами разработчика (рис. П2.15).

Рис. П2.15. Корешок вкладки **Network** на панели с отладочными инструментами веб-обозревателя

На открывшейся вкладке **Network** (рис. П2.16) находятся список файлов, загруженных по Сети при открытии текущей страницы, и график их загрузки.

3. Установим находящийся над графиком флажок **Disable cache** (рис. П2.17).

> ***Внимание!***
>
> Кэширование будет отключено, пока панель с инструментами разработчика присутствует на экране. Если закрыть ее, кэширование вновь активизируется, независимо от состояния флажка **Disable cache**.

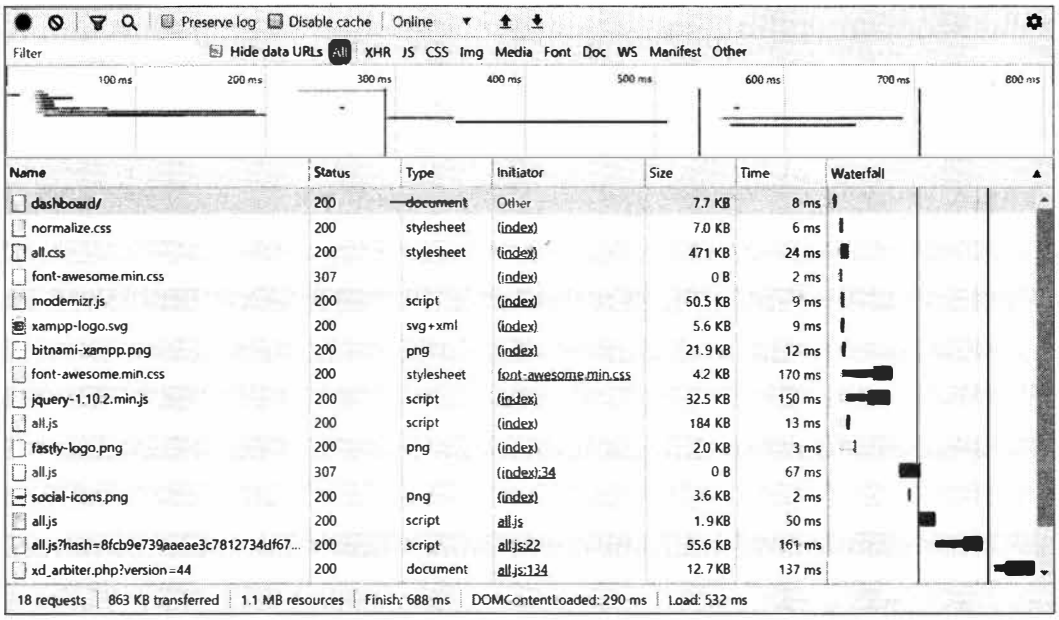


Рис. П2.16. Вкладка **Network** панели с инструментами разработчика

Рис. П2.17. Панель инструментов вкладки **Network** — устанавливаем флажок **Disable cache**

4. Напоследок очистим кэш веб-обозревателя, чтобы тот использовал самые актуальные версии файлов.

   Щелкнем для этого на списке файлов правой кнопкой мыши, выберем в появившемся на экране меню пункт **Clear browser cache** (рис. П2.18) и нажмем кнопку **ОК** в появившемся окне-предупреждении.

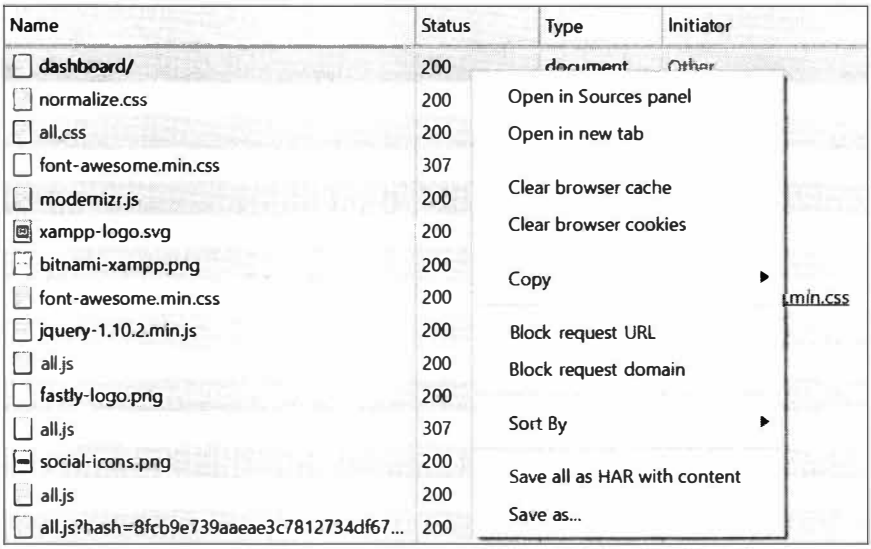


Рис. П2.18. Контекстное меню списка файлов на вкладке **Network** — выбираем пункт **Clear browser cache**

Приложение 3
Описание электронного архива

Электронный архив, сопровождающий книгу, опубликован на FTP-сервере издательства «БХВ-Петербург» по интернет-адресу: **ftp://ftp.bhv.ru/9785977566513.zip**. Ссылка на него доступна и со страницы книги на сайте **http://www.bhv.ru/**.

Список папок и файлов, имеющихся в архиве (вложенность папок и файлов показана отступами):

- *<номер урока>* — папка с материалами урока с указанным *номером*. Состав папки:
  - !sources — папка с исходными материалами, необходимыми для выполнения упражнений в текущем уроке;
  - *<номер урока>.<номер раздела>* — результаты выполнения упражнений из раздела с указанным *номером*. Состав папки:
    - ex — результаты упражнения, выполняемого под руководством автора книги;
    - s — результаты выполнения самостоятельного упражнения;
- readme.txt — текстовый файл с описанием электронного архива.

Предметный указатель

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

А

Б

В

Г

Д

Е

Ж

З

И

К

Л

М

Н

О

П

Р

С

Т

У

Ф

Х

Ц

Ч

Ш

Э

PHP и MySQL

25 уроков для начинающих

Дронов Владимир Александрович, профессиональный программист, писатель и журналист, работает с компьютерами с 1987 года. Автор более 30 популярных компьютерных книг, в том числе «HTML и CSS. 25 уроков для начинающих», «JavaScript: 20 уроков для начинающих», «Django 2.1. Практика создания веб-сайтов на Python», «HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера», «Python 3. Самое необходимое», «Python 3 и PyQt 5. Разработка приложений» и книг по продуктам Adobe Flash и Adobe Dreamweaver различных версий. Его статьи публикуются в журналах «Мир ПК» и «ИнтерФейс» (Израиль) и интернет-порталах «IZ City» и «TheVista.ru».

Простым языком, наглядно, без ненужных подробностей рассказано о программировании динамических веб-сайтов на языке PHP с применением СУБД MySQL и MariaDB. В книге 25 иллюстрированных уроков и более 30 практических упражнений.

Сжато, емко, наглядно — только самое необходимое

Вы узнаете, как

- генерировать веб-страницы программно;
- получать и проверять на корректность данные, отправленные посетителем;
- выводить страницы с сообщениями об ошибках, применяя исключения;
- создавать базы данных MySQL и MariaDB в программе phpMyAdmin;
- программно рисовать графику;
- отправлять электронные письма;
- защитить сайт от несанкционированного проникновения;
- шифровать конфиденциальные данные;
- перевести сайт на безопасный протокол HTTPS;
- противодействовать сетевым атакам;
- писать веб-службы REST;
- разделить код на модели, шаблоны и контроллеры;
- написать свой PHP-фреймворк;
- создать полнофункциональный веб-сайт.

Коды всех примеров и пяти учебных веб-сайтов можно скачать по ссылке ftp://ftp.bhv.ru/9785977566513.zip, а также со страницы книги на сайте www.bhv.ru.

bhv®

191036, Санкт-Петербург,
Гончарная ул., 20
Тел.: (812) 717-10-50,
339-54-17, 339-54-28
E-mail: mail@bhv.ru
Internet: www.bhv.ru

ISBN 978-5-9775-6651-3