Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

Д. В. ШЕВЧЕНКО М. И. ОЗЕРОВА

ОСНОВЫ WEB-ПРОГРАММИРОВАНИЯ

Лабораторный практикум

Владимир 2017

УДК 004.438
ББК 32.973.2
Ш37

Рецензенты:
Кандидат технических наук, доцент
зам. зав. кафедрой геоинформационных систем
Нижегородского государственного технического университета
им. Р. Е. Алексеева, зав. лабораторией ИПИ-технологий
Л. И. Райкин

Кандидат технических наук, доцент
профессор кафедры автоматизированных систем обработки
информации и управления Санкт-Петербургского государственного
электротехнического университета «ЛЭТИ»
Н. Г. Мустафин

Печатается по решению редакционно-издательского совета ВлГУ

Шевченко, Д. В.

Ш37 Основы web-программирования : лаб. практикум / Д. В. Шевченко, М. И. Озерова ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2017. – 154 с.

ISBN 978-5-9984-0778-9

Представляет собой ознакомительный цикл работ по web-программированию, в котором описаны основные термины и понятия, характеризующие современный web, а также технологии, применяемые для web-разработки, такие как HTML, PHP, Javascript. Рассмотрено использование платформы AMP для создания web-сайтов с динамическим содержимым и комплексными сценариями взаимодействия с пользователем, а также взаимодействия с базами данных. Позволяет самостоятельно закрепить и проверить полученные теоретические знания и приобрести практические навыки в разработке современных web-сайтов.

Предназначен для студентов, начинающих осваивать web-программирование в рамках дисциплин «Основы разработки web-приложений», «Основы информационного дизайна», «Системы разработки сайтов», «Технологии разработки мобильных приложений» направлений 09.03.02, 09.04.02 «Информационные системы и технологии» и 09.03.04, 09.04.04 «Программная инженерия».

Рекомендовано для формирования профессиональных компетенций в соответствии с ФГОС ВО.

Библиогр.: 7 назв.

УДК 004.438
ББК 32.973.2

ISBN 978-5-9984-0778-9

ВВЕДЕНИЕ

Лабораторный практикум включает ознакомительный цикл работ по следующим темам:

- формат HTML и представление страницы в виде дерева узлов. Использование HTML для создания статических страниц;
- язык PHP и развертывание платформы AMP для разработки динамических web-страниц;
- использование баз данных в качестве источника содержимого динамических web-страниц. Работа с базами данных на уровне PHP;
- способы отправки запроса от клиента к серверу. Реализация отправки запроса посредством HTML-страниц. Обработка запроса на стороне сервера интерпретатором PHP;
- сессии и способы их организации. Работа с сессиями на уровне PHP;
- безопасность. Способы хранения паролей на уровне базы. Использование техник хэширования и солей. Корректная валидация данных и предотвращение XSS-атак. Добавление специальных временных меток и предотвращение CSRF-атак;
- передача файлов со стороны клиента серверу и наоборот. Хранение файлов на сервере и доступ к ним со стороны PHP;
- архитектура приложений. Модульность и использование объектов и функций в PHP. Понятие слоев web-приложения и изоляции слоев;
- языки выполнения сценариев на стороне клиента. Работа с моделью документа в Javascript. Организация запросов к серверу без перегрузки страницы в целом.

Данное руководство не предполагает использование какого-либо фреймворка. Авторы исходят из предположения, что web-разработчик должен изначально научиться разрабатывать приложение самостоятельно, реализуя вручную весь необходимый функционал, чтобы получить более глубокое понимание процессов, происходящих на сто-

роне сервера. Только после этого он может использовать вспомогательные фреймворки типа JQuery или YUI для облегчения процесса разработки. В противном случае разработчик рискует остаться привязанным навсегда к одной платформе и даже ее не сможет использовать полноценно.

В практикуме предложены работы по использованию платформы AMP для создания web-сайтов с динамическим содержимым и комплексными сценариями взаимодействия с пользователем, а также с базами данных.

Платформа AMP является одной из наиболее распространенных и используемых. Она выбрана также и потому, что требует ручной настройки и установки всех компонентов и налаживания их взаимодействия, что считается более предпочтительным для первоначального использования, чем альтернативные решения вроде Joomla.

Лабораторная работа № 1

ЯЗЫК HTML

1.1. Цель работы

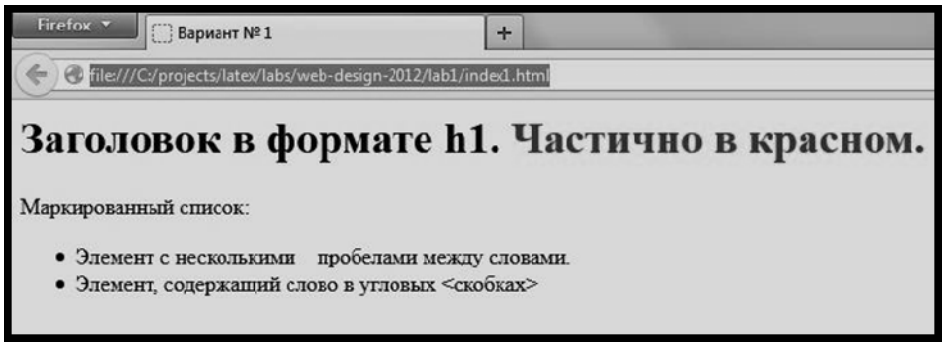
Освоение базового синтаксиса языка HTML и знакомство с наиболее распространенными тэгами.

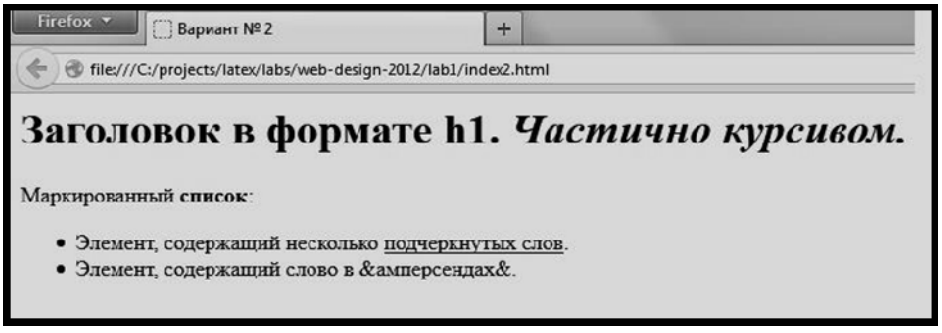
Предварительные требования

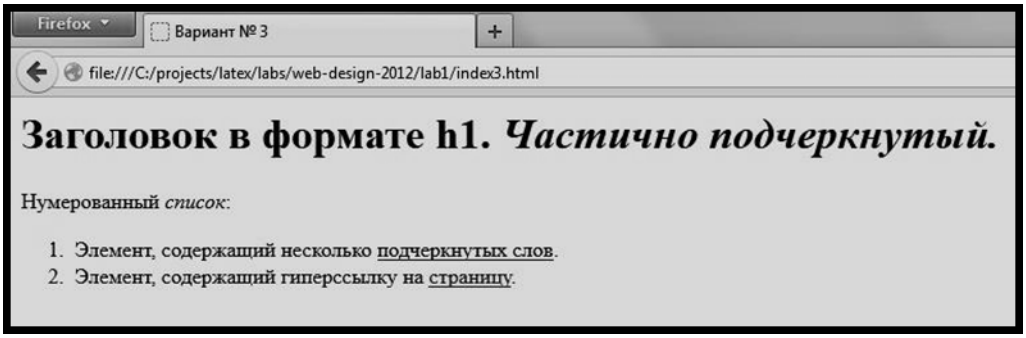
В данной работе ничего устанавливать дополнительно не требуется. Достаточно иметь установленный браузер и любой текстовый редактор.

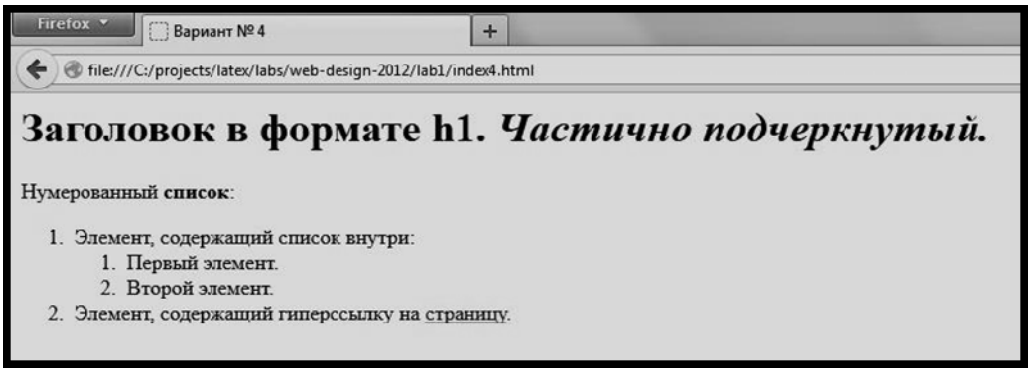
1.2. Варианты заданий

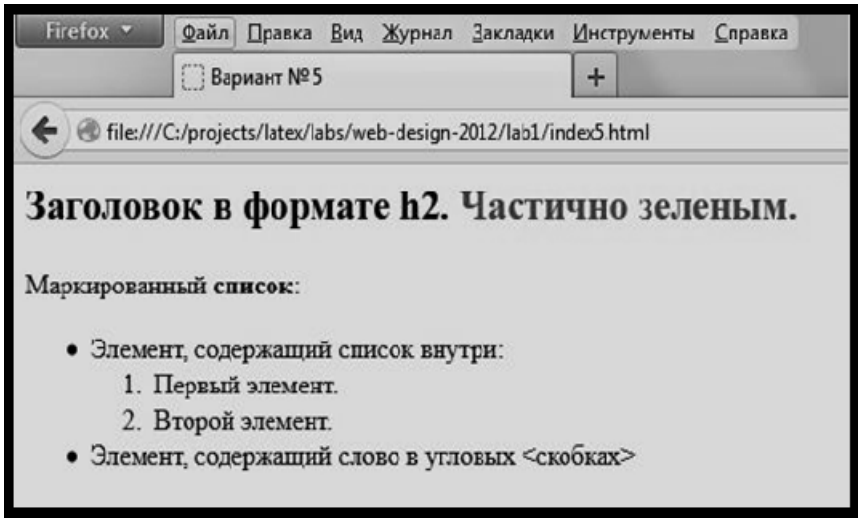
Имеется 10 вариантов заданий, представленных ниже. Необходимо проанализировать каждую страницу, выявить, какие тэги требуются для ее реализации, и написать исходный код этой страницы, после чего убедиться, что браузер отображает эту страницу именно так, как она выглядела изначально. Для каждого варианта следует проставить заголовок страницы, который отображает браузер на рисунке.

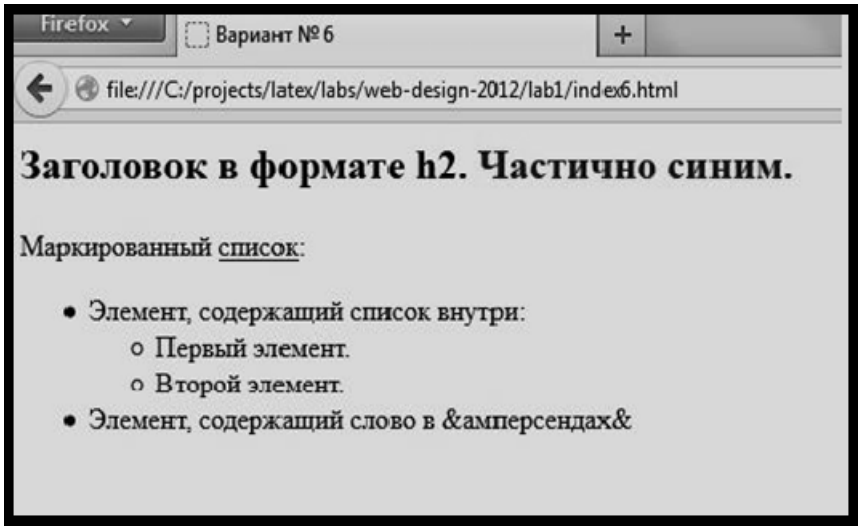


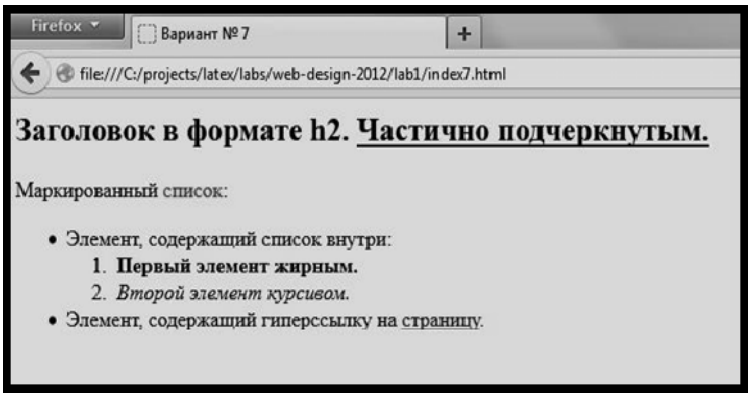


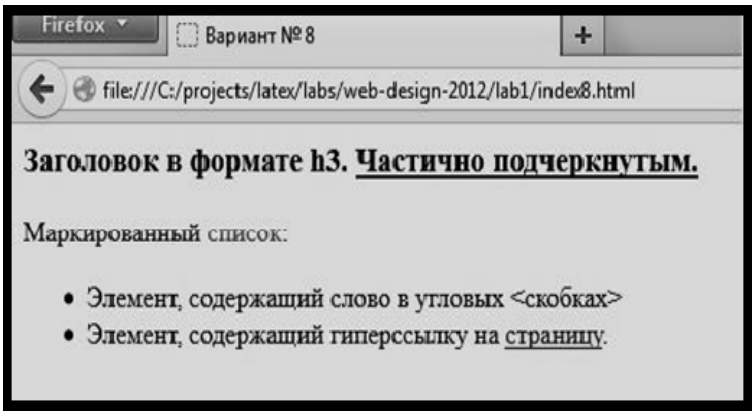

Firefox
Вариант № 3
file:///C:/projects/latex/labs/web-design-2012/lab1/index3.html
Заголовок в формате h1. Частично подчеркнутый.
Нумерованный список:
1. Элемент, содержащий несколько подчеркнутых слов.
2. Элемент, содержащий гиперссылку на страницу.

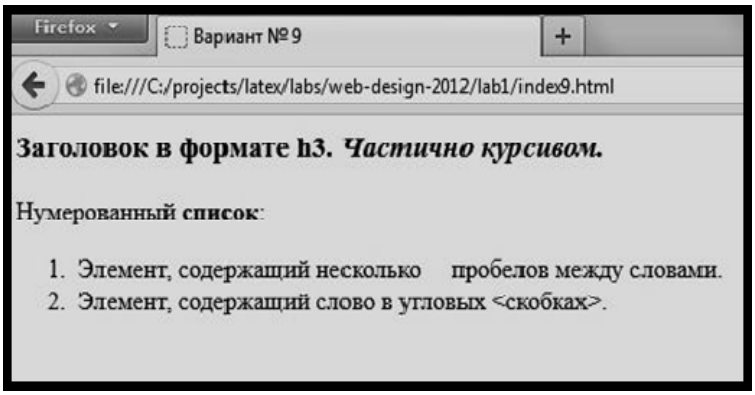

Firefox
Вариант № 4
file:///C:/projects/latex/labs/web-design-2012/lab1/index4.html
Заголовок в формате h1. Частично подчеркнутый.
Нумерованный список:
1. Элемент, содержащий список внутри:
1. Первый элемент.
2. Второй элемент.
2. Элемент, содержащий гиперссылку на страницу.

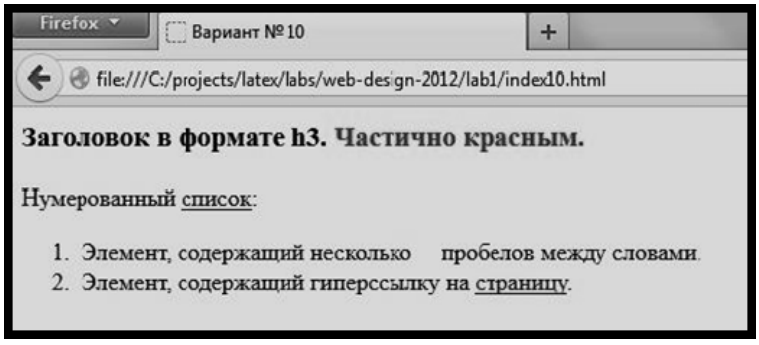

Firefox
Файл Правка Вид Журнал Закладки Инструменты Справка
Вариант № 5
file:///C:/projects/latex/labs/web-design-2012/lab1/index5.html
Заголовок в формате h2. Частично зеленым.
Маркированный список:
• Элемент, содержащий список внутри:
1. Первый элемент.
2. Второй элемент.
• Элемент, содержащий слово в угловых <скобках>


Firefox
Вариант № 6
file:///C:/projects/latex/labs/web-design-2012/lab1/index6.html
Заголовок в формате h2. Частично синим.
Маркированный список:
• Элемент, содержащий список внутри:
o Первый элемент.
o Второй элемент.
• Элемент, содержащий слово в &амперсендах&









1.3. Содержание отчета

1. Цель работы.
2. Вариант задания. Изображение требуемой страницы.

3. Исходный текст страницы.
4. Изображение полученной страницы.
5. Выводы.

1.4. Контрольные вопросы

1. Каково назначение тэга title? Где можно указывать данный тэг?
2. Каково назначение тэга meta? Какого рода информацию можно указывать в данном тэге?
3. Каково назначение тэга body? Какие атрибуты могут быть у данного тэга?
4. С помощью каких тэгов можно создавать маркированные и нумерованные списки? Можно ли для маркированного списка указать способ маркировки, отличный от стандартного? Можно ли для нумерованного списка указать способ нумерации, отличный от стандартного (например, с помощью римских цифр)?
5. Какова структура тэга в HTML-документе? Что он может содержать?
6. Допустимо ли в HTML оставить тэг незакрытым? Как браузер поведет себя в этом случае?
7. Как в HTML трактуются пробелы и новые строки? Каким образом можно принудительно заставить браузер добавить в отображение новый пробел или строку?
8. Как заставить в браузере отображать различные служебные символы HTML-страницы (например, угловые скобки, кавычки, апострофы)?
9. Как тэги преобразуются в дерево документа браузером? Нарисуйте дерево для документа ниже.

```
<html>
   <head>
      <title>Sample tree</title>
   </head>
   <body>
      <ul>
         <li>First element</li>
         <li>Second element</li>
      </ul>
   </body>
</html>
```

Лабораторная работы № 2

ЯЗЫК PHP

2.1. Цель работы

Освоение базовых конструкций языка PHP:

- средства вывода;
- организация циклов;
- операторы ветвления.

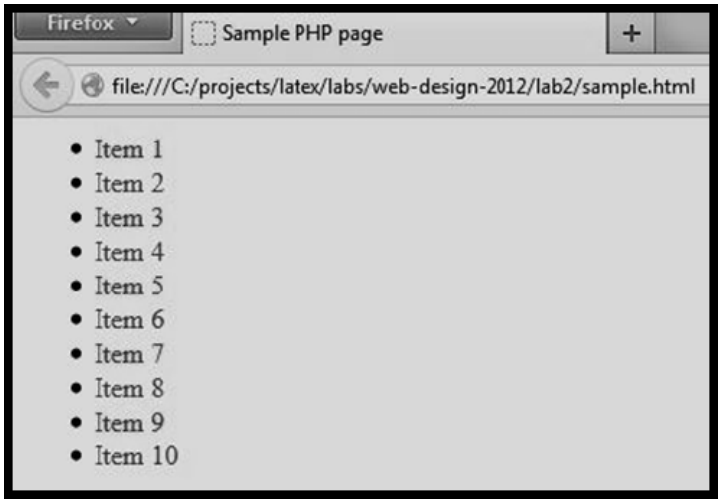
Предварительные требования

Необходимо установить следующие программные компоненты:

- Apache 2.2.x.
- PHP 5.x.

2.2. Пример выполнения задания

Предположим, нам необходимо сделать страницу, показанную на рисунке ниже. У всех элементов списка есть закономерности. Во-первых, номер элемента указывается в тексте элемента. Во-вторых, четные элементы списка выводятся зеленым, а нечетные красным.



Можно написать такую страницу вручную, а можно прибегнуть к помощи интерпретатора PHP. Одно из главных преимуществ, которое дает этот интерпретатор, – это возможность писать страницу напрямую

в формате HTML (эти фрагменты интерпретатор передает серверу в качестве ответа сразу же без обработки), вынеся в PHP только те фрагменты, которые генерируются динамически:

```
<html>
   <head><title>Sample PHP page</title></head>
   <body>
      <ul>
         <<PHP fragment goes here>>
      </ul>
   </body>
</html>
```

Поскольку большая часть страницы остается неизменной и всегда должна выдаваться в одном и том же формате, она записана напрямую в HTML. В PHP записано только то, что нужно сгенерировать программно.

2.2.1. Фрагмент PHP – вариант первый

Рассмотрим простейшую реализацию этого списка из 10 элементов с помощью цикла с условиями:

```
<?php
for ($i = 1; $i <= 10; $i++) {
   // Вывод тэга элемента
   echo "<li>";
   // Вывод тэга шрифта в зависимости от четности
   if ($i%2 == 0) echo "<font color = \"green\">";
   else echo "<font color = \"red\">";
   // Вывод содержимого элемента
   echo "Item ".$i;
   // Вывод закрывающего элемента шрифта
   echo "</font>";
   // Вывод закрывающего элемента списка с переводом на новую строку
   echo "</li>\n";
}
?>
```

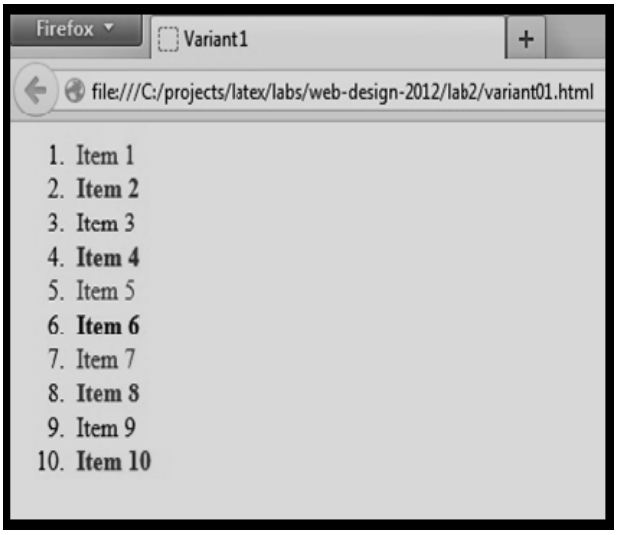
2.2.2. Фрагмент PHP – вариант второй

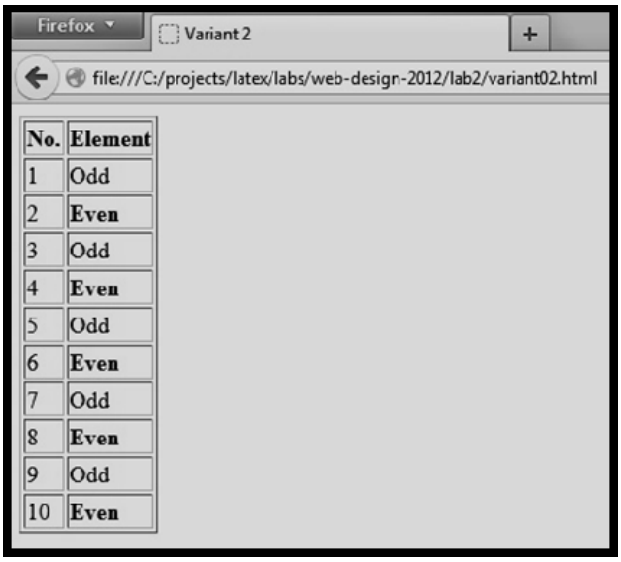
Предыдущий вариант реализует то, что нужно, однако он выглядит не очень понятно. PHP позволяет работать с выводом строк гораздо более изящно, указывая переменные напрямую в строке без всякой конкатенации:

```
<?php
for ($i = 1; $i <= 10; $i++) {
   // Установка переменной цвета
   if ($i%2 == 0) $color = "green";
   else $color = "red";
   // Вывод содержимого в совокупности
   echo "<li><font color = \"$color\">Item $i</font></li>\n";}
?>
```

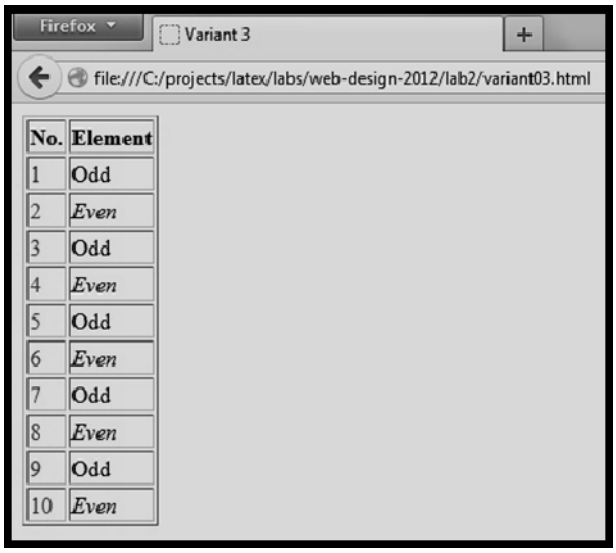
2.3. Варианты заданий

Имеется 10 вариантов заданий, представленных ниже. Необходимо проанализировать каждую страницу, выявить, какие тэги требуются для ее реализации, и написать исходный код этой страницы, после чего убедиться, что браузер отображает эту страницу именно так, как она выглядела изначально. Для каждого варианта следует проставить заголовок страницы, который отображает браузер на рисунке.

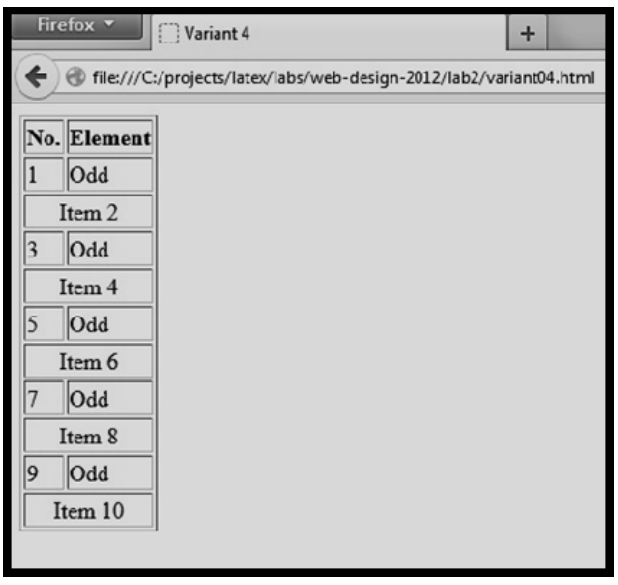




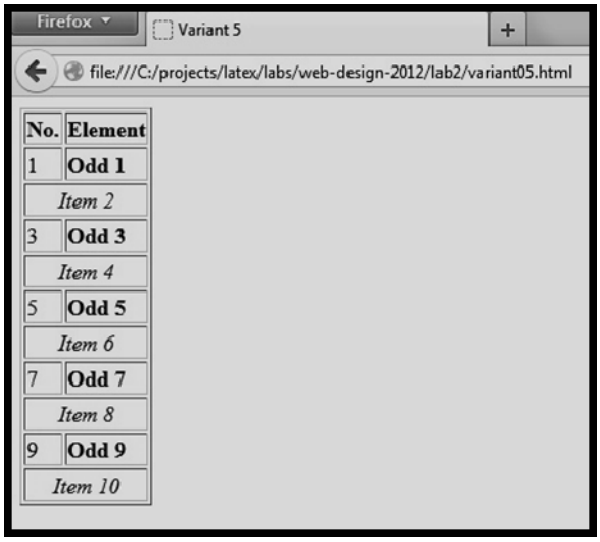
No.	Element
1	Odd
2	**Even**
3	Odd
4	**Even**
5	Odd
6	**Even**
7	Odd
8	**Even**
9	Odd
10	**Even**

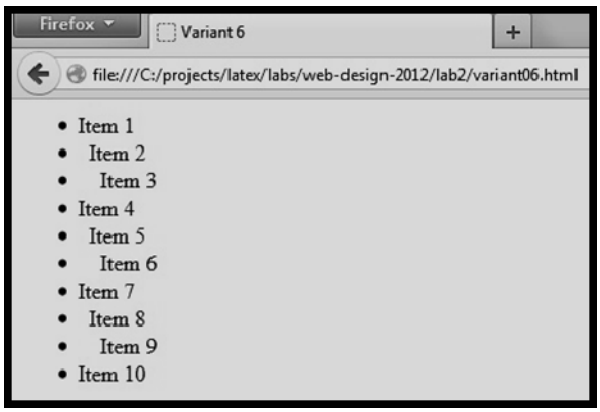


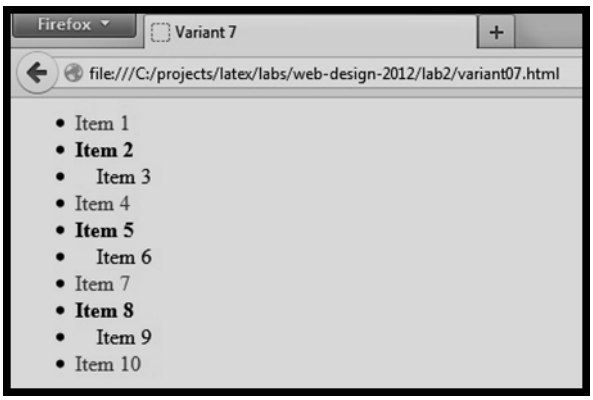
No.	Element
1	Odd
2	*Even*
3	Odd
4	*Even*
5	Odd
6	*Even*
7	Odd
8	*Even*
9	Odd
10	*Even*

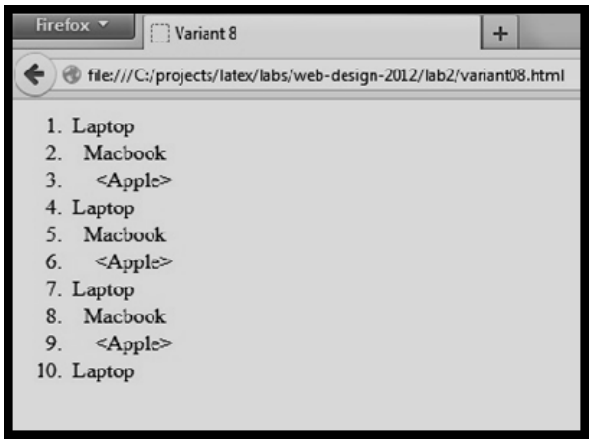


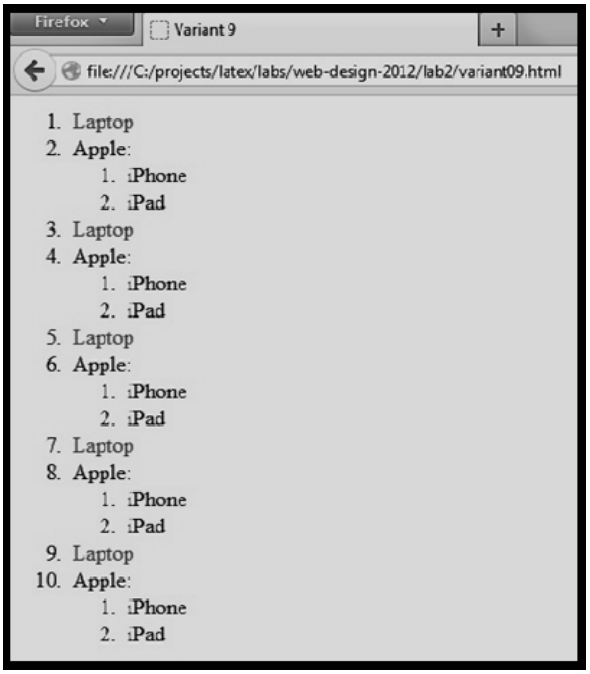
No.	Element
1	Odd
Item 2	
3	Odd
Item 4	
5	Odd
Item 6	
7	Odd
Item 8	
9	Odd
Item 10	

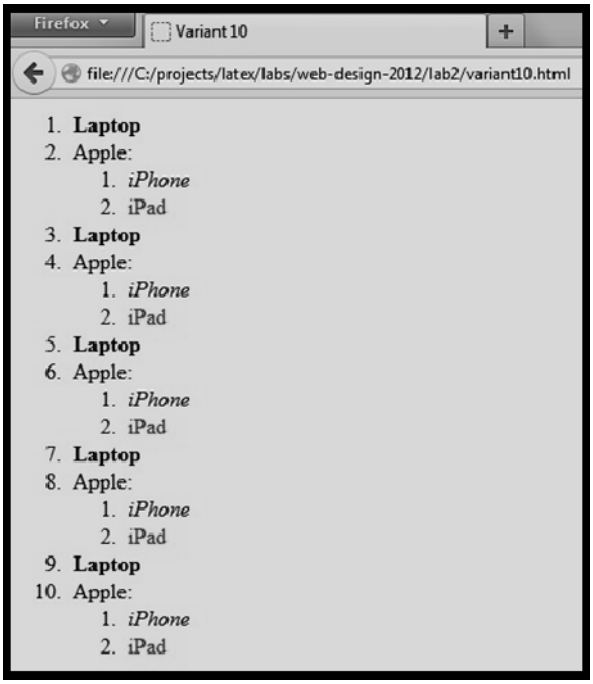

Firefox
Variant 5
file:///C:/projects/latex/labs/web-design-2012/lab2/variant05.html
No. Element
1 Odd 1
Item 2
3 Odd 3
Item 4
5 Odd 5
Item 6
7 Odd 7
Item 8
9 Odd 9
Item 10


Firefox
Variant 6
file:///C:/projects/latex/labs/web-design-2012/lab2/variant06.html
Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7
Item 8
Item 9
Item 10


Firefox
Variant 7
file:///C:/projects/latex/labs/web-design-2012/lab2/variant07.html
Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7
Item 8
Item 9
Item 10


Firefox
Variant 8
file:///C:/projects/latex/labs/web-design-2012/lab2/variant08.html
1. Laptop
2. Macbook
3. <Apple>
4. Laptop
5. Macbook
6. <Apple>
7. Laptop
8. Macbook
9. <Apple>
10. Laptop





2.4. Содержание отчета

1. Цель работы.
2. Вариант задания. Изображение требуемой страницы.
3. Исходный текст страницы.
4. Изображение полученной страницы.
5. Выводы.

2.5. Контрольные вопросы

1. Синтаксис оператора **for**. В чем его отличие от **foreach**?
2. Синтаксис оператора **echo**. Каково его отличие от оператора **print**?
3. В чем различие в обработке строк в двойных и ординарных кавычках оператором **echo**?
4. Служебные символы (кавычки, перенос строк и т. д.) и их вывод оператором **echo**.
5. Операторы присваивания и сравнения. В чем отличие оператора == от оператора ===?

Лабораторная работа № 3

УГЛУБЛЕННОЕ ЗНАКОМСТВО С AMP

3.1. Цель работы

Установка и настройка компонентов платформы AMP, знакомство с базовыми конструкциями языка PHP.

3.2. Компоненты платформы AMP

3.2.1. Описание Apache

Для того чтобы сервер мог обрабатывать запросы от клиентов и отсылать ответы по протоколу HTTP, необходимо, чтобы порт, к которому обращаются клиенты (по умолчанию 80-й для данного протокола), кто-то обслуживал (или, говоря иначе, *привязывался* к этому порту).

К порту привязывается какое-либо программное обеспечение, которое в дальнейшем принимает все запросы, приходящие на этот порт, разбирает содержимое запроса и в зависимости от него генерирует ответ, который отсылается клиенту. Два различных приложения не могут одновременно обслуживать один и тот же порт, это должно делать одно приложение.

Естественно, это приложение должно выполняться в *фоновом* режиме, даже если на сервере не запущен сеанс работы с пользователем. В

Windows такое приложение обычно регистрируется в виде *сервиса*, который запускается автоматически при запуске системы. В Linux такое приложение запускается в виде *демона*, который работает в фоновом режиме и может запускаться автоматически.

Есть несколько вариантов подобного обеспечения. Одним из наиболее распространенных является Apache (его версии существуют и под Linux, и под Windows). В его конфигурационном файле прописываются следующие настройки (самые важные в рамках данной лабораторной работы):

- какой порт обслуживать (Apache может одновременно обслуживать и несколько портов);
- какой каталог является корневым для сервера;
- какие обработчики использовать для различных типов страниц;
- какие страницы пытаться найти по умолчанию, если адрес указан *без страницы*.

3.2.2. Описание PHP

Язык PHP, созданный в конце 90-х гг., предназначен в первую очередь для web-программирования. Модуль PHP обрабатывает страницы (обычно с расширением .php), которые содержат специальные тэги вида

```
<?php
//...
?>
```

Файл преобразуется модулем PHP следующим образом. Все, что вне этих тэгов, остается неизменным, а содержимое тэгов заменяется на результат выполнения кода внутри них. Этот модуль можно запускать, например, из командной строки

```
php.exe index.php
```

Результатом будет то, что модуль отправит в консоль вывода результат обработки файла.

3.2.3. Описание MySQL

Web-сайт чаще всего не является статическим, т. е. его содержимое меняется в зависимости от действий пользователей. При этом сама структура сайта (его страниц, стилей и прочего) не меняется, меняется лишь внутреннее содержимое некоторых фрагментов. Например, лента новостей сама по себе внешне не меняется, меняются лишь сами новости.

Соответственно эти изменяемые данные должны где-то храниться, желательно отдельно от самого сайта. Обычно для этого используют базы данных. Одной из наиболее популярной является MySQL. Подобно Apache, MySQL при установке регистрируется в виде сервиса или демона (в зависимости от ОС). Она привязывается к специальному порту (по умолчанию 3306). Далее те приложения, которые желают использовать инструментарий баз данных, должны обращаться по этому порту по специальному протоколу к базе и получать от нее ответ. Внутреннее содержимое базы данных от внешних приложений скрыто, они могут лишь пересылать по протоколу различные команды и получать в ответ от базы результаты выполнения этих команд.

3.2.4. Межвзаимодействие компонентов

Рассмотрим теперь, как все это работает в совокупности. Пусть на сервер приходит запрос вида

```
http://www.yourserver.com/index.php
```

Этот запрос обработает Apache. В первую очередь он проверит, есть ли в корневом каталоге страница index.php. Если нет, то сгенерирует ошибку 404 (файл не найден). Если же есть, то он проведет сопоставление расширения страницы с теми расширениями, что прописаны в его конфигурационном файле. Если он не найдет сопоставления, то просто выдаст клиенту в ответ содержимое страницы "as-is".

Если в конфигурационном файле прописано, что страницы с расширением .php должен обрабатывать специальный модуль PHP, Apache запустит для обработки страницы этот модуль, попутно пе-

редав ему дополнительную информацию (окружение сервера, содержимое запроса от клиента и т. д.). Модуль обработает страницу и перешлет Apache результат обработки. Этот результат Apache и отправит назад клиенту.

Если на странице с расширением .php в коде имеются фрагменты работы с базой MySQL, модуль PHP соединится с ней посредством драйвера, получит необходимые данные и добавит их в страницу, которую перешлет по итогам Apache (оформив эти данные в необходимом формате).

Таким образом, сам Apache с базой данных никак не взаимодействует, а только с PHP. PHP может при необходимости взаимодействовать с базой данных, связываясь с ней по 3306-му порту посредством драйвера.

3.3. Пример выполнения задания

Рассмотрим следующую задачу:

Пример 1. *В базе имеется набор существующих в системе групп. Необходимо вывести этот набор в алфавитном порядке на странице в виде маркированного списка.*

3.3.1. Схема и заполнение базы данных

Будем предполагать, что у группы есть поле **name**, других важных для пользователя полей в ней нет. Тогда скрипт по созданию таблицы групп может быть таким:

```
CREATE TABLE groups (
   id INT(11) NOT NULL AUTO_INCREMENT,
   name VARCHAR(100) NOT NULL,
   PRIMARY KEY (id asc)
) ENGINE = InnoDB CHARSET = utf8;
```

Код создает таблицу с двумя полями, одно из них выступает в качестве первичного ключа. В реальных системах содержимое базы определяют пользователи, которые выполняют различные сценарии

на страницах. Однако в данном случае заполним таблицу вручную посредством следующего скрипта:

```
INSERT INTO groups(name) VALUES
   ('CCC'), ('AAA'), ('DDD'), ('BBB');
```

Скрипт специально вставляет данные не в алфавитном порядке, чтобы отразить тот факт, что данные могут добавляться пользователями в произвольном порядке.

3.3.2. Пример простой страницы на PHP

Прежде чем делать требуемую страницу с отображением информации, сделаем более простой вариант. Этот вариант будет отображать общую информацию о сервере, на котором установлен модуль PHP. Для этого используем системную функцию **phpinfo**. Вот как выглядит данная страница **info.php**:

```
<html>
   <head>
      <title>Информация</title>
   </head>
   <body>
   <?php
   phpinfo();
   ?>
   </body>
</html>
```

Обработчик действует очень просто. Все, что идет до открывающего тэга php, он пересылает Apache без изменений. Далее обработчик выполняет код внутри этого тэга. Все, что данный код отправит в консоль вывода (в примере ниже этот вывод проводится явно), обработчик перешлет Apache как продолжение страницы. То что идет после закрывающего тэга php, обработчик пересылает Apache также без изменений.

По итогам Apache отправит клиенту преобразованную страницу, которую он получил от обработчика PHP.

3.3.3. Пример сложной страницы на PHP

Теперь рассмотрим нужную нам страницу. Необходимо выбрать из базы названия групп, отсортировав их в алфавитном порядке. Далее в нужном месте страницы мы должны вставить эти названия в виде элементов списка. Рассмотрим эту страницу **groups.php**. Она будет сразу начинаться с фрагмента, который подключается к базе и получает нужные сведения, сохраняя их в виде массива:

```
<?php
// Соединение с сервером
// В случае ошибки прерываем выполнение скрипта
$link = mysql_connect("localhost", "root", "12345")
   or die("Could not connect: " . mysql_error());
// Выбираем нужную базу, в случае ошибки
// прерываем выполнение скрипта
mysql_select_db("testbase")
   or die("Could not select db: " . mysql_error());
// Заводим массив, в который будем сохранять данные
$groups = array();
// Выполняем запрос к базе, в случае ошибки
// прерываем выполнение скрипта
$rs = mysql_query("SELECT name FROM groups ORDER BY name")
   or die("Failed to retrieve data: " . mysql_error());
// Извлекаем данные из выборки и сохраняем в массиве
while ($row = mysql_fetch_assoc($rs)) {
   $name = $row["name"];
   // Добавляем имя в конец массива
   $groups []= $name;
}
// Не забываем закрыть соединение и избавиться
// от ресурсов, которые нам уже не нужны
mysql_free_result($rs);
mysql_close($link);
?>
```

Как видно, по итогам выполнения этого фрагмента был создан массив **groups** (он будет доступен в фрагменте PHP ниже на этой же странице). По итогам выполнения в консоль вывода ничего не отправляется, следовательно, по завершении этого фрагмента обработчик не пе-

решлет Apache ровным счетом ничего. Следующую часть страницы (заголовок и начало списка) выводим вне тэга PHP, чтобы обработчик переслал ее Apache без изменений (это продолжение страницы **groups.php**):

```
<html>
   <head>
      <title>Список групп</title>
   </head>
   <body>
      <ul>
```

Далее начинаем новый фрагмент PHP. Вообще PHP должен отправлять в консоль вывода только то, что реально генерируется динамически и зависит от содержимого базы. В данном случае сама структура страницы будет неизменной, поэтому выводится вне PHP. В рамках PHP выводятся лишь элементы списка

```
<?php
foreach ($groups as $value) {
   // А вот и команда вывода
   echo "<li>$value</li>\n";
}
?>
```

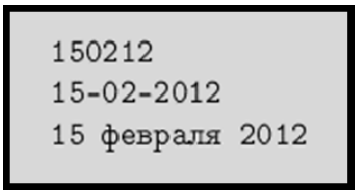
При использовании команды вывода в строковой константе производится замена названия переменной на ее реальное значение. Каждое значение упаковывается в виде элемента списка. Оставшаяся часть страницы вне PHP закрывает список и тело документа:

```
      </ul>
   </body>
</html>
```

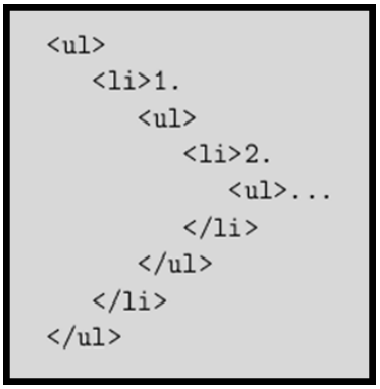
3.4. Варианты заданий

В каждом варианте сделать полноценную страницу (с заголовками, тэгами тела документа), которая должна содержать PHP-фрагмент, реализующий то, что требуется по варианту:

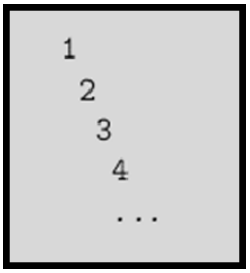
1) на странице в отдельных строках должна быть выведена текущая дата в следующих форматах: "ДДММГГ", "ДД-ММ-ГГГГ" и "ДД <название месяца> ГГГГ" (с использованием системных функций PHP)



2) выведите маркированный список чисел от 1 до 10 следующего вида:



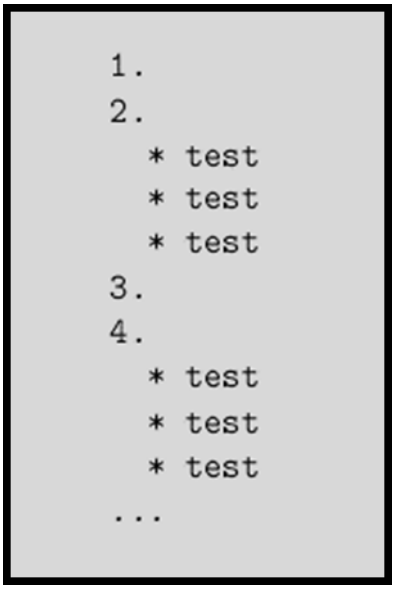
3) выведите числа от 1 до 10 в разных строках с увеличивающимся отступом



4) выведите таблицу из двух столбцов и десяти строк. В первом столбце содержится номер строки. Во втором – слово **test**. Для четных строк слово должно быть красного цвета, для нечетных – зеленого.

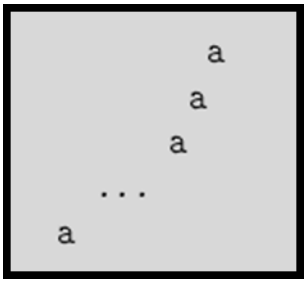
5) выведите таблицу из двух столбцов и десяти строк. В первом столбце должен содержаться номер строки, во втором – даты, начиная с сегодняшней, в формате "ДД-ММ-ГГГГ"

6) выведите нумерованный список из 10 элементов. Четные элементы должны содержать маркированные списки со словом **test**, нечетные – просто номера



7) выведите таблицу из двух столбцов и десяти строк. Четные строки должны быть выделены курсивом, нечетные – жирным шрифтом. Первый столбец должен содержать номер строки, второй – значение $i^2 + i + 2$, где i – номер строки.

8) выведите следующую перевернутую "лесенку" из 10 элементов:



9) выведите маркированный список из десяти элементов. Четные элементы должны быть красного цвета, нечетные – черного. Каждую третью строку вдобавок выделяют курсивом. В качестве текста в каждом элементе списка должно содержаться слово **test**.

10) выведите маркированный список из десяти элементов. Четные строки должны содержать нумерованные списки с датами в двух разных форматах, начиная с сегодняшней. Все строки имеют слово **test**

```
* test
* test
  1. 15-02-2012.
  2. 15 февраля 2012.
* test
* test
  1. 16-02-2012.
  2. 16 февраля 2012.
...
```

3.5. Содержание отчета

1) Цель работы.
2) Содержание варианта задания.
3) Код программы на PHP.
4) Скриншоты результатов выполнения страниц.
5) Выводы.

3.6. Контрольные вопросы

Для более глубокого понимания процессов, происходящих во время выполнения скриптов PHP, необходимо ответить на следующие контрольные вопросы. Эти вопросы будут перечислены в качестве тестов в дистанционном курсе. Для подготовки следует использовать прикрепленную справочную информацию.

1. Является ли PHP *интерпретатором* или *компилятором*? В чем разница между двумя способами выполнения программ?
2. Какой неявный механизм управления памятью используется в PHP? В каких ситуациях этот механизм может не срабатывать (в таком случае программисту придется управлять памятью явно в своем программном коде)?
3. Объявление и инициализация переменных в PHP.
4. Преобразование типов в PHP. В чем разница между операторами "==" и "==="?
5. Комплексные типы в PHP на основе stdClass. Как задаются поля в таких комплексных типах?

6. Каково назначение функций **isset** и **unset**?
7. В чем разница между функциями **isset** и **isempty**? Какие из утверждений, приведенных ниже, верны:
   а) переменная может быть одновременно неопределена и пуста;
   б) переменная может быть одновременно неопределена и не пуста;
   в) переменная может быть одновременно определена и пуста;
   г) переменная может быть одновременно определена и не пуста.
8. Каковы общепринятые правила наименования в PHP (для переменных, функций и классов)?
9. Строки в PHP. Какова разница между строковыми константами в двойных и ординарных кавычках?
10. Механизм замены в строках. Как подставить реальное значение переменной в строковую константу при выводе, если это необходимо?

Лабораторная работа № 4

БАЗЫ ДАННЫХ

4.1. Цель работы

Знакомство с базами данных и способами извлечения данных на уровне PHP.

4.2. Скрипт для создания таблиц

Рассмотрим следующую задачу. Пусть нам нужно хранить сведения о студентах. Для этого создадим две таблицы: первая таблица о группах, вторая – о самих студентах (со ссылкой на группу). Предположим, что у нас уже есть база, нужная нам, и требуется создать только сами таблицы

```
CREATE TABLE `group` (
   `id` INT(10) NOT NULL AUTO_INCREMENT,
   `name` VARCHAR(250) NOT NULL,
   PRIMARY KEY (`id`)
) ENGINE = InnoDB DEFAULT_CHARSET = utf8;

CREATE TABLE `student` (
   `id` INT(10) NOT NULL AUTO_INCREMENT,
   `name` VARCHAR(250) NOT NULL,
   `group` INT(10) NOT NULL,
   PRIMARY KEY (`id`),
   FOREIGN KEY (`group`) REFERENCES `group` (`id`) ON DELETE CASCADE
) ENGINE = InnoDB DEFAULT_CHARSET = utf8;
```

4.3. Скрипт для заполнения таблиц

Поскольку здесь необходимо проставлять ссылки на таблицу групп, идентификаторы у нее нужно задавать явно. Для второй таблицы идентификаторы можно не задавать, они будут сгенерированы автоматически

```
INSERT INTO `group` (`id`, `name`) VALUES
   (1, 'Group A'), (2, 'Group B');

INSERT INTO `student` (`name`, `group`) VALUES
   ('Ivanov', 1), ('Petrov', 1), ('Sidorov', 2);
```

4.4. Скрипт для удаления таблиц

Заодно на случай переустановки базы необходимо сделать скрипт для удаления таблиц

```
DROP TABLE `student`;
DROP TABLE `group`;
```

4.5. Страница с данными

Страница будет выглядеть несколько иначе, чем в предыдущих работах. Первым в ней будет фрагмент попытки подключения к базе. Если этого сделать не удалось, то выдается сообщение об ошибке. За-

тем – фрагмент HTML. Внутри него есть фрагмент прохода по выборке из базы. После фрагмента HTML идет фрагмент очистки соединения.

4.5.1. Подключение к базе

Вот как выглядит этот фрагмент:

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '123';

$link = mysql_connect($host, $user, $pass)
   or die('Failed to connect to the database. Error: '
      . mysql_error());
mysql_select_db('test') or
   die('Failed to select database. Error: ' . mysql_error());
$query = "SELECT student.name as name, `group`.name as gname"
   . "FROM student LEFT JOIN `group`"
   . "ON student.`group` = `group`.id";
$rs = mysql_query($query) or
   die("Failed to execute query. Error: "
      . mysql_error());
?>
```

По итогам этого фрагмента мы имеем выборку данных и подключение к базе.

4.5.2. Фрагмент HTML

Вот как выглядит страница. Это обычный одноуровневый список:

```
<html>
   <head><Title>Database</title></head>
   <body>
      <ul>
      <!-- PHP Fragment goes here-->
      </ul>
   </body>
</html>
```

4.5.3. Проход по выборке

Вот так выглядит проход по выборке:

```
<?php
while ($row = mysql_fetch_assoc($rs)) {
   $name = $row['name'];
   $groupname = $row['gname'];

   echo "<li>Student - $name. Group - $groupname.</li>\n";
}
?>
```

4.5.4. Освобождение ресурсов

После того как данные выведены и соединение более не нужно, необходимо освободить ресурсы, чтобы база могла получить соединение назад

```
<?php
mysql_free_result($rs);
mysql_close($link);
?>
```

4.6. Варианты заданий

В каждом варианте надо написать скрипт для создания таблиц базы, наполнения базы данными и удаления таблиц базы. Заодно нужно написать страницу согласно заданию. Каждая таблица в базе должна содержать нормальный первичный ключ, который генерируется автоматически. Если таблица ссылается на другую, следует прописать нормальный внешний ключ.

1. Необходимо создать две таблицы. Первая хранит в себе сведения о фирмах, а именно название фирмы (до 250 символов). Вторая – сведения об автомобилях, производимых фирмами: название автомобиля (до 150 символов), его стоимость (целое число) и ссылка на фирму. Далее вывести сведения о фирмах и автомобилях в виде двухуровневого списка. Первый уровень содержит названия фирм в алфа-

витном порядке. На втором расположены сведения об автомобилях для данной фирмы, отсортированные по названиям автомобилей в алфавитном порядке. Все данные страница должна извлечь из базы одним запросом, т. е. нельзя сделать запрос чисто для фирм, а потом для каждой фирмы извлекать дополнительным запросом сведения об автомобилях. Выглядеть страница должна так:

```
1. Ford.
   1. Name - extended. Cost - 2500.
   2. Name - T-model. Cost - 2700.
2. Mercedes.
   1. Name - advanced. Cost - 2800.
   2. Name - Basic. Cost - 2900.
```

2. Необходимо создать две таблицы. Первая хранит в себе сведения о фирмах, а именно название фирмы (до 100 символов). Вторая – сведения об автомобилях, производимых фирмами: название автомобиля (до 150 символов), его стоимость (целое число) и ссылка на фирму. Далее вывести сведения о фирмах и автомобилях в виде двухуровневого списка. Первый уровень содержит названия фирм в алфавитном порядке. На втором расположены сведения об автомобилях для данной фирмы, отсортированные по стоимости автомобилей в убывающем порядке. Все данные страница должна извлечь из базы одним запросом, т. е. нельзя сделать запрос чисто для фирм, а потом для каждой фирмы извлекать дополнительным запросом сведения об автомобилях. Выглядеть страница должна так:

```
* Ford.
   1. Name - T-model. Cost - 2700.
   2. Name - extended. Cost - 2500.
* Mercedes.
   1. Name - Basic. Cost - 2900.
   2. Name - advanced. Cost - 2800.
```

3. Необходимо создать две таблицы. Первая хранит в себе сведения о фирмах, а именно название фирмы (до 150 символов). Вторая – сведения об автомобилях, производимых фирмами: название автомобиля (до 200 символов), его стоимость (целое число) и ссылка на фирму. Далее вывести сведения о фирмах и автомобилях в виде

одноуровневого списка. Каждый элемент списка должен содержать сведения о фирме, названии модели и стоимости. Отсортировывают элементы по названию фирмы и автомобиля. Выглядеть страница должна так:

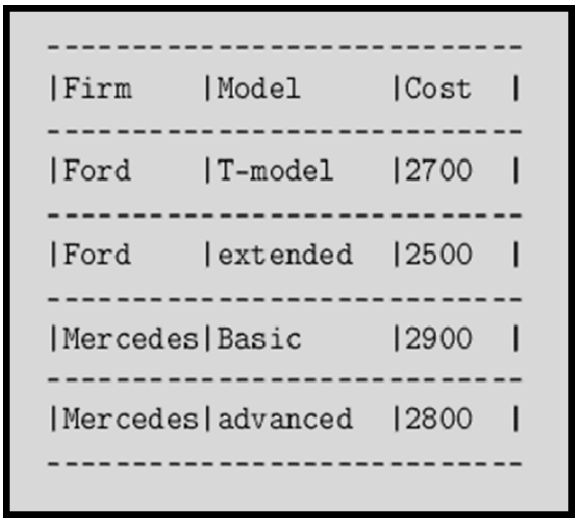
```
1. Firm - Ford. Name - extended. Cost - 2500.
2. Firm - Ford. Name - T-model. Cost - 2700.
3. Firm - Mercedes. Name - advanced. Cost - 2800.
4. Firm - Mercedes. Name - Basic. Cost - 2900.
```

4. Необходимо создать две таблицы. Первая хранит в себе сведения о фирмах, а именно название фирмы (до 250 символов). Вторая – сведения об автомобилях, производимых фирмами: название автомобиля (до 150 символов), его стоимость (целое число) и ссылка на фирму. Далее вывести сведения о фирмах и автомобилях в виде таблицы из двух столбцов. Сведения о каждой фирме даны в отдельных строках, объединяющих сразу два столбца. Эти строки отсортировывают в алфавитном порядке. Далее для каждой фирмы следуют сведения об автомобилях, остортированные по названиям автомобилей. Все данные страница должна извлечь из базы одним запросом, т.е. нельзя сделать запрос чисто для фирм, а потом для каждой фирмы извлекать дополнительным запросом сведения об автомобилях. Выглядеть страница должна так:

```
-----------------
|Model    |Cost |
-----------------
|   Ford        |
-----------------
|extended |2500 |
-----------------
|T-model  |2700 |
-----------------
|  Mercedes     |
-----------------
|advanced |2800 |
-----------------
|Basic    |2900 |
-----------------
```

5. Необходимо создать две таблицы. Первая хранит в себе сведения о фирмах, а именно название фирмы (до 250 символов), вторая – сведения об автомобилях, производимых фирмами: название

автомобиля (до 150 символов), его стоимость (целое число) и ссылка на фирму. Необходимо вывести сведения о фирмах и автомобилях в виде таблицы из трех столбцов. Сведения отсортировывают по названию фирмы (в алфавитном порядке) и стоимости автомобиля (в порядке убывания). Выглядеть страница должна так:



Firm	Model	Cost
Ford	T-model	2700
Ford	extended	2500
Mercedes	Basic	2900
Mercedes	advanced	2800

6. Необходимо создать две таблицы для хранения сведений об отелях. Первая хранит сведения о фирмах – название фирмы (до 150 символов). Вторая – сведения об отелях: название отеля (до 150 символов), стоимость проживания в отеле и название страны, где расположен отель (до 150 символов), а также ссылка на фирму. Сведения об отелях выводятся в виде двухуровневого списка. На первом уровне должны располагаться названия стран в алфавитном порядке. На втором – сведения об отелях для данной страны, отсортированные по названию фирмы и отеля в алфавитном порядке. Все данные страница должна извлечь из базы одним запросом, т. е. нельзя сделать запрос чисто для стран, а потом для каждой страны извлекать дополнительным запросом сведения об отелях. Выглядеть страница должна так:

```
1. Egypt.
   1. Firm - Arch. Hotel - Awesome. Cost - 1000.
   2. Firm - Arch. Hotel - Best. Cost - 800.
   3. Firm - Tour. Hotel - Epic. Cost - 100.
2. France.
   1. Firm - Arch. Hotel - Ritz. Cost - 1200.
   2. Firm - Arch. Hotel - Zimmer. Cost - 700.
   3. Firm - Tour. Hotel - Fail. Cost - 50.
```

7. Необходимо создать две таблицы для хранения сведений об отелях. Первая хранит сведения о фирмах – название фирмы (до 150 символов). Вторая – сведения об отелях: название отеля (до 150 символов), стоимость проживания в отеле и название страны, где расположен отель (до 150 символов), а также ссылка на фирму. Сведения об отелях выводятся в виде двухуровневого списка. На первом уровне должны располагаться названия фирм в алфавитном порядке. На втором – сведения об отелях для данной фирмы, отсортированные по названию страны и отеля в алфавитном порядке. Все данные страница должна извлечь из базы одним запросом, т. е. нельзя сделать запрос чисто для фирм, а потом для каждой фирмы извлекать дополнительным запросом сведения об отелях. Выглядеть страница должна так:

```
* Arch.
   1. Country - Egypt. Hotel - Awesome. Cost - 1000.
   2. Country - Egypt. Hotel - Best. Cost - 800.
   3. Country - France. Hotel - Ritz. Cost - 1200.
   4. Country - France. Hotel - Zimmer. Cost - 700.
* Tour.
   1. Country - Egypt. Hotel - Epic. Cost - 100.
   2. Country - France. Hotel - Fail. Cost - 50.
```

8. Необходимо создать две таблицы для хранения сведений об отелях. Первая хранит сведения о фирмах – название фирмы (до 150 символов). Вторая – сведения об отелях: название отеля (до 150 символов), стоимость проживания в отеле и название страны, где расположен отель (до 150 символов), а также ссылка на фирму. Необходимо вывести сведения об отелях в виде таблицы из трех столбцов. Сведения о фирмах должны идти в алфавитном порядке в строках, объединяющих все три столбца. Для каждой фирмы далее идут строки со сведениями об отелях, отсортированные по названию страны (в алфавитном порядке), и стоимости отеля (в порядке возрастания). Все данные страница должна извлечь из базы одним запросом, т. е. нельзя сделать запрос чисто для фирм, а потом для каждой фирмы извлекать дополнительным запросом сведения об отелях. Выглядеть страница должна так:

Country	Hotel	Cost
Arch		
Egypt	Best	800
Egypt	Awesome	1000
France	Zimmer	700
France	Ritz	1200
Tour		
Egypt	Epic	100
France	Fail	50

9. Необходимо создать две таблицы для хранения сведений об отелях. Первая хранит сведения о фирмах – название фирмы (до 150 символов). Вторая – сведения об отелях: название отеля (до 150 символов), стоимость проживания в отеле и название страны, где расположен отель (до 150 символов), а также ссылка на фирму. Сведения об отелях выводятся в виде таблицы из трех столбцов. Сведения о странах должны идти в алфавитном порядке в строках, объединяющих все три столбца. Для каждой страны далее идут строки со сведениями об отелях, отсортированные по названию фирмы (в алфавитном порядке) и стоимости отеля (в порядке возрастания). Все данные страница должна извлечь из базы одним запросом, т. е. нельзя сделать запрос чисто для стран, а потом для каждой страны извлекать дополнительным запросом сведения об отелях. Выглядеть страница должна так:

Firm	Hotel	Cost
Egypt		

10. Необходимо создать две таблицы для хранения сведений об отелях. Первая хранит сведения о фирмах – название фирмы (до 150 символов). Вторая – сведения об отелях: название отеля (до 150 символов), стоимость проживания в отеле и название страны, где расположен отель (до 150 символов), а также ссылка на фирму. Сведения об отелях выводятся в виде таблицы из четырех столбцов. Сведения отсортировывают по названиям стран, фирм и отелей. Выглядеть страница должна так:

4.7. Содержание отчета

1. Цель работы.
2. Содержание варианта задания.
3. Скрипты для базы.
4. Код программы на PHP.
5. Скриншоты результатов выполнения страниц.
6. Выводы.

4.8. Контрольные вопросы

1. Сигнатура функции **mysql_connect**. Какие аргументы она принимает на вход? Что она возвращает в случае, если соединение произведено успешно? Что она возвращает, если соединение с базой не удалось осуществить? Как в таком случае посмотреть детальные сведения об ошибке?
2. Сигнатура функции **mysql_select_db**. Какие аргументы она принимает на вход? Что она возвращает в случае, если удалось выбрать требуемую базу? Что она возвращает, если требуемую базу выбрать не удалось? Как в таком случае посмотреть детальные сведения об ошибке?
3. Сигнатура функции **mysql_query**. Какие аргументы она принимает на вход? Что она возвращает для следующих запросов:
    а) запрос на вставку;
    б) запрос на удаление;
    в) запрос на модификацию;
    г) запрос на выборку.
4. Приведите примеры соединений таблиц в SQL-запросах. Каковы отличительные характеристики левого, правого и внутреннего соединения таблиц?
5. Сигнатура функции **mysql_fetch_assoc**. Что именно она возвращает?

Лабораторная работа № 5

GET-ЗАПРОСЫ

5.1. Цель работы

Знакомство с взаимодействием между сервером и клиентом с помощью GET-запросов.

5.2. Адресная строка браузера

В предыдущих работах рассматривались динамические страницы, которые генерировали содержимое для выдачи пользователю, на основе источника данных. Однако пользователь не имел возможности каким-либо образом влиять на содержимое этого источника. Если быть более точным, то пользователь не имел возможности вообще как-то взаимодействовать с сервером. В данной работе рассматривается первая из таких возможностей на основе GET-запросов. GET-запросы к серверу можно генерировать разными способами. Именно их генерирует браузер, когда пользователь нажимает на какую-либо ссылку или вводит адрес напрямую в адресной строке браузера. GET-запрос содержит следующую информацию:

- адрес страницы на сервере, к которой следует обратиться;
- набор дополнительных параметров в виде пар 'key=value', которые идут после адреса страницы, отделенные от нее знаком вопроса. Сами пары друг от друга отделяются знаком амперсенда.

Рассмотрим три примера ссылок:

```
<!-- Первая ссылка, только адрес-->
http://www.example.com/info.php
<!-- Вторая ссылка, с дополнительным параметром-->
http://www.example.com/info.php?id=7
<!-- Третья ссылка, с набором дополнительных параметров-->
http://www.example.com/info.php?id=7&language=ru&status=1
```

Первая ссылка наиболее употребляемая. В GET-запросе содержится только адрес. Во втором варианте помимо адреса имеется еще

и дополнительный параметр под названием **id**, значение которого равно 7. В третьем варианте дополнительных параметров несколько. Следует заметить, что все параметры в GET-запросе строковые и передаются на сервер именно как строки.

В интерпретаторе PHP есть возможности обработки параметров GET-запроса, которые рассматриваются в примере ниже.

5.3. Пример выполнения задания

Рассмотрим следующую задачу:

Пример 2. *Имеется таблица с данными об автомобилях, где содержатся:*

- *первичный ключ автомобиля;*
- *название автомобиля;*
- *год выпуска;*
- *название фирмы;*
- *стоимость;*
- *мощность двигателя.*

Необходимо реализовать две страницы. Первая позволяет просмотреть детальные сведения о конкретном автомобиле по его ключу. На ней в виде маркированного списка должны отображаться все сведения об автомобиле. На второй странице помещена таблица со списком автомобилей, в которой для автомобилей указываются только названия. Чтобы просмотреть детальные сведения, пользователь должен перейти по ссылке "Просмотр", содержащейся в каждой строке.

5.3.1. Страница с детальным просмотром сведений

Рассмотрим еще раз детально, что происходит, когда браузеру нужно обратиться по ссылке вида:

```
http://www.example.com/info.php?id=7&language=ru&status=1
```

Сначала браузер производит разбор ссылки. Он формирует GET-запрос, специальную структуру, содержащую несколько служебных полей, а также поля адреса и информации. В поле адреса в

специальном формате записывается адрес страницы на сервере, к которой мы хотим обратиться. В поле информации записываются пары ’key-value’, которые также отсылаются на сервер.

Когда на сервер приходит GET-запрос, сервер производит его расшифровку, извлекает параметры GET-запроса и передает их интерпретатору PHP, который и выполняет требуемую страницу. В результате код страницы получает возможность обратиться к глобальному окружению интерпретатора и извлечь дополнительные параметры запроса. Рассмотрим страницу детального просмотра **show.php**. Ей необходимо передать один параметр – ключ автомобиля. Он должен иметь какое-то фиксированное название. Назовем его **pk**. В первую очередь на странице нужно произвести проверку, передан ли действительно такой дополнительный параметр

```
<?php
$pk = $_GET['pk'];
if (empty($pk)) {
   die("The required param 'pk' not specified");
}
?>
```

Далее при работе с базой нужно иметь в виду, что хотя ключ и указан, но записи с таким ключом, возможно, не существует:

```
<?php
$link = mysql_connect($host, $user, $pass)
   or die("Failed to connect to server. Error: " . mysql_error());
mysql_select_db("testdb")
   or die("Failed to select database. Error: " . mysql_error());

$query = "SELECT name, year, firm, power FROM auto WHERE id = " . $pk;
$rs = mysql_query($query)
      or die("Failed to select data from database. Error: " . mysql_error());
while ($row = mysql_fetch_assoc($rs)) {
   $auto = new stdClass;
   $auto->name = $row["name"];
   $auto->year = $row["year"];
   $auto->firm = $row["firm"];
   $auto->power = $row["power"];
}
mysql_free_result($rs);
mysql_close_link($link);

if (empty($auto)) {
   die("The record with key $pk does not exist");
}
?>
```

Обратите внимание, что данная страница поддерживает неявно следующий *программный контракт*:

- странице должен быть передан непустой параметр под названием **pk**;
- этот параметр числовой;
- в базе должна существовать запись с таким параметром.

Если этот контракт нарушен тем приложением, которое обратилось к странице, то выполнение будет досрочно прекращено. Если же контракт выполнен, то будет отображена страница с детальными сведениями:

```
<html>
   <head><title>Detailed info</title></head>
   <body>
      <h1>Detailed info about auto</h1>
      <ul>
         <!-- The php fragment goes here-->
      </ul>
   </body>
</html>
```

Обратите внимание, что в этом последнем фрагменте PHP уже не происходит никакого взаимодействия с базой, это было сделано раньше до фазы отображения данных

```
<?php
echo "<li>Name - {$auto->name}</li>";
echo "<li>Firm - {$auto->firm}</li>";
echo "<li>Year - {$auto->year}</li>";
echo "<li>Power - {$auto->power}</li>";
?>
```

5.3.2. Страница со списком

Что касается страницы со списком, то здесь отсутствует фаза проверки контракта, есть лишь две фазы – обращение к базе и отображение сведений. Назовем эту страницу **list.php**

```
<?php
$link = mysql_connect($host, $user, $pass)
   or die("Failed to connect to server. Error: " . mysql_error());
mysql_select_db("testdb")
   or die("Failed to select database. Error: " . mysql_error());

$query = "SELECT id, name FROM auto ORDER BY name";
$rs = mysql_query($query)
   or die("Failed to select data. Error: " . mysql_error());
?>
```

Далее идет фаза отображения

```
<html>
   <head><title>List of autos</title></head>
   <body>
      <table>
         <th>Название</th>
         <th>Операция</th>
         <!-- The php fragment goes here-->
      </table>
   </body>
</html>
```

Что касается названия, то здесь ясно, что выводить. Но что выводить в операции? Это должна быть ссылка на страницу **show.php**, которая соблюдает контракт, т. е. эта ссылка должна передавать идентификатор записи, назвав его именно **pk**, а не как-то иначе. Надо не забыть также очистить соединение и ресурсы

```
<?php
while ($row = mysql_fetch_assoc($rs)) {
   $name = $row["name"];
   $id = $row["id"];
   $url = "<a href = \"show.php?pk=$id\">Show</a>";
   echo "<tr>";
   echo "<td>$name</td>";
   echo "<td>$url</td>";
   echo "</tr>";
}

mysql_free_result($rs);
mysql_close_link($link);
?>
```

5.4. Варианты заданий

В каждом варианте заданий необходимо написать скрипт для создания таблиц и скрипт для заполнения этих таблиц данными. Важно также реализовать страницы, поддерживающие указанный контракт.

1. Необходимо создать три таблицы – автомобилей, фирм и стран. Таблица автомобилей имеет название, ссылки на фирму и страну. Требуется реализовать две страницы.

Первая отображает список автомобилей в виде таблицы из трех столбцов. В первом столбце дано название автомобиля, во втором – название фирмы со ссылкой на страницу просмотра о фирме. В третьем столбце – название страны. Список автомобилей должен быть упорядочен по названию автомобиля.

Вторая страница отображает детальные сведения о фирме по ее ключу, а именно: название фирмы и количество автомобилей данной фирмы

```
Firm detailed info:
1. Name - Mercedes.
2. Total number of autos - 502.
```

2. Необходимо создать три таблицы – автомобилей, фирм и стран. Таблица автомобилей имеет название, ссылки на фирму и страну. Требуется реализовать две страницы.

Первая страница отображает список автомобилей в виде таблицы из трех столбцов. В первом столбце дано название автомобиля, во втором – название фирмы. В третьем столбце – название страны со ссылкой на страницу просмотра о стране. Список автомобилей должен быть упорядочен по названию автомобиля.

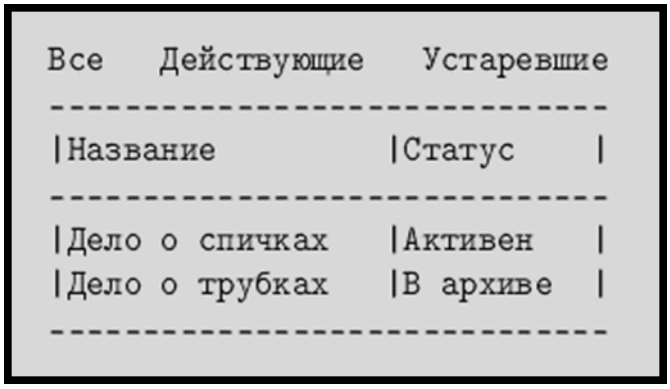
Вторая страница отображает детальные сведения о стране по ее ключу, а именно: название страны и количество автомобилей данной страны

```
Country detailed info:
1. Name - France.
2. Total number of autos - 75.
```

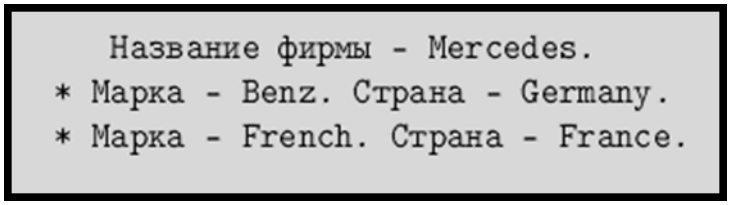
3. Необходимо создать одну таблицу – таблицу документов. У каждого документа есть название и статус. Статусы могут иметь следующие значения:

- **ACTIVE** – документ активно используется.
- **ARCHIVE** – документ находится в архиве.

Требуется реализовать страницу, на которой документы отображаются в виде таблицы из двух столбцов: ”Название“ и ”Статус“. Документы следует упорядочить по названию. Вверху страницы должны быть три ссылки, идущие в ряд. При нажатии на ссылку ”Все“ отображаются все документы независимо от их статуса. При нажатии на ссылку ”Действующие“ отображаются только документы с активным статусом, при нажатии на ссылку ”Устаревшие“ – только документы с архивным статусом



4. Необходимо создать три таблицы – автомобилей, фирм и стран. Таблица автомобилей имеет название, ссылки на фирму и страну. Требуется реализовать две страницы. На первой дан перечень фирм в виде маркированного списка, упорядоченного по названию фирм. Каждый элемент списка содержит название фирмы и переход на страницу детальных сведений о фирме, которые содержат в качестве заголовка название фирмы, а далее идет список всех автомобилей данной фирмы, упорядоченный по названию автомобилей:



5. Необходимо создать три таблицы – автомобилей, фирм и стран. Таблица автомобилей имеет название, ссылки на фирму и

страну. Следует реализовать две страницы. На первой дан перечень фирм в виде таблицы, упорядоченной по названию фирмы. В таблице два столбца, в первом – название фирмы, во втором – ссылка на страницу просмотра детальных сведений о фирме, которые содержат в качестве заголовка название фирмы, а далее идет список всех автомобилей данной фирмы, упорядоченный по названию стран:

```
   Название фирмы - Mercedes.
* Марка - Benz. Страна - Germany.
* Марка - French. Страна - France.
```

6. Необходимо создать одну таблицу – таблицу документов. У каждого документа есть название и статус. Статусы могут иметь следующие значения:

- **ACTIVE** – документ активно используется.
- **ARCHIVE** – документ находится в архиве.

Необходимо реализовать страницу, на которой документы отображаются в виде маркированного списка с названиями и статусами. Документы следует упорядочить по статусу. Вверху страницы должны быть три ссылки, идущие в ряд. При нажатии на ссылку ”Все“ отображаются все документы независимо от их статуса. При нажатии на ссылку ”Действующие“ должны быть отображены только документы с активным статусом, при нажатии на ссылку ”Устаревшие“ – только документы с архивным статусом

```
  Все   Действующие   Устаревшие
* Документ 'Дело о спичках'. Статус 'Активен'.
* Документ 'Дело о трубках'. Статус 'Активен'.
* Документ 'Дело о сигаретах'. Статус 'Архивный'.
```

7. Необходимо создать три таблицы – автомобилей, фирм и стран. Таблица автомобилей имеет название, ссылки на фирму и страну. Следует реализовать две страницы. На первой дан перечень стран в виде маркированного списка, упорядоченного по названию стран. Каждый элемент списка содержит название страны и переход на страницу детальных сведений о стране. Детальные сведения о стране содержат в качестве заголовка название страны, а далее идет список

всех автомобилей данной страны, упорядоченный по названию автомобилей:

```
   Название страны - Россия.
* Марка - ВАЗ-2110. Фирма - Волжский автомобильный завод.
* Марка - ГАЗ-53. Фирма - Горьковский автомобильный завод.
```

8. Необходимо создать три таблицы – автомобилей, фирм и стран. Таблица автомобилей имеет название, ссылки на фирму и страну. Следует реализовать две страницы. На первой дан перечень стран в виде таблицы, упорядоченной по названию страны. В таблице два столбца, в первом – название страны, во втором – ссылка на страницу просмотра детальных сведений о стране, которые содержат в качестве заголовка название страны, а далее идет список всех автомобилей данной страны, упорядоченный по названию фирм:

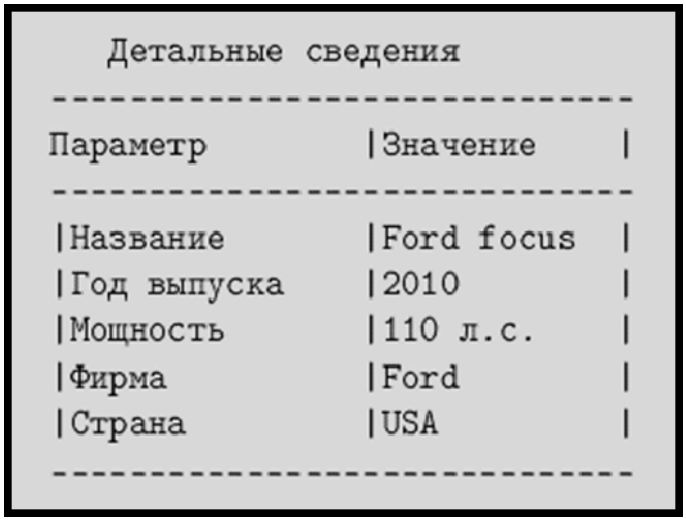
```
   Название страны - Россия.
* Марка - ВАЗ-2110. Фирма - Волжский автомобильный завод.
* Марка - ГАЗ-53. Фирма - Горьковский автомобильный завод.
```

9. Необходимо создать три таблицы – автомобилей, фирм и стран. Таблица автомобилей имеет название, год выпуска, мощность, ссылки на фирму и страну. Необходимо реализовать две страницы. На первой дан перечень автомобилей в виде таблицы, упорядоченной по названию автомобиля. В таблице два столбца, в первом – название автомобиля, во втором – ссылка на страницу просмотра детальных сведений об автомобиле, которые содержат следующий перечень в виде маркированного списка:

```
   Детальные сведения
* Название - Ford Focus.
* Год выпуска - 2010.
* Мощность двигателя - 110 л.с.
* Фирма - Ford.
* Страна - United States of America.
```

10. Необходимо создать три таблицы – автомобилей, фирм и стран. Таблица автомобилей имеет название, год выпуска, мощность,

ссылки на фирму и страну. Следует реализовать две страницы. На первой дан перечень автомобилей в виде маркированного списка, упорядоченный по названию. Каждый элемент списка содержит название автомобиля, а также ссылку на страницу просмотра детальных сведений об автомобиле, которые содержат следующий перечень в виде таблицы:



5.5. Содержание отчета

1. Цель работы.
2. Содержание варианта задания.
3. Скрипты для базы.
4. Код программы на PHP.
5. Скриншоты результатов выполнения страниц.
6. Выводы.

5.6. Контрольные вопросы

1. Каким образом в ссылке URL при формировании GET-запроса можно указать знак пробела?
2. Каким образом в ссылке URL при формировании GET-запроса можно указать символы в нестандартной кодировке?
3. Каковы ограничения на использование GET-запросов?

Лабораторная работа № 6

POST-ЗАПРОСЫ

6.1. Цель работы

Знакомство с взаимодействием между сервером и клиентом с помощью POST-запросов.

6.2. Формы HTML

В предыдущей лабораторной работе рассматривалось взаимодействие клиента с сервером посредством GET-запросов. Этот метод прост в использовании, однако имеет существенные недостатки:

1. Все содержимое пакета, передаваемого серверу, указывается явно в адресной строке. Это не всегда приемлемо. Например, пользователь вводит на странице пароль в поле, которое отображается звездочками (т. е. пароль не виден посторонним). Далее он нажимает на кнопку отправки, и содержимое отправляется в виде GET-запроса, отображая пароль в адресной строке в открытом виде.

2. Существует ограничение на максимальную длину такого типа запроса. С его помощью нельзя передавать данные большой размерности (например, файлы).

3. Возникают серьезные проблемы с кодировкой передаваемых данных, так как этот тип запроса поддерживает только кодировку **Latin1**. Строковые данные, которые нельзя передать в этой кодировке, приходится переделывать в HEX-формат, что приводит к существенному увеличению данных для передачи.

4. Та же проблема, что и в предыдущем пункте, возникает при передаче бинарных данных, содержащих элементы, которые нельзя закодировать в **Latin1**.

Эти проблемы решаются с помощью использования **POST**-запросов. Базовым для передачи таких запросов является понятие элементов *форм* внутри страниц HTML. Рассмотрим пример такой формы:

```
<form method = "POST" action = "upload.php">
<!-- Содержимое формы-->
</form>
```

То, что находится внутри формы, может быть отправлено путем нажатия на кнопку отправки данных, располагающуюся внутри этой же формы:

```
<input type = "submit" value = "Save">
```

Когда этот запрос приходит на сервер, Apache его разбирает и передает содержимое интерпретатору PHP, который сохраняет его в специальной глобальной переменной аналогично GET-запросу.

6.3. Пример выполнения задания

Рассмотрим следующую задачу:

Пример 3. *В системе имеется набор групп. Необходимо разработать страницы добавления и просмотра записей о студентах, принадлежащих к этим группам.*

Рассмотрим страницу добавления записей под названием **add.php**. Сценарий ее использования выглядит следующим образом:

- пользователь попадает на эту страницу со страницы со списком групп с помощью GET-запроса, в котором указан идентификатор группы под названием **groupid**;
- система извлекает из базы сведения о группе (название группы). Это название она отображает вверху страницы, а под ним отображает форму добавления. В форме располагаются два текстовых поля (имя и фамилия) и кнопка отправки данных;
- при нажатии на кнопку отправки данных пользователь переводится на эту же страницу с указанием того, что данные были отправлены. Система проверяет эти данные на валидность (т. е. что пользователь ввел непустые строки, и их содержимое не превышает по длине допустимый предел);

- если данные валидны, система сохраняет их в базе и получает первичный ключ. С помощью этого ключа она генерирует GET-запрос для перевода на страницу **show.php**, где отображаются детальные сведения о студенте;
- если данные не валидны, система снова отображает форму, указывая рядом с текстовыми полями сведения о допущенных ошибках. Пользователь должен ввести их повторно.

Здесь страница проверяет сразу два типа нарушений:

- нарушения *контракта*, как и для GET-запросов. Возникают в случае попыток взлома или ошибок в программном коде;
- нарушения *правил валидации*. Возникают в том случае, если пользователь ошибся при вводе данных. Тогда надо ввести эти данные повторно.

Рассмотрим код этой страницы. Предположим, что мы находимся на первой стадии, и пользователь только попал на страницу. Прежде всего нужно попробовать подключиться к базе, так как мы будем работать с ней в любом случае

```
<?php
$link = mysql_connect($server, $user, $pass)
   or die("Failed to connect to database. Error: " . mysql_error());
mysql_select_db($database)
   or die("Failed to select the database. Error: " . mysql_error());
?>
```

Теперь делаем следующее. Если пользователь попал на страницу в результате GET- или POST-запроса, то ему должен был быть передан ключ группы. Нужно проверить контракт, что такая группа действительно существует в базе.

```
<?php
if (!empty($_GET['groupid'])) {
   $groupid = $_GET['groupid'];
} else if (!empty($_POST['groupid'])) {
   $groupid = $_POST['groupid'];
} else {
   mysql_close($link);
   die("The parameter 'groupid' is not specified");
}
```

```
$query = "SELECT name FROM groups WHERE id = $groupid";
$rs = mysql_query($query)
   or die("Failed to select the group. Error: " . mysql_error());
while ($row = mysql_fetch_assoc($rs)) {
   $groupname = $row['name'];
}
mysql_free_result($rs);
if (empty($groupname)) {
   mysql_close($link);
   die("The parameter 'groupid' points to non-existing group");
}
?>
```

Итак, мы проверили контракт. Теперь нужно проверить условия валидации, если пользователь нажал на кнопку отправки, если все в порядке, сохранить данные в базе, после чего перевести пользователя на страницу просмотра:

```
<?php
$errors = array();
$success = true;
if (!empty($_POST['action'])) {
   $firstname = $_POST['firstname'];
   $lastname = $_POST['lastname'];

   if (empty($firstname)) {
      $success = false;
      $errors['firstname'] = "Empty field";
   }
   if (empty($lastname)) {
      $success = false;
      $errors['lastname'] = "Empty field";
   }

   if ($success) {
      $query = "INSERT INTO students(groupid, lastname, firstname)"
         . " VALUES ($groupid, '$lastname', '$firsname')";
      mysql_query($query)
         or die("Failed to add student. Error: " . mysql_error());
      $studentid = mysql_insert_id($link);
      mysql_close($link);
      header("Location: show.php?id=$studentid");
      die;
   }
}
?>
```

Теперь, выполнив все стадии, можно перейти к самой странице на HTML

```
<html>
   <head><title>Add student</title></head>
   <body>
   <h1>Add student to group: <?php echo $groupname;?></h1>
   <form method = "post" action = "add.php">
      <!-- Содержимое формы-->
   </form>
   </body>
</html>
```

В первую очередь в форме отображаются два текстовых поля

```
First name: <input type = "text" name = "firstname"
   value = "<?php echo $firstname;?>">  
<font color = "red"><?php echo $errors['firstname'];?></font><br/>
Last name: <input type = "text" name = "lastname"
   value = "<?php echo $lastname;?>">  
<font color = "red"><?php echo $errors['lastname'];?></font><br/>
```

Далее располагаем кнопку отправки и ссылку для возврата назад на страницу списка групп

```
<input type = "submit" value = "Add">  
<a href = "grouplist.php">Back</a>
```

Наконец, на форме располагаются два скрытых поля. В первом хранится идентификатор группы, чтобы передать его повторно при POST-запросе, во втором – указание странице на то, что данные были отправлены:

```
<input type = "hidden" name = "groupid" value = "<?php echo $groupid;?>">
<input type = "hidden" name = "action" value = "true">
```

Наконец, не забываем в конце выполнения скрипта закрыть явным образом соединение с базой

```
<?php
mysql_close($link);
?>
```

6.4. Варианты заданий

В каждом варианте заданий необходимо написать скрипт для создания таблиц и скрипт для заполнения этих таблиц данными. Следует реализовать страницы, поддерживающие указанный контракт.

1. Имеются две таблицы – фирмы и автомобили. У фирмы есть название, у автомобилей – ссылка на фирму, название, стоимость и мощность двигателя. Необходимо реализовать две страницы. Первая – страница добавления нового автомобиля. Она ожидает в качестве входного параметра ключ **firmid**, который на следующих шагах передается на страницу в виде POST-запроса. На странице в качестве заголовка должно быть написано название фирмы, а далее три текстовых поля для информации об автомобиле. После того как пользователь нажмет на кнопку ”Отправить“, производится валидация. Если она не проходит, пользователь остается на этой же странице. В противном случае его переводят на страницу просмотра сведений об автомобиле.

Страница сведений об автомобиле ожидает в качестве входного параметра ключ **autoid**, на основании которого она извлекает из базы и выводит исчерпывающие сведения об автомобиле в виде маркированного списка, включая название фирмы.

2. Имеются две таблицы – фирмы и автомобили. У фирмы есть название, у автомобилей – ссылка на фирму, название, стоимость и мощность двигателя. Необходимо реализовать две страницы. Первая – страница добавления нового автомобиля. На странице в форме должны идти три текстовых поля для информации об автомобиле, ниже – выпадающий список возможных вариантов фирм. После того как пользователь нажмет на кнопку ”Отправить“, производится валидация. Если она не проходит, пользователь остается на этой же странице. В противном случае его переводят на страницу просмотра сведений об автомобиле.

Страница сведений об автомобиле ожидает в качестве входного параметра ключ **autoid**, на основании которого она извлекает из базы и выводит исчерпывающие сведения об автомобиле в виде нумерованного списка, включая название фирмы.

3. Имеются две таблицы – фирмы и автомобили. У фирмы есть название, у автомобилей – ссылка на фирму, название, стоимость и мощность двигателя. Необходимо реализовать две страницы. Первая –

страница добавления нового автомобиля. На странице в форме должны идти три текстовых поля для информации об автомобиле, ниже – список возможных вариантов фирм в виде набора радио-кнопок. После того как пользователь нажмет на кнопку ”Отправить“, производится валидация. Если она не проходит, пользователь остается на этой же странице. В противном случае его переводят на страницу просмотра сведений об автомобиле.

Страница сведений об автомобиле ожидает в качестве входного параметра ключ **autoid**, на основании которого она извлекает из базы и выводит исчерпывающие сведения об автомобиле в виде таблицы, включая название фирмы.

4. Имеются две таблицы – фирмы и автомобили. У фирмы есть название, у автомобилей – ссылка на фирму, название, стоимость и мощность двигателя. Необходимо реализовать две страницы. Первая – страница добавления нового автомобиля. Она ожидает в качестве входного параметра ключ **firmid**, который на следующих шагах передается на страницу в виде POST-запроса. На странице в качестве заголовка должно быть написано название фирмы, а далее идут три текстовых поля для информации об автомобиле. После того, как пользователь нажмет на кнопку ”Отправить“, производится валидация. Если она не проходит, пользователь остается на этой же странице. В противном случае его переводят на страницу просмотра сведений о фирмах.

Страница сведений о фирмах содержит список фирм в виде таблицы из трех столбцов. В первом содержится название фирмы, во втором – список названий автомобилей для данной фирмы, в третьем – ссылка перехода на страницу добавления нового автомобиля для данной фирмы.

5. Имеются две таблицы – фирмы и автомобили. У фирмы есть название, у автомобилей – ссылка на фирму, название, стоимость и мощность двигателя. Необходимо реализовать две страницы. Первая – страница добавления нового автомобиля. На странице в форме должны идти три текстовых поля для информации об автомобиле, а ниже – список возможных вариантов фирм в виде набора радиокнопок. После того как пользователь нажмет на кнопку ”Отправить“, производится валидация. Если она не проходит, пользователь остается на этой же странице. В противном случае его переводят на страницу просмотра сведений об автомобиле.

Страница сведений об автомобиле ожидает в качестве входного параметра ключ **autoid**, на основании которого она извлекает из базы и выводит исчерпывающие сведения об автомобиле в виде таблицы, включая название фирмы. Внизу страницы имеется кнопка удаления. При нажатии на нее данный автомобиль удаляется из базы, а пользователя переводят на страницу добавления нового автомобиля.

6. Имеются три таблицы – фирмы, отели, страны. У фирм и стран есть названия, у отелей – название, ссылки на страну и фирму, а также стоимость пребывания. Необходимо реализовать две страницы. Первая – страница добавления нового отеля в фирму. В качестве входного параметра она ожидает ключ фирмы **firmid**. На странице в заголовке должно отображаться название этой фирмы. Далее идут два текстовых поля (название и стоимость отеля), а также выпадающий список с выбором стран. После того как пользователь нажмет на кнопку ”Отправить“, производится валидация. Если она не проходит, пользователь остается на этой же странице. В противном случае его переводят на страницу просмотра сведений об отеле.

Данная страница ожидает в качестве входного параметра ключ **hotelid**, на основании которого она извлекает из базы и выводит исчерпывающие сведения об отеле в виде таблицы, включая названия фирмы и страны. Внизу страницы должна быть кнопка удаления, при нажатии на которую данный отель удаляется из базы, а пользователя переводят на страницу добавления нового отеля для той же фирмы, к которой принадлежал старый отель.

7. Имеются три таблицы – фирмы, отели, страны. У фирм и стран есть названия, у отелей – название, ссылки на страну и фирму, а также стоимость пребывания. Необходимо реализовать две страницы. Первая – страница добавления нового отеля в страну. В качестве входного параметра она ожидает ключ страны **countryid**. На странице в заголовке отображается название этой страны. Далее идут два текстовых поля (название и стоимость отеля), а также выпадающий список с выбором фирм. После того как пользователь нажмет на кнопку ”Отправить“, производится валидация. Если она не проходит, пользователь остается на этой же странице. В противном случае его переводят на страницу просмотра сведений об отеле.

Страница сведений об отеле ожидает в качестве входного параметра ключ **hotelid,** на основании которого она извлекает из базы

и выводит исчерпывающие сведения об отеле в виде таблицы, включая названия фирмы и страны. Внизу страницы имеется кнопка удаления, при нажатии на которую данный отель удаляется из базы, а пользователя переводят на страницу добавления нового отеля для той же страны, к которой принадлежал старый отель.

8. Имеются три таблицы – фирмы, отели, страны. У фирм и стран есть названия, у отелей – название, ссылки на страну и фирму, а также стоимость пребывания. Необходимо реализовать две страницы. Первая – страница добавления нового отеля в фирму. В качестве входного параметра она ожидает ключ фирмы **firmid**. На странице в заголовке должно отображаться название этой фирмы. Далее идут два текстовых поля (название и стоимость отеля), а также выпадающий список с выбором стран. После того как пользователь нажмет на кнопку ”Отправить“, производится валидация. Если она не проходит, пользователь остается на этой же странице. В противном случае его переводят на страницу просмотра сведений о фирмах, содержащую данные в виде таблицы из трех столбцов. В первом столбце дано название фирмы, во втором перечислены названия отелей для данной фирмы, в третьем находится ссылка перехода на страницу добавления нового отеля для данной фирмы.

9. Имеются три таблицы – фирмы, отели, страны. У фирм и стран есть названия, у отелей – название, ссылки на страну и фирму, а также стоимость пребывания. Необходимо реализовать две страницы. Первая – страница добавления нового отеля в страну. В качестве входного параметра она ожидает ключ страны **countryid**. На странице в заголовке должно отображаться название этой страны. Далее идут два текстовых поля (название и стоимость отеля), а также выпадающий список с выбором фирм. После того как пользователь нажмет на кнопку ”Отправить“, производится валидация. Если она не проходит, пользователь остается на этой же странице. В противном случае его переводят на страницу просмотра сведений о странах.

Страница просмотра сведений о странах включает данные в виде таблицы из трех столбцов. В первом столбце содержится название страны, во втором перечислены названия отелей для данной страны, в третьем отображается ссылка перехода на страницу добавления нового отеля для данной страны.

10. Имеются три таблицы – фирмы, отели, страны. У фирм и стран есть названия, у отелей – название, ссылки на страну и фирму, а

также стоимость пребывания. Необходимо реализовать две страницы. Первая – страница добавления нового отеля в страну. В качестве входного параметра она ожидает ключ страны **countryid**. На странице в заголовке должно отображаться название этой страны. Далее идут два текстовых поля (название и стоимость отеля), а также выпадающий список с выбором фирм. После того как пользователь нажмет на кнопку ”Отправить“, производится валидация. Если она не проходит, пользователь остается на этой же странице. В противном случае его переводят на страницу просмотра сведений о странах, содержащую данные в виде таблицы из трех столбцов. В первом столбце дано название страны, во втором перечислены названия фирм, где в скобках после названия каждой фирмы указано количество отелей, которыми фирма располагает в данной стране. В третьем столбце отображается ссылка перехода на страницу добавления нового отеля для данной страны.

6.5. Содержание отчета

1. Цель работы.
2. Содержание варианта задания.
3. Скрипты для базы.
4. Код программы на PHP.
5. Скриншоты результатов выполнения страниц.
6. Выводы.

6.6. Контрольные вопросы

1. Как можно создать чекбоксы на HTML-странице внутри форм?
2. Как создать выпадающие списки на HTML-странице внутри форм?
3. Как создать элементы радио на HTML-странице внутри форм?
4. Может ли на странице быть сразу несколько форм?
5. Каково назначение скрытых полей внутри форм?
6. Когда согласно рекомендациям W3C следует использовать GET- и POST-запросы?
7. Можно ли отправить содержимое формы серверу посредством GET-запроса?

Лабораторная работа № 7

РАБОТА С СЕССИЯМИ

7.1. Цель работы

Знакомство с сессиями и реализация сложных сценариев сохранения информации при последовательных переходах пользователя со страницы на страницу.

7.2. Введение

Созданная в результате предыдущих лабораторных работ система позволяет пользователю осуществлять базовые сценарии взаимодействия с сущностями на стороне сервера – их просмотр и редактирование. Однако данная система пока не позволяет пользователю осуществлять авторизацию в системе, а также хранить какую-либо информацию, введенную пользователем однократно, на протяжении всей его работы с системой.

Рассмотрим по шагам простейший механизм авторизации, используемый в большинстве систем.

1. Пользователь заходит на одну из страниц сайта.
2. Система проверяет, был ли пользователь предварительно авторизован в системе. Если нет, пользователь перенаправляется на страницу авторизации.
3. На странице авторизации пользователю предлагается ввести свои логин и пароль. Система проверяет корректность введенных данных и, если все прошло успешно, авторизует пользователя в системе.
4. После этого пользователь может заходить на любую страницу в системе, при этом в правом верхнем углу отображается сообщение ”You are logged in as [name] / Logout“. Он может выйти из системы, нажав ссылку ’Logout’.

Такую систему невозможно реализовать только на основе GET/POST-запросов, потому что придется перекодировать все страницы так, чтобы ссылки и формы отправки содержали дополнительные параметры со служебной информацией о пользователе, что зачастую невозможно и прямо противоречит правилам безопасности. Необходимо, чтобы на время работы с клиентом сервер запоминал

некоторые требующиеся данные, которые удалялись бы из памяти сервера только по явному требованию клиента либо по истечении некоторого заданного администратором временного промежутка. Для этого и был разработан механизм *сессии*.

7.3. Работа с сессиями в PHP

Для того чтобы страница могла обращаться к данным текущей сессии, необходимо, чтобы в коде страницы в самом начале происходил вызов специальной функции

```
<?php
session_start();
// Дальнейший код страницы
?>
```

После вызова этой функции странице предоставляется доступ к хранилищу памяти для данной сессии. Хранилище доступно путем адресации к специальной глобальной переменной под названием **SESSION**. Как и переменные **GET** или **POST**, данная переменная представляет собой ассоциативный массив, где индексы – это названия, а значения – объекты. Вот пример кода, который выдает пользователю сведения о том, сколько раз он посетил конкретную страницу:

```
<?php
session_start();
// Если объекта с названием count
// в сессии нет - инициализируем его
if (!isset($_SESSION['count']) )
  $_SESSION['count'] = 0;
// Увеличиваем счетчик на 1
$_SESSION['count']++;
// Выводим пользователю сообщение
echo "You have visited this page "
  .$_SESSION['count']." times";
?>
```

Обратите внимание, что к этой глобальной переменной страница не имеет право обращаться до вызова функции **session_start**, иначе

интерпретатором PHP будет сгенерирована ошибка. Также стоит заметить, что в сессии можно сохранять не только переменные простых типов, но и сложные объекты с полями. Рассмотрим пример. Пусть на какой-то странице пользователь в форме заполняет сведения, которые надо хранить на протяжении сессии:

```
<form method = "POST" name = "store.php">
Фамилия: <input type = "text" name = "lastname" value = ""/><br/>
Имя: <input type = "text" name = "firstname" value = ""/><br/>
Возраст: <input type = "text" name = "age" value = ""/><br/>
<input type = "submit" value = "Send"/>
<input type = "hidden" name = "send" value = "true"/>
</form>
```

Вот как адресуемая страница **store.php** может сохранить данные в рамках сессии:

```
<?php
// Если в запросе содержится поле send
if (isset($_POST['send'])) {
  session_start();
  // Если в сессии был уже объект с таким
  // именем, то удаляем его
  unset($_SESSION['info']);
  // Создаем объект с тремя полями
  $user = new stdClass;
  $user->lastname = $_POST['lastname'];
  $user->firstname = $_POST['firstname';
  $user->age = $_POST['age'];
  $_SESSION['info'] = $user;
}
?>
```

А таким образом отображается информация о пользователе в рамках сессии на странице просмотра:

```
<?php
session_start();
// Если объект не задан, выдаем сообщение об ошибке
if (!isset($_SESSION['info']))
  echo "There is no user";
else {
  $user = $_SESSION['info'];
  echo "Lastname - {$user->lastname}. Firstname - {$user->firstname}. Age - {$user->age}.";
}
?>
```

7.4. Варианты заданий

В каждом варианте заданий необходимо написать скрипт для создания таблиц и скрипт для заполнения этих таблиц данными и реализовать страницы, поддерживающие указанный контракт.

1. Необходимо реализовать в базе две таблицы. Первая хранит в себе данные о пользователях – логин пользователя и пароль (для хранения пароля использовать MD5 без соли), вторая – сведения о страницах, посещенных данным пользователем: внешний ключ со ссылкой на пользователя, название страницы, которую он посетил, а также время посещения.

Необходимо реализовать три страницы. Первая содержит форму ввода логина и пароля, а также кнопку отправки. При попытке отправки данных пользователя переводят на эту же страницу, данные сопоставляются с базой, после чего следует начать сессию для этого пользователя (если данные верны).

Вторая страница проверяет содержимое сессии. Если пользователь еще не авторизовался, она сразу автоматически переводит его на страницу авторизации. Если пользователь авторизовался, то страница на основе данных сессии обращается к базе и обновляет сведения о том, сколько раз данный пользователь посещал эту страницу. После чего она выдает пользователю сообщение вида ”Вы посетили данную страницу *N* раз“.

Третья страница содержит статистические сведения в виде маркированного списка и сведения о последних 10 заходах на предыдущую страницу пользователей вообще. Каждый элемент маркированного списка отражает сведения вида ”Пользователь *username* заходил на страницу *pagename* в последний раз *date*“.

2. Необходимо реализовать в базе две таблицы. Первая хранит в себе данные о пользователях – логин пользователя и пароль (для хранения пароля использовать MD5 с солью длиной в 8 символов), вторая – сведения о страницах, посещенных данным пользователем: внешний ключ со ссылкой на пользователя, название страницы, которую он посетил, а также время посещения.

Необходимо реализовать три страницы. Первая содержит форму ввода логина и пароля, а также кнопку отправки. При попытке отправки данных пользователя переводят на эту же страницу, данные

сопоставляются с базой, после чего следует начать сессию для этого пользователя (если данные верны).

Вторая страница проверяет содержимое сессии. Если пользователь еще не авторизовался, она сразу автоматически переводит его на страницу авторизации. Если пользователь авторизовался, то страница на основе данных сессии обращается к базе и обновляет сведения о том, сколько раз данный пользователь посещал эту страницу. После чего она выдает пользователю сообщение вида ”Вы посетили данную страницу *N* раз“.

Третья страница содержит статистические сведения в виде маркированного списка и сведения о последних 20 заходах на предыдущую страницу пользователей вообще. Каждый элемент маркированного списка отображает сведения вида ”Пользователь *username* заходил на страницу *pagename* в последний раз *date*“.

3. Необходимо реализовать в базе две таблицы. Первая хранит в себе данные о пользователях – логин пользователя и пароль (для хранения пароля использовать SHA-1 без соли), вторая – сведения о страницах, посещенных данным пользователем: внешний ключ со ссылкой на пользователя, название страницы, которую он посетил, а также время посещения.

Необходимо реализовать три страницы. Первая содержит форму ввода логина и пароля, а также кнопку отправки. При попытке отправки данных пользователя переводят на эту же страницу, данные сопоставляются с базой, после чего необходимо начать сессию для этого пользователя (если данные верны).

Вторая страница проверяет содержимое сессии. Если пользователь еще не авторизовался, она сразу автоматически переводит его на страницу авторизации. Если пользователь авторизовался, то страница на основе данных сессии обращается к базе и обновляет сведения о том, сколько раз данный пользователь посещал эту страницу. После чего она выдает пользователю сообщение вида ”Вы посетили данную страницу *N* раз“.

Третья страница содержит статистические сведения в виде маркированного списка и сведения о последних 10 заходах на предыдущую страницу пользователей вообще. Каждый элемент маркированного списка содержит сведения вида ”Пользователь *username* заходил на страницу *pagename* в последний раз *date*“.

4. Необходимо реализовать в базе две таблицы. Первая хранит в себе данные о пользователях – логин пользователя и пароль (для хранения пароля использовать SHA-1 с солью длиной в 8 символов), вторая – сведения о страницах, посещенных данным пользователем: внешний ключ со ссылкой на пользователя, название страницы, которую он посетил, а также время посещения.

Необходимо реализовать три страницы. Первая содержит форму ввода логина и пароля, а также кнопку отправки. При попытке отправки данных пользователя переводят на эту же страницу, данные сопоставляются с базой, после чего следует начать сессию для этого пользователя (если данные верны).

Вторая страница проверяет содержимое сессии. Если пользователь еще не авторизовался, она сразу автоматически переводит его на страницу авторизации. Если пользователь авторизовался, то страница на основе данных сессии обращается к базе и обновляет сведения о том, сколько раз данный пользователь посещал эту страницу. После чего она выдает пользователю сообщение вида ”Вы посетили данную страницу *N* раз“.

Третья страница содержит статистические сведения в виде маркированного списка и сведения о последних 10 заходах на предыдущую страницу пользователей вообще. Каждый элемент маркированного списка отображает сведения вида ”Пользователь *username* заходил на страницу *pagename* в последний раз *date*“.

5. Необходимо реализовать в базе две таблицы. Первая хранит в себе данные о пользователях – логин пользователя и пароль (для хранения пароля использовать MD5 без соли), вторая – сведения о страницах, посещенных данным пользователем: внешний ключ со ссылкой на пользователя, название страницы, которую он посетил, а также время посещения.

Необходимо реализовать три страницы. Первая содержит форму ввода логина и пароля, а также кнопку отправки. При попытке отправки данных пользователя переводят на эту же страницу, данные сопоставляются с базой, после чего следует начать сессию для этого пользователя (если данные верны).

Вторая страница проверяет содержимое сессии. Если пользователь еще не авторизовался, она сразу автоматически переводит его на страницу авторизации. Если пользователь авторизовался, то страница

на основе данных сессии обращается к базе и обновляет сведения о том, сколько раз данный пользователь посещал эту страницу. После чего она выдает пользователю сообщение вида ”Вы посетили данную страницу *N* раз“.

Третья страница содержит статистические сведения в виде таблицы и сведения о последних 10 заходах на предыдущую страницу пользователей вообще. Таблица состоит из трех столбцов: логин пользователя, название страницы, на которую он заходил, и дата захода на страницу.

6. Необходимо реализовать в базе две таблицы. Первая хранит в себе данные о пользователях – логин пользователя и пароль (для хранения пароля использовать MD5 с солью длиной в 8 символов), вторая – сведения о страницах, посещенных данным пользователем: внешний ключ со ссылкой на пользователя, название страницы, которую он посетил, а также время посещения.

Необходимо реализовать три страницы. Первая содержит форму ввода логина и пароля, а также кнопку отправки. При попытке отправки данных пользователя переводят на эту же страницу, данные сопоставляются с базой, после чего следует начать сессию для этого пользователя (если данные верны).

Вторая страница проверяет содержимое сессии. Если пользователь еще не авторизовался, она сразу автоматически переводит его на страницу авторизации. Если пользователь авторизовался, то страница на основе данных сессии обращается к базе и обновляет сведения о том, сколько раз данный пользователь посещал эту страницу. После чего она выдает пользователю сообщение вида ”Вы посетили данную страницу *N* раз“.

Третья страница содержит статистические сведения в виде таблицы и сведения о последних 10 заходах на предыдущую страницу пользователей вообще. Таблица состоит из трех столбцов: логин пользователя, название страницы, на которую он заходил, и дата захода на страницу.

7. Необходимо реализовать в базе две таблицы. Первая хранит в себе данные о пользователях – логин пользователя и пароль (для хранения пароля использовать SHA-1 без соли), вторая – сведения о страницах, посещенных данным пользователем: внешний ключ со ссылкой на пользователя, название страницы, которую он посетил, а также время посещения.

Необходимо реализовать три страницы. Первая содержит форму ввода логина и пароля, а также кнопку отправки. При попытке отправки данных пользователя переводят на эту же страницу, данные сопоставляются с базой, после чего следует начать сессию для этого пользователя (если данные верны).

Вторая страница проверяет содержимое сессии. Если пользователь еще не авторизовался, она сразу автоматически переводит его на страницу авторизации. Если пользователь авторизовался, то страница на основе данных сессии обращается к базе и обновляет сведения о том, сколько раз данный пользователь посещал эту страницу. После чего она выдает пользователю сообщение вида ”Вы посетили данную страницу *N* раз“.

Третья страница содержит статистические сведения в виде таблицы и сведения о последних 10 заходах на предыдущую страницу пользователей вообще. Таблица состоит из трех столбцов: логин пользователя, название страницы, на которую он заходил, и дата захода на страницу.

8. Необходимо реализовать в базе две таблицы. Первая хранит в себе данные о пользователях – логин пользователя и пароль (для хранения пароля использовать SHA-1 с солью длиной в 8 символов), вторая – сведения о страницах, посещенных данным пользователем: внешний ключ со ссылкой на пользователя, название страницы, которую он посетил, а также время посещения.

Необходимо реализовать три страницы. Первая содержит форму ввода логина и пароля, а также кнопку отправки. При попытке отправки данных пользователя переводят на эту же страницу, данные сопоставляются с базой, после чего следует начать сессию для этого пользователя (если данные верны).

Вторая страница проверяет содержимое сессии. Если пользователь еще не авторизовался, она сразу автоматически переводит его на страницу авторизации. Если пользователь авторизовался, то страница на основе данных сессии обращается к базе и обновляет сведения о том, сколько раз данный пользователь посещал эту страницу. После чего она выдает пользователю сообщение вида ”Вы посетили данную страницу *N* раз“.

Третья страница отображает статистические сведения в виде таблицы и сведения о последних 10 заходах на предыдущую страницу пользователей вообще. Таблица состоит из трех столбцов: логин пользователя, название страницы, на которую он заходил, и дата захода на страницу.

9. Необходимо реализовать в базе две таблицы. Первая хранит в себе данные о пользователях – логин пользователя и пароль (для хранения пароля использовать MD5 без соли), вторая – сведения о страницах, посещенных данным пользователем: внешний ключ со ссылкой на пользователя, название страницы, которую он посетил, а также время посещения.

Необходимо реализовать три страницы. Первая содержит форму ввода логина и пароля, а также кнопку отправки. При попытке отправки данных пользователя переводят на эту же страницу, данные сопоставляются с базой, после чего необходимо начать сессию для этого пользователя (если данные верны).

Вторая страница проверяет содержимое сессии. Если пользователь еще не авторизовался, она сразу автоматически переводит его на страницу авторизации. Если пользователь авторизовался, то страница на основе данных сессии обращается к базе и обновляет сведения о том, сколько раз данный пользователь посещал эту страницу. После чего она выдает пользователю сообщение вида ”Вы посетили данную страницу *N* раз“.

Третья страница содержит статистические сведения в виде нумерованного списка и сведения о том, сколько раз каждый пользователь заходил на предыдущую страницу вообще. Каждый элемент списка отображает сведения вида ”Пользователь *username* заходил на страницу *pagename N* раз“.

10. Необходимо реализовать в базе две таблицы. Первая хранит в себе данные о пользователях – логин пользователя и пароль (для хранения пароля использовать MD5 с солью длиной в 8 символов), вторая – сведения о страницах, посещенных данным пользователем: внешний ключ со ссылкой на пользователя, название страницы, которую он посетил, а также время посещения.

Необходимо реализовать три страницы. Первая содержит форму ввода логина и пароля, а также кнопку отправки. При попытке отправки данных пользователя переводят на эту же страницу, данные сопоставляются с базой, после чего следует начать сессию для этого пользователя (если данные верны).

Вторая страница проверяет содержимое сессии. Если пользователь еще не авторизовался, она сразу автоматически переводит его на страницу авторизации. Если пользователь авторизовался, то страница на основе данных сессии обращается к базе и обновляет сведения о том, сколько

раз данный пользователь посещал эту страницу. После чего она выдает пользователю сообщение вида ”Вы посетили данную страницу *N* раз“.

Третья страница содержит статистические сведения в виде нумерованного списка и сведения о том, сколько раз каждый пользователь заходил на предыдущую страницу вообще. Каждый элемент списка отображает сведения вида ”Пользователь *username* заходил на страницу *pagename N* раз“.

7.5. Содержание отчета

1. Цель работы.
2. Содержание варианта задания.
3. Скрипты для базы.
4. Код программы на PHP.
5. Скриншоты результатов выполнения страниц.
6. Выводы.

7.6. Контрольные вопросы

1. Какие способы хранения пользовательских сессий существуют? Какой наиболее предпочтительный?
2. Каким образом сервер определяет на основе полученного запроса сессию пользователя, который прислал этот запрос?
3. Как можно принудительно завершить сессию (например, если пользователь решил нажать кнопку ’Выход’)?

Лабораторная работа № 8

УСТАНОВКА AMP И ВЫПОЛНЕНИЕ ПРОСТЕЙШИХ ПРИМЕРОВ

8.1. Цель работы

Установка и настройка компонентов платформы AMP и реализация базового примера на PHP для ознакомления с синтаксисом языка и особенностями библиотек.

8.2. Установка компонентов

Установите компоненты в следующей последовательности:

- MySQL 5.0 (стандартная конфигурация, прописывать путь к каталогу в системной переменной PATH не нужно);
- PHP 5.1 (стандартная конфигурация, прописать путь к каталогу в системной переменной PATH нужно обязательно);
- Apache 2.2.4 в стандартной конфигурации.

Далее откройте конфигурационный файл Apache httpd.conf и добавьте в него следующие строки для подключения модуля PHP:

```
#############
# Загружаем модуль PHP в составе модулей Apache
LoadModule php5_module "[Путь к каталогу PHP]\php5apache2.dll"
# Следующая строка уже есть, ее надо подправить, добавив нее index.php
DirectoryIndex index.php index.html
# Указываем Apache, что страницы с расширением .php нужно обрабатывать данным модулем
AddType application/x-httpd-php .php
# Указываем Apache путь к каталогу, где расположен файл php.ini
PHPIniDir "[Путь к каталогу PHP]"
#############
```

После этого протестируйте конфигурацию Apache и перезапустите его, чтобы убедиться, что все интегрировалось нормально.

8.3. Вывод информации о среде

Теперь в домашнем каталоге Apache создайте страницу index.php со следующим содержимым:

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

В браузере введите адрес http://localhost и убедитесь, что действительно отображается информация о системе PHP, а также сведения о системе AMP в целом. Убедитесь, что расширения для работы с MySQL действительно подключены и перечислены в составе компонентов, а также, что путь к конфигурационному файлу указан корректно.

Чтобы удостовериться, что с базой действительно удается установить соединение, создайте в домашнем каталоге Apache страницу dbtest.php со следующим содержимым:

```
<?php
$link = mysql_connect("localhost", "root", "[Пароль к базе]")
        or die("Could not connect : " . mysql_error());
echo "<b>Connected succesfully!</b>";
?>
```

В браузере введите адрес http://localhost/dbtest.php и убедитесь, что соединение с базой действительно устанавливается (т. е. выводится сообщение об успехе).

8.4. Базовые сведения о PHP

Детальные сведения о синтаксисе языка PHP можно посмотреть в прилагаемом к сведениям справочном руководстве, стоит отметить следующие особенности:

1. В целом синтаксис языка PHP соответствует синтаксису языка C. Однако имеется ряд отличий, самое существенное для начинающих разработчиков состоит в том, что все переменные начинаются со знака доллара. Хотя стандарт языка и допускает другие варианты наименования, однако это общепринятое соглашение, которого следует придерживаться.

2. PHP является интерпретируемой, а не компилируемой системой, т. е. среда работает со страницами PHP напрямую, разбирает их и выполняет требуемый код. Страницы не компилируются в машинные коды и не интегрируются в систему. Это упрощает добавление и редактирование страниц, однако приводит к падению производительности из-за необходимости интерпретации. Однако для современных аппаратных решений такое падение производительности не играет критической роли.

3. В PHP имеется встроенный сборщик мусора, поэтому разработчики могут не беспокоиться о высвобождении используемой их приложениями памяти, хотя рекомендуется явно избавляться от объектов, которые уже не нужны, с помощью специальных функций.

4. В PHP отсутствует явная типизация. Это порождает ряд проблем для разработчиков, поскольку их функции может вызвать любой код и передать им любые аргументы, поэтому при разработке критически важных функций в их коде обязательно проводятся дополнительные проверки для валидации аргументов.

8.5. Варианты заданий

Задания для данной лабораторной работы предполагают знакомство со следующими конструкциями языка PHP:

- циклы;
- операторы условий;
- средства вывода на экран.

В каждом задании необходимо создать страницу на PHP, которая генерирует HTML-страницу. Допустим смешанный стиль, т. е. заголовки HTML, не имеющие отношения к сути задания, можно выводить не с помощью средств PHP, а прописать их явно в виде HTML-фрагментов. Варианты заданий следующие.

1. Необходимо сгенерировать таблицу. Заголовок и строки генерируются с помощью PHP, а не прописываются вручную. Заголовок содержит 5 ячеек, *i*-я ячейка – название "Header i". Помимо заголовка в таблице 10 строк. Каждая четная строка выделяется цветом Aqua. В первом столбце каждой строки отображается номер ячейки, остальные ячейки строки хранят только знак решетки. В каждой третьей строке текст должен быть выделен курсивом.

2. Необходимо сгенерировать двухуровневый пронумерованный список. Первый уровень содержит 10 элементов. *i*-й элемент – это текст "Odd i" или "Even i" в зависимости от того, является ли номер элемента четным числом. Каждый элемент первого уровня содержит 5 дочерних элементов второго уровня. *i*-й дочерний элемет – это текст "Child i". Нечетные дочерние элементы нужно выделить жирным. Каждый третий элемент первого уровня должен быть красного цвета.

3. Необходимо сгенерировать маркированный список из первых 10 элементов последовательности Фибонначи. Каждый элемент с нечетным значением выделить курсивом, каждый третий элемент – цветом Aqua. *i*-й элемент должен содержать текст вида "F(i) = [Значение i-го элемента]".

4. Вывести список-"лесенку" из 10 элементов, где каждый следующий элемент располагается строкой ниже предыдущего и на 3 отступа правее предыдущего. Каждый нечетный элемент выделить жирным, каждый третий элемент – красным цветом. *i*-й элемент – это текст вида "Item i".

8.6. Содержание отчета

1. Цель работы.
2. Вариант задания.
3. Исходный текст программы на PHP.
4. Скриншот сгенерированной страницы.
5. Выводы.

8.7. Контрольные вопросы

1. Как сервер обрабатывает адрес, указанный ему в запросе? Организация иерархии страниц и каталогов.
2. Каковы назначение и способ задания виртуальных каталогов в Apache (модуль **mod_alias**)?
3. Организация сборщика мусора в PHP. Каково время существования переменных?
4. Какие проблемы порождают в PHP отсутствие явной типизации?

Лабораторная работа № 9

ВЗАИМОДЕЙСТВИЕ С БАЗОЙ

9.1. Цель работы

Создание структуры базы данных, заполнение таблиц данных и отображение данных из базы на страницах PHP.

9.2. Краткие сведения о базах данных

В ходе предыдущей лабораторной работы были созданы страницы, генерирующие содержимое динамически. Однако подлинно

динамическими они не являются, так как каждый раз генерируется одно и то же содержимое для страницы, и сколько к ней не обращайся и не обновляй, содержимое будет одно и то же.

Для того чтобы страница генерировала различное содержимое, важно, чтобы она его откуда-то извлекала. Чаще всего таким источником данных служат реляционные базы данных. Это такие базы, в которых данные представлены в виде таблиц, и таблицы могут ссылаться друг на друга. Такие базы удобнее всего использовать в приложениях, где предметная область представляется в виде диаграмм "сущность – связь". Есть три вида связей.

1. "Один-к-одному". Например, у каждого человека есть страховой полис. У человека не может быть двух или более полисов, а полис не может быть оформлен сразу на нескольких человек.
2. "Один-ко-многим". Например, существуют группы и студенты. В группе может быть много студентов, но студент не может числиться одновременно в нескольких группах.
3. "Многие-ко-многим". Наиболее редко используемый тип связи. Например, в фирме есть сотрудники и есть заказы. Над одним заказом может работать одновременно несколько сотрудников, и сотрудник может работать одновременно над множеством заказов.

Как правило, в каждой таблице существует столбец с первичным ключом – числовым значением, которое является уникальным для каждой строки данной таблицы. Ссылки на другие таблицы – внешние ключи – хранятся в других столбцах, которые содержат также числовые значения. Существуют определенные правила наименования: MySQL-столбец с первичным ключом должен называться "id", а внешний ключ на другую таблицу должен называться "[Название таблицы]_id". То есть если таблица customers ссылается на таблицу orders, то в таблице customers скорее всего будет столбец "orders_id".

9.3. Пример программы на PHP

Предположим, в базе есть таблица customers, в которой имеются поля name и address (название клиента и адрес соответственно). Тогда программа, отображающая эти данные в виде таблицы, могла бы выглядеть следующим образом:

```
<html>
  <head><title>Test DB</title></head>
  <body>
    <?php
      // Проверяем, есть ли соединение с базой, если нет, то прерываем выполнение
     $link = mysql_connect("localhost", "root", "[Пароль к базе]")
       or die ("Could not connect: " . mysql_error());
     mysql_select_db("[Название базы]") or die("Could not select database");
    ?>

    <table>
      <th>Name</th>
      <th>Address</th>
      <?php

        $query = "SELECT name, address FROM customers ORDER BY name";
        $rs = mysql_query($query) or die("Query failed: " . mysql_query());

        // Проходим по всем  строкам выборки
        while ($row = mysql_fetch_array($rs, MYSQL_ASSOC)) {
           $name = $row["name"];
           $address = $row["address"];
           echo "<tr><td>$name</td><td>$address</td></tr>";
        }
        // Очищаем память от выборки и закрываем соединения
        mysql_free_result($result);
        mysql_close($link);

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

9.4. Варианты заданий

В ходе каждого задания нужно будет выполнить следующее.

1. Написать скрипт создания необходимых таблиц и взаимосвязей. Рабочее название скрипта – schema.sql. Первичные ключи должны быть автоинкрементными, т. е. при вставке новой записи в базу первичный ключ не обязательно должен указываться.

2. Написать скрипт заполнения таблиц тестовыми данными. Рабочее название скрипта – data.sql. Скрипт строится на предположении, что схема уже создана. При заполнении данными можно указывать первичные ключи явно (чтобы правильно потом задать взаимосвязи с дочерними таблицами), однако схема должна поддерживать возможность добавления записей с автоматической генерацией ключа.

3. Написать скрипт очистки таблиц от текстовых данных. Рабочее название скрипта – flush.sql. Скрипт должен соблюдать порядок удаления данных, т. е. если таблица *A* ссылается на таблицу *B*, то сначала данные должны быть удалены из таблицы *A* и только затем из таблицы *B*, поскольку иначе может возникнуть нарушение целостности (т. е. этот запрос сгенерирует исключение).

4. Написать скрипт удаления созданных таблиц из базы. Рабочее название скрипта – drop.sql. Скрипт должен соблюдать порядок удаления сущностей, т. е. если таблица *A* ссылается на таблицу *B*, то сначала надо удалить таблицу *A* и только затем таблицу *B* (причины те же, что и для предыдущего пункта).

5. Написать страницу display.php, которая будет подключаться к базе (если доступа нет, то страница должна прекратить генерацию), извлекать из нее данные и отображать их в требуемом формате.

Варианты заданий

1) В базе должны быть производители и автомобили. У каждого производителя есть название, у каждого автомобиля – название, производитель и цвет (цвета должны содержаться в отдельной таблице и на них должна быть ссылка). Сведения нужно отображать в виде двухуровневого списка. На первом уровне – производители (отсортированные в алфавитном порядке), на втором – автомобили для данного производителя (также отсортированные в алфавитном порядке). На втором уровне данные нужно представлять в виде строки "[Название марки авто]. Цвет – [название цвета]".

2) В базе должны быть группы и студенты. У каждой группы есть название, у каждого студента – ФИО. Студент всегда принадлежит к одной группе, но не к нескольким сразу или вообще может быть вне группы (т. е. используется русский вариант организации, а не американский). Сведения нужно отображать в виде двухуровневого списка. На первом уровне – группы (отсортированные в алфавитном порядке), на втором – студенты (также отсортированные в алфавитном порядке). Если в группе меньше пяти человек (т. е. ее нужно расформировать), выделить ее название красным.

3) В базе должны быть фирмы и отели. У каждой фирмы есть название, у отеля – название, цена за день проживания и страна, в которой он располагается (страны перечислены в отдельной таблице).

Данные по отелям должны быть выведены в виде одноуровневого списка, отсортированы по названию страны, затем по стоимости проживания, а потом по названию отеля. Данные представить в виде строк "Страна – [название страны]. Стоимость в евро – [цена проживания]. Название отеля – [наименование]".

4) В базе должны быть улицы и адреса. У каждой улицы есть название, у каждого адреса – ссылка на улицу, номер дома и номер квартиры. Сведения нужно отображать в виде двухуровневого списка. На первом уровне идут улицы (отсортированные в алфавитном порядке). На втором – адреса для улицы (также отсортированные в алфавитном порядке). После перечисления адресов для улицы на первом уровне нужно вывести суммарные сведения – строку "Всего адресов: [количество адресов для данной группы]".

9.5. Содержание отчета

1. Цель работы.
2. Вариант задания.
3. Исходный текст скриптов для работы с базой данных.
4. Исходный текст программы на PHP.
5. Скриншот сгенерированной страницы.
6. Выводы.

9.6. Контрольные вопросы

1. В чем отличие DDL от DML?
2. Для чего необходимы первичные ключи? Из каких принципов они выбираются?
3. Для чего необходимы индексы в базе?
4. Назначение внешних ключей. Какие механизмы возможны для предотвращения нарушения целостности при попытке удаления родительской записи, на которую есть ссылки в дочерних таблицах?
5. Сигнатура функции **mysql_query**.
6. Сигнатура функции **mysql_fetch_array**.

Лабораторная работа № 10

ОБРАБОТКА ДАННЫХ ВВОДА ПОЛЬЗОВАТЕЛЯ СЕРВЕРОМ

10.1. Цель работы

Знакомство с GET- и POST-запросами от клиента к серверу и создание простейших сценариев взаимодействия.

10.2. Введение

В предыдущих лабораторных работах рассматривалось построение страниц, которые способны генерировать динамически содержимое, предоставляемое клиенту, в зависимости от содержимого базы данных. Однако при этом не рассматривался вопрос, откуда берется это содержимое – в рамках второй лабораторной работы оно было создано посредством скрипта. Очевидно, что реальные сайты работают по иной схеме:

- существуют страницы *просмотра*, на которых пользователь может просмотреть какие-либо сведения, которые извлекаются из базы и выдаются сервером в удобном для пользователя виде;
- существуют страницы *добавления/редактирования* информации, на которых пользователь может добавить новые данные либо модифицировать уже имеющиеся.

Очевидно, что оба типа страниц работают с содержимым базы, первый тип осуществляет запросы на выборку данных из базы. Второй – осуществляет запросы на добавление/редактирование/удаление записей в базе. Кроме того, страницы могут быть смешанного типа, т. е. отображать какие-то сведения и предоставлять функционал по добавлению новых сведений. Вот наиболее распространенные примеры таких страниц:

- форум. Пользователь после авторизации может зайти в любую ветвь, просмотреть ее содержимое и добавить новое сообщение в конце ветви;
- блог. Пользователь может добавлять новые статьи в собственный блог и просматривать остальные блоги.

В данной лабораторной работе рассматривается инструментарий для взаимодействия клиента с сервером, даются ответы на вопросы:

- как разместить в HTML-странице элементы, которые позволили бы пользователю вводить данные и отправлять их серверу?
- Как сервер обрабатывает данные от пользователя?

Существуют два основных способа передачи данных от клиента к серверу – это GET- и POST-запросы (есть и другие варианты, но они здесь не рассматриваются).

10.3. GET-запросы

В предыдущих лабораторных работах упоминалось, что браузер клиента общается с сервером, формируя пакет с запросом, который отправляется серверу. Сервер, получив такой пакет, отправляет назад пакет с ответом. В пакете запроса содержится многое, в первую очередь адрес страницы на сервере, к которой нужно обратиться. Помимо этого пакет может содержать дополнительные данные. Рассмотрим это на примере простой ссылки:

```
<a href = "remove.php?article_id=96&confirmed=1">Удалить статью</a>
```

Что происходит, когда пользователь нажимает на эту ссылку? Браузер отправляет содержимое ссылки (значение атрибута **href**) в виде GET-запроса на сервер. В качестве адреса сервер использует то, что идет в ссылке до знака вопроса, т. е. адрес **remove.php** (учтите, что этот адрес вычисляется относительно местонахождения страницы, на которой расположена ссылка). То что идет после знака вопроса – это параметры GET-запроса, которые представляются в формате **ключ=значение**. Такие пары разделяются знаком амперсенда. То есть в данном случае в пакете будет передано помимо самого адреса еще два параметра – **article_id** (со значением 96) и **confirmed** (со значением 1).

Следует учитывать, что эти параметры передаются в виде строковых значений. Apache, получая такой пакет, вызывает интерпретатор PHP для страницы **remove.php** и передает ему параметры GET-запроса. Интерпретатор может обратиться к ним через глобальную переменную-массив

```
<?php
$article = $_GET['article_id'];
$confirmed = $_GET['confirmed'];
// Произвести далее операцию удаления
?>
```

Недостатки такого способа передачи следующие:

- дополнительные параметры указываются явно в ссылке, что зачастую неправильно (приходится отображать пользователю служебные данные);
- проблемы с кодировкой – символы, которые не являются символами английского алфавита, приходится кодировать с помощью шестнадцатеричной нотации, которой предшествует знак процента. Например, знак пробела (код 32) кодируется как "%20". Это приводит к тому, что адресная строка может быть очень большой. Кроме того, часть символов (которая не может быть закодирована в ISO Latin 1) нельзя передать таким запросом;
- невозможность передавать в адресной строке содержимое файлов;
- многие серверы содержат ограничения по длине принимаемой адресной строки и слишком длинную адресную строку обрабатывать просто не будут.

10.4. POST-запросы

Для решения подобного рода проблем были созданы POST-запросы. В них содержимое параметров хранится в пакете запроса, но не в адресной строке, а в отдельном поле сообщения, которое может быть потенциально очень большим, и оно не отображается при запросе в адресной строке. Для того чтобы отправить такое содержимое, нужно разместить необходимые элементы (с помощью которых пользователь вводит данные) в *форму*. Форма задается с помощью следующих тэгов:

```
<form method = "POST" action = "admin/test.php">
<!-- Содержимое формы -->
</form>
```

Теперь при отправке данных пользователем через эту форму будет создан POST-запрос с адресом **admin/test.php**, где в теле сообщения (но не в адресной строке) будет передано содержимое формы. А содержимое может включать в себя следующие поля (рассмотрены наиболее распространенные).

10.4.1. Флаги

Чаще всего на формах в виде опций предоставляются флаги – галочки, которые пользователь может пометить. Чтобы ее создать, нужно использовать следующий тэг:

```
<input name = "[Имя флага]" type = "checkbox" value = "[Значение флага]"/>
```

Если пользователь выбрал флаг, то при отправке формы в тело сообщения пакета будет добавлена пара **Имя флага=Значение флага**. Если пользователь не выбрал флаг, то параметр с его именем просто не будет добавлен. По умолчанию флаг отображается как не выбранный пользователем.

10.4.2. Переключатели

Переключатели **radio** предлагают пользователю ряд вариантов, причем выбрать нужно один из них:

```
<input name = "[Имя переключателя]" type = "radio" value = "[Значение]"/>
```

При выборе пользователем одного из *группы* переключателей (группа – это все переключатели с одинаковым именем) и отправке данных через форму в тело сообщения пакета будет добавлена пара **Имя переключателя=Значение выбранного переключателя**.

10.4.3. Кнопка сброса формы

При нажатии на кнопку сброса все элементы формы будут установлены в то состояние, которое было задано в атрибутах по умолчанию, причем отправка формы не производится

```
<input type = "reset"  value = "Очистить форму"/>
```

10.4.4. Выпадающий список

Тэг **select** позволяет отображать на форме выпадающие списки, из которых пользователь может выбрать одно или несколько значений

```
<select name = "[Имя списка]">
<option value = "[Значение первого элемента]">Первый элемент</option>
<option value = "[Значение второго элемента]">Второй элемент</option>
</select>
```

При отправке пользователем данных через форму в случае, когда в списке что-то выбрано, в тело сообщения будет добавлена пара **Имя списка=Выбранное значение**. Естественно, содержимое списка можно генерировать динамически с помощью PHP, как и весь остальной код HTML.

10.4.5. Текстовое поле

Данные поля позволяют пользователю вводить различную текстовую информацию:

```
<input type = "text" name = "[Имя поля]" value = "Текст по умолчанию"/>
```

При вводе пользователем значения в тело сообщения будет добавлена пара **Имя поля=Введенный текст**. При этом производится шифрование в многобайтовой кодировке, поэтому таким способом можно передать любые символы.

10.4.6. Поле ввода пароля

Практически идентично текстовому полю, за тем исключением, что вводимый пользователем текст не отображается в поле (точнее, отображается в виде зведочек)

```
<input type = "password" name = "[Имя поля]"/>
```

10.4.7. Скрытое текстовое поле

Помимо всего прочего в форме могут располагаться скрытые поля, которые не отображаются пользователю, но при этом все равно передаются в случае отправки данных формы

```
<input type = "hidden" name = "[Название поля]" value = "[Значение поля]"/>
```

10.4.8. Кнопка отправки формы

Отправить данные в форме можно посредством специального элемента – кнопки отправки:

```
<input type = "submit" name = "[Название кнопки]" value = "[Текст кнопки]"/>
```

При нажатии на эту кнопку все данные формы кодируются в виде пакета и отправляются серверу. Если у кнопки есть имя, то в тело сообщения будет добавлена пара **Название кнопки=Текст кнопки**. Это необходимо, когда в форме несколько кнопок отправки, чтобы сервер смог определить, что именно ему отправляют.

10.5. Примеры обработки POST-запросов

Пример 1. Первая страница содержит в себе два текстовых поля, которые будет отображать вторая страница:

```
<form action = "action.php" name = "myform" method = "post">
  <input type = "text" name = "mytext"/><br/>
  <textarea name = "msg" cols = "20" rows = "10"/>
  <input type = "submit" value = "Отправить данные"/>
</form>
```

Вот как выглядит код страницы **action.php**:

```
<?php
$text = $_POST['mytext'];
$msg = $_POST['msg'];
echo "$text<br/>$msg";
?>
```

Пример 2. На первой странице генерируется динамически список выбора по годам:

```
<form action="action.php" name = "myform" method = "post">
  <select name = "year">
<?php
for ($i = 2000; $i <= 2015; $i++)
  echo "<option value = $i>$i</option>\n";
?>
  </select>
  <input type = "submit" value = "Отправить данные"/>
</form>
```

Вот как выглядит код страницы **action.php**:

```
<?php
$year = $_POST['year'];
echo "Selected value is: $year";
?>
```

10.6. Варианты заданий

Задания для данной лабораторной работы являются логическим развитием предыдущей лабораторной. Страницы добавления/редактирования сущностей должны позволять редактировать все поля записи (за исключением первичного ключа). Ссылки (внешние ключи) представляются с помощью выпадающих списков, все остальное – с помощью текстовых полей. Страница редактирования должна хранить в себе первичный ключ сущности (в виде скрытого поля). Страницы добавления/редактирования должны производить валидацию и в случае ошибки оставлять пользователя на той же странице. Каждый табличный список содержит в качестве первого столбца "№ п/п".

Варианты заданий следующие.

1. Страница просмотра списка фирм отображает в виде таблицы со столбцами "Название", "Просмотр". В первом – название фирмы. Во втором – ссылка, которая ведет на страницу просмотра сведений о фирме (описана детально ниже). Внизу страницы просмотра списка фирм имеется кнопка "Добавить", при нажатии на которую пользователь переходит на страницу добавления сведений о фирмах.

Страница просмотра отдельной фирмы отображает следующие сведения: название фирмы, маркированный список автомобилей, где каждый элемент списка – это ссылка с названием автомобиля. Нажав на эту ссылку, пользователь попадает на страницу просмотра сведений об автомобиле. Внизу страницы есть кнопки "Редактировать" и "Удалить" (эта кнопка отключена, если у фирмы есть автомобили).

При нажатии на кнопку "Редактировать" пользователь попадает на страницу редактирования сведений о фирме, которая описана ниже, при нажатии на кнопку "Удалить" – на страницу удаления записи о фирме, описанной ниже. Страница просмотра ожидает в качестве параметра GET-запроса параметр **id**, который хранит идентификатор фирмы. Если идентификатор не задан или такой фирмы в базе нет, то страница должна остановить выполнение и выдать сообщение об ошибке. Внизу страницы имеется ссылка "Назад", ведущая на страницу просмотра списка фирм.

Страница просмотра сведений об автомобиле отображает следующие сведения: название фирмы, название автомобиля и цвет. Никаких опций добавления/редактирования/удаления не нужно. Внизу

страницы имеется ссылка "Назад к фирме", которая будет переводить пользователя назад на страницу просмотра сведений о фирме-производителе данного автомобиля. Страница просмотра сведений ожидает в качестве параметра GET-запроса параметр **id**, который хранит идентификатор автомобиля. Если идентификатор не задан или такого автомобиля в базе нет, то страница должна остановить выполнение и выдать сообщение об ошибке.

Страница добавления новой фирмы содержит заголовок "Добавление новой фирмы", одно текстовое поле, в которое пользователь введет название. Внизу страницы имеются две кнопки "Добавить" и "Отмена", нажатие на них возвращает назад на эту же страницу, но обрабатывать эти события страница должна по-разному. Если нажата кнопка "Отмена", то происходит принудительный возврат пользователя на общую страницу просмотра списка фирм. Если нажата кнопка "Добавить", то производится валидация. Если она пройдена успешно, то фирма добавляется в базу и осуществляется принудительный переход на страницу просмотра сведений для этой фирмы. Валидация не проходит в следующих случаях: название фирмы пустое, название фирмы превышает допустимую длину текстового поля в базе, фирма с таким названием уже есть. Если валидация не проходит, то пользователь остается на странице добавления и под текстовым полем ему выводится красным цветом сообщение об ошибке – что именно неправильно.

Страница редактирования сведений о фирме выглядит практически так же, как предыдущая. Отличия следующие: заголовок "Редактирование сведений о фирме [Название фирмы]", текстовое поле будет изначально не пустым (в нем название фирмы, извлеченное из базы). При нажатии на кнопку "Отмена" пользователь возвращается на страницу просмотра сведений о фирме. Вместо кнопки "Добавить" будет кнопка "Сохранить". Страница ожидает в качестве параметра POST-запроса параметр **id**, который хранит идентификатор фирмы. Если идентификатор не задан или такой фирмы в базе нет, то страница останавливает выполнение и выдает сообщение об ошибке. Этот параметр должен сохраняться в форме в виде скрытого поля, чтобы при отправке данных из формы это поле снова присутствовало в POST-запросе.

Страница удаления фирмы ожидает в качестве параметра POST-запроса параметр **id**, который хранит идентификатор фирмы. Если идентификатор не задан или такой фирмы в базе нет или у этой фирмы есть автомобили, то страница останавливает выполнение и выдает сообщение об ошибке. В противном случае страница удаляет записи из базы и выдает сообщение "Фирма [Название фирмы] удалена успешно". Внизу страницы имеется ссылка "Назад", ведущая на страницу просмотра списка фирм.

2. Страница просмотра списка групп должна быть представлена в табличной форме со столбцами "Название", "Просмотр", "Удалить". Название будет красного цвета, если в группе меньше пяти студентов. В столбце "Просмотр" находится ссылка, нажав на которую пользователь попадает на страницу просмотра детальных сведений о группе, которая описана ниже. В столбце "Удалить" должны быть флаги, которые пользователь может пометить (т. е. удалить за один раз можно несколько групп). Внизу страницы кнопка "Удалить" переводит на страницу удаления групп. Флаги должны быть только для тех групп, у которых нет студентов, т. е. для группы со студентами в столбце "Удалить" не должно быть ничего.

Страница просмотра сведений о группе содержит следующие сведения: название группы, затем в виде маркированного списка идет список студентов, принадлежащих к данной группе, отсортированный в алфавитном порядке. Внизу страницы имеются кнопка "Редактировать", ведущая на страницу редактирования сведений о группе, а также ссылка "Назад", ведущая к странице просмотра списка групп. Страница ожидает в качестве параметра GET-запроса параметр **id**, который хранит идентификатор группы. Если идентификатор не задан или такой группы в базе нет, то страница должна остановить свое выполнение и выдать сообщение об ошибке.

Страница редактирования сведений о группе выглядит следующим образом: заголовок "Редактирование сведений о группе [Название группы]", далее текстовое поле, в котором изначально хранится название группы, извлеченное из базы. Внизу страницы две кнопки "Сохранить" и "Отмена", нажатие на которые ведет назад на эту же страницу, но обрабатывать эти события страница должна по-разному. Если нажата кнопка "Отмена", то осуществляется принудительный возврат пользователя на страницу просмотра сведений о группе. Если нажата

кнопка "Сохранить", то производится валидация. Если валидация пройдена успешно, запись обновляется в базе, и происходит принудительный переход на страницу просмотра сведений для этой группы.

Валидация не проходит в следующих случаях: название группы пустое, название группы превышает допустимую длину текстового поля в базе, есть другая группа с таким же названием. Если валидация не проходит, то пользователь остается на странице редактирования, и под текстовым полем ему выводится красным цветом сообщение об ошибке – что именно неправильно. Страница ожидает в качестве параметра POST-запроса параметр **id**, который хранит идентификатор группы. Если идентификатор не задан или такой группы в базе нет, то страница останавливает выполнение и выдает сообщение об ошибке. Этот параметр должен сохраняться в форме в виде скрытого поля, чтобы при отправке данных из формы это поле снова присутствовало в POST-запросе.

Страница удаления групп ожидает в качестве параметра POST-запроса параметр **id**, который хранит массив идентификаторов групп. Для каждого идентификатора в массиве проводится проверка: если в группе нет студентов, то группа удаляется из базы и выводится новая строка с сообщением "Группа [Название группы] успешно удалена". Если в группе есть студенты (недопустимая ситуация, но все же стоит проверить), то страница должна остановить выполнение и выдать сообщение об ошибке. Внизу страницы должна быть ссылка "Назад", ведущая на страницу просмотра списка групп.

3. Страница просмотра стран строится в табличной форме со столбцами "Название", "Просмотр", "Удалить". Название выделяется курсивом, если у страны нет отелей. В столбце "Просмотр" имеется ссылка, нажав на нее пользователь попадает на страницу просмотра детальных сведений о стране, которая описана ниже. В столбце "Удалить" должны быть флаги, которые пользователь может пометить (т. е. удалить за один раз можно несколько стран). Внизу страницы расположена кнопка "Удалить", которая переводит на страницу удаления выбранных стран. Флаги отмечают только те страны, у которых нет отелей, т. е. для страны с отелями в столбце "Удалить" не должно быть ничего.

Страница удаления стран ожидает в качестве параметра POST-запроса параметр **id**, который хранит массив идентификаторов стран.

Для каждого идентификатора в массиве проводится проверка: если в стране нет отелей, то она удаляется из базы и выводится новая строка с сообщением "Страна [Название страны] успешно удалена". Если в стране есть отели (недопустимая ситуация, но все же стоит проверить), то страница должна остановить выполнение и выдать сообщение об ошибке. Внизу страницы должна быть ссылка "Назад", ведущая на страницу просмотра списка стран.

Страница просмотра сведений о стране выглядит следующим образом: сначала название страны, затем в виде маркированного списка названия отелей, отсортированные в алфавитном порядке. Каждое название – это ссылка, нажав на которую пользователь попадает на страницу просмотра детальных сведений об отеле. Внизу страницы расположены кнопка "Добавить новый отель", нажав на которую пользователь попадает на страницу добавления нового отеля для данной страны, а также ссылка "Назад", ведущая на страницу просмотра списка стран. Страница ожидает в качестве параметра GET-запроса параметр **id**, который хранит идентификатор страны. Если идентификатор не задан или такой страны в базе нет, то страница должна остановить выполнение и выдать сообщение об ошибке.

Страница просмотра сведений об отдельном отеле выглядит следующим образом: сначала название отеля, затем название страны, далее название фирмы-владельца и стоимость проживания в день. Внизу страницы должна быть ссылка "Назад к стране", ведушая на страницу просмотра сведений о стране, к которой принадлежит отель. Страница ожидает в качестве параметра GET-запроса параметр **id**, который хранит идентификатор отеля. Если идентификатор не задан или такого отеля в базе нет, то страница должна остановить выполнение и выдать сообщение об ошибке.

Страница добавления сведений об отеле выглядит следующим образом: заголовок "Добавление отеля для страны [Название страны]", далее текстовое поле с названием отеля, текстовое поле со стоимостью и выпадающий список фирм. Внизу страницы есть две кнопки "Добавить" и "Отмена", при нажатии на которые пользователь попадает на эту же страницу, но она обрабатывает эти два события поразному. При нажатии на кнопку "Отмена" пользователь возвращается на страницу просмотра сведений о стране. При нажатии на кнопку "Добавить" производится валидация, если она прошла успешно, то

запись добавляется в базу и происходит принудительный переход на страницу просмотра сведений о новом отеле.

Валидация может не пройти в следующих случаях: название отеля пустое, название отеля слишком длинное, отель с таким названием уже есть. Если валидация не пройдена, то пользователь остается на этой же странице, и под текстовым полем ему выводится сообщение об ошибке – что именно неправильно. Страница ожидает в качестве параметра POST-запроса параметр **id**, который хранит идентификатор страны. Если идентификатор не задан или такой страны в базе нет, то страница должна остановить выполнение и выдать сообщение об ошибке. Этот параметр должен сохраняться в форме в виде скрытого поля, чтобы при отправке данных из формы это поле снова присутствовало в POST-запросе.

4. Страница просмотра списка улиц в виде таблицы со столбцами "Название", "Просмотр". При нажатии на ссылку "Просмотр" пользователь попадает на страницу просмотра сведений об улице. Внизу после таблицы есть кнопка "Добавить", нажав на которую пользователь попадает на страницу добавления новой улицы.

Страница добавления новой улицы содержит одно текстовое поле, в которое пользователь вводит название новой улицы. Внизу кнопка "Добавить" и ссылка "Назад", которая возвращает пользователя на страницу просмотра списка улиц. При нажатии на кнопку "Добавить" пользователь оказывается на этой же странице и производится валидация. Если валидация прошла успешно, то улица добавляется в базу, и осуществляется принудительный переход на страницу просмотра сведений о новой улице. Валидация не проходит в следующих случаях: название пустое, название слишком длинное, улица с таким названием уже есть. Если валидация не проходит, то пользователь остается на странице добавления, и под текстовым полем ему выводится красным цветом сообщение об ошибке – что именно неправильно.

Страница просмотра сведений об улице выглядит следующим образом: сначала название улицы, затем в виде маркированного списка дано перечисление адресов на этой улице. Внизу страницы после списка расположена кнопка "Добавить адрес", нажав на которую пользователь переходит на страницу добавления адреса. Также имеется ссылка "Назад", которая возвращает пользователя на страницу просмотра списка улиц. Страница ожидает в качестве

параметра GET-запроса параметр **id**, который хранит идентификатор улицы. Если идентификатор не задан или такой улицы в базе нет, то страница должна остановить выполнение и выдать сообщение об ошибке.

Страница добавления сведений об адресе выглядит следующим образом: сначала заголовок "Добавление адреса для улицы [Название улицы]", затем идут два текстовых поля, в которые пользователь вводит номера дома и квартиры. Внизу есть кнопка "Добавить". Нажав на нее, пользователь попадает на эту же страницу и производится валидация. Если валидация прошла успешно, то запись об адресе добавляется в базу, и осуществляется принудительный переход на страницу просмотра сведений об улице. Валидация не проходит в следующих случаях: номера пустые, номера слишком длинные, на этой улице уже есть такой адрес. Если валидация не проходит, то пользователь остается на странице добавления, и под текстовым полем ему выводится красным цветом сообщение об ошибке – что именно неправильно. Страница ожидает в качестве параметра POST-запроса параметр **id**, который хранит идентификатор улицы. Если идентификатор не задан или такой улицы в базе нет, то страница должна остановить выполнение и выдать сообщение об ошибке. Этот параметр сохраняется в форме в виде скрытого поля, чтобы при отправке данных из формы это поле снова присутствовало в POST-запросе.

10.7. Содержание отчета

1. Цель работы.
2. Вариант задания.
3. Исходный текст программы на PHP.
4. Скриншоты сгенерированных страниц.
5. Выводы.

10.8. Контрольные вопросы

1. В каком случае следует использовать GET-, а в каком случае POST-запросы согласно стандарту? Когда можно сделать исключение?
2. Может ли форма отправлять данные в GET-запросе?

3. Каким образом элемент флага сделать видимым на странице, но недоступным для модификации? Как выполнить то же самое для текстовых полей и переключателей?
4. Какая процедура позволяет, чтобы флаг по умолчанию был помечен на странице с самого начала при ее загрузке?
5. Перечислите порядок действий, чтобы при загрузке страницы в выпадающем списке сразу был выбран один из элементов.
6. Какая процедура обеспечивает в выпадающем списке выбор нескольких элементов?
7. Как задать размер текстового поля и указать допустимую длину вводимого текста?
8. Как задать на странице многострочное текстовое поле? Какие атрибуты возможны у данного тэга?
9. Если на странице две формы, то попадают ли данные из одной формы в пакет при отправке данных из другой?
10. Предположим, что мы хотим выбрать нескольких пользователей и удалить их нажатием одной кнопки. Для этого у каждого пользователя есть флаг на странице, который можно пометить. При нажатии на кнопку отправки все флаги сохраняются в сообщении POST-запроса. Как следует назвать эти флаги, чтобы они правильно передались в POST-запрос в виде массива элементов с одинаковым названием?
11. Каким образом производить принудительный переход на страницу с помощью функции **header**? Какие ограничения есть на применение этой функции?

Лабораторная работа № 11

ОРГАНИЗАЦИЯ РАБОТЫ С УЧЕТНЫМИ ЗАПИСЯМИ

11.1. Цель работы

Знакомство с организацией работы в системе с учетными записями, хранение сведений о пользователе, функции хэширования.

11.2. Хранение паролей в базе

Из предыдущего материала становится ясно, как организовать работу с учетными записями в системе.

1. Если пользователь хочет зарегистрироваться в системе, он вводит свои данные в специальной форме и отправляет серверу.
2. Сервер проверяет корректность этих данных (т. е. что пользователя с таким логином еще нет в системе, что пароль задан и удовлетворяет стандартным требованиям безопасности и т. д.) и сохраняет эти данные в базе.
3. После (возможной) специальной процедуры подтверждения по почте (которая здесь детально рассматриваться не будет) пользователь может авторизоваться в системе, отправив специальной формой серверу свои логин и пароль.
4. Система проверит, что в базе действительно есть пользователь с таким паролем и извлечет из базы дополнительные данные о нем (ФИО, например), после чего поместит их в сессию под специальным служебным названием.
5. Все страницы, на которые заходит пользователь, проверяют наличие в сессии объекта с таким названием и, если он не задан, перенаправляют пользователя на страницу авторизации.

Возникает вопрос, как хранить сведения о пользователе в базе? Естественно, они хранятся в таблице. Особый интерес представляет столбец, в котором хранится пароль. На первый взгляд надо просто хранить пароль в базе так, как ввел его пользователь – в виде текста. Тогда серверу при авторизации пользователя достаточно будет сравнить присланный пароль с тем, что хранится в базе. Однако это

неверный подход и крайне опасный, так как если кто-то получит несанкционированный доступ хотя бы на чтение к этой таблице базы, то он сможет подделываться под любого пользователя и совершать любые действия от его имени. Не будем вдаваться в подробности криптографии, однако стоит заметить, что согласно международным стандартам для серьезных систем категорически запрещается хранить пароли в *открытом* виде, т. е. в том виде, в каком их ввел пользователь. Они должны быть зашифрованы, причем так, чтобы по этому шифру, даже зная способ шифрования, злоумышленник не смог восстановить исходный пароль (точнее говоря, не смог это сделать за приемлемое время).

Итак, предположим, что данные шифруются с помощью некоторого алгоритма A. Когда пользователь регистрируется, система шифрует его пароль с помощью алгоритма A и этот шифр сохраняет в базе. Когда пользователь авторизуется, система применяет алгоритм A к паролю, присланному пользователем, и сравнивает полученный шифр с тем, что есть в базе. Если шифры совпадают, то пароль верен.

Одним из наиболее распространенных способов шифрования является алгоритм MD5. Для произвольной текстовой строки s он вычисляет *хэш* $hash(s)$ – 128-битное число. Таким образом, такое шифрование является *необратимым,* так как строк бесконечное множество, а количество хэшей ограничено диапазоном от 0 до $2^{128} - 1$. Отсюда несколько важных следствий:

1. злоумышленнику необязательно знать пароль пользователя, достаточно знать любой пароль, хэш которого совпадает с хэшем пароля пользователя, тогда он сможет авторизоваться (такая ситуация в хэшировании называется *коллизией*). Однако особенность алгоритма MD5 в том, что вычислить хэш для строки гораздо быстрее, чем подобрать строку, дающую такой же хэш;

2. даже администратор системы не имеет доступа к настоящим паролям пользователей. Поэтому, если пользователь забыл пароль, то администратор может сгенерировать ему новый, но не может прислать старый пароль. Из-за этого во всех современных системах авторизации при регистрации просят указывать почтовый адрес, чтобы на него отсылать новый сгенерированный пароль (так как вероятность

того, что злоумышленник одновременно взломает почтовую службу и систему, крайне мала).

Функции хэширования реализованы в большинстве языков программирования в виде библиотек. Кстати, они есть и в MySQL, поэтому можно записать пароль в базу с помощью такого запроса:

```
INSERT INTO users(login, password) values('ivanov', md5('12345'));
```

На уровне PHP эта функция называется так же:

```
<?php
$pass = $_POST['pass'];
$hash = md5($pass);
// Сравниваем хэш с извлеченным из базы.
?>
```

11.3. Варианты заданий

Задания для данной лабораторной работы являются логическим развитием предыдущей лабораторной. Во всех вариантах необходимо добавить в базу таблицу **users**, в которой будут следующие поля: фамилия, имя, логин и пароль пользователя. Необходимо на уровне SQL-скрипта прописать в базе одну запись пользователя, чтобы была возможность авторизоваться. Следует также иметь страницу авторизации, куда вводят логин пользователя и пароль. Кнопка отправки переводит пользователя на эту же страницу. Если логин и пароль подходят, то пользователь авторизуется в системе и переводится на основную страницу варианта. Все страницы в варианте должны проверять, авторизован ли пользователь, если нет, то перенаправлять его на страницу авторизации.

Если же пользователь авторизован, то вверху каждой страницы в правом углу должно быть сообщение "Вы вошли как [Фамилия Имя]/Выйти". Ссылка "Выйти" ведет на страницу авторизации и передает дополнительный параметр. Система проверяет наличие этого параметра в GET-запросе и при положительном ответе удаляет данные о пользователе из сессии с помощью функции **unset**.

Также необходимо разработать аналог корзины заказов пользователя, когда он переходит со страницы на страницу и выбирает для различных объектов "Добавить в список". В конце работы он может

перейти на страницу списка и оформить общий заказ. Здесь надо сделать аналогичный функционал. На главной странице каждого варианта будет ссылка на страницу просмотра списка. Варианты следующие.

1. На странице просмотра отдельной фирмы в отображении списка автомобилей рядом с каждым автомобилем имеется ссылка "Добавить в заказ". Она ведет на эту же страницу (т. е. в ней надо передавать идентификатор фирмы) с параметрами указания того, что этот автомобиль должен быть добавлен в заказ. Если такие параметры есть, то система помещает идентификатор в поле сессии (поскольку автомобилей может быть добавлено несколько, то это поле должно быть множеством). Когда генерируется список автомобилей для фирмы, ссылки не генерируются для тех автомобилей, которые уже были добавлены в заказ.

На странице просмотра заказа отображается список выбранных автомобилей в виде таблицы со столбцами "Фирма", "Марка", отсортированный по названиям фирмы и автомобилей. Внизу страницы есть ссылка, ведущая назад на главную страницу. Помимо этого имеется кнопка "Очистить заказ", ведущая на эту же страницу. При ее нажатии система должна удалить все заказы из памяти (не из базы, естественно).

2. На странице просмотра отдельной группы в отображении списка студентов рядом с каждым студентом помещается ссылка "Добавить в ведомость". Она ведет на эту же страницу (т. е. в ней надо передавать идентификатор группы) с параметрами указания, что этот студент должен быть добавлен в ведомость. Если такие параметры есть, то система помещает идентификатор в поле сессии (поскольку студентов может быть добавлено несколько, то это поле должно быть множеством). Когда генерируется список студентов для группы, ссылки не генерируются для тех студентов, которые уже были добавлены в ведомость.

На странице просмотра ведомости отображается список выбранных студентов в виде таблицы со столбцами "Группа", "ФИО", отсортированный по группам и студентам. Внизу страницы расположены ссылка, ведущая назад на главную страницу, и кнопка "Очистить ведомость", ведущая на эту же страницу. При ее нажатии система удаляет всех студентов из памяти (не из базы, естественно).

3. На странице просмотра отдельной страны в отображении списка отелей рядом с каждым отелем находится ссылка "Добавить в тур", которая ведет на эту же страницу (т. е. в ней надо передавать идентификатор страны) с параметрами указания, что этот отель должен быть добавлен в тур. Если такие параметры есть, то система помещает идентификатор в поле сессии (поскольку отелей может быть добавлено несколько, то это поле должно быть множеством). Когда генерируется список отелей для страны, ссылки не должны генерироваться для тех отелей, которые уже были добавлены в тур.

На странице просмотра тура отображается список выбранных отелей в виде таблицы со столбцами "Страна", "Отель", отсортированный по странам и отелям. Внизу страницы помещаются ссылка, ведущая назад на главную страницу, и кнопка "Очистить тур", ведущая на эту же страницу. При ее нажатии система удаляет все отели из памяти (не из базы, естественно).

4. На странице просмотра отдельной улицы в отображении списка адресов рядом с каждым адресом имеется ссылка "Добавить в отчет", которая ведет на эту же страницу (т. е. в ней надо передавать идентификатор улицы) с параметрами указания, что этот адрес должен быть добавлен в отчет. Если такие параметры есть, то система помещает идентификатор в поле сессии (поскольку адресов может быть добавлено несколько, то это поле должно быть множеством). Когда генерируется список адресов для улицы, ссылки не генерируются для тех адресов, которые уже были добавлены в отчет.

На странице просмотра отчета отображается список выбранных адресов в виде таблицы со столбцами "Улица", "Адрес", отсортированный по улицам и адресам. Внизу страницы расположены ссылка, ведущая назад на главную страницу, и кнопка "Очистить отчет", ведущая на эту же страницу. При ее нажатии система удаляет все адреса из памяти (не из базы, естественно).

11.4. Содержание отчета

1. Цель работы.
2. Вариант задания.
3. Исходный текст программы на PHP.
4. Скриншоты сгенерированных страниц.
5. Выводы.

11.5. Контрольные вопросы

1. Какая сигнатура функции session_start и что она возвращает?
2. Какое назначение функции md5 и что она возвращает?
3. Дайте определение функции isset и что она возвращает. В чем ее отличие от функции empty?
4. Какая сигнатура функции unset и что она возвращает?
5. Какая сигнатура функции **usort** и что она возвращает?

Лабораторная работа № 12

ЗНАКОМСТВО С МОДУЛЬНОСТЬЮ

12.1. Цель работы

Знакомство с функциями в PHP и способами разбиения программ на модули.

12.2. Введение

В ходе предыдущих лабораторных работ была создана система, которая позволяет выполнять базовые операции, доступные в большинстве систем такого рода – авторизацию и CRUD-операции над сущностями. Однако с точки зрения разработчика (а не конечного пользователя) данная система имеет ряд недостатков.

1. Код большинства страниц содержит дублирующие фрагменты, которые повторяют друг друга. Это в первую очередь проверка авторизации. Также совпадают фрагменты подключения к базе. Если появится необходимость изменить проверку авторизации или подключиться к базе на основе не MySQL, а PostgreSQL, то изменения придется вносить во все страницы. Недостаток данного кода в наличии *антипаттерна* "copy-and-paste".

2. Параметры подключения к базе прописаны явно в коде всех страниц. Обычно системы организуются так, чтобы в них было явное разделение на конфигурационные и выполняемые файлы. Тогда администратор в случае необходимости меняет только параметры кон-

фигурации. Выполняемые файлы должны изменяться в исключительных случаях.

3. Антипаттерн ухудшает код страницы "spaghetti code", так как в нем смешаны несколько слоев системы. Необходимо произвести их разделение.

Однако с точки зрения пользователя данная система тоже имеет существенные недостатки: она спроектирована исходя из предположения, что объем данных в базе будет очень небольшим. Данное предположение не высказывалось нигде явно в предыдущих лабораторных работах, однако это действительно так. Рассмотрим главные страницы системы. Там в виде таблицы отображается набор некоторых сущностей, с которыми можно производить различные операции. Когда этих сущностей станет хотя бы 100, работа с главной страницей крайне осложнится. Даже ее генерация будет занимать существенное время. А попытки просмотра всей страницы и кнопок внизу под таблицей будут крайне трудоемкими. Пользователь нуждается в поддержке двух операций:

- *фильтрация.* Необходимо разместить в начале страницы форму с текстовым полем и кнопкой "Поиск". Если пользователь что-то введет в поле и нажмет кнопку поиска, то в таблице должны отображаться данные только о тех сущностях, названия которых содержат в себе введенный пользователем фрагмент;
- *постраничное разделение*. На странице должны отображаться не все сущности, а лишь их порция, например 10 элементов. Внизу под *таблицей* дано меню из ссылок "Назад", "Вперед", "К первой странице". При нажатии на одну из этих ссылок должен осуществляться перевод на требуемую часть данных (т. е. переход производится на эту же страницу, но ей указывается новый диапазон).

12.3. Создание функций

Как и большинство современных императивных языков, PHP поддерживает создание функций. Синтаксис создания выглядит следующим образом:

```
function <Имя функции>(<Список аргументов>) {
  <Тело функции>
}
```

Естественно, аргументов функции может и не быть. Что касается переменных, объявленных в теле функции, их область видимости ограничивается этой функцией, и после ее выполнениия эти аргументы будут удалены из памяти сборщиком мусора. PHP придерживается специального стиля наименования функций, отличного от C++ или Java. Соглашения о наименованиях следующие:

- имена содержат только символы английского алфавита в нижнем регистре, цифры и знаки подчеркивания;
- имена должны начинаться с букв;
- имена не должны заканчиваться знаками подчеркивания.

Поскольку PHP интерпретирует страницы, то в коде функции можно использовать вызовы функций, определенных в коде далее:

```
<?php
function first($str) {
  if (second($str)) {
    echo "Correct";
  }
}

function second($str) {
  if (isset($str)) return TRUE;
  else return FALSE;
}
?>
```

Обратите внимание, что функция необязательно должна возвращать какое-то значение. Этот подход отличается от Pascal, где функция обязательно должна возвращать какое-либо значение, в противном случае ее необходимо оформлять как процедуру. Еще несколько важных замечаний по поводу функций.

1. В определении функции отсутствуют типы аргументов. Это позволяет при вызове функции передавать ей любые аргументы. Главное, чтобы количество аргументов совпало с тем, что указано в списке аргументов в определении функции. Если функция используется только самим разработчиком внутри собственного кода, то валидацию данных можно не производить (она гарантируется вызывающим функцию кодом). Но если функция входит в состав библиотеки, доступной для сторонних разработчиков, то внутри нее обязательно должна производиться валидация аргументов.

2. Тип самой функции также не задается явно, поэтому она может возвращать значения *различных* типов в зависимости от сценария выполнения. Например, функция **mysql_query** может возвращать **FALSE** в случае ошибки либо извлеченную из базы выборку данных.

12.4. Создание модулей

PHP позволяет подключать в страницы дополнительные файлы, а эти файлы могут, в свою очередь, подключать дополнительные файлы (будем называть далее эти файлы *модулями*). Рассмотрим, как можно выделить в системе конфигурационный файл с параметрами подключения к базе, а также библиотеку для подключения к базе. Пусть в корне системы имеется файл **config.php**:

```
<?php
// Определим глобальную переменную $CFG,
// в которой будут храниться параметры
// подключения к базе. Если переменная уже
// задана, удалим ее и создадим заново
if (isset($CFG)) unset($CFG);

$CFG = new stdClass;
$CFG->host = 'localhost';
$CFG->user = 'root';
$CFG->password = '12345';
$CFG->database = 'mydb';
?>
```

Итак, этот файл содержит в себе описание переменной, поля которой хранят настройки подключения к базе. В случае необходимости администратор должен будет изменить настройки только в этом файле, не меняя весь остальной код системы. Теперь предположим, что в системе имеется каталог **lib**, в котором располагаются необходимые библиотеки. Пусть библиотека, работающая с базой, называется **datalib.php**, и ее код выглядит следующим образом:

```
<?php
require_once('../config.php');
```

```
function init_connection() {
  global $CFG;
  mysql_connect($CFG->host, $CFG->user, $CFG->password)
    or die("Failed to connect: ".mysql_error());
  mysql_select_db($CFG->database)
    or die("Failed to choose db: ".mysql_error());
}
?>
```

В начале страницы производится вызов функции **require_once**, которая подключает конфигурационный файл. Поскольку он находится на уровень выше, то используется относительная ссылка с помощью многоточия, чтобы интерпретатор смог правильно определить его положение. Далее идет определение функции.

В определении функции первой строкой идет указание, что переменная **CFG** является *глобальной*, т. е. она определена вне кода функции. В противном случае при попытке работы с этой переменной интерпретатор будет пытаться отыскать ее в области локальных переменных функции, а локальной переменной с таким названием нет. Теперь предположим, что в корне системы есть файл **index.php**, которому необходимо работать с базой. Тогда он может использовать для подключения код из библиотеки:

```
<?php
require_once('lib/datalib.php');

init_connection();
// Дальнейший код по работе с базой
?>
```

Обратите внимание, что здесь не подключается явно модуль **config.php**, поскольку на самой странице переменная **CFG** не используется явно. Кроме того, исчезли явные вызовы функций MySQL.

12.5. Разделение системы на слои

Вооружившись данным инструментарием, попробуем провести *рефакторинг* имеющейся системы, которая извлекает данные из таблицы и отображает их пользователю. Согласно формальному опреде-

лению, рефакторинг – это такое изменение кода системы, которое не приводит к изменению поведения системы с точки зрения пользователей (т. е. пользователи не должны заметить, что система переписана заново). Это понятие было введено Мартином Фаулером.

Итак, предположим, что у нас имеется страница **index.php**, которая отображает пользователю данные о студентах в виде таблицы из столбцов "Фамилия", "Имя", "Отчество". Эти данные извлекаются из базы из таблицы **students** в виде столбцов **lastname**, **firstname** и **surname**. При этом страница поддерживает возможность фильтрации, т. е. в текстовом поле можно ввести значение, нажать на кнопку "Поиск", и система выведет только те записи, где фамилия (либо имя, либо отчество) содержит это значение в качестве подстроки. Исходный код этой страницы не приводится (однако из предыдущих лабораторных работ его можно представить – это так называемый спагетти-код, где перемешаны вывод страницы, обращения к базе и обработка введенных пользователем значений). Будем считать, что у нас есть каталог **lib**, в котором имеется библиотека **datalib.php**, описанная ранее. Нам нужна функция, которая будет принимать на вход один аргумент – текстовое значение. Если текстовое значение пустое, то функция выбирает из базы все записи, отсортировав их по ФИО. Если непустое, то выбираются только те записи, которые удовлетворяют критерию фильтрации. Возвращаться должен список объектов, а не элементов ассоциативного массива. Каждый объект должен иметь три поля – **lastname**, **firstname** и **surname**.

12.5.1. Первый вариант функции

Попробуем реализовать функцию впрямую и разместим ее в **datalib.php**:

```
function get_students_listing($filter) {
  if (!isset($filter) || strlen($filter) == 0) {
    $query = "SELECT * FROM students ORDER BY lastname, firstname, surname";
  } else { // Задан фильтр
    $query = "SELECT * FROM students WHERE lastname LIKE '%$filter%'"
  ." OR firstname LIKE '%$filter%' OR surname LIKE '%$filter%'"
  ." ORDER BY lastname, firstname, surname";
  }
```

```
 $rs = mysql_query($query) or die("Query failed: ".mysql_error());
 $result = array();
 while ($row = mysql_fetch_array($rs, MYSQL_ASSOC)) {
  $item = new stdClass;
$item->lastname = $row["lastname"];
$item->firstname = $row["firstname"];
$item->surname = $row["surname"];
$result []= $item;
 }
 return $result;
}
```

12.5.2. Второй вариант функции

Предыдущий вариант действительно работает, однако поскольку таких функций выборки в системе чаще всего много, полезно выделить некоторые общие фрагменты, которые можно генерировать автоматически.

1. Фрагменты запроса на нечеткое сопоставление с помощью оператора **LIKE**. В разных базах синтаксис этого оператора может отличаться. Чтобы не пришлось при портировании системы на новую базу переписывать весь код слоя **datalib.php**, нужно выделить генерацию этих частей запроса в отдельный фрагмент.

2. Запросы состоят из нескольких обязательных частей (выборка и сортировка), а также необязательных частей (условие фильтрации). Было бы оптимально не повторять общие обязательные части для всех сценариев.

Нам понадобится функция, которая бы объединила несколько подстрок в одну строку, включив между отдельными подстроками символы-разделители (в нашем случае пробелы). Для этого заведем дополнительную библиотеку **textlib.php**

```
<?php

function concat_strings($arr, $separator) {
 $result = '';
 $first = TRUE;
 foreach ($arr as $item) {
  if (!first) $result .= $separator;
```

```
 $first = FALSE;
 $result .= $item;
  }
  return $result;
}
?>
```

Заодно заведем дополнительную функцию в самой библиотеке **datalib.php**, которая будет генерировать **LIKE** запрос для требуемого столбца

```
function get_like_part($column, $value) {
  return "$column LIKE '%$value%'";
}
```

Тогда новый вариант требуемой функции будет выглядеть следующим образом:

```
require_once('textlib.php');

function get_students_listing($filter) {
  $arr = array();
  $arr []= "SELECT * FROM students";
  if (isset($filter) && strlen($filter) > 0) {
    $arr []= "WHERE ".get_like_part('lastname', $filter);
 $arr []= "OR ".get_like_part('firstname', $filter);
 $arr []= "OR ".get_like_part('surname', $filter);
  }
  $arr []= "ORDER BY lastname, firstname, surname";
  $query = concat_strings($arr, " ");
  $rs = mysql_query($query) or die("Query failed: ".mysql_error());
  $result = array();
  while ($row = mysql_fetch_array($rs, MYSQL_ASSOC)) {
    $item = new stdClass;
 $item->lastname = $row["lastname"];
 $item->firstname = $row["firstname"];
 $item->surname = $row["surname"];
 $result []= $item;
  }
  return $result;
}
```

Конечно, можно на этом не останавливаться и совершенствовать функции дальше, однако здесь необходимо соблюдать баланс между требованиями проекта и перфекционизмом разработчика (это одна из основных задач руководителя программного проекта). Посмотрим, как будет выглядеть код **index.php**:

```
<?php
require_once('lib/datalib.php');
init_connection();
$filter = '';
if (isset($_POST['filter'])) $filter = $_POST['filter'];
$students = get_students_listing($filter);
?>
<html>
<head>
<title>Список студентов</title>
</head>
<body>
<form method = "post" action = "index.php">
Фильтр: <input type = "text" name = "filter" value = "<?php echo $filter;?>">
<input type = "submit" value = "Поиск">
</form>
<br/>
<table>
<th>Фамилия</th><th>Имя</th><th>Отчество</th>
<?php
foreach ($students as $student) {
  echo "<tr>";
  echo "<td>{$student->lastname}</td>";
  echo "<td>{$student->firstname}</td>";
  echo "<td>{$student->surname}</td>";
  echo "</tr>\n";
}
?>
</table>
</body>
</html>
```

Обратите внимание, что в коде основной страницы нет никаких обращений к базе напрямую и обработки специальных случаев, когда пользователь ввел значение и нажал на кнопку (небольшое упражнение: попробуйте представить, как выглядел бы код этой же страницы

в случае без библиотек). Слой страницы обращается к промежуточному слою, но напрямую с базой не работает. Обычно в системе таких слоев несколько, и существует следующее правило: каждый слой может взаимодействовать напрямую только с предыдущим и следующим слоями и ни с какими другими.

12.6. Варианты заданий

Все варианты из предыдущих лабораторных работ должны подвергнуться рефакторингу. Во-первых, необходимо завести библиотеку **securitylib.php**, которая будет содержать следующие функции:

- **require_login()** – функция будет использоваться на всех страницах, кроме самой страницы авторизации. Она должна вызываться в самом начале и, если пользователь не авторизован, переводить его на страницу авторизации;
- **login(username, password)** – функция будет авторизовывать пользователя, когда он указал свои данные. Функция возвращает **TRUE**, если пользователь указал корректные данные, и **FALSE** в противном случае;
- **logout()** – функция должна завершать сессию пользователя (при нажатии на ссылку "Выйти").

Во-вторых, требуется завести конфигурационный файл, в котором хранить настройки подключения к базе. Также нужно завести библиотеку работы с данными, в которую поместить функции подключения к базе и работы с сущностями.

В-третьих, основная страница должна подвергнуться переработке, чтобы поддерживать функции фильтрации и сортировки, т. е. в форме над таблицей должны быть поле "Фильтр" и кнопка "Поиск". Если пользователь нажал на кнопку "Поиск", то значение фильтра сохранится в сессии. Внизу под таблицей находятся ссылки "Назад" и "Вперед". Если это первая страница, то ссылки "Назад" не должно быть. Если это последняя страница, то ссылки "Вперед" не должно быть. При нажатии на одну из этих ссылок производится переход на эту же страницу с указанием нового диапазона (точнее, сколько записей в начале выборки должно быть пропущено). Если пользователь жмет на кнопку "Поиск" (т. е. меняется сама выборка), то нужно осуществлять переход на первую страницу. Следует отображать порции данных по 10 записей за один раз.

С точки зрения разработки это значит, что на уровне библиотеки работы с данными нужно реализовать функцию следующей сигнатуры:

```
<?php
/**
 * Данная функция считывает данные из основной таблицы.
 * Она возвращает их в виде списка объектов с полями,
 * как в примере.
 * @param $skip Численное значение. Указывает, сколько
 * записей с начала выборки нужно пропустить. Если
 * речь идет о первой странице, то это значение 0.
 * @param $filter Значение фильтра, если оно пустое, то
 * генерируется обычный запрос, как в предыдущих лабораторных.
 * В противном случае генерируется запрос с использованием LIKE.
 **/
function get_main_listing($skip, $filter) {
 // ..
}
?>
```

На самой странице не должно быть никаких SQL-запросов и работы с базой напрямую. Все осуществляется путем взаимодействия через промежуточный слой библиотеки. Аналогично все страницы варианта также должны измениться и использовать промежуточные функции. Для каждого варианта функции следующие:

```
//***************Вариант 1**/
/**
 * Данная функция по идентификатору
 * фирмы извлекает из базы сведения
 * о фирме. А именно, возвращается
 * объект с полями id, name, count,
 * где count – кол-во автомобилей для
 * данной фирмы.
 **/
function get_firm($id);

/**
 * Данная функция по идентификатору
 * фирмы производит ее удаление из базы.
 * Если у фирмы есть автомобили, то
 * функция должна прервать выполнение
```

```
 * с сообщением об ошибке.
 **/
function delete_firm($id);

/**
 * Данная функция по идентификатору фирмы
 * возвращает список автомобилей для
 * данной фирмы. Каждый элемент списка
 * содержит поля id и name автомобиля.
 **/
function get_auto_list($id);

/**
 * Данная функция по идентификатору
 * автомобиля должна вернуть детальные
 * сведения о нем. А именно, объект
 * со следующими полями: id, name,
 * firmname, colorname, firmid.
 **/
function get_auto($id);

/**
 * Данная функция производит
 * валидацию предлагаемого имени
 * фирмы для добавления. Если валидация
 * прошла успешно, то возвращается TRUE,
 * в противном случае возвращается строка
 * с описанием ошибки.
 **/
function validate($name);

/**
 * Данная функция добавляет в базу
 * фирму с указанным именем. Возвращает
 * первичный ключ новой записи.
 **/
function add($name);

/**
 * Данная функция обновляет в базе
 * запись о фирме. Входной аргумент содержит
 * два поля: id и name.
 **/
function update($firm);
```

```
//***************Вариант 2**/
/**
 * Данная функция по идентификатору
 * группы извлекает из базы сведения
 * о группе. А именно, возвращается
 * объект с полями id, name, count,
 * где count – кол-во студентов для
 * данной группы.
 **/
function get_group($id);

/**
 * Данная функция по идентификатору группы
 * возвращает список студентов для
 * данной группы. Каждый элемент списка
 * содержит поля id и name студента. Список
 * должен быть отсортирован в алфавитном порядке.
 **/
function get_student_list($id);

/**
 * Данная функция производит
 * валидацию предлагаемого имени
 * группы для редактирования. Если валидация
 * прошла успешно, то возвращается TRUE,
 * в противном случае возвращается строка
 * с описанием ошибки. Аргумент содержит два
 * поля: id и name.
 **/
function validate($group);

/**
 * Данная функция обновляет в базе
 * запись о группе. Входной аргумент содержит
 * два поля: id и name.
 **/
function update($group);

/**
 * Данная функция по идентификатору
 * группы производит ее удаление из базы.
 * Если в группе есть студенты, то
 * функция должна прервать выполнение
```

```
 * с сообщением об ошибке.
 **/
function delete_group($id);

//***************Вариант 3**/
/**
 * Данная функция по идентификатору
 * страны извлекает из базы сведения
 * о стране. А именно, возвращается
 * объект с полями id, name, count,
 * где count – кол-во отелей для
 * данной страны.
 **/
function get_country($id);

/**
 * Данная функция по идентификатору страны
 * возвращает список отелей для
 * данной страны. Каждый элемент списка
 * содержит поля id и name отеля. Список
 * должен быть отсортирован в алфавитном порядке.
 **/
function get_hotel_list($id);

/**
 * Данная функция по идентификатору
 * отеля должна вернуть детальные
 * сведения о нем. А именно, объект
 * со следующими полями: id, name, price,
 * countryname, countryid, firmname.
 **/
function get_hotel($id);

/**
 * Данная функция по идентификатору
 * страны производит ее удаление из базы.
 * Если в стране есть отели, то
 * функция должна прервать выполнение
 * с сообщением об ошибке.
 **/
function delete_country($id);

/**
 * Данная функция производит
```

```
 * валидацию предлагаемого имени
 * отеля для добавления. Если валидация
 * прошла успешно, то возвращается TRUE,
 * в противном случае возвращается строка
 * с описанием ошибки.
 **/
function validate($name);

/**
 * Данная функция добавляет в базу
 * новый отель. Входной аргумент
 * содержит три поля: name, price, firmid.
 * Возвращает первичный ключ новой записи.
 **/
function add($hotel);

//***************Вариант 4**/
/**
 * Данная функция производит
 * валидацию предлагаемого имени
 * улицы для добавления. Если валидация
 * прошла успешно, то возвращается TRUE,
 * в противном случае возвращается строка
 * с описанием ошибки.
 **/
function validate_street($name);

/**
 * Данная функция добавляет в базу
 * улицу с указанным именем. Возвращает
 * первичный ключ новой записи.
 **/
function add_street($name);

/**
 * Данная функция по идентификатору
 * улицы извлекает из базы сведения
 * об улице. А именно, возвращается
 * объект с полями id, name, count,
 * где count – кол-во адресов для
 * данной улицы.
 **/
function get_street($id);
/**
```

```
 * Данная функция по идентификатору улицы
 * возвращает список адресов для
 * данной улицы. Каждый элемент списка
 * содержит поля id, apartment, house адреса. Список
 * должен быть отсортирован в алфавитном порядке.
 **/
function get_address_list($id);

/**
 * Данная функция производит
 * валидацию предлагаемого адреса. Если валидация
 * прошла успешно, то возвращается TRUE,
 * в противном случае возвращается строка
 * с описанием ошибки. Аргумент содержит поля
 * streetid, house, apartment
 **/
function validate_address($address);

/**
 * Данная функция добавляет в базу
 * новый адрес. Входной аргумент
 * содержит три поля: streetid, house, apartment.
 * Возвращает первичный ключ новой записи.
 **/
function add_address($address);
```

12.7. Содержание отчета

1. Цель работы.
2. Вариант задания.
3. Исходный текст программы на PHP.
4. Скриншоты сгенерированных страниц.
5. Выводы.

12.8. Контрольные вопросы

1. В чем отличие функции **require_once** от **include_once**?
2. Каков синтаксис выражений, используемых в операции **LIKE**?
3. Какая процедура позволяет MySQL считывать из базы не больше *n* записей в выборке?
4. Какая процедура позволяет MySQL пропускать в выборке первые *n* записей?

Лабораторная работа № 13

ОБРАБОТКА ФАЙЛОВ

13.1. Цель работы

Знакомство с файловыми функциями PHP и организацией файлового хранилища.

13.2. Введение

В ходе предыдущих работ была создана система, которая выглядит достаточно хорошо с точки зрения модульности и позволяет пользователям заполнять ее содержимое самостоятельно. Есть лишь одно исключение: они не могут добавлять файлы, что является неотъемлемым атрибутом современных web-систем.

Передача файлов определяется стандартом RFC-1867 и любой браузер, который поддерживает этот стандарт, может отсылать файлы серверу. Согласно этому стандарту файлы пересылаются в виде отдельных фрагментов внутри POST-запроса. Это означает, что пользователь может одновременно переслать несколько файлов, если в форме отправки расположены несколько элементов загрузки файлов. Однако файлы отличаются от остальных полей, передаваемых через POST-запрос, по нескольким пунктам:

1. недостаточно просто передать название элемента и содержимое файла, как это делалось с другими полями. Дело в том, что у файла есть и другие атрибуты, например, его название на компьютере пользователя. Следовательно, необходимо пересылать и эту дополнительную информацию. Поэтому с файлами нельзя работать через глобальную переменную **POST**;

2. файлы нельзя хранить в базе данных, так как их содержимое не является текстовым или числовым, это набор байтов. Хотя ряд баз поддерживают хранение таких типов, как **BLOB**, использование этого типа порождает серьезные проблемы. Резко возрастает размер базы данных, и даже простое резервирование становится существенной проблемой. Поэтому наиболее распространенным считается вариант, когда файлы хранятся в отдельном файловом хранилище, а в базе –

ссылки к этим файлам. По этим ссылкам система может найти в хранилище интересующий пользователя файл и выдать его содержимое с помощью скрипта;

3. файлы могут быть очень большого размера, и путем их передачи можно вызвать сбой сервера. Необходимо каким-то образом задавать ограничения на размеры пересылаемых файлов.

13.3. Загрузка файлов на сервер в PHP

Размер файла, который может принимать система, контролируется в **php.ini** следующими параметрами:

- **upload_max_filesize** – максимальный размер каждого файла, пересылаемого пользователем. Если размер файла превышает это значение, то генерируется ошибка, которая записывается в глобальную переменную, отвечающую за работу с файлами;
- **max_file_uploads** – сколько файлов можно отправлять через один запрос. По умолчанию количество неограничено;
- **post_max_size** – максимальный размер POST-запроса в целом. Если он состоит из нескольких файлов, то полезно ограничить их суммарный вес;
- **upload_tmp_dir** – директория для временной загрузки файлов.

Остановимся подробнее на последней директиве. Когда на сервер приходит POST-запрос с файлами внутри, они для начала попадают во временную директорию, указанную этой директивой. Для них генерируются специальные временные названия, которые вместе с исходными названиями файлов помещаются в специальную глобальную переменную. Также в эту переменную помещается статус ошибки, чтобы скрипт мог определить в случае необходимости, что ему делать.

13.3.1. Загрузка одного файла

Итак, рассмотрим первый пример, как система может обработать файл, загруженный пользователем. Предположим, что на странице располагается форма, в которой один элемент выбора файла и кнопка отправки

```
<form enctype = "multipart/form-data" action = "upload.php" method = "post">
  Выбрать файл: <input name = "userfile" type = "file">
  <input type = "submit" value = "Отправить">
</form>
```

Обратите внимание, что в данном случае в форме прописывается дополнительный атрибут **enctype**, который может принимать два значения. Значение по умолчанию гласит, что содержимое формы отправляется закодированным по стандартам URL. Однако в данном случае это привело бы к резкому росту размера пакета, поэтому данные формы кодируются в виде комплексного сообщения, где часть содержимого представляется в виде потока байтов (это содержимое файла). Сам элемент загрузки представляется в виде тэга **input** со специальным типом **file**. Как этот элемент отображается на странице, зависит от браузера. Обычно это текстовое поле, рядом с которым располагается кнопка, нажав на нее, пользователь видит диалоговое окно. После выбора в этом окне файла абсолютный путь к нему прописывается в текстовом поле.

Когда пользователь нажимает на кнопку отправки, на сервер уходит POST-запрос. Перед тем как запустить скрипт upload.php, PHP проверяет файл на корректность (т. е. что он дошел полностью и его размер не превышает допустимый предел). Если все прошло хорошо, он помещает его во временный каталог и прописывает необходимые сведения о нем в глобальной переменной. Вот как скрипт может обработать эту информацию:

```
<?php
// Укажем каталог, в котором располагается наше файловое хранилище
$uploaddir = 'var/www/uploads';
// Вычислим имя файла, указанное пользователем,
// убрав абсолютный путь к нему
$uploadfile = basename($_FILES['userfile']['name']);

echo "<pre>";
// Попробуем записать файл из временного каталога
// в файловое хранилище. Для этого используем
// специальную функцию
if (move_uploaded_file($_FILES['userfile']['tmp_name'],
   "$uploaddir/$uploadfile")) {
  echo "Файл загружен успешно";
```

```
} else {
  echo "Возможная атака на сервер!";
}
echo "</pre>";
?>
```

Обратите внимание, что файл не находится во временной директории PHP постоянно, он будет удален во время очередного очищения каталога, поэтому файл необходимо переписать в какой-то постоянный каталог (который обычно прописывается в настройках конфигурационного файла).

13.3.2. Ограничение размера файла на стороне клиента

Существует способ изначально ограничить размер файла на стороне самого клиента, чтобы тот даже не пытался отправить его серверу в случае превышения этого размера. Например, у нас есть сайт, куда пользователь может «закачивать» аудиозаписи и свои собственные рисунки. Предельный размер файла в PHP можно установить только один. Предположим, администратор поставил его равным 10 МБт, исходя из размеров аудиозаписей. Однако очевидно, что рисунки по размеру должны быть гораздо меньше. Предположим, разработчик системы планирует, чтобы рисунки, размером больше 100 кБт, нельзя было загружать в систему. Можно на стороне скрипта производить обработку типа файла и его размера и выдавать ошибку, однако тогда пользователь может начать загрузку большого файла, ждать длительное время, после чего получить сообщение, что размер превышен. Существует способ, когда браузер может контролировать размер файла для конкретной формы:

```
<form enctype = "multipart/form-data" action = "upload.php" method = "post">
  <input type = "hidden" name = "MAX_FILE_SIZE" value = "100000">
  Выбрать файл: <input name = "userfile" type = "file">
  <input type = "submit" value = "Отправить">
</form>
```

В этой форме задано специальное скрытое поле **MAX_FILE_SIZE**, оно должно располагаться до элемента выбора файла, чтобы идти первым в POST-запросе. В этом случае браузер проверяет размер

файла перед отправкой. Если размер превышает указанное значение, то файл не отправляется POST-запросом. Вместо этого в запрос помещается сообщение об ошибке, которое скрипт может обработать и выдать пользователю сообщение, что именно не так.

Следует обратить особое внимание, что данная проверка на стороне клиента не отменяет необходимости в ограничениях на стороне сервера, так как злоумышленник может сгенерировать POST-запрос самостоятельно. Кроме того, некоторые старые версии браузеров могут не распознать это скрытое поле и отправить файл бóльшего размера. Поэтому в итоге на разработчиков ложится необходимость проверки всех файлов, что присылает пользователь. Однако данный прием позволяет отсечь отправку больших файлов добропорядочными пользователями, которые составляют основную массу пользователей на сайте.

13.3.3. Отправка множества файлов в одной форме

Предположим, что пользователь хочет сразу загрузить целую коллекцию файлов, например альбом с фотографиями. Для этого можно воспользоваться приемом, схожим с тем, что использовался для чекбоксов в предыдущих лабораторных работах:

```
<form enctype = "multipart/form-data" method = "post" action = "upload.php">
Рисунок 1: <input type = "file" name = "pictures[]"><br/>
Рисунок 2: <input type = "file" name = "pictures[]"><br/>
Рисунок 3: <input type = "file" name = "pictures[]"><br/>
<input type = "submit" value = "Отправить">
</form>
```

Теперь в качестве элемента **pictures** в глобальной переменной будет фигурировать уже целый массив. Обработать его скрипт может следующим образом:

```
<?php
foreach ($_FILES["pictures"]["error"] as $key => $error) {
  if ($error == UPLOAD_ERR_OK) {
 $tmp_name = $_FILES["pictures"]["tmp_name"][$key];
 $name = basename($_FILES["pictures"]["name"][$key]);
 move_uploaded_file($tmp_name, "$uploaddir/$name");
```

```
 } else if ($error != UPLOAD_ERR_NO_FILE) {
 echo "Ошибка при загрузке файла!<br/>";
 }
}
?>
```

Что происходит здесь? PHP всегда генерирует элемент с названием **error** для полученного файла. Если загрузка произведена успешно, то его значение равно специальному статусу **UPLOAD_ERR_OK**. В противном случае нужно удостовериться, что пользователь вообще выбрал какой-то файл. Поскольку файлов может быть выбрано несколько, они помещаются в специальные массивы. Путем прохода по массиву для **error** скрипт получает не только сами значения ошибок, но и их индексы в массивах. Далее по этим индексам в случае отсутствия ошибки скрипт вычисляет временное название файла и название, под которым этот файл загрузил пользователь.

13.4. Обработка файлов на сервере

В предыдущих работах было рассмотрено, как можно загружать файлы на сервер. Однако с файлами в дальнейшем необходимо как-то работать. Поскольку они располагаются в отдельном каталоге, на них нельзя просто сгенерировать ссылки. Необходимо через скрипт считать их содержимое из файла и выдать это содержимое клиенту. Кроме того, зачастую существует необходимость удаления файлов.

13.4.1. Генерация ответа в виде файла серверу

Предположим, что у каждого пользователя есть рисунок в профиле, который необходимо отображать при просмотре профиля. Для этого на соответствующей странице пропишем следующий код:

```
<img src = "image.php?id=<?=$user->id?>"/>
```

В файловом хранилище имеется специальный файл **none.jpg**, если у пользователя пока нет рисунка. Путь к рисунку (относительный) хранится в таблице **Users** в специальном текстовом поле **profile**. Тогда код **image.php** может выглядеть следующим образом:

```
<?php
$id = $_GET['id'];
$query = "select profile from Users where id = $id";
$rs = mysql_query($query);
$row = mysql_fetch_array($rs, MYSQL_ASSOC);
$profile = $row["profile"];

if (empty($profile)) {
  $profile = "none.jpg";
}

$filename = "$uploaddir/$profile";

// Попытаемся открыть файл
$handle = fopen($filename, "rb");
if ($handle === FALSE) {
  die("Ошибка выполнения скрипта";
}

// Необходимо очистить буфер вывода
// данных, чтобы избежать ошибок в ряде браузеров
ob_end_clean();
ob_start();
// Выставим заголовок с пустым содержимым
header("Content-Type:");

// Считываем файл поблочно
while (!feof($handle)) {
  $buffer = fread($handle, 1024);
  echo $buffer;
  ob_flush();
  flush();
}
?>
```

Стоит обратить внимание на то, что данный скрипт считывает содержимое файла поблочно порциями по 1024 байта, чтобы не повышать затраты по памяти для выполнения скрипта. Кроме того, приходится производить периодическую очистку буфера (т. е. выдачу содержимого сразу после записи в буфер), чтобы опять же не возникало увеличения затрат по памяти в скрипте.

13.4.2. Удаление файлов из системы

Предположим, пользователь решил удалить свой профиль. Для этого надо сделать пометку на уровне базы, что у него нет профиля, и удалить рисунок из файлового хранилища, так как в противном случае он там и останется. Вот как это можно сделать:

```
<?php
$id = $_GET['id'];
$query = "select profile from Users where id = $id";
$rs = mysql_query($query);
$row = mysql_fetch_array($rs, MYSQL_ASSOC);
$profile = $row["profile"];

if (empty($profile)) {
  die("У данного пользователя нет профиля");
}

$filename = "$uploaddir/$profile";

if (unlink($filename)) {
  mysql_query("UPDATE Users SET profile = NULL WHERE id = $id");
  echo "Удаление произведено успешно";
}
?>
```

13.5. Варианты заданий

В каждом варианте в конфигурационный файл необходимо добавить настройки, которые будут указывать:

- где в системе располагается файловое хранилище;
- размер блока – какими порциями система будет считывать содержимое файла из хранилища;
- ограничивать размер файла на стороне браузера.

Необходимо определить в системе отдельную библиотеку **filelib.php**, в которой определены следующие функции:

```
<?php
/**
* Данная функция производит вывод
```

```
* содержимого файла поблочно на страницу.
* @param filename Относительный путь к файлу (относительно хранилища).
**/
function generate_content($filename);

/**
* Данная функция проверяет допустимость данного
* расширения согласно варианту.
* @param ext Расширение
* @return TRUE – если расширение допустимо,
* FALSE в противном случае.
function check_extension($ext);

/**
* Данная функция проверяет все каталоги
* в указанном пути и создает те из них, которые
* отсутствуют.
* Например, путь равен "users/profiles/1", а в хранилище
* есть только users. Тогда в этом каталоге будет создан
* каталог profiles, а в этом каталоге будет создан
* каталог 1.
**/
function create_dir($dir);
?>
```

В хранилище располагается каталог **profiles**, в котором хранятся рисунки профилей пользователей. В таблице **users** должно появиться новое поле, хранящее относительный путь к рисунку пользователя (относительно каталога **profiles**). На каждой странице рядом с именем пользователя следует отображать небольшой рисунок с его профилем (если профиля нет, то отображать специальный рисунок **none.jpg**). Размер рисунка должен быть фиксированным и не зависеть от размера файлов, загруженных пользователем.

Во всех вариантах каталоги для сущностей могут быть заранее не созданы, поэтому при загрузке файлов необходимо проверять существование этих каталогов и создавать их в случае необходимости. Не забывайте про удаление рисунков при удалении соответствующей сущности.

1. У каждой фирмы может быть эмблема (а может и не быть). Она должна отображаться на страницах просмотра сведений о списке

фирм (отдельный столбец) и просмотра сведений о фирме. Задавать это поле можно на страницах добавления/редактирования сведений о фирме. Допустимые расширения для рисунков "jpg/jpeg/gif".

2. У каждой группы может быть несколько фотографий группы. Они отображаются в виде галереи внизу страницы просмотра сведений о группе. Рисунки должны быть одинакового размера. Под рисунками должна быть форма отправки, в которой будут располагаться выбор файла и кнопка отправки. Допустимые расширения для рисунков "jpg/jpeg".

3. У каждой страны может быть несколько рисунков. Они отображаются в виде галереи внизу страницы просмотра сведений о стране. Рисунки должны быть одинакового размера. Под рисунками должна быть форма отправки, в которой расположены выбор файла и кнопка отправки. Допустимые расширения для рисунков "jpg".

4. У каждого адреса может быть фотография дома для данного адреса. Она отображается на странице просмотра сведений об улице для соответствующих адресов. Рядом с каждой фотографией должен отображаться чекбокс. Внизу расположена кнопка "Удалить фотографии", которая будет убирать из хранилища фотографии для данных адресов. Фотографию можно задавать на странице добавления адреса. Допустимые расширения "jpg/jpeg".

13.6. Содержание отчета

1. Цель работы.
2. Вариант задания.
3. Исходный текст программы на PHP.
4. Скриншоты сгенерированных страниц.
5. Выводы.

13.7. Контрольные вопросы

1. Какие поля заполняются в глобальной переменной для загружаемого файла?
2. Какие коды ошибок возможны при передаче файла?
3. Сигнатура функции **move_uploaded_file**.

4. Сигнатура функции **fopen**. В каких режимах можно открывать файл?
5. Сигнатура функции **fread**. Как определить, что конец файла достигнут?
6. Сигнатура функции **flush**. Почему необходимо вызывать еще и **ob_flush**?

Лабораторная работа № 14

ВЫПОЛНЕНИЕ СЦЕНАРИЕВ НА СТОРОНЕ КЛИЕНТА

14.1. Цель работы

Знакомство со сценариями, выполняемыми на стороне клиента, и базовыми возможностями языка Javascript.

14.2. Введение

Созданная в ходе предыдущих лабораторных работ система предоставляет все необходимые возможности пользователям – заполнение базы своим собственным содержимым, добавление произвольных файлов и т. д. Однако в плане дизайна она не очень удобна, так как в ней все сценарии выполняются на стороне *сервера*.

Это означает, что при любом нажатии на кнопку или ссылку генерируется запрос к серверу и происходит перегрузка всей страницы. Однако в ряде случаев необходимо сделать так, чтобы обработка производилась на стороне клиента без перегрузки страницы в условиях, когда обращения к серверу вообще не происходит. Рассмотрим несколько вариантов такого взаимодействия с пользователем на стороне клиента.

1. В нескольких вариантах пользователь выбирает набор сущностей, помечая их галочками, и удаляет, нажимая на кнопку. С точки зрения безопасности стоит сделать так, чтобы система осуществляла запрос на подтверждение у пользователя и только затем передавала серверу этот запрос на удаление.

2. Во всех вариантах пользователь добавляет новые сущности на уровень базы, при этом валидация производится путем перезагрузки всей страницы. Однако проверить введенные пользователем значения по критериям пустоты и максимальной длины можно и на стороне клиента, не обращаясь к серверу.

3. Предположим, на странице расположен скрытый блок, который должен быть виден пользователю при каких-то определенных условиях. Например, на странице профиля по умолчанию стоит галочка "Гражданин РФ". Если пользователь при заполнении информации сделал эту галочку непомеченной (т. е. он не гражданин РФ), то необходимо, чтобы на странице появилось текстовое поле, в которое он мог бы ввести информацию о стране, из которой он прибыл.

Для обработки подобных сценариев были созданы различные скриптовые языки, самым распространенным из которых является Javascript.

14.3. Основы языка Javascript

Данный язык унаследовал синтаксис от семейства C-подобных языков. Подобно PHP в нем отсутствует строгая типизация, т. е. переменные и функции не имеют явно заданного типа. Так же как и в PHP, память выделяется под объекты по требованию и удаляется специальным сборщиком мусора.

14.3.1. Объявление переменных

Переменные не являются строго типизированными и в любой переменной можно хранить любое значение. Название переменной может начинаться с буквы, знака подчеркивания либо знака доллара. Остальные символы могут быть и цифрами. Для того чтобы объявить переменную, следует поставить перед ней ключевое слово **var**. Внутри функции можно адресоваться к глобальным переменным, не объявляя их явно как глобальные. Вот пример кода с использованием переменных различной области видимости:

```
// Объявление глобальной переменной
var x = 0;
```

```
function f() {
  // Объявляем две локальные переменные
  var z = 'foxes', r = 'birds';
  // Следующая переменная будет объявлена
  // как глобальная, поскольку нет слова var
  m = 'fish';
  // Объявляем внутреннюю функцию
  function child() {
    // Эта переменная является локальной
    // и не влияет на переменную r,
    // объявленную выше,
    var r = 'monkeys';
    // Дочерняя функция видит переменную z,
    // объявленную в родительской функции
    z = 'penguins';
  }

  child();
  // Здесь можно использовать глобальную переменную x
  return x;
}

// Вызов функции
f();
// Следующий код вызовет исключение,
// т. к. локальная переменная z более недоступна
alert(z);
```

Механизм таков: когда Javascript-интерпретатор пытается найти переменную, значение которой он должен использовать, он ищет переменную с таким названием в области видимости функции. Если это имя не найдено, поиск производится в родительской области видимости (внутри которой объявлена эта функция) и так далее вверх по цепочке, пока не дойдет до глобальной области видимости переменных. Если и в этом случае интерпретатор не найдет данную переменную, он сгенерирует исключение.

При присваивании переменной значения интерпретатор действует так же, с той лишь разницей, что если он не нашел переменной даже в глобальной области видимости, то создаст такую переменную, причем именно в глобальной области видимости. Таким образом, необъявленная переменная становится глобальной при операции присваивания.

14.3.2. Типы данных

Подобно PHP в Javascript встроено несколько фундаментальных типов данных (числа, символы, строки, булевские значения). Имеется специальное значение **undefined**, которое присвоено всем непроинициализированным переменным, а также их полям, которые в данный момент не существуют. При этом **undefined** не равен **null**, так как переменную можно проинициализировать таким значением:

```
// Переменная объявлена,
// но не проинициализирована
var test;
// Переменная объявлена
// как пустой объект
var testObj = {};

// Следующие команды выдадут
// undefined
alert(test);
alert(test.myProp);
```

14.3.3. Объявление функций

В отличие от C++ или Pascal, в Javascript каждая функция является экземпляром класса **Function**, т. е. *объектом*. Поэтому функцию можно объявить несколькими способами:

```
// Стандартное объявление
function add(x, y) {
  return x + y;
}
var t = add(1, 2);
alert(t);

// Нестандартное объявление
var add = new Function('x', 'y', 'return x + y');
var t = add(1, 2);
alert(t);
```

Поскольку функции являются объектами, то они вполне могут назначаться в качестве полей других объектов, приписываться и са-

мим переменным, а также возвращаться в качестве значения другими функциями. Помимо этого функции в качестве аргументов также могут принимать функции. Это так называемые функции *высшего порядка*.

Определение 1. *Функция, которая принимает на вход только объекты, но не другие функции, называется функцией первого порядка. Функцией n-го порядка называется функция, у которой хотя бы один аргумент является функцией (n−1)-го порядка.*

Эти функции пригодятся впоследствии при изучении AJAX-технологии.

14.3.4. Анонимные функции

В коде можно определять и анонимные функции, у которых нет названия. Это вкупе с возможностью функций принимать и возвращать функции в качестве значений позволяет делать весьма интересные вещи. Рассмотрим несколько примеров.

Пример 4. *Дан список книг в виде массива bookList, у каждого из элементов которого есть поле sales – количество проданных экземпляров. Необходимо реализовать функцию, возвращающую из этого списка только те элементы, количество проданных экземпляров которого не меньше порогового значения threshold.*

Приведем сразу код этой функции, он очень компактен по размеру

```
function getBestSellers(bookList, threshold) {
  return bookList.filter(
    function(book) {return book.sales >= threshold;}
  );
}
```

Рассмотрим, что здесь происходит. Переменная bookList является массивом, а у всех объектов типа **Array** в Javascript есть функция **filter**, которая принимает на вход функцию булевского типа с одним аргументом. **Filter** проходит последовательно по всем элементам исходного массива и для каждого вызывает внутреннюю функцию, переданную в качестве аргумента. Если функция возвращает **true**, то элемент будет добавлен в итоговую коллекцию, возвращаемую **filter**. Вообще в функциональном программировании есть три базовые

функции (еще со времен языков семейства Lisp), которые считаются функциями высшего порядка – это **filter**, **map** и **fold**, однако здесь они детально рассматриваться не будут. Внутри в качестве аргумента используется анонимная функция – создается объект класса **Function**, который ожидает один входной аргумент, но при этом не имеет названия. Это нужно в комплексных приложениях, где создаются объекты сложной структуры с большим количеством функций-обработчиков различных ситуаций. Рассмотрим еще один пример:

Пример 5. *Необходимо разработать функцию, которая принимает на вход в качестве аргумента произвольную функцию f и генерирует ее производную, аппроксимируя ее на узком отрезке dx, который передается ей в качестве второго входного аргумента.*

Код этой функции может выглядеть следующим образом:

```
function derivative(f, dx) {
  return function(x) {
    return (f(x+dx) - f(x))/dx;
  };
}
```

Как видно из кода, мы возвращаем новую функцию (но не даем ей названия), которая принимает на вход один аргумент *x* – число. При этом внутри функции доступны значения объектов *f* и *dx*, даже после завершения работы внешней функции **derivative**, т. е. можно использовать далее эту функцию следующим образом:

```
var f = function(x) {return x*x;};
var der = derivative(f, 0.01);
t = der(1);
alert(t);
```

14.4. Встраивание скриптов в код страницы

Итак, с базовыми особенностями языка мы ознакомились. Теперь разберемся, как встраивать Javascript в код страницы. Его можно включить напрямую

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

```
<title>Первая страница с Javascript</title>
<script type = "text/javascript">
var x = {};
function f() {
  return 0;
}
</script>
</head>
<body>
</body>
</html>
```

Это не всегда удобно, особенно если на множестве страниц используются одни и те же фрагменты кода Javascript, поэтому обычно код выносят в файлы с расширением .js и размещают в каталоге сайта. Их включают в страницу следующим образом:

```
<script language = "javascript" src = "scroller.js"/>
```

14.5. Событийная модель обработки

Итак, мы определили какие-то функции и объекты в коде Javascript на странице, а как же их теперь использовать? Javascript использует для этого событийную модель, т. е. объекты на странице могут иметь какие-то события (эти события прописываются в качестве атрибутов тэгов). В качестве обработчиков событий могут выступать различные функции javascript. Типичные примеры событий:

1. Нажатие кнопки или ссылки пользователем.
2. Загрузка страницы или отдельного рисунка.
3. Курсор наведен над определенным элементом на странице.
4. Выбор какого-либо элемента на странице.
5. Попытка отправки данных формы.
6. Ввод текста в текстовом поле.

Рассмотрим наиболее распространенные варианты. В ряде вариантов требуется удалить данные, которые пользователь помечает галочками после того, как он нажмет на кнопку. Было бы желательно встроить защиту от неосторожных действий пользователя, чтобы страница запрашивала подтверждение и в случае отказа пользователя не производила отправку данных на сервер (и заодно не перезагружала страницу). Для этого можно использовать следующий код:

```
<script type = "text/javascript">
function check_form() {
  return confirm('Вы уверены, что хотите удалить данные записи?');
}
</script>

<form method = "post" action = "upload.php"
  onsubmit = "javascript:return check_form()">
  <input type = "checkbox" name = "records[]" value = "1"> запись 1<br/>
  <input type = "checkbox" name = "records[]" value = "2"> запись 2<br/>
  <input type = "checkbox" name = "records[]" value = "3"> запись 3<br/>
  <input type = "submit" value = "Удалить">
</form>
```

Что здесь происходит? В Javascript есть возможность вызова нескольких типов всплывающих окон, здесь используется **confirm**. Эта функция генерирует всплывающее окно с текстом, который передан ей в качестве аргумента, и двумя кнопками "ОК/Отмена". Если пользователь нажал "ОК", то возвращается значение **true**, в случае отмены возвращается **false**.

Далее в форме определено событие **onsubmit**. Если код события возвращает **true**, то произойдет отправка данных. В противном случае страница не будет перезагружена и пользователю ничего отправлено не будет. Важно также, что вызов функций является *синхронным*, т. е. при вызове функции **check_form** страница блокируется; пока пользователь не нажмет одну из кнопок, не будет возвращено значение функцией **confirm**. Это значит, что если код, выполняемый внутри Javascript, требует продолжительного времени, то страница будет просто заблокирована и браузер не будет реагировать на действие пользователя. Поэтому для таких длительных операций должен использоваться асинхронный способ вызова (он рассматривается в следующей лабораторной работе).

14.6. DOM-модель документа

Предыдущий пример кода позволял реагировать на событие, однако само по себе событие зачастую несет мало информации. Было бы желательно знать значения других элементов на странице (поскольку их мог отредактировать пользователь). Кроме того, необходимо каким-

то образом перерисовывать часть страницы, не перегружая ее полностью и не обращаясь за этим к серверу. Для того чтобы это было можно сделать, в Javascript используется специальный инструментарий DOM (Document Object Model). Это значит, что HTML-документ представляется в виде дерева. Каждый тэг соответствует узлу дерева. Атрибуты тэга являются атрибутами узла, а внутреннее содержимое тэга представляется в виде специального информационного поля узла. Дочерние тэги становятся потомками исходного узла. Рассмотрим на примере:

```
<input type = "text" name = "fio" value = "">
```

Данный тэг будет представлен в виде дерева из одного узла. Этот узел будет иметь три атрибута – **type, name, value** с соответствующими значениями. Каждый атрибут доступен как для чтения, так и для записи, т. е. при необходимости, найдя соответствующий узел, можно модифицировать его атрибуты. Рассмотрим более сложный пример:

```
<select name = "target">
  <option value = "1"><li>Запись</li> 1</option>
  <option value = "2">Запись 2</option>
  <option value = "3">Запись 3</option>
</select>
```

Это фактически преобразуется в следующее дерево:

```
select
--option
----li
--option
--option
```

Что касается атрибутов узлов, то здесь все понятно. Как выставлены дочерние узлы в иерархии, тоже очевидно из данного примера. Неочевидно то, чему равно внутреннее содержимое для каждого узла. Оно равно тексту, который располагается между открывающим и закрывающим тэгами для данного узла. Таким образом, Javascript должен предоставлять следующие возможности:

1) находить один из узлов в дереве страницы по какому-то критерию;
2) считывать и записывать внутреннее содержимое узла;
3) иметь доступ к списку атрибутов узла с возможностью модификации;
4) иметь доступ к дочерним элементам узла с возможностью модификации.

Обычно для того чтобы получить доступ к элементам, система адресуется к ним по идентификаторам. Не стоит путать их с названиями, передаваемыми в POST-запросе. Идентификатор в тэге задается атрибутом **id**. Рассмотрим следующий пример:

Пример 6. *Необходимо разработать форму, в которой имеются два поля – фамилия и имя пользователя, а также есть кнопка отправки. При отправке пользователем данных должна производиться валидация на стороне клиента. В случае, если значения полей пустые либо длина значения превышает 30 символов, отправка не осуществляется. Вместо этого рядом с соответствующим полем возникает сообщение красного цвета об ошибке.*

Форма будет выглядеть следующим образом:

```
<form method = "post" action = "add.php" onsubmit = "return check_form()">
Фамилия: <input name = "lastname" id = "lastname" value = "">
<div id="error_lastname"></div><br/>

Имя: <input name = "firstname" id = "firstname" value = "">
<div id = "error_firstname"></div><br/>

<input type = "submit" value = "Отправить">
</form>
```

А вот скрипт, который проверяет значение полей и производит валидацию:

```
<script type = "text/javascript">

function isEmpty(text) {
  return text === "";
}
```

```
function isTooLong(text) {
  return text.length > 30;
}

function check_form() {
  var lastname = document.getElementById("lastname");
  var firstname = document.getElementById("firstname");
  var error_empty = "<font color = \"red\">Поле не может быть
пустым</font>";
  var error_too_long =
    "<font color = \"red\">Длина поля не может превышать 30
символов</font>";
  var result = true;
  var error_lastname = document.getElementById("error_lastname");
  var error_firstname = document.getElementById("error_firstname");


  if (isEmpty(lastname.value)) {
    error_lastname.innerHTML = error_empty;
    result = false;
  } else if (isTooLong(lastname.value)) {
    error_lastname.innerHTML = error_too_long;
    result = false;
  } else error_lastname.innerHTML = "";

  if (isEmpty(firstname.value)) {
    error_firstname.innerHTML = error_empty;
    result = false;
  } else if (isTooLong(firstname.value)) {
    error_firstname.innerHTML = error_too_long;
    result = false;
  } else error_firstname.innerHTML = "";

  return result;
}
</script>
```

14.7. Варианты заданий

Во всех вариантах нужно сделать следующее:

1. Страница авторизации должна производить валидацию вводимых пользователем значений на стороне клиента. Если логин или

пароль пустые или их длина превышает 25 символов, то страница не должна отправлять данные серверу, а вместо этого приписывать рядом с проблемным полем сообщение об ошибке.

2. Для всех кнопок, вызывающих удаление сущностей, должно всплывать окно подтверждения, которое запрашивает у пользователя, действительно ли он хочет удалить сущности.

3. То же самое должно производиться для страниц добавления/редактирования сущностей. Когда пользователь жмет на кнопку "Сохранить", страница запрашивает подтверждение у пользователя.

14.8. Содержание отчета

1. Цель работы.
2. Вариант задания.
3. Исходный текст программы на PHP.
4. Скриншоты сгенерированных страниц.
5. Выводы.

14.9. Контрольные вопросы

1. Какие типы событий наиболее часто используются в Javascript?
2. Какие типы всплывающих окон может генерировать Javascript?
3. Какая сигнатура функции **filter** и что она принимает?
4. Сигнатура функции **map**.
5. Дайте определение функции **fold**.
6. Как можно в Javascript получить список всех элементов для определенного типа тэга, например **input**, имеющихся в документе?
7. Как получить в Javascript список атрибутов для данного узла?
8. Каким образом получить в Javascript список всех дочерних элементов для данного узла?
9. Как добавить новый дочерний элемент в узел, не прибегая к модификации **innerHTML**?
10. Как удалить один из дочерних элементов узла?

Лабораторная работа № 15

ЗНАКОМСТВО С AJAX

15.1. Цель работы

Знакомство с технологией асинхронного взаимодействия с сервером без перегрузки страницы в целом.

15.2. Введение

В рамках предыдущей лабораторной работы были реализованы простейшие варианты взаимодействия браузера с клиентом без перегрузки страницы в целом (посредством Javascript). Однако, хотя это значительно улучшило функциональность, это взаимодействие ограничено текущим содержимым документа и не может использовать дополнительную информацию от сервера, не производя при этом перегрузки страницы. Рассмотрим ряд сценариев, когда такой функциональности недостаточно.

1. Необходимо реализовать поиск. Поисковая форма содержит текстовое поле и кнопку, располагающуюся рядом с полем. Было бы желательно реализовать автозаполнение, т. е. когда пользователь начинает что-то вводить в поле, нужно, чтобы браузер реагировал на события, когда в форме что-то меняется (это событие **onkeyup**). Браузер должен обратиться к серверу и передать ему то, что ввел пользователь.

Сервер осуществляет поиск по базе и присылает браузеру сообщение, в котором имеются десять наиболее популярных фраз, содержащих введенный пользователем текст в качестве подстроки. Получив ответ от сервера, браузер декодирует ответ и генерирует на его основе блок под текстовым полем, в который добавляет полученные сведения в формате, понятном пользователю.

2. Необходимо реализовать каскадные списки. Например, пользователь в анкете должен указать, какое учебное заведение он закончил. Для этого в анкете есть три каскадных списка. В первом он выбирает область, где находится заведение, во втором – город области,

где находится данное заведение, в третьем – имеющиеся в данном городе учебные заведения.

Очевидно, что всю эту информацию заранее можно записать на странице, однако тогда ее объем станет чрезмерным. Более предпочтительным считается вариант, когда на странице изначально размещается только список областей, а содержимое остальных списков генерируется динамически. На основе выбора пользователя в первом списке генерируется содержимое второго, а на основе выбора пользователя во втором списке генерируется содержимое третьего списка.

В эпоху первых браузеров поддержка таких сценариев также была возможна, но она неизбежно приводила к перегрузке всей страницы. В 1999 г. Microsoft представила специальный **ActiveX** объект, который позже был реализован во всех наиболее распространенных браузерах под названием **XmlHttpRequest**, который позволял реализовывать взаимодействие клиента с сервером. Отметим особенности этого взаимодействия.

1. Вызов к серверу является *асинхронным*. Это означает, что интерпретатор Javascript, дойдя до строки, где производится вызов к серверу, не будет дожидаться окончания вызова, поскольку этот процесс может быть очень длинным. При передаче обращения к серверу браузеру указывается, какую функцию вызвать после получения ответа от сервера (либо по истечении тайм-аута).

Это указание возможно благодаря тому, что функции являются объектами (можно указать и анонимную функцию при необходимости). При вызове к серверу Javascript продолжит выполнение со следующей строчки после вызова. При этом объект, на который была передана ссылка браузеру, не будет удален из памяти, пока не придет ответ. При получении ответа осуществляется вызов этой функции, после чего она будет удалена из памяти.

2. Взаимодействие с сервером происходит в несколько этапов, и на каждом этапе может быть вызвана функция обработки. Поэтому она должна проверять ответ от сервера и производить перерисовку страницы только после получения конечного ответа (о том, является ли ответ конечным, можно судить по статусу).

3. В качестве ответа может приходить сообщение в формате XML (отсюда и буква **"X"** в аббревиатуре **Ajax**). Однако формат не обязательно должен быть именно XML. Возможно использование

других форматов, например, сообщение в виде обычного текста либо в формате JSON (этим список не ограничивается, но это наиболее распространенные форматы).

4. Для асинхронного взаимодействия с сервером обычно используется JavaScript (отсюда и буква **"J"** в аббревиатуре **Ajax**). Однако это требование также не является обязательным. Можно применять другие языки обработки сценариев на стороне клиента, если они поддерживают класс **XmlHttpRequest** (например, VBScript).

5. Обращение к объекту **XmlHttpRequest** может осуществляться через объекты **ActiveX** (в IE) либо напрямую к самому объекту (в браузерах Chrome, Opera, Firefox и др.). Это необходимо учитывать при создании скриптов, и перед вызовом объектов проверять наличие их в памяти интерпретатора.

6. Страница, сгенерированная на стороне сервера и переданная клиенту, не может производить асинхронные вызовы к третьей стороне – только к серверу, от которого она получена. Существует ряд возможностей для обхода этого ограничения (чтобы отображать на сайте информацию из третьих источников в режиме онлайн), однако в данной лабораторной работе они не рассматриваются.

15.3. Примеры использования

Рассмотрим наиболее распространенные примеры использования технологии **Ajax**. Все они строятся на передаче сообщений в виде прямого текста, т. е. скрипт на стороне сервера фактически генерирует часть HTML-страницы. Эта часть возвращается клиенту и чаще всего просто вставляется напрямую в нужный элемент, который уже существует на странице.

Автозаполнение в текстовом поле

Представим, что в базе содержится набор пользователей с именами и фамилиями. Нужно реализовать страницу, на которой пользователь в поле поиска начинает вводить часть фамилии, а ему под полем высвечиваются ответы, кто это может быть, без перезагрузки страницы. Для начала спроектируем страницу на стороне сервера.

1. Страница ожидает в качестве входного GET-параметра имя-параметр с названием **name**. Если параметр не задан или он пустой, то страница ничего не возвращает в ответ, она прерывает выполнение, так как подобный вызов с пустым параметром рассматривается как исключительная ситуация.

2. Если параметр задан, то производятся запрос к базе с помощью конструкции **LIKE** Пи поиск всех пользователей, у кого в фамилии содержится присланный текст. При этом пользователи в выборке сортируются в алфавитном порядке по фамилии и имени. При этом не должно выбираться более 10 пользователей за один раз (поскольку в качестве части фамилии может быть прислана одна буква).

3. Если выборка пустая, то страница не должна ничего возвращать.

4. Если выборка не пустая, то страница сформирует набор строк, каждая из которых содержит фамилию и имя пользователя.

Так выглядит данная страница:

```
<?php
$name = $_GET['name')
if (/**Имя не задано**/) {
  die;
}
$name = strtolower($name);

// Подключаемся к базе

$query = "SELECT lastname, firstname FROM users"
  ." WHERE lastname LIKE '%$name%'"
  ." ORDER BY lastname, firsname"
  ." LIMIT 10";

// Специально гасим предупреждения,
// чтобы не искажать ответ клиенту
$rs = @mysql_query($query);
$result = array();
while ($row = mysql_fetch_assoc($rs)) {
  $lastname = $row['lastname'];
  $firstname = $row['firstname'];
  $result []= "$lastname $firstname";
}
```

```
if (empty($result)) {
  die;
}

echo implode("<br/>", $result);
?>
```

Код достаточно прост. Гораздо более интересен код функции в JavaScript, который производит обращение к этой странице. Код действует на основе следующих предположений:

- на странице имеется текстовое поле, которое при вводе пользователем символов внутри этого поля будет производить вызов функции **showHint**, которой передается в качестве аргумента значение, введенное пользователем;
- если значение не пустое, то функция генерирует обращение к странице, описанной выше;
- после получения ответа функция пытается записать его в качестве внутреннего содержимого в элемент с идентификатором **txtHint** (предполагается, что на странице есть такое поле).

Вот как выглядит эта функция:

```
<script type = "text/javascript">
function showHint(str) {
  if (str.length == 0) {
    document.getElementById("txtHint").innerHTML = "";
    document.getElementById("txtHint").style.border = "0px";
    return;
  }
  if (window.XMLHttpRequest) {
    // Код для большинства браузеров
    xmlhttp = new XMLHttpRequest();
  } else {
    // Код для IE5, IE6
    xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
  }

  // Задаем анонимную функцию
  // в качестве обработчика события
  xmlhttp.onreadystatechange = function() {
    // Если это последняя стадия и сервер
    // успешно возвратил ответ
```

```
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
      document.getElementById("txtHint").innerHTML =
        xmlhttp.responseText;
      document.getElementById("txtHint").style.border =
        "1px solid #A5ACB2";
    }
  }
  // Открываем соединение с нужной страницей браузера
  xmlhttp.open("GET", "gethint.php?name=" + str, true);
  // Отправляем запрос
  xmlhttp.send();
}
</script>
```

Оставшуюся часть страницы предлагается дописать самостоятельно.

15.4. Каскадные списки

Предположим, что на странице находится два списка. В первом перечислены регионы Российской Федерации. Когда пользователь выбирает из этого списка один из регионов, во втором перечисляются города, которые есть в данном регионе. Естественно, заранее всю эту информацию загружать на странице не имеет смысла, так как объем страницы возрастет. Проще держать на странице информацию лишь по первому списку, а второй динамически генерировать заново каждый раз, когда пользователь сделает выбор в первом списке. Для начала напишем страницу, которая генерирует элементы дочернего списка по асинхронному запросу. Она ожидает идентификатор региона в качестве входного параметра и возвращает список, где первым элементом (со значением 0) идет **"не выбрано"**. Реализуем данную страницу с помощью некоторых возможностей функционального программирования, применяя функцию **map**.

```
<?php

/**
 * Возвращает строковое
 * представление одного элемента
 * массива в виде элемента списка
```

```
 * HTML.
 */
function get_element($elem) {
  return "<option value = \"{$elem->id}>"
    ."{$elem->name}</option>";
}

$id = $_GET['id'];
// Проводим валидацию
// и подключение к базе

$query = "SELECT id, name FROM cities"
  ." WHERE region = $id"
  ." ORDER BY name";
$rs = @mysql_query($query);

$result = array();
$elem = new stdClass;
$elem->id = 0;
$elem->name = "не выбрано";
$result []= $elem;

while ($row = mysql_fetch_assoc($rs)) {
  $elem = new stdClass;
  $elem->id = $row['id'];
  $elem->name = $row['name'];
  $result []= $elem;
}

echo implode("\n", array_map("get_element", $result));
?>
```

Что касается страницы, то скрипт на ней практически идентичен предыдущему примеру, поэтому часть его будет пропущена. Функция скрипта привязывается к событиям первого списка, и ей передается в виде строки значение идентификатора выбранного пользователем элемента в списке. По этому идентификатору производится запрос к серверу

```
<script type = "text/javascript">
function fillCities(str) {
  // Производим валидацию строки
  // и генерацию объекта
```

```
  xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
      document.getElementById("secondList").innerHTML =
        xmlhttp.responseText;
    }
  }
  xmlhttp.open("GET", "getlist.php?id=" + str, true);
  xmlhttp.send();
}
</script>
```

Валидация на стороне сервера

Инструментарий **Ajax** позволяет произвести валидацию на стороне сервера без перегрузки всей страницы. Напомним, что в предыдущей работе с помощью Javascript проверялось, являются ли введенные пользователем значения пустыми и не превышена ли их длина. Для таких проверок не нужно обращение к серверу, следует использовать исключительно инструменты самого браузера. Однако в этом случае невозможны более сложные проверки. Например, нельзя проверить без обращения к серверу корректность данных для добавления (нет ли в базе записей с таким названием и т. д.). **Ajax** позволяет осуществить такую проверку без перегрузки всей страницы.

1. Внутри формы, данные которой нужно отправить на сервер, располагается кнопка отправки. При нажатии кнопки производится гашение отправки – перегрузки страницы не возникает.

2. При нажатии на кнопку генерируется запрос к серверу, в котором содержится необходимая информация (например, введенное пользователем название). Запрос отправляется с помощью **XmlHttpRequest**.

3. При получении ответа от сервера он декодируется. Если проверка на сервере прошла успешно, данные отправляются в форме с помощью специальных возможностей Javascript. Если же проверка не пройдена, то генерируется сообщение об ошибке, которое добавляется в качестве элемента на страницу.

Рассмотрим серверную часть системы в этом случае. Она ожидает GET-запрос, а в качестве ответа возвращает открытым текстом два возможных значения: **"SUCCESS"/"FAILURE"** (кавычки для наглядности). Код ее может выглядить следующим образом:

```
<?php
// По умолчанию страница будет
// генерировать сообщение об ошибке
$success = "FAILURE";

if (isset($_GET["add"])
  && !empty($_GET["add"])) {
  $name = $_GET["add"];
  // Производим подключение к базе
  // Осуществляем запрос к базе
  $query = "SELECT COUNT(*) AS cnt"
    ." FROM autos WHERE name = '$name'";
  $rs = mysql_query($query) or die($success);
  $row = mysql_fetch_assoc($rs);
  $cnt = $row["cnt"];
  if ($cnt == 0) {
    $success = "SUCCESS";
  }
}
die($success);
?>
```

Сценарий на Javascript сначала проведет проверку на стороне браузера. Если текстовое поле не пустое и его длина не превышает допустимое значение, то только в этом случае отправляется запрос на сервер. Иначе будет просто выставлено сообщение об ошибке.

```
<script type = "text/javascript">
// Данная функция вставляет в
// элемент ошибки нужный текст
function set_message(msg) {
  if (msg == "")
    document.getElementById("errorName").innerHTML = "";
  else document.getElementById("errorName").innerHTML =
    "<font color = \"red\">" + msg + "</font>";
}

// Данная функция проводит
// проверку пришедшего от сервера
// сообщения и в случае необходимости
// сообщает пользователю, что
// такой автомобиль уже есть в базе.
// В противном случае производится отправка
```

```
// данных формы на сервер с перегрузкой страницы.
function validate_response(response, name) {
  if ($response !== "SUCCESS") {
    set_message("В базе уже есть автомобиль с названием "
      + name);
  } else {
    // Находим нужную форму по ее идентификатору
    document.forms["myform"].submit();
  }
}

function validate_form() {
  set_message("");

  var name = document.getElementById("name").value;
  if (is_empty(name)) {
    set_message("Поле не может быть пустым");
  } else if (is_too_long(name)) {
    set_message("Превышена допустимая длина поля");
  } else {
    // Создаем объект XmlHttpRequest
    xmlhttp.onreadystatechange = function() {
      if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        validate_form(xmlhttp.responseText, name);
      }
    }
    // Отправляем запрос серверу
  }
  return false;
}
</script>
```

15.5. Форматы передачи данных

Все рассмотренные выше варианты использовали формат передачи в виде обычного текста. Это не всегда удобно и правильно. С точки зрения архитектуры правильно отправлять от сервера клиенту только данные без тэгов и конструкций HTML, а уже клиент будет решать, как ему отображать эти данные – в виде списка, таблицы и т. д. Это позволит множеству страниц использовать один и тот же источник данных на сервере, отображая их по-разному в соответствии с

требованиями. В принципе такие данные можно передать прямым текстом. Однако это порождает много неудобств:

1) сложно передавать коллекции однотипных элементов. Для этого надо сгенерировать строку, являющуюся конкатенацией текстовых значений этих элементов, разделенных символами-разделителями, которые не должны встречаться ни в одном из элементов. Это означает большую трудоемкость программирования подобных систем;

2) ситуация еще больше усложняется, когда элементы являются *структурированными* (элемент состоит из набора полей, где каждое поле, в свою очередь, тоже может быть набором полей). Для этого нужно, чтобы сервер для коллекции объектов произвел *сериализацию* каждого объекта (преобразование объекта в текстовый эквивалент), а клиентская часть должна производить *десериализацию* полученных строк (восстановление исходных объектов).

Для этого необходим инструментарий, который позволял бы кодировать данные в каком-либо специальном формате на стороне сервера, передавать их клиенту и декодировать на стороне клиента.

15.5.1. Передача данных в XML-формате

Рассмотрим еще раз задачу о каскадных списках. Сервер фактически возвращает пары "ключ – значение" для городов, которые необходимо отобразить. Их и надо передавать без HTML оформления. Пусть они кодируются в XML-файле следующего вида:

```
<list>
  <city id = "1">Амстердам</city>
  <city id = "2">Антверпен</city>
  <city id = "3">Берлин</city>
</list>
```

Это есть корневой элемент, в котором располагается коллекция дочерних элементов. Идентификаторы представлены в виде атрибутов (это числа, и их кодирование не представляет проблемы), а названия – в виде внутреннего содержимого атрибутов (они могут содержать различные символы, которые нельзя закодировать в значении атрибута). Страница на стороне сервера, генерирующая такой ответ клиенту, может выглядеть следующим образом:

```
<?php
if (!isset($_GET["id"])) {
  die;
}
$id = $_GET["id"];
// Открываем соединение с базой
// Получаем нужную выборку

// Создаем новый объект XML в памяти
header("Content-type: text/xml; charset=UTF-8");
$writer = new  XMLWriter;
$writer->openMemory();
$writer->startDocument("1.0", "UTF-8");

// Создаем корневой элемент
$writer->startElement("list");

// Добавляем нулевой элемент
$writer->startElement("city");
$writer->writeAttribute("id", 0);
$writer->text("не выбрано");
$writer->endElement();

// Добавляем коллекцию
while ($row = mysql_fetch_assoc($rs)) {
  $writer->startElement("city");
  $writer->writeAttribute("id", $row["id"]);
  $writer->text($row["name"]);
  $writer->endElement();
}

// Закрываем корневой элемент
$writer->endElement();
// Отправляем документ в выходной поток
echo $writer->outputMemory(TRUE);
?>
```

Как видно, большую часть работы по генерации документа проделывает инструментарий **XMLWriter**. На стороне клиента его должен обрабатывать Javascript, в котором также есть возможности по обработке документов в формате XML. Для этого объект **XmlHttpRequest** должен использовать не **responseText**, а **responseXML** как поле ответа, потому что содержимое ответа будет храниться именно в нем:

```
<script type = "text/javascript">
// Пропускаем код по созданию объектов
// и отправке запроса серверу

// Передаем ответ функции обработки
xmlhttp.onreadystatechange = function() {
  if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
    fill_list(xmlhttp.responseXML.documentElement);
  }
}

// А вот сама функция,
// которая очищает список,
// и заполняет его новыми элементами
function fill_list(response) {
  list = document.getElementById("citylist");
  // Удаляем старые дочерние элементы
  while (list.hasChildNodes()) {
    list.removeChild(list.lastChild);
  }
  coll = response.getElementsByTagName("city");
  for (var i = 0; i<coll.length;i++) {
    var opt = document.createElement("option");
    opt.value = coll[i].id;
    opt.text = coll[i].nodeValue;
    list.options.add(opt);
  }
}
</script>
```

15.5.2. Передача данных в формате JSON

Изначально клиент взаимодействовал с сервером исключительно в XML-формате (поэтому и сам класс называется XmlHttpRequest). Однако он слишком громоздкий и при определенных обстоятельствах (когда объект состоит из набора полей, где размер каждого поля не слишком велик) может быть чересчур громоздким, т. е. большую часть сообщения будет занимать не собственно информация, а служебные элементы. Есть более компактный вариант передачи сообщений – формат JSON. Он позволяет представлять структурированные

данные. Вот как можно было бы представить один из объектов, описанных в предыдущем разделе:

```
{
  "id" : 1,
  "name" : "Амстердам"
}
```

Вот страница, где более сложный объект с комплексными полями:

```
{
  "name" : "product",
  "properties" :
  {
    "id" : 13,
    "code" : "LLL"
  }

}
```

Массивы объектов можно представить так:

```
[
  {
    "id" : 1,
    "name" : "Амстердам"
  },
  {
    "id" : 2,
    "name" : "Антверпен"
  },
  {
    "id" : 3,
    "name" : "Берлин"
  }
]
```

Формат интуитивно понятный и более компактный по сравнению с XML, однако разница несущественна, если применяется компрессия трафика. Однако при этом JSON является подмножеством языка Javascript и может быть преобразован в него напрямую, применяя функцию **eval**. Но данный способ не рекомендуется (лучше

использовать специальный парсер), если только разработчик не уверен абсолютно в надежности источника, который прислал это сообщение. Вот как выглядит серверная часть с кодированием формата в JSON:

```
<?php
if (!isset($_GET["id"])) {
  die;
}
$id = $_GET["id"];
// Открываем соединение с базой
// Получаем нужную выборку

//Создаем новый массив
$arr = array();

// Добавляем нулевой элемент
$elem = new stdClass;
$elem->id = 0;
$elem->name = "не выбрано";
$arr []= $elem;

// Добавляем коллекцию
while ($row = mysql_fetch_assoc($rs)) {
  $elem = new stdClass;
  $elem->id = $row["id"];
  $elem->name = $row["name"];
  $arr []= $elem;
}

// Отправляем в выходной поток закодированное представление
die(json_encode($arr));
?>
```

Как видим, данный способ гораздо компактнее с точки зрения объема требуемого кода. В данном случае ответ клиенту опять придет в поле **responseText** и его необходимо будет декодировать:

```
<script type = "text/javascript">
// Пропускаем код по созданию объектов
// и отправке запроса серверу

// Передаем ответ функции обработки
xmlhttp.onreadystatechange = function() {
```

```
  if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
    fill_list(xmlhttp.responseText);
  }
}

// А вот сама функция,
// которая очищает список
// и заполняет его новыми элементами
function fill_list(response) {
  // Очищаем список от старых дочерних элементов
  // Получаем коллекцию путем декодирования
  coll = JSON.parse(response);
  for (var i = 0; i<coll.length;i++) {
    var opt = document.createElement("option");
    opt.value = coll[i].id;
    opt.text = coll[i].name;
    list.options.add(opt);
  }
}
</script>
```

15.6. Варианты заданий

В данной работе задания не являются продолжениями предыдущих работ, а должны быть реализованы с "чистого листа".

1. Необходимо в базе создать три таблицы: стран, городов и учебных заведений. Каждый город связан с одной из стран, каждое заведение с одним из городов. На странице будут три каскадных списка и два текстовых поля. Первый каскадный список позволяет выбирать страну. Рядом с ним (слева) располагается текстовое поле. Изначально список пуст из-за большого количества стран. Когда пользователь введет в текстовом поле какой-либо непустой текст, в первом списке появятся страны, названия которых начинаются с этого текста (без учета регистра). После выбора страны можно будет переходить к выбору города.

Город выбирается также с помощью текстового поля и списка, но список заполняется сразу как только будет выбрана страна. С помощью текстового поля можно сузить список городов в стране, выбрав только те, названия которых начинаются с введенного пользователем текста во втором текстовом поле. Третий список заполняется

набором учреждений после того, как пользователь выбрал город. Для формата передачи данных использовать XML.

2. Реализовать первый вариант, но с использованием JSON.

3. В базе должно быть две таблицы: стран и пользователей (у пользователей есть фамилии и имена). Каждый пользователь связан с одной из стран. Необходимо реализовать страницу добавления, где должно быть два текстовых поля – фамилия и имя пользователя. Ниже представлен выпадающий список имеющихся в базе стран, который можно отфильтровать, введя в текстовом поле справа от списка начало названия страны (как в первом и втором вариантах).

При нажатии на кнопку отправки производится валидация на стороне клиента на предмет пустоты полей и ограничений на длину. После этого запрос на добавление отправляется серверу, но без перегрузки страницы. Если все прошло успешно, то высвечивается сообщение зеленым цветом, что запись была добавлена в базу. Если пользователь с такими именем и фамилией уже есть в базе, то высвечивается сообщение красным цветом о недопустимости добавления.

Клиент, отправляя серверу сообщения, кодирует их в URL в обычном виде. Ответ от сервера должен приходить в XML-формате.

4. Реализовать третий вариант, но ответ должен быть в формате JSON.

15.7. Содержание отчета

1. Цель работы.
2. Вариант задания.
3. Исходный текст программы на PHP.
4. Скриншоты сгенерированных страниц.
5. Выводы.

15.8. Контрольные вопросы

1. Какие значения может принимать **readyState**?
2. Какие значения может принимать **status**?
3. В каком случае предпочтительнее использовать XML, а в каком JSON в качестве формата передачи данных?
4. Сигнатура функции **json_encode** в PHP. Какие аргументы она может принимать на вход и что именно возвращать?

ЗАКЛЮЧЕНИЕ

В лабораторном практикуме рассматривались основы web-программирования: базовые конструкции языка PHP, компоненты платформы AMP, способы извлечения данных на уровне PHP, взаимодействие между сервером и клиентом с помощью GET-запросов, настройка компонентов платформы AMP.

При этом упор был сделан на практическую сторону, приведено большое количество примеров программ, упражнений, контрольных вопросов. В результате выполнения лабораторных работ студенты приобретают теоретические знания и практические навыки по web-программированию.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Харрис, Э.* PHP/MySQL для начинающих / Э. Харрис. – М.: Кудиц-Образ, 2007. – 384 с.
2. *Зандстра, Мэт*. PHP. Объекты, шаблоны и методики программирования / Мэт Зандстра. – СПб. : Вильямс, 2011. – 560 с. – ISBN 978-5-8459-1689-1.
3. *Веллинг, Л.* Разработка веб-приложений с помощью PHP и MySQL / Л. Веллинг, Л. Томсон. – СПб. : Вильямс, 2010. – 848 с. – ISBN 978-5-8459-1574-0.
4. *Никсон, Р.* Создаем динамические веб-сайты с помощью PHP, MySQL, Javascript и CSS / Р. Никсон. – СПб. : Питер, 2013. – 560 с. – ISBN 978-5-496-00187-8.
5. *Ленгсторф, Дж*. PHP и Jquery для профессионалов / Дж. Ленгсторф. – СПб. : Вильямс, 2011. – 362 с. – ISBN 978-5-8459-1693-8.
6. *Флэнаган, Д*. Javascript. Подробное руководство / Д. Флэнаган. – М. : Символ-Плюс, 2013. – 1080 с. – ISBN 978-5-93286-215-5.
7. *Бейтс, М.* CoffeeScript. Второе дыхание Javascript / М. Бейтс. – М. : ДМК-Пресс, 2016. – 310 с. – ISBN 978-5-97060-240-9.

ОГЛАВЛЕНИЕ

Учебное издание

ШЕВЧЕНКО Дмитрий Васильевич
ОЗЕРОВА Марина Игоревна

ОСНОВЫ WEB-ПРОГРАММИРОВАНИЯ

Лабораторный практикум

Редактор А. П. Володина
Технический редактор С. Ш. Абдуллаева
Корректор Е. П. Викулова
Компьютерная верстка Е. А. Кузьминой

Подписано в печать 20.06.17.
Формат 60×84/16. Усл. печ. л. 9,07. Тираж 82 экз.
Заказ
Издательство
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых.
600000, Владимир, ул. Горького, 87.