РАЗРАБОТКА ИНФОРМАЦИОННЫХ СИСТЕМ ДЛЯ СЕТИ ИНТЕРНЕТ С ИСПОЛЬЗОВАНИЕМ РНР и MySQL

ВВЕДЕНИЕ

Начиная с первых страниц методической разработки, мы познакомимся с новым языком программирования, который в настоящее время является основным средством разработки скриптов для Web-серверов. При рассмотрении примеров предполагается, что Вы уже знакомы с технологией написания *HTML*-страниц. После получения запроса от клиентского компьютера на определенный файл Web-сервер начинает данный запрос обрабатывать. Если запрашивается чисто *HTML*-документ, то обработка достаточна простая – при наличии на сервере такого файла он пересылается на компьютер пользователя и отображается в браузере. При этом браузер представляет информацию на основании *HTML-тегов,* содержащихся в файле. В результате перед пользователем на экране предстает готовый документ. В издании рассматривается другой тип файлов, которые может запрашивать клиентский компьютер, а именно файлы с расширением *РНР.* Они включают программу на языке *РНР* (такая программа может составлять часть файла либо занимать весь файл целиком). Язык *РНР* является языком серверных скриптов (сценариев), которые можно встраивать в обычный *HTML*-документ. Фактически сервер, получив запрос на файл с кодом на языке *РНР,* обрабатывает его препроцессором – программой *php.exe*. А результат обработки (который уже содержит только текстовую информацию, изображения, теги и клиентские сценарии) передается на компьютер, с которого данный файл был запрошен. Можно сказать, что пользователь получает обычный *HTML*-документ.

Теперь поясним – как все-таки препроцессор узнает о том, что в файле располагается именно код на *РНР*, который нужно обрабатывать. Условие, что файл имеет расширение *РНР,* является явно недостаточным. Основная подсказка для препроцессора заключается в том, что код *РНР* оформляется специальными тегами. Так, в начале фрагмента на *РНР* должна располагаться конструкция <?php , а после завершения фрагмента программы необходимо поместить тег ?>. В принципе существуют и другие варианты оформления кода *РНР*, но этот более распространенный и далее в тексте будем пользоваться только им. Работа препроцессора на сервере по шагам подробно может быть описана следующим образом:

* на вход препроцессора поступает файл (запрашиваемый браузером пользователя), который он начинает просматривать (более правильно – обрабатывать) его содержимое;
* если в тексте встречается тег, обозначающий начало программы (<?php ), препроцессор переходит к выполнению команд;
* если встречается закрывающий тег, обозначающий окончание кода на *РНР*, препроцессор опять переключается в режим просмотра кода (пока ему не встретится очередной участок кода на *РНР* или пока не будет достигнут конец файла).

Возможна ситуация, когда файл целиком состоит из команд на языке *РНР*, но более часто встречается вариант, кода присутствуют и *HTML*-код, и строки на *РНР*. Важное отличие от обыкновенных *HTML*-страничек заключается в том, что при использовании *РНР* – программ пользователь в своем браузере не видит исходный текст, а получает только результат обработки файла препроцессором.

# 1.Среда разработки PHP - программ

Как и при разработке знакомых *HTML*-страниц для написания *РНР-программ* можно использовать обыкновенный *Блокнот*. В данном тексте используется именно этот вариант для разработки. В принципе для небольших разработок *Блокнота* вполне достаточно. Однако следует заметить, что в настоящее время существует огромное количество редакторов, которые позволяют более удобным способом создавать и редактировать файлы *РНР*. В таблице 1 приведена информация о том, где можно найти необходимые редакторы (если *Блокнот* не устраивает) для работы.

Таблица 1

|  |  |
| --- | --- |
| Название редактора | Комментарий |
| **PHP Expert Editor** | Официальный сайт:  *http://www.ankord.com/*  Русскоязычный вариант доступен по адресу *http://www.ankord.com/ru* |
| **PHP Designer** | Официальный сайт:  *http://www.mpsoftware.dk/phpdesigner.php* |
| **NOTEPAD++** | Официальный сайт:  *http://notepad-plus.sourceforge.net*  Данный редактор можно использовать как альтернативу обычному блокноту, только в более продвинутом варианте. |

Для выполнения *РНР* - программ необходим *Web-сервер*, который мог бы принимать запросы пользователя и отправлять ему соответствующие ответы. Для того, чтобы Web-сервер выполнял скрипты, написанные именно на РНР, необходимо установить на нем РНР. Кроме того, в дальнейшем (во второй половине издания) нам потребуется работа с базой данных. В связи с этим требуется установить еще и систему управления данными.

Одним из самых распространенных *Web*-серверов является *Apache*, а одной из самых популярных систем управления базами данных является *MySql*. Именно эти два программных продукта наиболее часто используются в связке с *PHP*. Поэтому для того, чтобы приступить к работе, в первую очередь необходимо установить эти программные продукты. В таблице 2 представлены сайты, где можно найти дистрибутивы для установки *Web-сервера* с поддержкой *PHP* и *MySQL*.

Таблица 2.

|  |  |  |
| --- | --- | --- |
| Название программы | Официальный сайт (откуда можно скачать дистрибутив) | Размер дистрибутива |
| Apache (Web-сервер) | www.apache.org | 4,9 Mб |
| PHP | www.php.net | 8,9 Мб |
| MySQL | www.mysql.com  (справка на русском языке доступна на  www.mysql.com/doc/ru/index.html) | 16,9 Мб |
| phpMyAdmin | www. phpMyAdmin.net | 3,4 Мб |

Однако, использование данного варианта для начала работы не очень эффективно. Во-первых, большой объем информации. А во-вторых, сам процесс установки для не слишком опытного человека может оказаться достаточно сложным. Существует более удобный вариант для работы с указанными технологиями. Это система – *Денвер*. Так с помощью *Денвера* можно без каких-либо усилий установить *Apache, PHP, MySQL, phpAdmin* буквально за пять минут. При этом важно, что размер дистрибутива, который необходимо скачать занимает чуть более 3 Мбайт.

На официальном сайте *www.denwer.ru* система *Денвер* характеризуется как джентльменский набор Web-разработчика – набор инструментов, используемый Web-разработчиками для отладки сайтов на домашней (локальной) Windows-машине без необходимости выхода в Интернет. Важно отметить, что *Денвер* устанавливается в один (единственный) каталог и вне его ничего не изменяет. Он не пишет файлы в Windows-директорию и не прописывает никакой информации в реестре. При необходимости Вы можете поставить даже два *Денвера*, и они не будут конфликтовать. Для удаления *Денвера* достаточно просто удалить каталог, где он расположен. Также все конфигурирование и настройка под конкретную машину происходит автоматически.

Для начала необходимо скачать последнюю версию дистрибутива. Для этого можно зайти в сеть по ссылке *http://www.denwer.ru/dis/latest* и скачать предложенный файл. После перекачки файла можно приступить к его установке. Сам файл представляет собой самораспаковывающийся архив. После окончания распаковки Вам будет предложено нажать клавишу *Enter*, а затем ввести имя директории. По умолчанию предлагается директория *C:\WebServers* и с этим вариантом можно согласиться. Далее в процессе установке программа Вам предложит выбрать букву для обозначения виртуального диска. В этом случае также лучше согласить с предложенной буквой *Z*. После этого начнется процесс копирования информации, по завершению которого Вам будет предложено выбрать режим запуска Денвера:

* *автоматический* – в этом случае комплекс будет запускаться автоматически при входе в операционную систему;
* *ручной* – чтобы начать работу с комплексом необходимо будет запустить его с помощью специального ярлыка. Для завершения работы комплекса также отключение следует выполнить с помощью другого ярлыка.

Второй вариант предпочтительней – в этом случае вы можете запускать комплекс только тогда, когда он Вам нужен. После этого согласитесь с размещением ярлыков на рабочем столе. В результате установки на рабочем столе появятся три кнопки:

* *Start servers* - запуск комплекса;
* *Stop servers* - остановка работы комплекса;
* *Restart servers* - перезапуск комплекса (в процессе работы эта функция нам также потребуется).

Итак, для того, чтобы запустить *Денвер* (начать работу с Вашим персональным *Web-сервером*), необходимо воспользоваться пиктограммой *Start servers*. После этого в правой нижней части экрана появится пиктограмма в виде *пера*, обозначающая работу *Web-сервера Apache*. Важно отметить, что после запуска *Денвера* на Вашем компьютере будет создан специальный виртуальный диск, и если Вы не сменили его имя при установке Денвера, то он будет называться *Z*. Ярлык *Restart servers* осуществляет перезапуск *Web-сервера*, что необходимо, если вы совершили некоторые изменения и хотите, чтобы они вступили в силу. Например, изменениями считается создание нового хоста (по другому – добавление нового сайта).

Посмотрим, что представляет собой *Денвер*. Для этого следует открыть окно браузера и набрать *localhost*. В результате перед вами откроется окно, которое показано на рис.1. Здесь представлено несколько закладок и справочная информация. Если Вы в дальнейшем решите разместить свой сайт в Интернете, то можете обратиться по ссылкам, размещенным на основной закладке сайта. Причем на сайте *www.holm.ru* можно воспользоваться бесплатным хостингом.

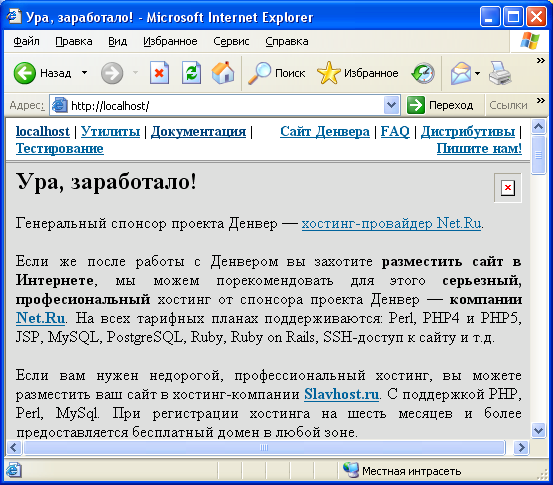


Рис.1.

Сейчас выберем пункт *Утилиты* и посмотрим – какие утилиты и инструменты включает в себя Денвер. В дальнейшем мы воспользуемся *phpMyAdmin* – системой администрирования *MySQL* через Web-интерфейс. Этот инструмент позволяет создавать базы данных. А пока для тестирования системы можно перейти на закладку *Тестирование* и выбрать один из тестов – *test1.ru* (рис.2), результат выполнения которого представлен на рис.3.

* *etc* – хранит скрипты конфигурирования Денвера, и здесь же находится его конфигурационный файл, который править не рекомендуется.
* *home* – с данной папкой вы будете работать часто, т.к. в ней будут храниться ваши сайты (виртуальные хосты).
* *tmp* – папка для хранения временной информации. В подкаталоге данной папки *!sendmail* будут храниться все письма, отправленные с помощью функции *mail().*

*usr* – системная папка, в которой хранятся *Apache, MySql* и *PHP*.

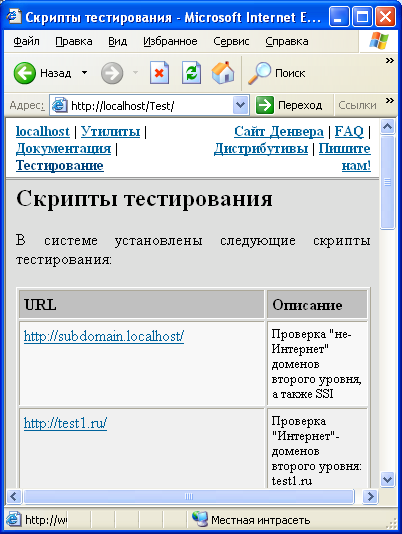


Рис.2.

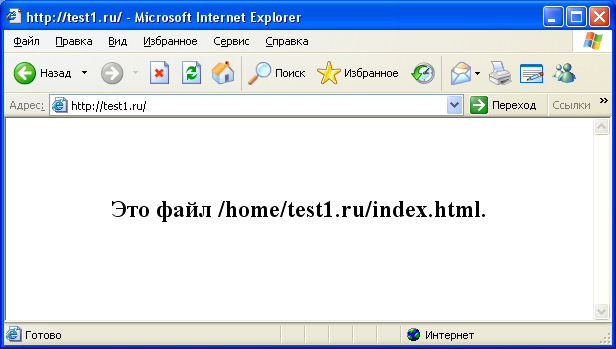
.

Рис.3.

# <https://zen.yandex.ru/media/id/5b2cd6035870da00a9db63ab/ustanovka-lokalnogo-servera-openserver-5b3a7efafc1b4f00a80a2164?utm_source=serp>

# 2. ОСНОВЫ РНР

По умолчанию, после установки *Денвера*, вам будут доступны несколько виртуальных хостов. Например, если вы наберете в браузере *www.test1.ru*, то отобразится тестовая страничка. Для того, чтобы добавить новый сайт, достаточно создать в каталоге *home* (виртуального диска *Z*) папку с именем будущего сайта, только ее имя должно быть без *www*. Например, вы хотите создавать ваши программы, работая с виртуальным сайтом *www.mysait.ru*. Для этого необходимо создать папку *mysait.ru* в каталоге home. Но кроме этого вам нужно создать еще одну папку с именем *www* (внутри папки *mysait.ru*). После этого следует перезапустить *Web-сервер* с помощью ярлыка *Restart servers*, и новый сайт готов к работе.

После создания папки *mysait.ru* следует во внутренней папке *www* разместить первую программу, которая будет включать вместе с уже знакомыми конструкциями и новые теги. Для этого воспользуемся *Блокнотом* (листинг 1).

Листинг 1. Файл index.php

<html> <head>

<title> Первая программа </title>

</head>

<body>

<?php

echo " Начало создания сайта по информационным технологиям ";

?>

</body> </html>

Теперь необходимо перезапустить *Денвер* (для регистрации в системе нового хоста), а затемв окне браузера следует обычным образом обратиться к нашему сайту (рис.4).

Хотя программа небольшая, ее необходимо прокомментировать. Часть текста занимает знакомый *HTML*-код. С тега <?php начинается участок кода на *PHP*, который завершается тегом ?>. Содержательная часть программы составляет единственную команду вывода сообщения на экран:

echo " Начало создания сайта по информационным технологиям ";

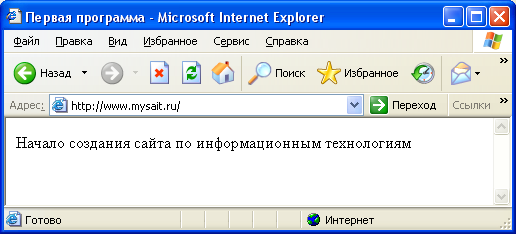


Рис.4.

Обратите внимание, что каждая команда в языке PHP заканчивается точкой с запятой. Также важно отметить, чтов строке браузера мы указали просто название сайта без информации о файле. Однако с сайта мы получили конкретный файл *index.php*. Дело в том, что по умолчанию при подобном обращении к сайту на пользовательский компьютер отправляется файл с именем *index.* Теперь немного изменим данную программу в плане использования уже знакомых тегов (листинг 2).

Листинг 2. Файл index.php (вариант 2)

<html><head>

<title> Первая программа </title>

</head><body>

<?php

echo " <B>Начало</B> <I> создания</I> <br>сайта по

информационным технологиям ";

?>

</body></html>

Здесь мы использовали теги языка разметки гипертекста, в результате чего после обращения к нашему сайту мы получим в браузере несколько другое оформление страницы.

Как и во всех языках программирования, в РНР можно использовать переменные. На листинге 3 приведена программа, дающая представление о работе с переменными. Вызов программы и результат ее выполнения в окне браузера показаны на рис.5.

Листинг 3. Файл prog1.php

<html><head>

</head><body>

<?php

$a="НГЛУ";

echo " <B> факультет</B><br>".$a." <br>

<I>Н.Новгород</I>";

?>

</body></html>

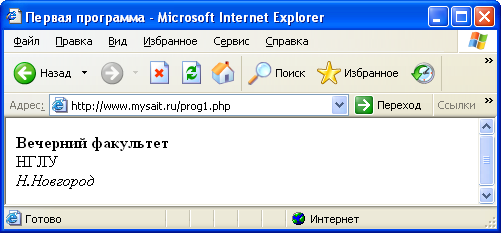


Рис.5.

В данной программе мы использовали переменную $a, которой присвоили значение (строку символов):

$a ="НГЛУ".

При определении переменой тип данных указывать необязательно – *РНР* определит его автоматически. Вообще в РНР существует 4 типа простых данных: *строковый*, *целочисленный*, *вещественный* и *логический*. В рассматриваемой программе вывод информации на экран осуществляется с использованием сложной конструкции. А именно с помощью точки происходит объединение трех конструкций.

Важно отметить, что *РНР* является регистрозависимым языком и поэтому строчные и заглавные буквы путать нельзя. Когда вы присваиваете переменной имя, то следует руководствоваться следующими правилами для исключения лишних ошибок:

* имя переменной должно содержать латинские буквы, цифры и символ подчеркивания;
* имя не должно начинаться с цифры.

Константа – это частный случай использования переменной. Она задается один раз и характеризует значение, которое будет неизменным на протяжении всей программы. Для определения константы предназначена функция define(), синтаксис которой записывается следующим образом:

define(“имя константы”,значение константы)

На листинге 4 приведен текст программы, который иллюстрирует использование констант.

Листинг 4. Файл prog2.php

<html><head>

</head><body>

<?php

define ("koeff",60);

$h=3.5;

echo " Число минут в указанном значении в часах =".$h\*koeff;

?>

</body></html>

В *РНР* используются так называемые предопределенные константы. Это константы созданные разработчиками *РНР*. Далее в тексте наиболее часто будут встречаются TRUE и FALSE, которые хранят 1 и 0 соответственно. Например, с помощью константы PHP\_VERSION вы можете увидеть номер версии *PHP*, которая используется вами. На листинге 5 приведен вариант программы с выводом сообщения о версии *PHP*, а на рис.6 продемонстрирован результат.

Листинг 5. Файл prog2.php (вариант 2)

<html> <body>

<?php

define ("koeff",60);

$h=3.5;

echo " Число минут в указанном значении в часах =".$h\*koeff."<br>";

echo PHP\_VERSION;

?>

</body></html>

Для работы с данными в PHP, как и в других языках, используются массивы. *Массив* – это набор значений, которые хранятся в определенной последовательности, а доступ к ним осуществляется с помощью специального ключа, который указывает - к какому именно значению мы хотим обратиться (какой элемент массива нам нужен). Именно ключ позволяет однозначно отделять данные друг от друга, поэтому каждый ключ массива должен быть уникальным. Ключ может быть либо текстом, либо числом.

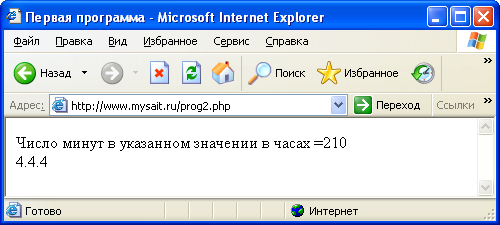


Рис.6.

Создать (объявить) массив можно несколькими способами. Так с помощью следующей конструкции происходит создание пустого массива:

$mas = array().

В данном случае используется специальное слово array с пустыми скобками. Это приведет к созданию массива, который не будет содержать никаких данных. Другой вариант заключается в последовательном перечислении элементов массива:

$mas = array(“Room1” => “Офис”, “Room1” => “Склад”,55=>100);

В подобной записи все данные перечисляются следующим образом: Ключ => Значение. В роли ключа может выступать как строка, так и число. Можно эти же элементы объявить и отдельно:

$mas [“Room1”] = “Офис”;

$mas [“Room2”] =“Склад”;

$mas [55] =100;

Здесь сначала указывается имя массива, затем в квадратных скобках его ключ, далее знак равно и значение этого элемента массива. Возможен вариант автоматической нумерации. Это можно сделать последовательным перечислением:

$mas = array(“Офис”, “Склад”,100);

То же самое можно реализовать и по-другому:

$mas [ ] = “Офис”;

$mas [ ] =“Склад”;

$mas [ ] =100;

В обоих случаях индекс первого элемента 0, второго 1, и т.д. На следующем листинге показан пример создания массива и вывода элементов в окне браузера.

Листинг 6. Файл prog3.php

<?php

$mas = array("Room1" => "Офис", "Room2" => "Склад",55=>100);

echo $mas["Room1"]."<br>";

echo $mas ["Room2"]."<br>";

echo $mas [55]."<br>";

?>

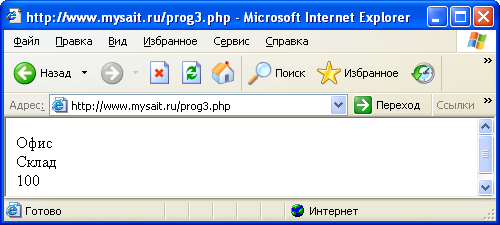


Рис.7.

У большинства сайтов размещенных в Интернете можно видеть счетчики, которые отображают число людей посетивших данную страницу. Логика работы подобной программы построена следующим образом: число посещений хранится в обычном текстовом файле (например, с именем *z.txt*). При каждом новом обращении к программе (далее мы ее приведем) это число увеличивается на единицу. Заметим, что в *РНР* с файлом можно работать в двух режимах:

* *текстовом* – означает, что вы работаете со всей информацией, содержащейся в файле, как с текстом, состоящим из строк.
* *бинарном* - подразумевается, что файл состоит из байтов данных. Этот режим предназначен для работы со всеми типами файлов, отличных от текстового (в частности, с графическими).

Перед тем как начать работу с файлом, его следует открыть, что выполняется с помощью функции fopen(), которая имеет следующий синтаксис:

fopen(имя файла, режим работы).

Данная функция возвращает специальное число, которое называется *дескриптор открытого файла*. После обращения к функции fopen()это число необходимо запомнить в переменной, чтобы в дальнейшем можно было обращаться к файлу. Фактически вся работа с файлом осуществляется через дескриптор. Скажем, для дескриптора может быть использована переменная $f, и далее мы можем работать с файлом через нее (нам не нужно постоянно указывать его имя и режим работы). Одним из параметров функции fopen является режим работы, который имеет две составляющие, об одной из которых мы говорили (текстовый или бинарный). Другую составляющую можно назвать способом работы с файлом. Именно от указания способа работы будет зависеть то, что мы можем делать с открытым файлом. Например, если он открыт для чтения, то при попытке записать в него информацию, возникнет ошибка. В таблице 3 перечислены различные способы работы с файлом. В зависимости от указания способа, приведенного в таблице, будет зависеть то, что мы сможем делать с открытым файлом. Фактически функция fopen предоставляет разработчикам разнообразные варианты работы с файловой информацией.

Таблица 3. Способы работы с файлом

|  |  |
| --- | --- |
| **Способ работы**  **с файлом** | **Описание** |
| r | Файл открывается для чтения, указатель текущей позиции помещается в начало файла. Если файл не существует, то возникает ошибка. |
| r+ | Файл открывается для чтения и записи, указатель текущей позиции помещается в начало файла. Если файл не существует, то возникает ошибка. |
| w | Создается пустой файл и открывается только для записи. Если файл с таким именем существует, то он перезаписывается. |
| w+ | Создается пустой файл и открывается для записи и для чтения. Если файл с таким именем существует, то он перезаписывается. |
| a | Файл открывается для записи, указатель текущей позиции помещается в конец файла. Если файла не существует, то он создается. |
| a+ | Файл открывается для записи и чтения, указатель текущей позиции помещается в конец файла. Если файла не существует, то он создается. |

Полная запись второго параметра данной функции выглядит следующим образом:

*СпособРаботыРежимРаботы,*

где *СпособРаботы* – это одно из значений, представленных в таблице 3. Другая составляющая (*РежимРаботы*) – обозначает один из двух вариантов:

* t– текстовый.;
* b – бинарный.

Например, a+t означает, что файл открывается для записи и чтения,а указатель помещается в конец файла. Если файла не существует, то он создается. Здесь tозначает, что работа с открытым файлом предполагается как с набором строк (т.е. как с обычным текстом). Если же указать a+b, то работать с файлом можно как с набором байтов. На листинге 7 приведена программа, выполняющая роль счетчика посещений. В плане взаимодействия с файлом z.txt она может упрощенно описана следующим образом:

* чтение значения счетчика в переменную;
* увеличение счетчика на единицу;
* запись нового значения в файл.

Листинг 7. Файл prog4.php

<?php

$f=fopen("z.txt","a+t") or die("Файл не открывается");

flock($f,2);

$s=fgets($f);

$s=$s+1;

ftruncate($f,0);

fputs($f,$s);

flock($f,3);

fclose($f);

echo $s;

?>

На рис.8 показан результат работы – на странице отображается число посещений.

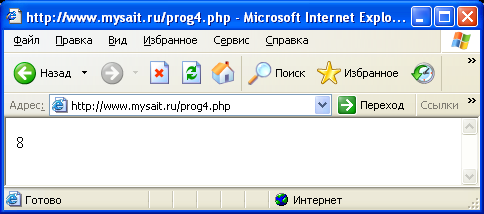


Рис.8.

В программе используется функция блокировки файла flock(). Благодаря этому, пока один пользователь работает с файлом, никто другой, кроме него не может внести в него изменения. Синтаксис функции flock() выглядит следующим образом:

flock(дескриптор файла, числовое значение).

Что касается дескриптора, то о нем мы уже говорили, а числовое значение определяет вариант блокировки. Возможны следующие ситуации (причем можно указать как числовое значение, так и его альтернативное значение в виде константы):

* 2 (или LOCK-EX) – пока один пользователь работает с файлом, никто другой не может к нему обратиться;
* 3 (или LOCK\_UN) – снятие установленной ранее блокировки.

Поясним, что представляет собой указатель текущей позиции. Его назначение подобно курсору при работе с текстовым редактором (обозначение текущей позиции в открытом файле). Если необходимо что-то записать, то запись данных осуществляется именно с места, где установлен указатель. Так же, если необходимо прочитать данные из файла, процесс чтения будет осуществлен, начиная с текущего положения указателя. В результате вся работа с файлом строится в соответствии с позицией указателя. В следующей строке программы после блокировки мы получаем данные из файла с помощью функции fgets(). Назначение ее заключается в чтении очередной строки из файла. Единственный параметр, который необходимо передать в функцию – это дескриптор файла. В нашем случае в результате выполнения функция fgets() возвратит первую (и единственную) строку файла z.txt. Это строку мы сохраним в переменой $s. Далее в программе происходит увеличение значения счетчика на единицу. После этого производится запись нового значения счетчика. Но перед этим удаляется предыдущее. Для этого используется функция ftruncate(файловый дескриптор, размер). В этой функции два параметра (первый нам знаком). Второй же определяет значение размера (в байтах) до какой границы необходимо файл обрезать. В данном случае мы указали ноль, что означает полную очистку файла. Далее производится запись нового значения счетчика в файл, а затем снятие блокировки. Работа с файлом заканчивается после выполнения функции fclose().

Разберем еще один момент, который мы упустили. В программе была использована функция die, которая имеет следующий синтаксис:

die(текст сообщения об ошибке).

Эта функция выводит на экран текст, переданный в качестве параметра, и осуществляет немедленный выход из программы. Поэтому данная функция не используется сама по себе, а применяется лишь в сочетании с другими. Так, в данном случае, если функции fopen не удалось открыть файл, то она вернет отрицательный результат (значение FALSE). И уже тогда выполнится функция Die.

Когда две функции используются рядом, строится логическое выражение, которое нужно разделить с помощью специально предназначенного для этого логического оператора. Рассматриваемая строка обрабатывается препроцессором определенным образом. Сначала выполняется функция fopen. Если она выполнена удачно и был возвращен файловый дескриптор, то выполняется следующая строка программы. Если же функции fopen не удалось открыть файл, то она вернет FALSE в качестве выходного значения. В этом случае сработает “защитный вариант” в лице функции die. В результате на экран будет выведен текст сообщения, после чего выполнение программы будет прекращено.

# 3. ФОРМЫ

С формами вы встречались много раз, когда заходили на различные сайты, т.к. они являются их обязательной частью. Вводите ли вы электронный адрес для подписки на рассылку, регистрируетесь ли на сайте – в любом случае вы используете формы.

На рис.9 показан вариант использования формы. Основное назначение формы – оживить страничку, придать ей интерактивность. По другому это означает возможность вести пользователю диалог (у посетителя запрашивается информация, которую он вводит). Например, при отправке файла появляется окно с просьбой подтвердить это действие. Важно заметить, что форма является элементом языка *HTML*, а не *РНР*. Технология *РНР* предоставляет только механизм работы. В дальнейшем по тексту мы практически в каждой разработке будем использовать формы.

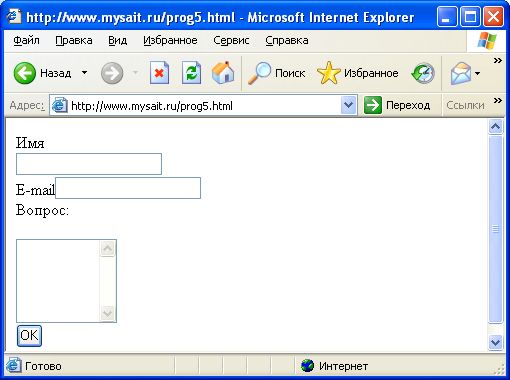


Рис.9.

Полная запись формы в тексте программы выглядит следующим образом:

*<FORM*

*NAME=”ИМЯ ФОРМЫ”*

*ACTION=”ИМЯ ПРОГРАММЫ ОБРАБОТЧИКА ДАННОЙ ФОРМЫ”*

*METHOD=”МЕТОД ПЕРЕДАЧИ ДАННЫХ”>*

*Элементы формы*

*</FORM>*

Рассмотрим назначение данных параметров формы. Что касается *NAME*, то это ключевой параметр, позволяющий идентифицировать конкретную форму. *ACTION* указывает имя программы (скрипта), которому будут отправлены данные, имеющиеся на форме. Заметим, что отправка формы – это обычный запрос к серверу, только вместе с запросом передается еще и дополнительная информация – данные формы. Если параметр *ACTION* не указать, то данные будут переданы текущей программе (самой себе). Параметр *METHOD* определяет способ отправки формы на сервер. Этих способов два – *GET* и *POST*, различия между ними мы рассмотрим позже, а пока вернемся к рис.9. Если пользователь внес информацию в текстовые поля и нажал на кнопку *ОК*, то к указанному в разделе *ACTION* скрипту произойдет обращение на сервере.

Основное назначение формы заключается в том, чтобы быть контейнером для различных элементов (подобных показанным на рис.9). Наиболее распространенным является *тестовое окно* (его также часто называют *поле ввода*). На форме этот элемент объявляется следующим образом:

*<INPUT type=”text” name=”Имя элемента”*

*value=”Значение элемента” title=”Всплывающая подсказка”*

*size=длина >*

Параметр *type* как раз и определяет тип элемента. В данном случае указание ”*text*” говорит именно о текстовом элементе. Параметр *name* – имя элемента. Когда информация будет отправляться на сервер, то данные именно от этого элемента передаются в виде:

*Имя элемента = Значение элемента.*

Менее важный параметр – это *title* (всплывающая подсказка, которая будет отображаться при наведении мыши), а последний (*size*) определяет длину поля. Все параметры за исключением *type* и *name* являются необязательными. Например, вместо записи

*<INPUT type=”text” name=”client”*

*value=”Здесь необходимо ввести свою фамилию”*

*Title=”Поле для ввода имени” Size=40 >*

можно использовать сокращенный вариант:

*<INPUT type=”text” name=”client” >*

Этот второй вариант заметно короче, но менее понятен пользователю. Перейдем теперь к другому элементу управления – к *кнопке* с помощью, которой данные отправляются на сервер. Программно *кнопка* создается следующим образом:

*<INPUT type=”submit” Name=”имя элемента”*

*Value=”текст на кнопке” Title=”Всплывающая подсказка”*

*Size=40 >*

Для того, чтобы другая программа могла получить данные формы можно использовать один из трех массивов:

* $\_GET – применяется, когда данные передаются методом GET;
* $\_POST – применяется, когда данные передаются методом POST;
* $\_REQUEST – применяется как альтернативный вариант двум предыдущим, т.к. заменяет их, не важно каким методом они передавались, они все равно будут доступны в этом массиве.

Имя элемента массива, в котором хранятся данные об имени пользователя такое же, как имя элемента формы, в котором было введено это само значение. Рассмотрим технические действия иллюстрирующие передачу информации на примере (рис.10). Для этого нам необходимо создать файл с prog6.html (именно с расширением html). Текст файла приведен на листинге 8.

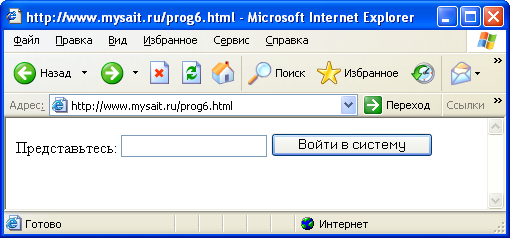


Рис.10.

Листинг 8. Файл prog6.html

<HTML><BODY>

<FORM name="my\_form" ACTION = "prog6.php">

Представьтесь:

<INPUT TYPE = "TEXT" NAME = "name\_of\_user"

value=""

title="Поле для ввода имени">

<input

type="submit" value="Войти в систему"

name="enter\_system"

title="Нажмите, если Вы ввели имя, чтобы увидеть

результат">

</form>

</BODY></HTML>

Далее нам потребуется создать файл prog6.php, который обрабатывает действия формы (листинг 9).

Листинг 9. Файл prog6.php

<html><body>

<?php

echo "Привет ".$\_GET['name\_of\_user'];

?>

</body></html>

В нашем случае для имени поля ввода было использовано name\_of\_user. В связи с этим значение, которое ввел пользователь можно получить с помощью:

$\_GET['name\_of\_user']

Заметим, что если метод отправки данных не указан, то они передаются методом GET. На рис. 11 и 12 показан результат взаимодействия рассмотренных программ.

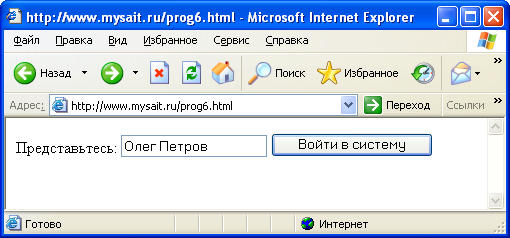


Рис.11.

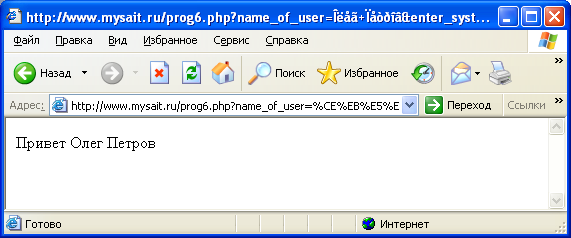


Рис.12.

Поговорим теперь о методах отправки информации представленной на форме. Как было сказано, форма может быть отправлена двумя методами - *GET* и *POST*. Метод *GET* означает, что данные будут передаваться присоединенными к имени программы обработчика (имя указано в параметре *action* формы), то есть формат для передачи будет следующий:

*название программы в разделе action плюс данные.*

Чтобы сервер мог данные отличить, они присоединяются по специальным правилам. Сначала после имени программы-обработчика ставится знак вопроса (?), что указывает передачу данных. После этого указывается имя первого объявленного элемента на форме (в нашем случае это текстовое окно), далее знак равенства и значение. Если элемент один, то формирование запроса считается оконченным. После этого он отправляется на сервер. Если же элементов несколько, то информация о следующем отделяется от предыдущего знаком &. Формат записи очередного элемента строится по описанной схеме: имя элемента, знак равно, значение элемента.

Метод *GET* удобен тем, что в этом случае вам видна сформированная браузером строка запроса для передачи на сервер. Таким образом, вы можете напрямую передавать обработчику информацию, минуя форму. Для этого достаточно сформировать вручную в браузере строку по описанному ранее правилу. Стоит отметить, что русские буквы передаются в закодированном виде, так что вы не увидите русских букв – они будут заменены кодами.

Поясним теперь в чем отличительная особенность метода *POST*. Так, при использовании метода *POST* данные отправляются незаметно для пользователя в теле запроса. Сам запрос будет состоять из трех частей:

* запрос к программе-обработчику, указанному в параметре *action* формы;
* сообщение о том, какой объем данных будет передаваться;
* передача самих данных.

# 4. ОБЗОР ЭЛЕМЕНТОВ ФОРМЫ

Ранее мы рассмотрели текстовое окно и кнопку, но ими ассортимент элементов не исчерпывается. Здесь мы приведем примеры использования других элементов в программе. Начнем с поля для ввода пароля, которое объявляется точно также как и поле ввода. Только в параметре *type* необходимо указать значение *password*. Например, следующая запись:

*<INPUT type=”password”*

*name=”pass\_user”title=”Поле для ввода имени” size=40 >*

создает поле для ввода пароля (все вводимые символы будут выглядеть как жирные точки). Изменим теперь несколько рассмотренные ранее файлы с целью продемонстрировать работу с данным элементом. Стартовый файл *prog7.html* приведен на листинге 10.

Листинг 10. Файл prog7.html.

<HTML> <BODY>

<FORM name="my\_form" ACTION = "prog7.php">

Введите пароль:

<INPUT TYPE = "password" NAME = "name\_of\_user"

value="" title="Поле для ввода пароля">

< INPUT TYPE ="submit" value="Войти в систему"

name="enter\_system">

</form> </BODY></HTML>

В этом случае вводимый пользователем текст скрыт от взгляда посторонних, а отправляемый серверу введенный пароль проверяется на соответствие истинному. Файл, обрабатывающий данный запрос представлен на листинге 11.

Листинг 11. Файл prog7.php (второй вариант)

<html><body>

<?php

if ($\_GET['name\_of\_user'] == "123")

echo "Привет ";

else

echo "Ошибка ";

?>

</body></html>

Результат работы данных модулей продемонстрирован на рис.13 и 14.

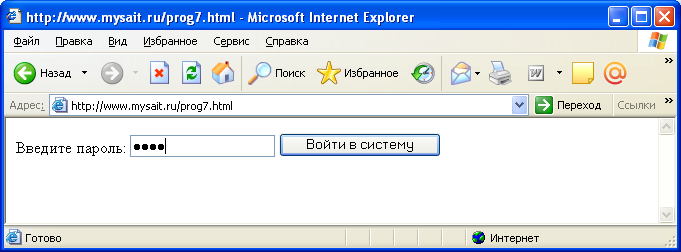


Рис.13.

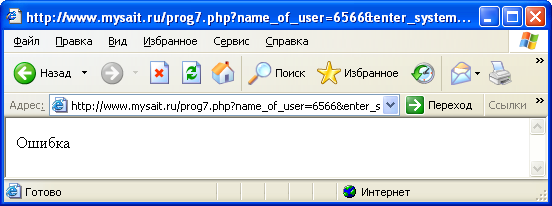


Рис.14.

Перейдем к другим элементам управления – к переключателям. Эти элементы используются для выбора одного варианта из нескольких заранее предопределенных. На сайтах переключатели используются очень часто и обеспечивают для пользователя очень удобный ввод данных отправляемых на сервер. Для создания переключателя используется запись:

*<INPUT type=”ratio” Name=”name1”*

*Value=”Значение” checked >Текст для переключателя*

Поясним параметры данного элемента:

* *type* – тип элемента, в данном случае *ratio*, что означает создание переключателя;
* *name* – имя элемента (данный параметр используется для создания группы переключателей);
* *checked* – является необязательным параметром, может указываться только для одного элемента из группы (элементы одной группы должны иметь одинаковое имя).

Под текстом для переключателя подразумевается любой текст, который располагается рядом с ним. Например, группу из двух переключателей можно организовать с помощью записи следующего вида:

*<INPUT type=”ratio” Name=”group1” Value=”Yes” checked>Да*

*<INPUT type=”ratio” Name=”group1” Value=”No” >Нет*

Данные от переключателей в программу обработки формы будут передаваться в виде *Name = Value*. Для каждой группы переключателей передается одна такая конструкция (т.к. у группы имя одинаковое). Рассмотрим конкретный пример. Цель заключается в том, что оформить страницу в виде показанном на рис.15. В этом случае от пользователя требуется внести свое имя и выбрать один из двух предлагаемых вариантов. На сервер после щелчка по кнопке *ОК* отправится запрос с информацией о конкретном включенном переключателе.

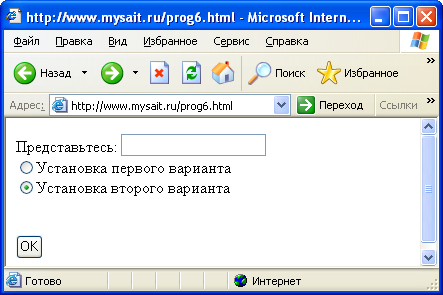


Рис.15.

Файл, соответствующий данной странице, приведен на листинге 12, а скрипт обрабатывающий форму ввода, на листинге 13.

Листинг 12. Файл prog8.html

<html><body>

<FORM name="my\_form" ACTION = "prog8.php">

Представьтесь:

<input type = "text" NAME = "name\_of\_user"

value="" title="Поле для ввода имени">

<br>

<input type="radio" Name="Variant" Value="1"

>Установка первого варианта

<br><input type="radio" Name="Variant" Value="2"

checked>Установка второго варианта

<br><br><br>

<input type="submit" value="OK"

name="enter\_system"

title="Нажмите, если Вы ввели имя, чтобы увидеть

результат">

</form>

</body></html>

Сама по себе форма достаточно несложная. В ней текстовое окно несет информацию о фамилии пользователя заполняющего форму, а группа переключателей отражает выбранный вариант. Скрипт (листинг 13) обрабатывающий форму сначала проверяет является ли он следствием щелчка по кнопке на форме. Если да, то далее открывается файл и в него записывается фамилия, которую пользователь ввел с клавиатуры. Кроме того, записывается значение того переключателя, который выбран пользователем из группы. Если в дальнейшем открыть файл zzz.txt, то мы увидим информацию подобную показанной на рис.16.

Листинг 13. Файл prog8.php

<?php

if ($\_GET['enter\_system']=="OK")

{ echo "Привет ".$\_GET['name\_of\_user'];

$f=fopen("zzz.txt","a+t") or die("Файл не открывается");

flock($f,2);

fputs($f,$\_GET['name\_of\_user']);

fputs($f," ");

fputs($f,$\_GET['Variant']);

fputs($f," ");

flock($f,3);

fclose($f);

}

else

echo "Ошибка ";

?>

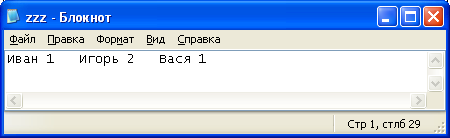


Рис.16.

Следующий элемент управления – этого список, который существует в трех вариациях:

* в виде выпадающего списка;
* в виде списка с фиксированным числом видимых элементов, когда элементов больше заданного числа (в этом случае доступен вертикальный скроллинг);
* в виде списка с фиксированным числом видимых элементов и возможностью выбора нескольких элементов (при помощи удержания клавиши Ctrl).

Все описанные виды списков объявляются с помощью тега <select>, а какой вариант будет доступен, зависит от указанных вами параметров. Схема создания списка:

<select

size=”количество видимых элементов”

name=”имя списка”

[multiple]>

<option value=”Значение 1” [selected]>

Отображаемая строка 1 </option>

…

<option value=”Значение N” [selected]>

Отображаемая строка N </option>

</select>

В теге <select> задаются параметры списка, далее указываются его элементы, каждый из которых объявляется парным тегом <option></option>. После того, как все элементы списка объявлены, ставится завершающий тег </select>. Параметр size задает количество отображаемых в списке строк. Если он равен 1, то получается выпадающий вариант списка. Если будет задан необязательный параметр multiple, то получится список с возможностью множественного выбора. Но при выпадающем списке данный параметр не работает. Если при объявлении элемента указывается необязательный параметр selected, то это будет означать, что данный элемент будет выбранным. Для множественного выбора это можно использовать несколько раз.

Рассмотрим пример организации формы ввода с использованием списка. На листинге 14 приведен файл prog9.htmlс формой ввода, а на листинге 15 приведен скрипт, который ее обрабатывает.

Листинг 14. Файл prog9.html содержащий форму ввода

<HTML> <BODY>

<FORM name="my\_form" ACTION = "prog9.php">

Фамилия: <INPUT TYPE = "text" NAME = "name\_of\_user"

value="" title="Поле для ввода фамилии">

<br>

<SELECT size="1" Name="spisok1">

<option value= "Оценка 5 "> Отлично </option>

<option value= "Оценка 4 "> Хорошо </option>

<option value= "Оценка 3 "> Удовлетворительно </option>

<option value= "Оценка 2 "> Неудовлетворительно </option>

</SELECT>

<br>

<input type="submit" value="OK" name="enter\_system"

title="Нажмите, если Вы ввели имя, чтобы увидеть результат">

</form>

</BODY></HTML>

Результат отображения файла prog9.html в окне браузера приведен на рис.17.

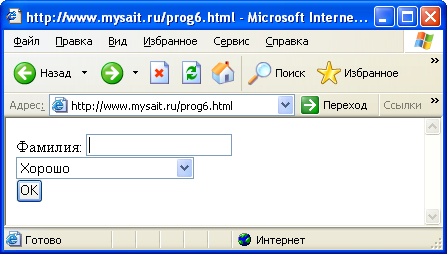


Рис.17.

Данный пример отражает реальную ситуацию, связанную с внесением в текстовый файл фамилии учащегося и оценки. Для этого скрипт, обрабатывающий данные, введенные пользователем должен быть оформлен так как показано на листинге 15.

Листинг 15. Файл prog9.php

<?php

if ($\_GET['enter\_system']=="OK")

{ echo "Внесена оценка студента с фамилией - ".$\_GET['name\_of\_user'];

$f=fopen("zzzzz.txt","a+t") or die("Файл не открывается");

flock($f,2);

fputs($f,$\_GET['name\_of\_user']);

fputs($f," ");

fputs($f,$\_GET['spisok1']);

fputs($f,"\n");

flock($f,3);

fclose($f);

}

else

echo "Ошибка ";

?>

Еще один элемент – поле многостраничного текста, который объявляется следующим образом:

<textarea Name=”Имя элемента” Rows= высота элемента

Cols=длина элемента> Текст, который будет содержать элемент </textarea>

Для удобства восприятия параметров приведем пример создания поля ввода многостраничного текста:

<textarea Name=”new\_element” Rows=10 Cols=40>

Список Элементов

Элемент 1

Элемент 2

Элемент 3

</textarea>

На рис.18 показано отображение рассматриваемого элемента на странице в окне браузера.

И последний элемент управления в данном разделе – скрытое поле. Данный элемент нельзя увидеть и его назначение заключается в передаче данных, которые не должны изменяться пользователем. Например, этот элемент часто используют при реализации скрипта по загрузке файлов на сервер. В этом случае создается скрытое поле, содержащее максимально возможный размер файла для загрузки. Таким образом, можно проверить соответствие условию еще перед тем, как начнется процесс загрузки файла.

Скрытое поле создается следующим образом:

<input name=”имя” type=”hidden” value=”значение”>

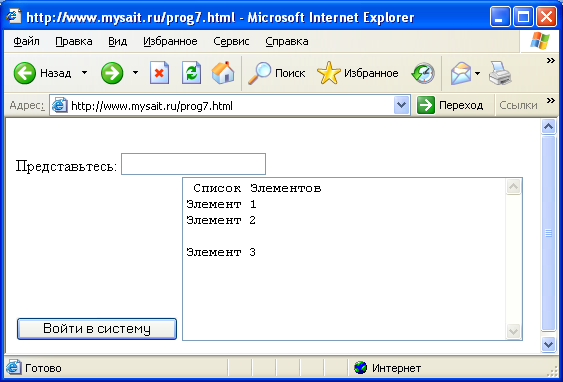


Рис.18.

# 5. ФОРМА ОБРАТНОЙ СВЯЗИ

Форма обратной связи представляет простейший механизм общения посетителя сайта с его администратором. На рис.19 представлено изображение в окне браузера, а на листинге 16 соответствующий текст.

Листинг 16. Файл prog10.html для формирования формы обратной связи

<form action="obr.php" method="post">

Имя <br> <input type="text" name="person"><br>

E-mail<input type="text" name="email">

<br> Вопрос:<br> <br>

<textarea name="q" cols=10 rows=5>

</textarea> <br>

<input type="submit" name="okb" value="OK">

</form>

Для обработки данной формы следует использовать скрипт, показанный на листинге 17.

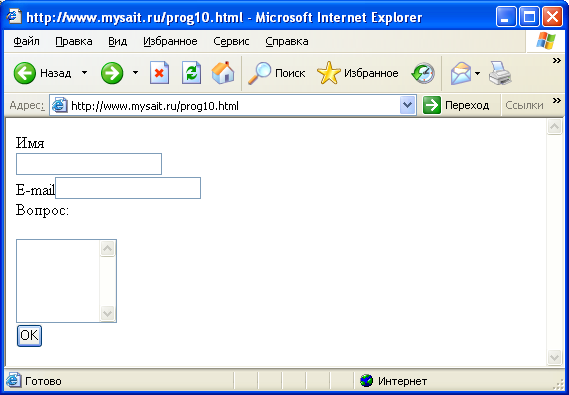


Рис.19.

Листинг 17. Файл obr.php для обработки формы обратной связи

<?php

if (isset($\_POST['okb']))

{

echo 'Получатель - '.$\_POST['person'];

$komu="cashaev@rambler.ru";

$tema="Вопрос от ".$\_POST['person']." ".$\_POST['email'];

$text\_pisma=$\_POST['q'];

$z=mail($komu,$tema,$text\_pisma);

}

?>

Здесь в строке if (isset($\_POST['okb'])) используется конструкция для проверки условия – есть элемент с именем okb в масииве $\_POST. Для этого используется функция isset возвращающая значение TRUE или FALSE. Зачем нужна такая проверка? Дело в том, что просто никто не мешает запустить скрипт obr.php напрямую. Тогда программа будет работать неправильно – у нее не будет необходимых данных (фамилии, адреса, сообщения). Далее производится отправка электронного письма с данными, которые ввел пользователь. Для этого используется стандартная функция mail(), и ее синтаксис выглядит так:

mail(адрес получателя, тема, сообщение);

Если отправка успешна, то функция вернет TRUE, в противном случае FALSE. Описание ее параметров:

* адрес получателя – в программе для этого отводится переменная $komu;
* тема – тема письма, соответственно используется переменная $tema;
* сообщение – текст письма ($text\_pisma).

В Денвере в качестве функции mail() используется отладочная заглушка, которая не отправляет письма, а складывает их в директорию tmp/!sendmail. Мы до этого рассматривали ситуации, когда форма и скрипт, ее обрабатывающий, размещаются в разных файлах. Приведем пример (листинг 18) в котором реализован вариант обработки в одном файле.

Листинг 18. Файл obrabotka.php

<?php

if (isset($\_POST['okb']))

{ if ($\_POST['person']=='' or $\_POST['email']=='' or $\_POST['q']=='')

{ echo "<font color='red'>Вы ввели не все данные</font>";

echo "<br><a href=obrabotka.php>назад</a>";

exit; }

echo 'Оправлено. Получатель - '.$\_POST['person'];

$komu="cashaev@rambler.ru";

$tema="Вопрос от".$\_POST['person']." ".$\_POST['email'];

$text\_pisma=$\_POST['q'];

$z=mail($komu,$tema,$text\_pisma); }

?>

<form action="obrabotka.php" method="post">

Имя <br>

<input type="text" name="person">

<br> E-mail<input type="text" name="email">

<br> Вопрос:<br>

<br><textarea name="q" cols=10 rows=5>

</textarea>

<br>

<input type="submit" name="okb" value="OK">

</form>

# 6. ГОСТЕВАЯ КНИГА

Это один из самых удобных способов оставить свое мнение на сайте. Разрабатываемая далее гостевая книга будет состоять из двух скриптов:

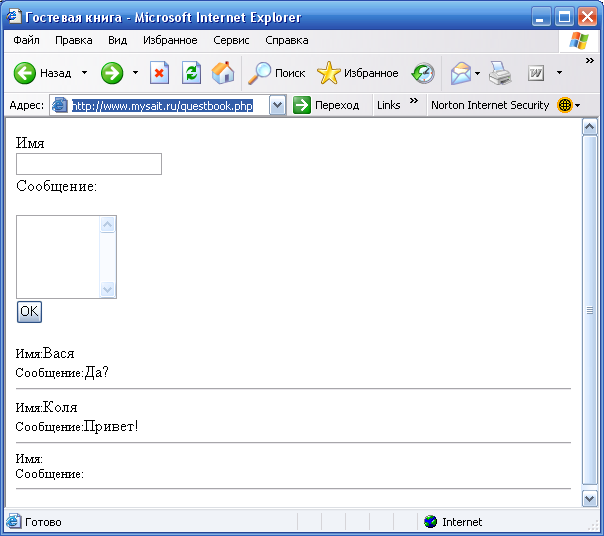
* questbook.php – осуществляет вывод формы, (рис.20), в которой будут выводиться все сообщения.
* add\_message.php – обработчик формы, который осуществляет добавление нового сообщения в гостевую книгу. 

Рис.20.

Теперь о формате записей посетителей, который будет иметь следующий вид:

* разделитель;
* имя посетителя;
* текст сообщения.

Постановка задачи ясна и теперь можно перейти к кодированию. Далее на листинге 19 приведен текст файла questbook.php.

Листинг 19. Файл questbook.php

<html><head>

<title> Гостевая книга </title></head>

<body>

<form action="add\_message.php" method="post">

Имя <br>

<input type="text" name="name\_of\_quest">

<br>Сообщение:<br>

<br>

<textarea name="message\_of\_quest" cols=10 rows=5>

</textarea>

<br><input type="submit" name="okbutton" value="OK">

</form>

<?php

$f=fopen("gost.txt","rt") or die("Не могу открыть файл");

while (!feof($f))

{

$data=fgets($f); // Получаем фамилию

echo "<small>Имя:</small>".$data."<br>";

$data=fgets($f); // Получаем сообщение

echo "<small>Сообщение:</small>".$data."<br>";

echo "<hr>";

}

fclose($f);

?>

Прокомментируем приведенный текст. Так в начале файла создается форма с полем ввода name\_of\_quest, полем ввода для многостраничного текста message\_of\_quest и кнопкой для добавления сообщения с именем okbutton. Данные будут передаваться методом POST и обрабатываться программой add\_message.php. В программе используется уже знакомая вам функция fopen() для открытия файла (этот файл мы создадим далее). В данном случае файл открывается как текстовый и только в режиме чтения. В операторе условия используется новая функция feof(), которая определяет, достигнут ли указатель окончания файла. Если да, то функция возвращает true, в противном случае false. Принцип работы функции прост: каждый раз, когда что-то читается из файла, указатель текущей позиции смещается ровно на столько, сколько информации из файла было прочитано. Для обозначения отрицания используется восклицательный знак. В цикле происходит чтение всех сообщений из файла gost.txt. Так как одно сообщение состоит из двух строк, то для вывода одного сообщения нужно прочитать эти три строки (разделитель на экран выводить не будем- вместо этого в программе будем использовать тег для отображения линии).

Перед тем как запустить файл questbook.php, необходимо создать текстовый файл для сообщений, о котором говорилось выше. В противном случае вы увидите сообщение об ошибке. Поэтому в блокноте создайте файл – вначале он будет пустым.

Теперь создадим скрипт, обрабатывающий информацию, полученную от формы.

Листинг 20. Файл add\_message.php

<?php

if (isset($\_POST['okbutton']))

{

if ($\_POST['name\_of\_quest']=='')

exit("Введите имя <a href=’questbook.php’>Назад!</a>");

if ($\_POST['message\_of\_quest']=='')

exit("Введите сообщение <a href='questbook.php'>Назад!</a>");

$f=fopen("gost.txt","at") or die("Не могу открыть файл");

flock($f,2);

fputs($f,$\_POST['name\_of\_quest']."\n");

fputs($f,$\_POST['message\_of\_quest']."\n");

flock($f,3);

fclose($f);

}

header('location:questbook.php');

?>

Сначала производится проверка, каким образом был запущен скрипт: если как обработчик формы, то сообщение будет добавлено, в противном случае пользователь сразу перенаправляется на скрипт questbook.php. Для этого используется функция header(), ее синтаксис выглядит следующим образом:

header(заголовок).

Данная функция отправляет браузеру пользователя заголовок, переданный в качестве параметра. Важно заметить, что до использования header()должна использоваться до вывода пользователю каких-либо сообщений.

**7. БАЗЫ ДАННЫХ**

В этом разделе мы рассмотрим технологию работы с базами данных. Для этого используется специальная программа, которая называется системой управления базами данных. В среде Web-программирования наибольшую распространенность получила *MySql*. Официальный сайт *MySql* http://www.mysql.com. Русскоязычная документация доступна по адресу www.mysql.com/doc/ru/index.html. У *MySql* нет удобного графического интерфейса, поэтому для работы с этой системой чаще всего используется специальный инструмент, который позволяет максимально упростить этот процесс. Его название — phpMyAdmin, официальный сайт — [www.phpmyadmin.net](http://www.phpmyadmin.net), а русскоязычный сайт доступен по адресу <http://www.php-myadmin.ru>.

Для того, чтобы воспользоваться этим инструментом в системе *Денвер* необходимо сначала набрать в строке браузера localhost, а затем выбрать пункт phpMyAdmin. В результате Вы увидите окно, которое представлено на рис. 21. С помощью phpMyAdmin можно делать практически все, начиная от создания базы данных и заканчивая формированием списка пользователей для доступа к ней. Рассмотрим технологию разработки несложной базы данных. Для создания базы данных необходимо в поле Создать новую БД ввести название будущей базы данных (для выполнения дальнейшей разработки выберем название – Firma) и нажать кнопку Создать. После этого база будет создана, а в левой части phpMyAdmin вы можете видеть (рис.22) ее имя. В центре экрана также можно наблюдать SQL-запрос, который характеризует только что выполненное действие. Стоит отметить, что язык SQL является связующим звеном между разработчиком и СУБД, т.к. последняя понимает именно этот язык. Поэтому, чтобы совершить какое-либо действие, необходимо составить SQL-запрос и отправить его СУБД. Система phpMyAdmin берет на себя эту заботу, предоставляя разработчику визуальный интерфейс работы для работы с MySQL.

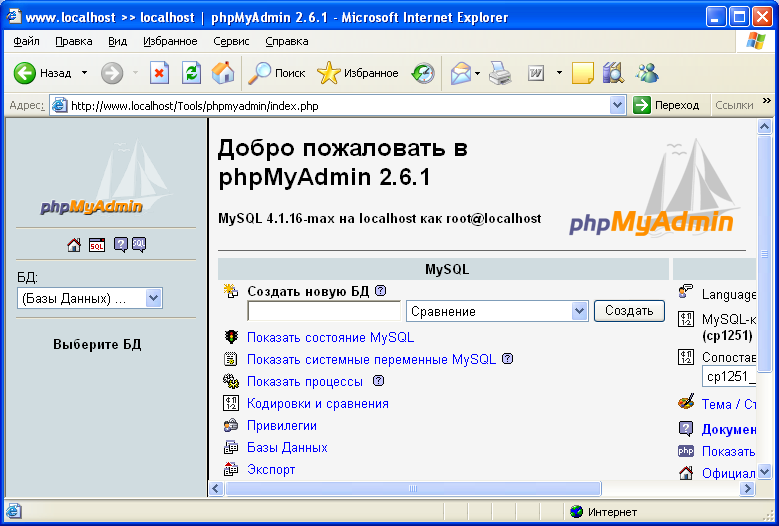


Рис.21.

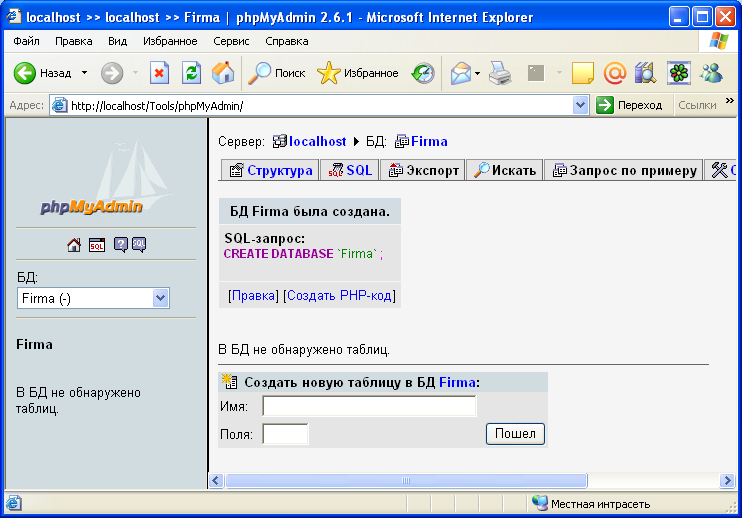


Рис.22.

Теперь начнем работу по разработке новой базы данных и для начала создадим таблицу USERS. Для этого в поле Имя расположенном под надписью Создать новую таблицу введите USERS, а в поле Поля число 3. После этого следует нажать кнопку Пошел и теперь перед Вами откроется окно показанное на рис.23, где присутствует набор из секций, каждая из которых состоит из однотипных элементов. Для первой строки в секцию Поле введите id, а в поле Тип выберите INT – это тип данных для столбца.

*Примечание.*

Приведем наиболее используемые типы данных:

* VARCHAR – текстовый тип, предназначенный для хранения до 255

символов;

* TEXT – текстовый тип данных, предназначенный для хранения до

65535 символов;

* INT – числовой тип;
* DATE - предназначенный для хранения даты.

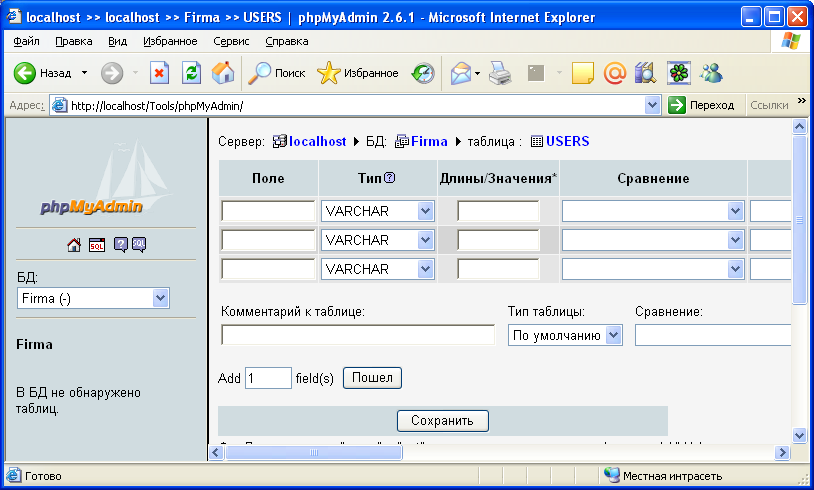


Рис.23.

Вернемся к разработке таблицы и добавим столбец name типа VARCHAR с максимально возможным количеством символов, равным 20. Это поле предназначено для фамилии пользователя. Также добавим столбец pass типа VARCHAR с максимальным количеством символов, равным 32 (это нужно, т.к. в нем будет храниться хеш пользовательского пароля, выдаваемый функцией md5(), а она возвращает 32-значное число).

Теперь настроим таблицу USERS. Сначала сделаем так, чтобы столбцу id значения присваивались автоматически. Для этого прокрутите окно браузера с помощью горизонтального скроллинга до выпадающего списка Дополнительно. В нем необходимо выбрать единственный доступный пункт auto\_incement. Также следует установить переключатель с изображением ключа (primary key или первичный ключ), назначение которого – указать системе управления базой данных, что значения столбца будут уникальными. Теперь структура таблицы создана и необходимо нажать кнопку Сохранить. На рис.24 показано окно, которое открывается после этого. В центре окна располагается текст запроса для создания таблицы.

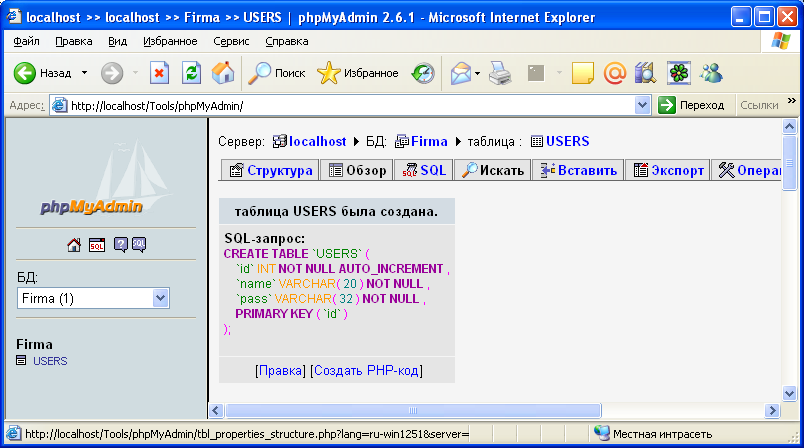


Рис.24.

Теперь можно добавить две записи для двух пользователей. Для этого щелкните левой кнопкой мыши на вкладке Вставить, расположенной в верхней части окна phpMyAdmin. Заполните значения в соответствие с рис.25.

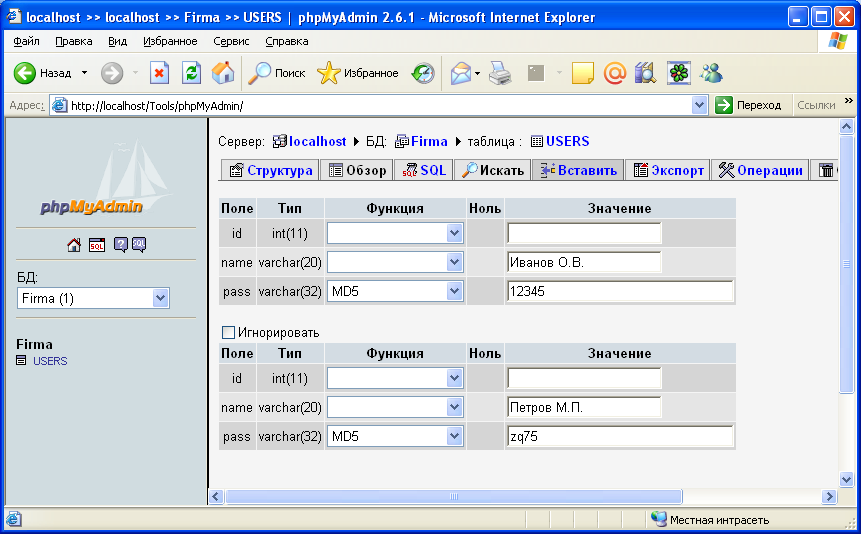


Рис.25.

Как было указано пароль не хранится в открытом виде. Вместо этого используется его хеш, полученный как результат функции md5(). И тут *phpMyAdmin* предоставляет нам удобный способ сделать это – в разделе Функция выберите в выпадающем списке, расположенном напротив слова pass, пункт MD5(). В результате в поле pass будет записано значение полученное после выполнения функции md5() с параметром 12345, т.е. хеш от введенного вами значения, в чем вы убедитесь чуть позже. После ввода двух записей нажмите на кнопку Пошел. Теперь если воспользоваться кнопкой Обзор, то можно убедиться, что записи добавлены (рис.26). Аналогичным образом создадим таблицу Scladi. В таблице установим 2 поля – id (для кода) и naz (для названия), после чего внесем информацию о двух складах (рис.27). Третьей таблицей в нашей базе будет Nomenclatura, которая предназначена для товаров (рис.28). Поле Tovar отводится для названия товара, а поле idScl для кода склада, где данный товар располагается. Поле Col будем использовать для фиксации количества единиц определенного товара на конкретном складе.

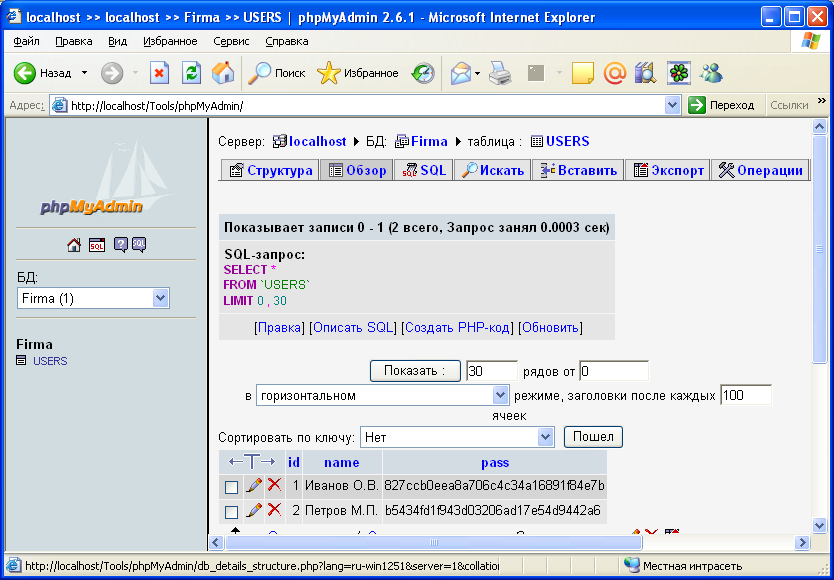


Рис.26.

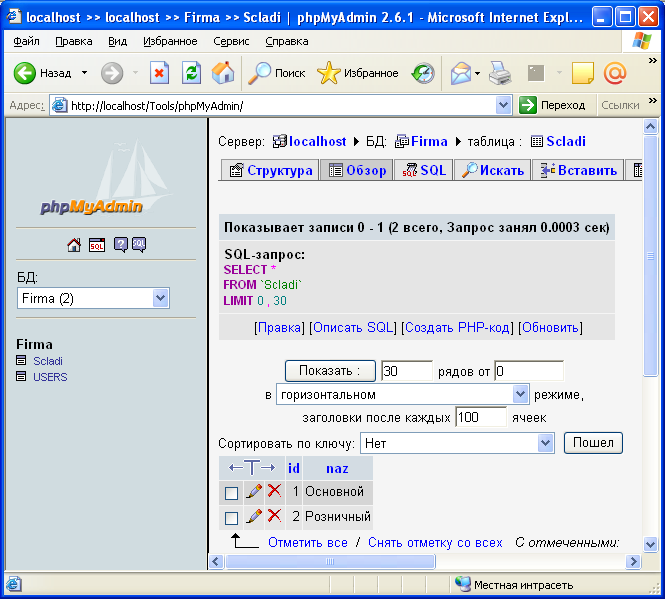


Рис.27.

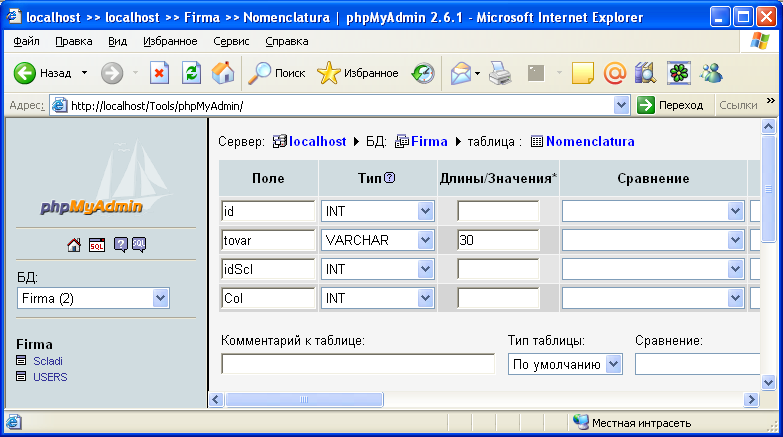


Рис.28.

Внесите в таблицу несколько товаров (к указанному на рис.29 добавьте еще несколько).

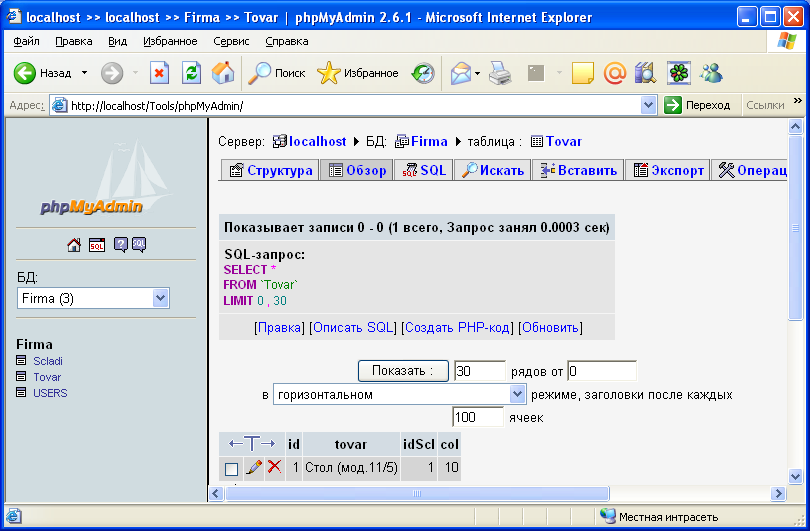


Рис.29.

Рассмотренная математическая модель, конечно, является достаточно упрощенной по сравнению с реальной ситуацией. Однако наша задача на этом примере продемонстрировать программные ресурсы для работы с базами данных. Теперь перейдем к написанию программного кода, но сначала нам потребуется создать новый виртуальный хост. Для этого в папке home добавьте папку sclad.ru, а уже там создайте еще одну папку под именем www. Перед дальнейшей работой перезапустим *Денвер* – в этом случае созданный хост будет зарегистрирован в системе. В дальнейшем все разрабатываемые файлы необходимо сохранять в папке sclad.ru\www.

Начнем с подключения к базе данных – это первое необходимое действие, которое мы должны выполнить. Для этого используется следующая функция:

mysql\_connect($sqlhost,$sqluser,$sqlpass)

Здесь $sqlhost – имя хоста, в качестве которого чаще всего указывается localhost, т.к. обычно MySQL и РНР находятся на одном и том же сервере. Параметр $sqluser представляет имя пользователя, под которым будет осуществлено подключение к серверу, а $sqlpass – пароль. Если произвести подключение не удалось, то функция вернет FALSE, поэтому очень часто mysql\_connect() используется следующим образом:

mysql\_connect($sqlhost,$sqluser,$sqlpass) or die("MySQL не доступен! ".mysql\_error()).

После того, как подключение к базе данных MySQL осуществлено, необходимо выбрать базу данных, с которой будет производиться работа. Для этого используется функция:

mysql\_db (database\_name),

где database\_name – имя базы данных.

Если mysql\_db была выполнена успешно, то эта функция вернет TRUE, а в противном случае FALSE. На листинге 21 приведен файл, позволяющий осуществить подключение к базе данных.

Листинг 21. Файл connect.php

<?php

$sqlhost="localhost";

$sqluser="root";

$sqlpass="";

$db="Firma";

mysql\_connect($sqlhost,$sqluser,$sqlpass) or die("MySQL не доступен! ".mysql\_error());

mysql\_select\_db($db) or die("Нет соединения с базой данных".mysql\_error());

?>

Для *Денвера* имя хоста будет localhost, имя пользователя – root, а пароль будет пустым. Этот файл следует использовать на различных страницах при подключении к базе данных. Начнем с первой страницы index.php , которая будет выполнять функцию авторизации. На рис.30 показана эта страница в окне браузера. Все элементы, которые Вы видите, составляют форму. От пользователя требуется правильно ввести пароль после выбора в верхнем списке своей фамилии. На листинге 22 приведен текст файла index.php.

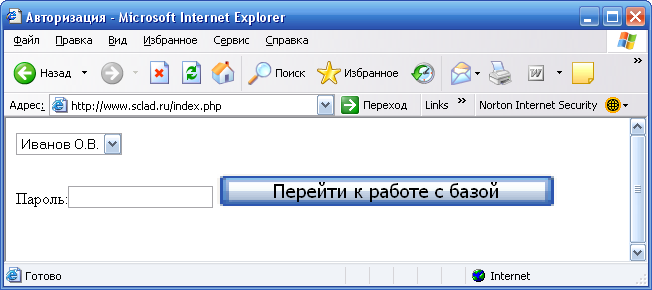


Рис.30.

Листинг 22. Файл index.php

<html><head><title> Авторизация </title>

<?php

require\_once('connect.php');

$sql="SELECT name FROM USERS ORDER BY name";

$result=mysql\_query($sql);

?>

<form action="start.php" method="post">

<select name="z1">

<?php

while ($line=mysql\_fetch\_row($result))

{ echo "<option value='".$line[0]."'>$line[0]";

}

?>

</select><br><br>

Пароль:<input type='password' name='pass1'>

<input name='enterStart' STYLE="font-size: 14pt; font-family: Tahoma, sans-serif"

type='submit' value='Перейти к работе с базой'>

</form>

Теперь прокомментируем строки. В начале процедуры используется функция require\_once, которая позволяет выполнить скрипт имя которого указано в качестве параметра:

require\_once('connect.php').

Это обеспечивает подключение к базе данных. Далее формируется текст запроса на языке SQL:

$sql="SELECT name FROM USERS ORDER BY name".

В следующей строке программы производится выполнение запроса для чего используется функция mysql\_query. Текст запроса передается ей в качестве параметра. Фактически мы с помощью данного запроса выбрали из базы данных список пользователей. Далее в программе начинается формирование формы для ввода информации. Первый элемент формы представляет собой список. Для его заполнения была использована функция

$line=mysql\_fetch\_row($result),

которая позволяет извлечь очередную строку из результата выполненного запроса. В данном случае мы получаем очередную фамилию пользователя, которая отражена в элементе массива $line[0]. Кроме списка на форме мы расположили поле для ввода пароля и кнопку для отправки результата ввода информации скрипту start.php. Данный скрипт осуществляет проверку пользователя. А именно в нем производится соответствие информации введенной пользователем и имеющейся в базе данных (пароли мы ранее установили). На рис.31 показан результат ошибочного указания пароля. В этом случае пользователю предлагается вернуться на стартовую страницу.

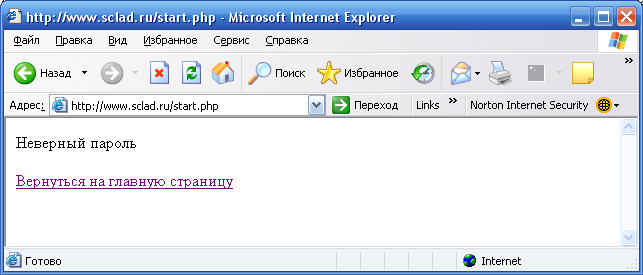


Рис.31.

На рис.32 приведена реакция программы на правильный ввод пароля. Непосредственно сам текст скрипта start.php реализущего авторизацию приведен на листинге 23.

Листинг 23. Файл start.php

<?php

if ( isset($\_POST['enterStart']))

// Проверка - является ли скрипт результатом ввода формы

{

require\_once('connect.php');

$par=md5($\_POST['pass1']);

$sql="SELECT name,pass FROM USERS

WHERE name='".$\_POST['z1']."' and pass='".$par."'";

$res=mysql\_query($sql);

if(!mysql\_num\_rows($res))

die('Неверный пароль <br><a href="index.php"><br>

Вернуться на главную страницу </a>');

else

{ echo "Авторизация прошла успешно!!!";

echo "<form action='pr.php' method='post'>";

echo "<input name='add1' type='submit'

value='Перейти к работе'>";

echo "</form>";

}

}

else

{

echo '<a href="index.php">Вернуться на главную страницу </a>';

}

?>

В начале программы располагается проверка – является ли скрипт результатом выполнения файла index.php. Это обеспечивает некоторую защиту. Далее производится запрос к базе данных. При этом используется функция mysql\_num\_rows(), которая в качестве своего значения возвращает число записей извлеченных из базы данных в результате выполнения запроса. И если ни одной записи найдено не было, то следовательно пароль ошибочный. В противном случае перед пользователем открывается кнопка (рис.32), которая позволяет перейти к скрипту pr.php.

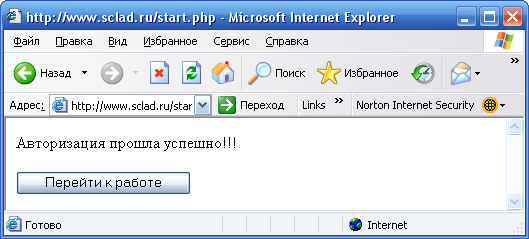


Рис.32.

На рис.33 приведена страница (в окне браузера), являющаяся результатом выполнения скрипта pr.php. В этом случае перед нами открываются имеющаяся информация и элементы для ввода данных новой позиции номенклатуры. Текст данного скрипта приведен на листинге 24.

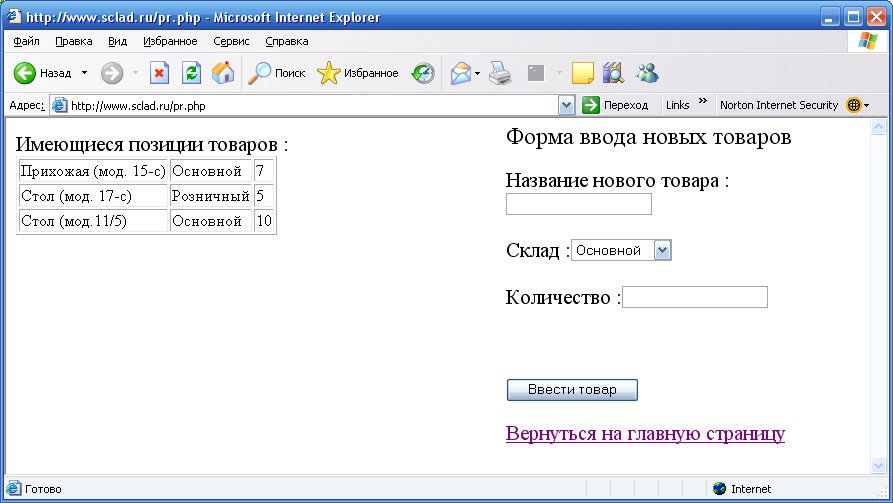


Рис.33.

Листинг 24. Файл pr.php

<?php

require\_once('connect.php');

if (isset($\_POST['enterNew']))

// Если это результат обработки формы данного файла

{

// Выбор индекса склада, где располагается новый товар

$sqlScl="SELECT id,naz FROM Scladi

WHERE naz='".$\_POST['SclNewTovar']."'";

$dataScl=mysql\_query($sqlScl);

$lineScl=mysql\_fetch\_row($dataScl);

// Запись нового товара

$zx=$\_POST['NewCol'];

$sqlNewTov="INSERT INTO Tovar SET tovar='"

.$\_POST['NewTovar']."',idScl=".$lineScl[0].",col=".$zx;

mysql\_query($sqlNewTov) or die(mysql\_error());

}

$sql="SELECT tovar,idScl,col FROM Tovar ORDER BY tovar";

$data=mysql\_query($sql);

?>

<font style="font-size:16pt">

Имеющиеся позиции товаров :</font>

<table border=1>

<?php

while ($line=mysql\_fetch\_row($data))

{

?>

<tr><td>

<?php

echo $line[0];

?>

</td><td>

<?php

$sql2="SELECT id,naz FROM Scladi WHERE id=".$line[1];

$data2=mysql\_query($sql2);

$line2=mysql\_fetch\_row($data2);

echo $line2[1];

?>

</td>

<td>

<?php

echo $line[2];

?>

</td></tr>

<?php

}

?>

</table>

<div style="position:absolute;top:5;left:500">

<font size="+2"> Форма ввода новых товаров</font>

<form method='post' action=''>

<font style="font-size:16pt">

Название нового товара :<input type='text'

name='NewTovar' value=''><br>

<br>

<?php

$sql3="SELECT id,naz FROM Scladi";

$result3=mysql\_query($sql3);

echo 'Склад :<select name="SclNewTovar">';

while ($line3=mysql\_fetch\_row($result3))

{ echo "<option value='".$line3[1]."'>$line3[1]";

}

?>

</select><br><br>

Количество :<input type='text' name='NewCol' value=''><br>

<br><br>

<br><input name='enterNew'

type='submit' value='Ввести товар'>

</form>

<a href="index.php">Вернуться на главную страницу </a>

</div>

В рассматриваемом скрипте, с одной стороны, производится отображение имеющихся товаров с указанием склада и количества. А с другой стороны расположенная в правой части страницы форма позволяет пользователю вводить информацию о новых товарах. Для свойства name кнонки формы установлено значение enterNew, а начальная часть скрипта выполняется именно при щелчке по кнопке. В этом случае в базу данных записывается информация о товаре. Общая часть скрипта позволяет отображать информацию о товарах в таблице. На рис.34 приведен результат ввод новой позиции номенклатуры.

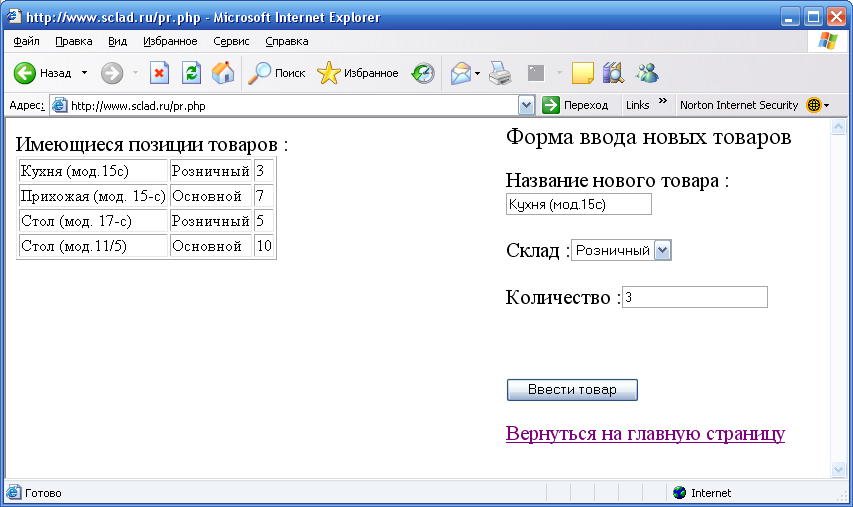


Рис.34.