МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ

ХАРЬКОВСКИЙ НАЦИОНАЛЬНЫЙ
УНИВЕРСИТЕТ РАДИОЭЛЕКТРОНИКИ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам

по дисциплине

"INTERNET ТЕХНОЛОГИИ"

2 часть

для студентов дневной и заочной форм обучения

направления 6.050102 — компьютерная инженерия

Утверждено кафедрой ЭВМ

протоколом №1 от 30.08.2013

Харьков 2013

Методические указания к лабораторным работам по дисциплине «INTERNET - технологии» (2 часть) для студентов дневной и заочной форм обучения направления 6.050102 — «Компьютерная инженерия» / Сост.: Лебёдкина А.Ю., Саранча С.Н. – Харьков: ХНУРЭ, 2013. – 123 с.

Составители: Лебёдкина А.Ю.,
Саранча С.Н.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Второй семестр дисциплины «INTERNET-технологии» посвящен задачам реализации многоуровневых асинхронных интернет-приложений с насыщенным, функциональным интерфейсом, поддержкой безопасности передаваемых данных и независимым доступом к ним. В рамках цикла лабораторных работ будут рассмотрены вопросы проверки корректности данных и безопасности web приложений, применения расширения объектов данных PDO для обеспечения абстракции доступа к базе данных, программных методов и протоколов веб-приложения, используемых для хранения данных в веб-браузере, освоения механизмов шаблонизации Smarty, а также изучения методов передачи данных с применением технологии AJAX и протокола передачи данных WebSockets.

1 ПРОГРАММНЫЕ СРЕДСТВА ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ ВХОДНЫХ ДАННЫХ

1.1 Цель работы

Исследование клиентских и серверных программных средств для обеспечения безопасности вводимых пользователем данных с целью построения надежных и устойчивых интернет-приложений.

1.2 Методические указания по организации самостоятельной работы студентов

При подготовке к выполнению лабораторной работы следует повторить способы создания формы с использованием стандарта HTML5 и CSS3 [1], ознакомиться с синтаксисом регулярных выражений, общими принципами валидации данных на клиентской и серверной сторонах, а также работы с DOM API HTML5 для валидации данных [2], и изучить программные методы хеширования и шифрования данных [3,4].

1.3 Программное обеспечение ПК

При выполнении лабораторной работы используется ПЭВМ под управлением операционной системы Windows XP и старше, веб-сервер xampp 2.4.4, библиотека phpseclib, среда разработки Notepad++ 6.4.5.

1.4 Методические указания по выполнению лабораторной работы

Наряду с общими вопросами безопасности, уязвимости в Web приложениях возникают в основном из-за недостаточной проверки вводимых данных и применения ненадёжных методов хэширования, реализуемых на уровне приложений.

С целью реализации надежных и устойчивых web-приложений разработчику рекомендуется выполнить ряд проверок:

– определение степени доверия к данным;
– типизация всех не доверенных данных;
– валидация всех типизированных не доверенных данных;
– фильтрация, экранирование и санитаризация данных;
– хеширование и/или шифрование данных.

1.4.1 Валидация данных

Регулярные выражения дают вам возможность построить шаблоны, используя набор специальных символов; эти шаблоны затем могут сопоставляться с текстом из файла, введенной в приложение датой или данными из формы, заполненной пользователем на сайте. В зависимости от того, есть ли совпадение, или нет, принимается соответствующее действие и вызывается соответственный программный код.

Например, одно из самых распространенных приложений регулярных выражений – проверка того, верен или нет введенным в форму пользователем адрес электронной почты; если да, форма будет принята, а если нет, появится сообщение, просящее пользователя исправить ошибку. Здесь регулярные выражения играют большую роль в процедурах интернет-приложений, принимающих решение, хотя, как вы увидите позже, они также могут иметь огромный эффект в сложных операциях поиска и замены.

Синтаксис регулярных выражений

Простейшее регулярное выражение можно записать так: ***"abc".***

Это выражение соответствует любой строке, которая содержит подстроку "abc".

Существует такое понятие, как выражение в квадратных скобках. Квадратные скобки ограничивают поиск теми символами, которые в них заключены: ***"[abc]".***

Этому регулярному выражению соответствует любая строка, содержащая abc либо вместе, либо каждый из них в отдельности.

Допустим, нам нужно создать регулярное выражение, соответствующее всем буквам русского алфавита. В этом случае можно, конечно, перечислить все эти буквы в регулярном выражении. Это допустимо, но утомительно и неэлегантно. Более коротко такое регулярное выражение можно записать следующим образом: ***"[а-Я]".***

Это выражение соответствует всем буквам русского алфавита, поскольку любые два символа, разделяемые дефисом, задают соответствие диапазону символов, находящихся между ними. Заметьте, что регулярное выражение "[а-Я]" описывает символы как нижнего, так и верхнего регистров, поэтому более подробно это выражение можно записать так: ***"[а-яА-Я]"***.

Точно таким же образом задаются регулярные выражения, соответствующие числам: ***"[0-9]"*** или ***"[0123456789]".***

Оба этих выражения эквивалентны и соответствует любой цифре.

При создании регулярных выражений часто удобно пользоваться групповым символом точки ".", который объединяет два одиночных символа, за исключением символа \n. К примеру, ***".ок"***. Это выражение, в частности соответствует строкам "кок", "док", "ток".

Выражение ***"x.[0-9]"*** соответствует строке, содержащей символ x, за которым идет любой другой символ и цифры от 0 до 9. Этому критерию, к примеру, удовлетворяют строки "xy1", "xz2" и т. д.

В регулярном выражении может быть несколько ветвей, которые разделяются символом |, действующим как оператор OR (ИЛИ). Т. е., если в выражении используются ветви, то для соответствия регулярного выражения какой-либо строке, достаточно, чтобы только одна из ветвей соответствовала этой строке: ***"abc|абв".***

Этому регулярному выражению соответствует любая строка, содержащая подстроки "abc" или "абв". Ветвление удобно применять при проверке расширений и имен файлов, зон доменных имен и т. д. К примеру, следующее регулярное выражение проверяет, содержатся ли в строке подстроки "ru", "com" или "net": "ru|com|net"

Для исключения последовательности символов из поиска перед этой ней ставится символ "^": ***"[^а-я]"***.

Это регулярное выражение отвечает любому символу, не содержащемуся в диапазоне а-я. Обратите внимание, что символ ^ находится внутри квадратных скобок, так как только в этом случае он имеет значение "не". При использовании символа ^ вне квадратных скобок, он имеет совсем иное значение, о чем пойдет речь немного позже.

Регулярное выражение можно уточнить при помощи квалификаторов - так называются символы +, ?, *. Квалификаторы говорят о том, сколько раз последовательность символов может встретиться в строке и указываются непосредственно после той части выражения, к которой они применяются:

– ***"а+"*** - хотя бы один а (строки "абв" и "абва" соответствуют этому выражению, а строка "укр" - нет);

– ***"а?"*** - ноль или один а (строки "абв" и "укр" соответствуют этому выражению, а строка "абва" - нет);

– ***"а*"*** - ноль или более а (строки "абв" и "абва" и "укр" соответствуют этому выражению).

Границы - это числа в фигурных скобках, указывающие количество вхождений в строку фрагмента выражения, непосредственно предшествующего границе:

– ***"ху{2}"*** соответствует строке, в которой за х следует два у;

– ***"ху{2,}"*** соответствует строке, в которой за х следует не менее двух у (может быть и больше);

– ***"ху{2,6}"*** соответствует строке, в которой за х следует от двух до шести у.

Для указания количества вхождений не одного символа, а их последовательности, используются круглые скобки:

– ***"х(yz){2,6}"*** соответствует строке, в которой за х следует от двух до шести последовательностей yz;

– ***"х(yz)*"*** соответствует строке, в которой за х следует ноль и более последовательностей yz;

Иногда бывает удобно создавать регулярное выражение таким образом, чтобы можно было, к примеру, сказать, что, по крайней мере, за одной из строк "морская", следует точно строка "волна". Для этого регулярное выражение разбивают на подвыражения с помощью круглых скобок: ***(морская)*волна.*** Это выражение соответствует строкам "волна", "морская волна", "морская морская волна" и т.д.

В регулярном выражении можно указать, должно ли конкретное подвыражение встречаться в начале, в конце строки или и в начале и в конце строки. Символ ^ соответствует началу строки: ***"^xz"***

Такое выражение соответствует любой строке, начинающейся с xz. Обратите внимание, что в этом случае символ ^ ставится за пределами выражения в скобках, к примеру: ***"^[a-z]"***

Знак доллара $ соответствует концу строки: ***"xz$"***

Это регулярное выражение соответствует любой строке, заканчивающейся на xz.

Валидация формы со стороны браузера (HTML5).

Основной атрибут тега <input>, позволяющий задавать различные элементы формы – type. Для каждого элемента существует свой список атрибутов, которые определяют его вид и характеристики. В HTML5 добавлено еще более десятка новых элементов.

Кроме того, до появления стандарта HTML5, при использовании формы на вашем сайте, вы должны были пропускать введенный текст через **JavaScript** для проверки. Атрибут pattern указывает регулярное выражение, согласно которому требуется вводить и проверять данные в поле формы. Если присутствует атрибут pattern, то форма не будет отправляться, пока поле не будет заполнено правильно.

<input type="text" title="По крайней мере восемь символов, содержащих хотя бы одну цифру, один символ нижнего и верхнего регистра" required pattern="(?=.*[a-z A-Z0-9]){8,}" />

Используйте глобальный атрибут title для дополнения встроенной ошибки своим описанием шаблона для помощи пользователю. Правда форма вывода атрибута title отличается от браузера к браузера.

Ниже приведены глобальные атрибуты событий, которые могут быть вставлены в элементы HTML5 для определения событий действия. События в таблице 1.1 выполняются по действиям внутри HTML form.

Таблица 1.1 – События формы

Атрибут	Статус	Описание
onblur		Скрипт выполняется, когда элемент теряет фокус
onchange		Скрипт выполняется, когда элемент изменился
oncontextmenu	Новый	Скрипт выполняется, когда контекстное меню срабатывает
onfocus		Скрипт выполняется, когда элемент получает фокус
onformchange	Новый	Скрипт выполняется, когда форма изменена
onforminput	Новый	Скрипт выполняется, когда форма получает пользовательский ввод
oninput	Новый	Скрипт выполняется, когда элемент получает пользовательский ввод
oninvalid	Новый	Скрипт выполняется, когда элемент недействителен
onreset	Не поддерживается в HTML5	Скрипт выполняется, когда форма сбрасывается
onselect		Скрипт выполняется, когда элемент выбран
onsubmit		Скрипт выполняется при отправке формы

API и DOM являются фундаментальными частями спецификации HTML5. С помощью нового DOM API можно обращаться к элементам валидации:

– **willValidate** true, если форма будет валидироваться:

```
<div id="one"></div>
<input type="text" id="two" />
<input type="text" id="three" disabled />
<script>
 document.getElementById('one').willValidate; //false
```

```
 document.getElementById('two').willValidate; //true
 document.getElementById('three').willValidate; //false
</script>
```

– **customError** true, если установлена кастомная ошибка:

```
<input id="foo" />
<input id="bar" />
<script>
 document.getElementById('foo').validity.customError; //false
 document.getElementById('bar').setCustomValidity('Invalid');
 document.getElementById('bar').validity.customError; //true
</script>
```

– **patternMismatch** true, если value не совпадает с паттерном:

```
<input id="foo" pattern="[0-9]{4}" value="1234" />
<input id="bar" pattern="[0-9]{4}" value="ABCD" />
<script>
 document.getElementById('foo').validity.patternMismatch; //false
 document.getElementById('bar').validity.patternMismatch; //true
</script>
```

– **rangeOverflow** true, если value больше чем max:

```
<input id="foo" type="number" max="2" value="1" />
<input id="bar" type="number" max="2" value="3" />
<script>
 document.getElementById('foo').validity.rangeOverflow; //false
 document.getElementById('bar').validity.rangeOverflow; //true
</script>
```

– **rangeUnderflow** true, если value меньше чем max:

```
<input id="foo" type="number" min="2" value="3" />
<input id="bar" type="number" min="2" value="1" />
<script>
```

document.getElementById('foo').validity.rangeUnderflow; //false

document.getElementById('bar').validity.rangeUnderflow; //true

</script>

– **stepMismatch** true, если value не совпадает с каждым step

<input id="foo" type="number" step="2" value="4" />

<input id="bar" type="number" step="2" value="3" />

<script>

document.getElementById('foo').validity.stepMismatch; //false

document.getElementById('bar').validity.stepMismatch; //true

</script>

– **tooLong** true, если символов в value больше чем указано в maxlength

<input id="foo" type="text" maxlength="1" value="A" />

<input id="bar" type="text" maxlength="1" value="AB" />

<script>

document.getElementById('foo').validity.tooLong; //false

document.getElementById('bar').validity.tooLong; //true в Opera, false в других поддерживаемых браузерах.

</script>

– **typeMismatch** true, если value не валидно для атрибута type

<input id="foo" type="url" value="http://foo.com" />

<input id="bar" type="url" value="foo" />

<script>

document.getElementById('foo').validity.typeMismatch; //false

document.getElementById('bar').validity.typeMismatch; //true

</script>

– **valueMissing** true, если элемент имеет атрибут require, но не имеет значение в value

<input id="foo" type="text" required value="foo" />

<input id="bar" type="text" required value="" />

```
<script>
 document.getElementById('foo').validity.valueMissing; //false
 document.getElementById('bar').validity.valueMissing; //true
</script>
```

– **valid** true, если элемент валиден по всем параметрам

```
<input id="valid-1" type="text" required value="foo" />
<input id="valid-2" type="text" required value="" />
<script>
 document.getElementById('valid-1').validity.valid; //true
 document.getElementById('valid-2').validity.valid; //false
</script>
```

Проверить валидацию всей формы можно так:

```
<form id="form-1">
 <input id="input-1" type="text" required />
</form>
<form id="form-2">
 <input id="input-2" type="text" />
</form>
<script>
 document.getElementById('form-1').checkValidity(); //false
 document.getElementById('input-1').checkValidity(); //false
 document.getElementById('form-2').checkValidity(); //true
 document.getElementById('input-2').checkValidity(); //true
</script>
```

Сообщение валидации можно посмотреть так:

```
<input type="text" id="foo" required />
<script>
```

```
document.getElementById('foo').validationMessage;
//Chrome --> 'Please fill out this field.'
//Firefox --> 'Please fill out this field.'
//Safari --> 'value missing'
//IE10 --> 'This is a required field.'
//Opera --> ''
</script>
```

Установить пользовательское сообщение об ошибке можно так:

```
<input type="password" id= "pass1" name="pass1" pattern="\w{6,}" required autofocus onchange="this.setCustomValidity(this.validity.patternMismatch ? 'Password must contain at least 6 characters' : ''); "/>
```

или

```
if (input.willValidate )
{
        if(input.checkValidity())
        {       document.getElementById('login').setCustomValidity('Login ('  + document.getElementById('login').value + ')  must contain at least 7 characters ');
else { document.getElementById('login').setCustomValidity('');}
```

Обработка данных на клиентской стороне.

Объект String в JavaScript дает набор методов, которые поддерживают регулярные выражения. Первый из них – это метод **search()**, используемый для поиска строки для соответствия определенному регулярному выражению. Посмотрите на следующий пример:

```
<script language="JavaScript">
// определяем строку для поиска
var str = "The Matrix";
```

```
// определяем шаблон поиска
var pattern = /trinity/;
// ищем и возвращаем результат
if(str.search(pattern) == -1)
{
  alert("Тринити не в Матрице");
} else
{
  alert("Трините в Матрице на символе " + str.search(pattern));
}
</script>
```

Метод search() возвращает позицию подстроки, соответствующую регулярному выражению или -1 в случае отсутствия такого соответствия. В нашем примере видно, что шаблона «trinity» в строке «The Matrix» нет, поэтому мы и получаем сообщение об ошибке. Если изменить регулярное выражение var pattern = /tri/, то результат поиска будет положительным.

Объект String также предоставляет метод **match()**, который может расцениваться, как близкий родственник метода search(). В чем же разница? Метод search() возвращает позицию, где находится соответствие. Метод match() работает немного по-другому: он применяет шаблон к строке и возвращает массив найденных значений.

```
<script language="JavaScript">
// определяем строку
var str = "Mississippi";
// определяем шаблон
var pattern = /is./;
// проверяем на вхождение
// помещаем результат в массив
```

```
var result = str.match(pattern);
// display matches
for(i = 0; i < result.length; i++)
{ alert("Соответствие #" + (i+1) + ": " + result[i]);}
</script>
```

Просмотрите этот пример в браузере, и вы получите сообщение, показывающее первый результат соответствия. Вот такой:

Соответствие #1: iss

В этом примере было задано регулярное выражение «is.». Он найдет строку «is», за которой следует любой символ (оператор «.» в конце шаблона находит все что угодно в строке). Если вы посмотрите на строку, в которой мы производили поиск, вы увидите, что в ней есть два вхождения этого шаблона. Для получения всех вхождений искомой подстроки необходимо добавить модификатор модификатор «g» (для поиска одного и более вхождений) в шаблон.

```
// определяем шаблон и глобальный модификатор
var pattern = /is./g;
```

Добавленный модификатор «g» обеспечивает нахождение всех вхождений шаблона в строку и сохранение в возвращаемый массив. Далее будут рассмотрены и другие модификаторы.

Предыдущий набор примеров демонстрировал возможности поиска объекта String. Вы также можете осуществлять операции поиска и замены с помощью метод **replace()**, который принимает регулярное выражение и значение для его замены. Вот так:

```
<script language="JavaScript">
// определяем строку
var str = "Welcome to the Matrix, Mr. Anderson";
// заменяем одну строку на другую
str = str.replace(/Anderson/,"Smith");
```

```
// показываем новую строку
alert(str);
</script>
```

Результатом выполнения скрипта является замена подстроки «Anderson» была строкой «Smith».

Модификатор «g» может использоваться для замены нескольких вхождений шаблона в строку:

```
<script>
var str = "yo ho ho and a bottle of gum";
// возвращает "yoo hoo hoo and a bottle of gum"
alert(str.replace(/o\s/g, "oo "));
</script>
```

Здесь мета-символ «\s» обозначает пробелы после «yo» и «ho» и заменяет на «oo».

Также вы можете использовать нечувствительный к регистру поиск по шаблону – просто добавьте модификатор «i» в конце шаблона.

Объект String также предоставляет метод **split()**, который может быть использован для разделения одной строки на отдельные части на основе особого значения разделения. Эти части затем помещаются в массив для дальнейшей обработки. Демонстрирует это следующий пример:

```
<script language="JavaScript">
// определяем строку
var friends = "Joey, Rachel, Monica, Chandler, Ross, Phoebe";
// разделяем на части с помощью запятых
var arr = friends.split(", ");
</script>
```

В JavaScript версии 1.1 и ниже, вы можете использовать только строковые значения в качестве разделителей. JavaScript 1.2 меняет все это, теперь вы можете разделять строки даже на основе регулярных выражений.

Чтобы лучше это понять, рассмотрим следующую строку, которая демонстрирует распространенную проблему: неравное количество пробелов между значениями разделения:

Neo | Trinity |Morpheus | Smith| Tank

Здесь символ «|» используется для разделения различных имен. И количество пробелов между разными «|» разное — это означает, что прежде, чем вы сможете использовать разные элементы строки, вы вынуждены удалить лишние пробелы вокруг них. Разделение с использованием регулярного выражения в качестве разделителя является элегантным решением этой проблемы, что мы и видим на следующем примере:

```
// определяем строку
var str = "Neo| Trinity  |Morpheus  |  Smith|  Tank";
// определяем шаблон
var pattern = /\s*|\s*/;
// разделяем строку с помощью регулярного выражения в
// качестве разделителя
result = str.split(pattern);
```

Результатом работы этого кода будет массив, содержащий имена, без всякого удаления пробелов.

Итак, все примеры в этой статье связаны с объектом String для демонстрации мощи реализации регулярных выражений в JavaScript. Но JavaScript также предоставляет базовый объект Regular Expression, смысл существования которого — поиск по шаблону в строках и переменных.

Этот объект имеет три полезных метода. Вот они:

– test() — проверяет строку на вхождение по шаблону;

– exec() — возвращает массив найденных вхождений в строке, позволяя расширенную работу с регулярными выражениями;

– compile() — после того, как регулярное выражение связано с объектом Regular Expression.

Рассмотрим простой пример:

```
<script language="JavaScript">
// определяем строку
var str = "The Matrix";
// создаем объект RegExp
var character = new RegExp("tri");
// ищем по шаблону в строке
if(character.test(str)) {
   alert("User located in The Matrix.");
} else {
   alert("Sorry, user is not in The Matrix.");
}
</script>
```

Это похоже на один из самых первых примеров этой статье. Тем не менее, как вы видите, он имеет совершенно другую реализацию.

Основное отличие находится в том, что создается объект Regular Expression для поиска с помощью регулярного выражения. Он создается с помощью ключевого слова «new», следующего за конструктором объекта. По определению, конструктор принимает два параметра: шаблон для поиска и модификаторы, если они имеют место быть (в этом примере их нет).

Следующим шагом после создания объекта, является его использование. Здесь мы использовали метод **test()** для поиска вхождения по шаблону. По умолчанию, этот метод

принимает строковую переменную и сравнивает её с шаблоном, переданным в конструктор объекта. В случае нахождения соответствия, он возвращает true, в противном же случае false. Очевидно, что это более логичная реализация, чем использование метода search() объекта String.

Поведение метода **exec()** похоже на то, что делает метод match() объекта String. Посмотрим:

```
<script language="JavaScript">
// определяем строку
var place = "Mississippi";
// указываем шаблон
var obj = /is./;
// ищем вхождение, помещаем результат в массив
result = obj.exec(place);
</script>
```

Метод exec() возвращает соответствие указанному регулярному выражению, если такое имеется, как массив. Вы можете обратиться к первому элементу массива, чтобы получить найденную подстроку, а также её расположение с помощью метода index().

Главное различие между методами match() и exec() в передаваемых параметрах. Первый требует шаблон в качестве аргумента, второй же требует строку для проверки.

И это ещё не все. У метода exec() есть возможность продолжить поиск по строке для нахождения аналогичного вхождения без указания модификатора «g». Протестируем эту возможность с помощью следующего примера:

```
<script language="JavaScript">
// определяем строку
var place = "Mississippi";
// определяем шаблон
```

```
var obj = /is./;
// ищем все вхождения в строку
// показываем результат
while((result = obj.exec(place)) != null)
{    alert("Found " + result[0] + " at " +  result.index);
}
</script>
```

В данном примере используется цикл «while» для вызова метода exec() до тех пор, пока не достигнут конец строки (на котором объект вернет null и цикл закончится). Это возможно, потому что каждый раз, вызывая exec(), объект Regular Expression продолжит поиск с того места, на котором закончил.

Другая интересная особенность этого кода заключается в создании объекта Regular Expression. Вы наверняка заметили, что, в отличие от предыдущего примера, здесь не используется конструктор для создания объекта. Вместо этого, шаблон просто применяется к переменной. Думайте об этом просто как о более коротком способе создания объекта Regular Expression.

В этом примере используется несколько регулярных выражений для проверки данных, введенных в форму пользователем, чтобы проверить правильность их формата. Этот тип проверки на стороне клиента крайне важен в Сети для того, чтобы быть уверенным в правильности и безопасности поступающих данных.

Для обеспечения удобства заполнения форм, а также корректности введенных данных необходимо правильно определять допустимые для ввода форматы данных. Также необходимо информировать пользователя о том, в каком формате следует вводить информацию.

В плане реализации проверки форм наиболее удачным является использование регулярных выражений, которые позволяют подобрать шаблон в подавляющем большинстве случаев.

Обработка данных на серверной стороне.

PHP поддерживает два вида записи регулярных выражений: POSIX (Portable Operating System Interface, интерфейс переносной операционной системы) и Perl (PCRE - Perl Compatible Regular Expression). По состоянию на PHP 5.3.0, расширение регулярные выражения POSIX считается устаревшим.

В общем случае, функции для работы с регулярными выражениями выполняются более медленно, чем строковые функции, предоставляющие аналогичные возможности. Поэтому, если можно без ущерба для эффективности приложения использовать строковые функции, их следует использовать.

PHP поддерживает ряд функций для работы с Perl-совместимыми регулярными выражениями.

int preg_match (string pattern, string subject [, array matches])

Эта функция ищет в строке subject соответствие регулярному выражению pattern. Если задан необязательный параметр matches, то результаты поиска помещаются в массив.

mixed preg_replace (mixed pattern, mixed replacement, mixed subject[int limit])

Эта функция ищет в строке subject соответствия регулярному выражению pattern, и заменяет их на replacement. Необязательного параметр limit задает число соответствий, которые надо заменить. Если этот параметр не указан, или равен -1, то заменяются все найденные соответствия.

int preg_match_all (string $pattern , string $subject [, array &$matches [, int $flags = PREG_PATTERN_ORDER [, int $offset = 0]]])

Функция ищет в строке subject все совпадения с шаблоном pattern и помещает результат в массив matches в порядке, определяемом комбинацией флагов flags. После нахождения первого соответствия последующие поиски будут осуществляться не с начала строки, а от конца последнего найденного вхождения.

Параметр flags регулирует порядок вывода совпадений в возвращаемом многомерном массиве:

– PREG_PATTERN_ORDER (по умолчанию) упорядочивает результаты так, что элемент *$matches[0]* содержит массив полных вхождений шаблона, элемент *$matches[1]* содержит массив вхождений первой подмаски, и так далее.

– PREG_SET_ORDER упорядочивает результаты так, что элемент $matches[0] содержит первый набор вхождений, элемент $matches[1] содержит второй набор вхождений, и так далее.

– PREG_OFFSET_CAPTURE для каждой найденной подстроки будет указана ее позиция в исходной строке. Необходимо помнить, что этот флаг меняет формат возвращаемого массива matches в массив, каждый элемент которого содержит массив, содержащий в индексе с номером 0 найденную подстроку, а смещение этой подстроки в параметре subject - в индексе 1.

Дополнительный параметр offset может быть использован для указания альтернативной начальной позиции для поиска.

array preg_split (string $pattern , string $subject [, int $limit = -1 [, int $flags = 0]])

Функция разбивает строку subject по регулярному выражению pattern.

Если указан параметр limit, функция возвращает не более, чем *limit* подстрок, оставшаяся часть строки будет возвращена в последней подстроке.

Параметр *flags* может быть любой комбинацией следующих флагов (объединенных с помощью побитового оператора |):

– PREG_SPLIT_NO_EMPTY – функция preg_split() вернет только непустые подстроки.

– PREG_SPLIT_DELIM_CAPTURE – выражение, заключенное в круглые скобки в разделяющем шаблоне, также извлекается из заданной строки и возвращается функцией.

– PREG_SPLIT_OFFSET_CAPTURE – для каждой найденной подстроки будет указана ее позиция в исходной строке. Необходимо помнить, что этот флаг меняет

формат возвращаемого массива: каждый элемент будет содержать массив, содержащий в индексе с номером *0* найденную подстроку, а смещение этой подстроки в параметре *subject* - в индексе *1*.

1.4.2 Методы шифрования данных

Большинство современных web приложений работает с важной и конфиденциальной информацией пользователя, которая хранится на сервере в базе данных (логин, пароль, IP-адреса, кредитные карточки) или в массиве session, на клиенте в качестве промежуточных данных – в cookies, WebStorege.

Если ваше приложение работает с финансовыми, медицинскими или просто с очень важными данными - используйте HTTPS - расширение протокола HTTP, поддерживающее шифрование. Данные, передаваемые по протоколу HTTPS, «упаковываются» в криптографический протокол SSL или TLS, тем самым обеспечивается защита этих данных. В отличие от HTTP, для HTTPS по умолчанию используется TCP-порт 443. Данные между браузером и веб-сервером передаются в зашифрованном виде и не могут быть расшифрованы в случае перехвата сниффером.

Однако протокол SSL/SSH защищает данные, которыми обмениваются клиент и сервер, но не защищают сами хранимые данные, поскольку SSL - протокол шифрования на уровне сеанса передачи данных. В этом случае, если злоумышление может получить непосредственный доступ к данным (в обход веб-сервера) и может извлечь интересующие данные, нарушить их целостность или подменить их.

В таком случае рекомендуется применять в web приложениях шифрование и/или хеширование данных. *Шифрованием* называется процесс преобразования данных в формат, в котором они могут быть прочитаны (во всяком случае, теоретически) только предполагаемым получателем сообщения. Получатель расшифровывает данные при помощи ключа или секретного пароля. *Хеширование* осуществляется хеш-функциями для получения хеш-кода.

PHP поддерживает большое количество алгоритмов шифрования и предоставляет расширения и библиотеки с готовыми методами, не требуя профессиональной подготовки в области криптографии и детального изучения алгоритмов шифрования, для решения такого рода задач. Некоторые из них представлены ниже.

string hash (string $algo , string $data [, bool $raw_output = false])

Эта функция возвращает строку, содержащую вычисленный хеш-код в шестнацатеричной кодировке в нижнем регистре на основе выбранного algo алгоритма хеширования и сообщение для хеширования data. Если *raw_output* задан как **TRUE**, то возвращается хеш-код в виде бинарных данных.

Генерация хеш-кода на основе ключа, используя метод HMAC, прямым методом шифрования осуществляется на основе

string hash_hmac (string $algo , string $data , string $key [, bool $raw_output = false])

Данная функция на основе выбранного алгоритма хеширования (список поддерживаемых алгоритмов можно узнать с помощью hash_algos()), сообщения data для хеширования и общего секретного ключа key. Возвращает строку содержащую вычисленный хеш-код в шестнадцатеричной кодировке в нижнем регистре. Если raw_output задан как TRUE, то возвращается хеш-код в виде бинарных данных.

Инициализация инкрементального контекста хеширования осуществляется с помощью функции

resource hash_init (string $algo [, int $options = 0 [, string $key = NULL]])

Параметрами функции являются имя `algo` выбранного алгоритма хеширования, необязательные настройки options для генерации хеша (в настоящее время поддерживается только один вариант: HASH_HMAC, причем в данном случае параметр key должен быть указан) и собственно общий секретный ключ key, который будет использоваться с методом хеширования HMAC. Возвращаемым значением данной

функции является ресурс хеширования для использования в функциях hash_update(), hash_update_stream(), hash_update_file() и hash_final().

bool hash_update (resource $context , string $data)

Эта функция на основе контекста хеширования context, возвращаемый hash_init(), добавляет данные data в активный контекст хеширования.

Завершает инкрементальное хеширование и возвращает результат в виде хеш-кода

string hash_final (resource $context [, bool $raw_output = false])

Эта функция возвращает строку содержащую вычисленный хеш-код в шестнадцатеричной кодировке в нижнем регистре. Если raw_output задан как TRUE, то возвращается хеш-код в виде бинарных данных.

string mhash (int hash, string data [, string key])

mhash() применяет хэш-функцию hash к данным data и возвращает результирующий хэш (называемый также digest/дайджест). Если key специфицирован, возвращается результирующий HMAC. HMAC это хэширование с ключам для аутентификации сообщения, либо просто дайджест сообщения, который зависит от специфицированного ключа. Не все алгоритмы, поддерживаемые в mhash, могут использоваться в режиме HMAC. При ошибке возвращает FALSE.

string mhash_keygen_s2k (int hash, string password, string salt, int bytes)

Генерирует ключ длиной bytes из заданного пользователем пароля в соответствии с заданным идентификатором hash (одна из констант MHASH_hash_name). Это алгоритм Salted S2K, как специфицировано в документе OpenPGP (RFC 2440),

Параметр salt обязан отличаться и быть достаточно произвольным для каждого ключа, генерируемого вами, чтобы создавать разные ключи. Salt имеет фиксированный размер в 8 байтов и будет заполняться нулями, если вы предоставите меньшее количество байтов.

Предоставляемые пользовательские пароли реально не подходят для использования в качестве ключей в алгоритмах криптографии, поскольку пользователи

выбирают ключи, которые они могут ввести с клавиатуры. Эти пароли используют только от 6 до 7 бит на символ (или менее). Рекомендуется использовать какую-нибудь трансформацию (вроде этой функции) предоставляемого пользователем ключа.

Mcrypt можно использовать для шифровки и дешифровки. Mcrypt может работать с четырьмя режимами шифровки (CBC, OFB, CFB и ECB). Если связь имеется с библиотекой libmcrypt-2.4.x или выше, эти функции могут также работать с блок-режимом шифровки nOFB и в режиме STREAM.

Для открытия модуля конкретного алгоритма и режима

resource mcrypt_module_open (string $algorithm , string $algorithm_directory , string $mode , string $mode_directory)

Имя алгоритма algorithm определяется одной из констант MCRYPT_ciphername, режим шифрования mode определяется одной из констант MCRYPT_MODE_modename, а параметры algorithm_directory и mode_directory используются для нахождения модуля шифрования. Вы можете указать имя каталога, или вы можете установить параметр в пустую строку (""), тогда используется значение, установленное в конфигурационном файле php.ini mcrypt.algorithms_dir Если он не установлен, используется тот какталог по умолчанию, который был составлен в Libmcrypt (обычно / USR / местные / Библиотека / Libmcrypt). Возвращает дескриптор шифрования, или FALSE при ошибке. Модуль закрывается с использованием функции **mcrypt_module_close()**, в качестве параметра которого указывается дескриптор шифрования.

Вы обязаны (в режимах CFB и OFB) или можете (в режиме CBC) поддерживать вектор инициализации /initialization vector (IV) для соответствующей функции шифровки. Этот IV обязан быть уникальным и обязан быть тем же самым при дешифровке/шифровке. Для данных, которые хранятся в шифрованном виде, вы можете получить вывод функции индекса, под которым данные хранятся (например, MD5-ключ имени файла).

Для случайной генерации вектора инициализации (IV) применяется

string mcrypt_create_iv (int $size [, int $source = MCRYPT_DEV_RANDOM])

Вектор IV может быть не секретным, хотя это может быть нежелательно. Кроме того вы даже можете отправить его вместе с зашифрованным текстом, не теряя безопасности.

В качестве параметров вы можете указать размер (size) IV и источник (source) генерации IV (MCRYPT_RAND - генератор случайных чисел, MCRYPT_DEV_RANDOM - чтение данных из */dev/random* и MCRYPT_DEV_URANDOM - чтение данных из */dev/urandom*). До 5.3.0, MCRYPT_RAND единственный поддерживается Windows. Вектор IV, как правило, должен иметь размер блока алгоритма, поэтому его размеры определяются вызовом mcrypt_enc_get_iv_size ().

Вы должны всегда вызывать эту функция перед вызовом mcrypt_generic() или mdecrypt_generic()

int mcrypt_generic_init (resource $td , string $key , string $iv)

В качестве параметров применяются дескриптор шифрования td, ключ key, максимальный размер которого определяется вызовом mcrypt_enc_get_key_size() и вектор IV.

Для деинициализации модуля шифрования применяется функция **mcrypt_generic_deinit()** с дескриптором шифрования в качестве параметра.

Для шифрования данных применяется функция

string mcrypt_generic (resource $td , string $data)

Данные **data** заполняются "\0" для того, чтобы убедиться, длина данных соответствует n*blocksize. Эта функция возвращает зашифрованные данные. Следует отметить, что длина возвращаемой строки может фактически быть больше, чем на входе, из-за заполнения данных.

Для дешифрования данных применяется функция **mdecrypt_generic().**

Шифровка input-значения шифром TripleDES под 2.4.x и выше в режиме ECB

```
<?php
   $key = "this is a secret key";
   $input = "Let us meet at 9 o'clock at the secret place.";
```

```
$td = mcrypt_module_open('tripledes', '', 'ecb', '');
$iv = mcrypt_create_iv (mcrypt_enc_get_iv_size($td), MCRYPT_RAND);
mcrypt_generic_init($td, $key, $iv);
$encrypted_data = mcrypt_generic($td, $input);
mcrypt_generic_deinit($td);
mcrypt_module_close($td);?>
```

Результатом является зашифрованная $encrypted_data строка.

Модуль использует функции OpenSSL для генерации и проверки подписи и отпечатков (шифровки) и открытия (дешифровки) данных. PHP-4.0.4pl1 требует OpenSSL >= 0.9.6, но PHP-4.0.5 и выше, также будет работать и с OpenSSL >= 0.9.5.

Функции openssl_get_xxx используют ресурсы key или certificate для формирования положительного идентификатора ключа ресурса.

resource openssl_pkey_get_private (mixed key [, string passphrase])

Эта функция возвращает положительный идентификатор ключа ресурса при успехе, FALSE при ошибке.

В качестве ключа key может использоваться:

-строка формата file://path/to/file.pem. Указанный файл должен содержать сертификат в кодировке PEM / закрытый ключ (или может содержать оба);

- закрытый ключ формата PEM.

Дополнительный параметр passphrase должен быть использован, если указанный ключ зашифрован (защищен паролем).

Функция openssl_pkey_get_public() извлекает открытый ключ из сертификата и подготавливает его для использования другими функциями.

resource openssl_pkey_get_public (mixed certificate)

Эта функция возвращает положительный идентификатор ключа ресурса при успехе, FALSE при ошибке.

В качестве сертификата certificate может использоваться:

- сертификат ресурса X.509;

- строка формата / / путь / к / file.pem. Указанный файл должен содержать сертификат в кодировке PEM / закрытый ключ (или может содержать оба);

- закрытый ключ формата PEM.

Для шифрования данных data с закрытым ключом key и сохранения результатов в crypted используется функция

bool openssl_private_encrypt (string $data , string &$crypted , mixed $key [, int $padding = OPENSSL_PKCS1_PADDING])

Шифрованные данные с помощью функции openssl_private_encrypt() могут быть расшифрованы функцией openssl_public_decrypt(). Параметр padding может быть OPENSSL_PKCS1_PADDING, OPENSSL_NO_PADDING. Возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки.

Для расшифровки данных, которые были предварительно зашифрованы функцией **openssl_public_encrypt()** используется

bool openssl_private_decrypt (string $data , string &$decrypted , mixed $key [, int $padding = OPENSSL_PKCS1_PADDING])

Эта функция расшифровывает данные **data** с закрытым ключом **key** и сохраняет результат в decrypted. В качестве параметра padding может быть использованы OPENSSL_PKCS1_PADDING, OPENSSL_SSLV23_PADDING, OPENSSL_NO_PADDING OPENSSL_PKCS1_OAEP_PADDING. Возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки.

Для расшифровки данных, которые были предварительно зашифрованы функцией openssl_private_encrypt **()** используется функция **openssl_public_decrypt().**

1.5 Варианты заданий

Задание №1. Построить клиентскую (HTML5, JS) и серверную (PHP) части для проверки правильности данных HTML-формы с указанным набором полей и ограничений (таблица 1.2, 1.3).

Таблица 1.2 – Варианты заданий для валидации

№ варианта	Поле1 (HTML5)	Поле2 (JS)	Поле3 (PHP)	Правила (JS и/или PHP)
1	Login	Pass1	Pass2	Pass1=Pass2
2	Credit Visa	Text	Email	
3	Credit MasterCard	Text	Date	Определить сколько дней прошло с даты Birthday по текущую дату
4	Credit MasterCard	Date	Email	Определить сколько дней до окончания срока действия кредитной карты
5	Email	IP adress	WebURL	
6	IP adress	Pass	Age	
7	Height, step=3	Weight, max=200	Credit MasterCard	Weight < Height-100
8	Login	Pass1	Pass2	Login≠Pass1
9	Credit Visa	Email	IP adress	
10	Credit Maestro	Email	WebURL	

Таблица 1.3 – Требования к значениям полей

Тип поля	Описание правил
Text	Непустая строка. В строке допустимыми символами считаются буквы, цифры, “_”, “?”, “!”.
Login	Текстовая строка, содержащая как минимум 7 символов букв и цифр, начинающаяся с буквы латинского алфавита
Pass	Текстовая строка, содержащая как минимум 6 символов букв и цифр
Height	Число для отображения роста в футах или см (1′ = 30,48 см или 0,3048 м)
Weight	Число для отображения веса в кг или фунтах (1 фунт = 0,454 кг)
Age	Целое число, указывающее возраст (число в диапазоне от 1 до 100)
Email	Адрес электронной почты – только один символ @, одна или более точек, проверка на существующие домены верхнего уровня.
IP adress	Любой IP-адрес в пределах диапазона 192.168.1.0–192.168.1.255.
WebURL	Адрес сайта с проверкой на существующие протоколы (например, http, https, ftp, file)
Date	Дата в формате ГГГГ.ММ.ДД или гг.ММ.ДД
Credit Visa	Номер карточки международной платежной системы VISA, всегда начинается с цифры «4», состоит всего из 16 цифр (четыре группы по четыре цифры) или из 13 цифр (первая группа из четырех цифр и три группы из трех).
Credit MasterCard	Номер карточки MasterCard начинается с цифры «5», вторая цифра находится в диапазоне от 1 до 5, состоит из 16 цифр
Credit Maestro	Номер карточки Maestro начинается с цифр “3”, “5” “6” и может состоять из 13,16 или 19 цифр.

Задание №2. Реализовать для одного из полей механизм безопасности данных с указанным методом шифрования (таблица 1.4).

Таблица 1.4 – Варианты заданий для реализации методов шифрования данных

№ варианта	Поле	Метод
1	Pass1	Метод hash с использованием секретного слова в качестве соли
2	Credit Visa	Шифрование методом hash_update и запись значения в COOKIES
3	Credit MasterCard	Метод mcrypt с использованием вектора шифрования для шифрования и дешифрования данных
4	Credit MasterCard	Метод hash_update с использованием ключа, генерируемого тегом HTML5 <keygen>
5	IP adress	Шифрование методом mcrypt и запись значения в COOKIES
6	IP adress	Шифрование методом hash_hmac идентификатора SESSION и запись значения в SESSION
7	Credit MasterCard	Метод mhash с использованием ключа, генерируемого mhash_keygen
8	Pass1	Метод hash_hmac с использованием ключа, генерируемого тегом HTML5 <keygen>
9	Credit Visa	Шифрование методом mcrypt и запись значения в SESSION
10	Credit Maestro	OpenSSL с использованием закрытого ключа и сертификата

1.6 Содержание отчета

Отчет по лабораторной работе должен содержать:

- цель работы;
- исходный вариант задания;
- HTML5 – файл с дизайном формы и реализацией валидации формы браузером;
- JS – файл с блоком JavaScript для клиентской проверки отсылаемых данных;

– PHP – файл для валидации и шифрования принимаемых данных на серверной стороне;

– результаты работы всех вышеприведенных модулей;

– выводы по работе.

1.7 Контрольные вопросы и задания

1) . Какими специальными символами определяется количество повторений группы символов при задании регулярного выражения?

2) . Каким образом реализовать с использованием HTML5 ограничение на вводимое значение даты, например, ввод даты, начиная с текущей?

3) . Каким образом реализовать с использованием HTML5 для одного элемента несколько обработчиков?

4) . Каким образом реализовать для HTML5 формы вывод кастомного сообщения в виде текста на веб-сайт?

5) . Какие функции JavaScript и PHP используются для проверки соответствия входных данных типа integer?

6) . Какие функции JavaScript и PHP используются для проверки соответствия входной текстовой строки заданному шаблону?

7) . В чем состоит отличие процесса шифрования от хеширования?

8) . Назовите способы решения задачи обратной хешированию?

9) . Что представляет собой криптографическая соль?

10) В чем состоят недостатки применения неизменной соли?

11) В чем отличие прямого и инкрементального способа хэширования?

12) Какие вы знаете способы генерации ключей для шифрования?

13) В чем заключается отличие расширения HASH и MHASH?

2 ИЗУЧЕНИЕ РАСШИРЕНИЯ PDO ДЛЯ ОБЕСПЕЧЕНИЯ АБСТРАКЦИИ ДОСТУПА К БАЗАМ ДАННЫХ

2.1 Цель работы

Изучение методов управления подключениями, выполнения запросов, транзакций и процедур расширения PDO для реализации основных возможностей различных СУБД в web приложении.

2.2 Методические указания по организации самостоятельной работы студентов

При подготовке к выполнению лабораторной работы следует повторить синтаксис операторов языка SQL применительно к операциям выборки (SELECT) и модификации (INSERT, DELETE, UPDATE) данных.

Кроме того, требуется ознакомиться с программными методами выполнения запросов к базе данных, а также этапами выполнения подготовленного запроса к базе данных с применением интерфейса PDO [5,6].

2.3 Программное обеспечение ПК

При выполнении лабораторной работы используется ПЭВМ под управлением операционной системы Windows XP и старше, веб-сервер xampp 2.4.4, среда разработки Notepad++ 6.4.5.

2.4 Методические указания по выполнению лабораторной работы

При подготовке к выполнению лабораторной работы следует повторить синтаксис операторов языка SQL применительно к операциям выборки (SELECT) и модификации (INSERT, DELETE, UPDATE) данных.

Расширение объекты данных PHP (PDO) определяет простой и согласованный интерфейс для доступа к базам данных в PHP. Каждый драйвер базы данных, в котором реализован этот интерфейс, может представить специфичный для базы данных функционал в виде стандартных функций расширения (рисунок 2.1). PDO не зря расшифровывается как PHP Data Object - так как взаимодействие с базами данных осуществляется посредством объектов.

Рисунок 2.1 – Схема реализации интерфейса для доступа к базам данных в PHP

К основным задачам PDO относятся следующие.

– обеспечить стандартизированное API для реализации основных возможностей различных СУБД;

– быть потенциально расширяемым, чтобы разработчики баз данных могли реализовывать новые возможности своих СУБД в PDO;

– обеспечить легкую переносимость приложений между различными СУБД;

– PDO призвано обеспечить абстракцию средств доступа к СУБД, но не абстракцию функционала самих СУБД. Другими словами: PDO *не* абстрагирует саму *базу данных*, это расширение не переписывает SQL запросы и не эмулирует отсутствующий в СУБД функционал. В PDO нет эмуляции тех возможностей, которые не поддерживаются одной СУБД, но есть в другой;

– упростить создание новых драйверов для работы с базами данных в PHP, обеспечив упрощенный интерфейс с "внутренностями" PHP, работа с которыми занимает больше всего времени.

Схема подготовки и выполнения подготовленного запроса к базе данных (БД) с применением интерфейса БД PDO, а также вывода результатов показана на рисунке 2.2.

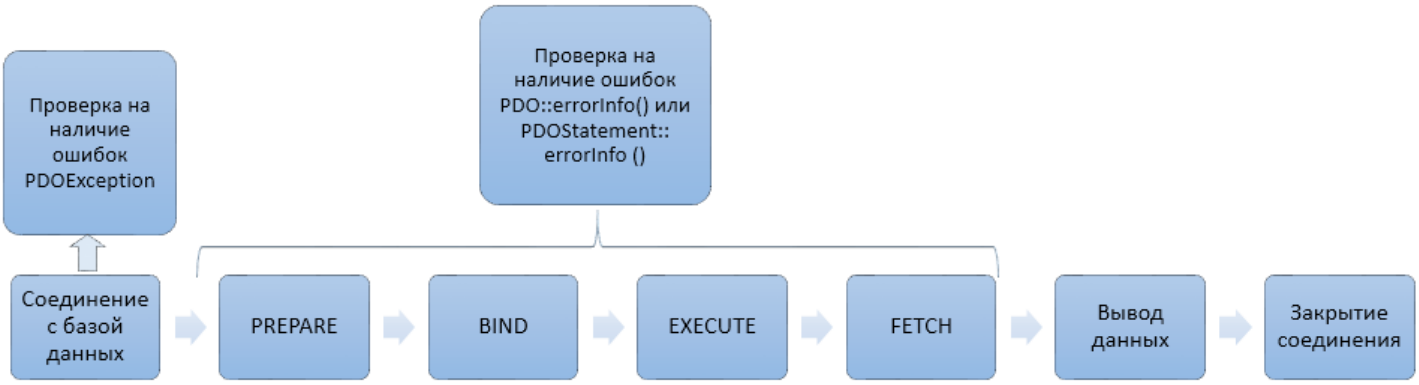

Рисунок 2.2 – Схема выборки строк из базы данных с применением подготовленного запроса и вывода результатов

Далее будут рассмотрены основные принципы работы с классами PDO и PDOStatement.

Поддерживает следующие драйверы БД (просмотреть список драйвер возможно с помощью метода PDO::getAvailableDrivers()):

PDO_CUBRID Cubrid

PDO_DBLIB FreeTDS / Microsoft SQL Server / Sybase

PDO_FIREBIRD Firebird/Interbase 6

PDO_IBM IBM DB2

PDO_INFORMIX IBM Informix Dynamic Server

PDO_MYSQL MySQL 3.x/4.x/5.x

PDO_OCI Oracle Call Interface

PDO_ODBC ODBC v3 (IBM DB2, unixODBC и win32 ODBC)

PDO_PGSQL PostgreSQL

PDO_SQLITE SQLite 3 и SQLite 2

PDO_SQLSRV Microsoft SQL Server / SQL Azure

PDO_4D 4D

Соединение и закрытие соединения с базой данных.

Соединение с базой данных устанавливается тогда, когда создаётся экземпляр класса PDO (например, объект $pdh)), причем не имеет значения, какой драйвер используется. Его конструктор принимает параметры для того, чтобы определить источник базы данных (известный как DSN), и необязательные параметры для имени пользователя (username), пароля (password) и массив устанавливаемых опций подключения (driver_options).

Формат записи конструктора следующий:

PDO::_construct (string dsn [, string username [, string password [, array driver_options]]])

Имя источника данных PDO_MYSQL (DSN) требуется в качестве первого параметра конструктора при создании нового объекта класса PDO, оно составлено из следующих элементов:

– DSN prefix - приставка DSN, например, "mysql:" или "mysqli";

– host - имя хоста, на котором находится сервер базы данных;

– port - номер порта, сервера базы данных;

– dbname - имя базы данных;

– unix_socket - сокет Unix MySQL (не должен использоваться с хостом или портом);

– charset – кодировка (до PHP 5.3.6 этот элемент игнорирован).

Варианты строки подключения драйвера PDO_MYSQL:

mysql:host=localhost;dbname=testdb

mysql:host=localhost;port=3307;dbname=testdb

mysql:unix_socket=/tmp/mysql.sock;dbname=testdb

Вариант строки подключения драйвера PDO_SQLITE: sqlite:/path/to/database.db

Для создания базы данных в памяти, используйте: sqlite::memory:

Вариант строки подключения драйвера для sqlite (version 2): sqlite2:/path/to/database.db

Для создания базы данных в памяти, используйте: sqlite2::memory:

Пример создания объекта PDO и подключения к СУБД MySQL ($dbh расшифровывается как «database handle).

```
<?php
 $dsn = 'mysql:host=localhost;dbname=testdb';   $username = 'имя пользователя';
 $password = 'пароль';
 $options = array( PDO:: MYSQL_ATTR_INIT_COMMAND => 'SET NAMES utf8');
 $dbh = new PDO ($dsn, $username, $password, $options);?>
```

В результате успешного подключения к базе данных возвращается экземпляр класса PDO. Соединение остается активным в течение жизни этого объекта PDO. Для того чтобы завершить соединение, вам нужно уничтожить объект, гарантируя, что все остальные ссылки на него будут удалены – это можно осуществить путем присвоения NULL переменной, содержащей объект. Если вы не делаете этого явно, PHP автоматически закрывает соединение после завершения скрипта.

```
<?php
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
// используем соединения здесь
// теперь закрываем его
$dbh = null;
?>
```

Опции подключения можно устанавливать тремя способами:

– При соединении с базой данных наиболее часто используемые опции соединения – это такие как PDO::ATTR_PERSISTENT для создания постоянных подключений к серверам баз данных, которые не закрываются в конце сценария, но кэшируются и использовать повторно, когда другой сценарий запрашивает соединение с

использованием тех же учетных данных, и PDO::MYSQL_ATTR_INIT_COMMAND, позволяющая частично реализовать указание кодировки при создании объекта PDO:

```
<?php
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass, array(
   PDO::ATTR_PERSISTENT => true, PDO::MYSQL_ATTR_INIT_COMMAND =>
'SET NAMES \'UTF8\"
));
?>
```

– При использовании метода PDOObj->setAttribute() для присваивания атрибута PDO объекту, метода PDOStatement->setAttribute() для установки опций конкретного драйвера:

```
<?php
$ dbh = new PDO($connection_string);
$ dbh ->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_OBJ);
 $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
?>
```

– При использовании некоторых функции выборки данных, например, у метода PDO::prepare() в качестве второго параметра учитываются опции драйвера (одна или более пар ключ=>значение для установки значений атрибутов объекта PDOStatement):

```
<?php
$sth = $dbh->prepare($sql, array(PDO::ATTR_CURSOR => PDO::CURSOR_FWDONLY));
?>
```

Если вы хотите написать переносимый код, не зависящий от типа подключаемой базы данных рекомендуется не использовать специфичные для конкретного драйвера константы и конструкции запросов. Больше информации по теме отличительных опций разных СУБД и методах подключения к ним можно найти на php.net.

Блок try/catch рекомендуется использовать для оборачивания в него всех PDO-операций и использования механизм исключений PDOException для вывода ошибок:

```
<?php
try {
  $dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
  foreach($dbh->query('SELECT * from FOO') as $row) {
     print_r($row);
  }
  $dbh = null;
} catch (PDOException $e) {
  print "Error!: " . $e->getMessage() . "<br/>";
  die();}
?>
```

Выполнение запросов к базе данных.

Метод **query()** выполняет SQL запрос statement без подготовки и возвращает результирующий набор в виде объекта PDOStatement или FALSE при ошибке.

PDOStatement PDO::query (string $statement)

Приятной особенностью **query()** является то, что после выполнения SELECT запроса можно сразу работать с результирующим набором посредством курсора.

Метод query() требует, чтобы вы должным образом экранировали все данных, чтобы избежать SQL инъекций и других вопросов. Поэтому его надо использовать в запросах без внешних данных.

```
<?php
$conn ->query("SET CHARACTER SET utf8");
function getFruit($conn) {
$sql = 'SELECT name, color, calories FROM fruit ORDER BY name';
foreach ($conn->query($sql) as $row) {
print $row['name'] . "\t";
print $row['color'] . "\t";
```

```
print $row['calories'] . "\n";
}}?>
```

Быстрое получение данных из запроса.

PDO предлагает некоторые дополнительные пути для получения информации из запроса без использования fetch*() функций и цикла по результату.

Метод **PDOStatement::rowCount** возвращает количество строк, которые были затронуты в ходе выполнения последнего запроса DELETE, INSERT или UPDATE, запущенного соответствующим объектом PDOStatement.

```
$stmt = $db->query('SELECT * FROM table');
$row_count = $stmt->rowCount();
```

В другом случае используйте метод **PDOStatement::columnCount()**, чтобы узнать количество столбцов в результирующем наборе, который представляет объект PDOStatement.

Если объект PDOStatement был возвращен из метода query(), число столбцов можно узнать сразу же. Если объект PDOStatement был возвращен из метода PDO::prepare(), точное количество столбцов можно будет узнать только после запуска метода PDOStatement::execute().

```
<?php
$sth = $dbh->prepare("SELECT name, colour FROM fruit");
/* Подсчет количества столбцов в (несуществующем) результирующем наборе */
$colcount = $sth->columnCount();
print("Перед вызовом execute(), в результирующем наборе $colcount столбцов
(должно быть 0)\n");
$sth->execute();
/* Подсчет количества столбцов в результирующем наборе */
$colcount = $sth->columnCount();
print("После вызова execute(), в результирующем наборе $colcount столбцов
```

```
(должно быть 2)\n");
?>
```

Метод **PDO::lastInsertId** возвращает ID последней вставленной строки либо последнее значение, которое выдал объект последовательности.

```
$db->query("INSERT INTO users SET name='Vasya',address='Here',email='vasya@test.com'");
$insertId=$db->lastInsertId();
```

Если вы не используете prepared statements, тогда необходимо для безопасного использования SQL запросов использовать метод **quote()**, который возвращает строку, в которой проставлены кавычки в строковых данных (если требуется) и экранированы специальные символы внутри строки подходящим для драйвера способом.

```
<?php
$conn = new PDO('sqlite:/home/lynn/music.sql3');
/* небезопасная строка */
$string = 'Naughty \' string';
print "Неэкранированная строка: $string\n";
print "Экранированная строка:" . $conn->quote($string) . "\n";
//Запрос с условием и экранированием
$conn->query('SELECT * FROM table WHERE id = ' . $conn->quote($id));
?>
```

Вывод результата:

Неэкранированная строка: Naughty ' string

Экранированная строка: 'Naughty '' string'

Метод **exec()** запускает SQL запрос statement на выполнение и используется для операций, которые не возвращают никаких данных, кроме количества затронутых ими записей. Данные внутри запроса должны быть правильно экранированы. Используется, например, при удалении данных из базы данных.

```
<?php
$dbh = new PDO('odbc:sample', 'db2inst1', 'ibmdb2');
/* Удаляем все записи из таблицы FRUIT */
$count = $dbh->exec("DELETE FROM fruit WHERE colour = 'red'");
/* Получим количество удаленных записей */
print("Удалено $count строк.\n");
?>
```

Выполнение подготовленных запросов.

Подготовленные выражения (англ. prepared statments) позволяют определить выражение один раз, а затем многократно его выполнять с разными параметрами, что повышает скорость выполнения и экономит траффик между приложением и базой данных. Также они позволяют отделить переменные от запроса, что позволяет сделать код безопаснее. С этой целью применяется метод **prepare()**, который подготавливает запрос statement к выполнению и возвращает ассоциированный с этим запросом объект PDOStatement.

Обратите внимание, что после создания объектом PDO подготовленного выражения в классе PDOStatement, используется только он, соответственно, в нем есть свои функции errorCode, errorInfo, а также результат выполнения запросов, также сразу же хранится в нем.

PDOStatement PDO::prepare (string $statement [, array $driver_options = array()])

Запрос может содержать именованные (:name) или неименованные (?) псевдопеременные, которые будут заменены реальными значениями во время запуска запроса на выполнение. Если вы определили переменные знаком вопроса, то в функцию

execute передайте массив значений, в той, последовательности, в которой стоят переменные. Использовать одновременно и именованные, и неименованные псевдопеременные в одном запросе нельзя, необходимо выбрать что-то одно.

```
<?php
/* Выполнение SQL запроса с передачей ему массива именованных параметров */
$sql = 'SELECT name, colour, calories FROM fruit
WHERE calories < :calories AND colour = :colour';
/*Предписание создать объект PDOStatement с последовательным курсором*/
$sth = $dbh->prepare($sql, array(PDO::ATTR_CURSOR =>
 PDO::CURSOR_FWDONLY));
$sth->execute(array(':calories' => 150, ':colour' => 'red'));
$red = $sth->fetchAll();

/* Выполнение запроса с передачей ему массива неименованных параметров */
$sth = $dbh->prepare('SELECT name, colour, calories FROM fruit
WHERE calories < ? AND colour = ?');
$sth->execute(array(150, 'red'));
$red = $sth->fetchAll();
?>
```

В предыдущем примере пропущена функция **bind*()**, задающая значение именованной или неименованной псевдопеременной в подготовленном SQL запросе.

Существует три вида функции типа bind*():

– **bindValue()** задает значение именованной или неименованной псевдопеременной в подготовленном SQL запросе

bool PDOStatement::bindValue (mixed $parameter , mixed $value [, int $data_type = PDO::PARAM_STR])

```
<?php
/* Выполнение подготовленного запроса с именованными псевдопеременными */
$sth = $dbh->prepare('SELECT name, colour, calories
FROM fruit WHERE calories < :calories);
$sth->bindValue(':calories', 150, PDO::PARAM_INT);
$sth->execute();
/* Выполнение подготовленного запроса с неименованными псевдопеременными (?) */
$sth = $dbh->prepare('SELECT name, colour, calories
FROM fruit WHERE calories < ? AND colour = ?');
$sth->bindValue(1, 150, PDO::PARAM_INT);
$sth->bindValue(2, ‘red’, PDO::PARAM_STR);
$sth->execute();
?>
```

– **bindParam()** привязывает именованный или неименованный параметр подготавливаемого SQL запроса с настоящей переменной, при изменении настоящей переменной, не нужно больше вызывать никаких дополнительных функций, можно сразу execute().В отличие от bindValue(), переменная привязывается по ссылке, и ее значение будет вычисляться во время вызова execute()

bool PDOStatement::bindParam (mixed $parameter , mixed &$variable [, int $data_type = PDO::PARAM_STR [, int $length [, mixed $driver_options]]])

```
<?php
/* Выполнение запроса с привязкой PHP переменных */
$calories = 150;
$colour = 'red';
/* Выполнение подготовленного запроса с именованными псевдопеременными */
$sth = $dbh-
>prepare('SELECT name, colour, calories FROM fruit WHERE calories < :calories);
```

```
$sth->bindParam(':calories', $calories, PDO::PARAM_INT);
$sth->execute();
/*Выполнение подготовленного запроса с неименованными псевдопеременными (?)*/
sth = $dbh->prepare('SELECT name, colour, calories
FROM fruit WHERE calories < ? AND colour = ?');
$sth->bindParam(1, $calories, PDO::PARAM_INT);
$sth->execute();
?>
```

– **bindColumn()** привязывает переменную к заданному столбцу в результирующем наборе запроса. Каждый вызов PDOStatement::fetch() или PDOStatement::fetchAll() будет обновлять все переменные, задавать им значения столбцов, с которыми они связаны

bool PDOStatement::bindColumn (mixed $column , mixed &$param [, int $type [, int $maxlen [, mixed $driverdata]]])

```
<?php
$sql = 'SELECT name, colour, calories FROM fruit';
$stmt = $dbh->prepare($sql);
/* Связывание по номеру столбца */
$stmt->bindColumn(1, $name);
$stmt->bindColumn(2, $colour);
/* Связывание по имени столбца */
$stmt->bindColumn('calories', $cals);
$stmt->execute();
?>
```

Первым параметром *parameter* каждой из функций является идентификатор параметра запроса. Для подготавливаемых запросов с именованными параметрами это будет имя в виде :name. Если используются неименованные параметры (знаки вопроса ?) это будет позиция псевдопеременной в запросе (начиная с 1).

Третьим параметром каждой из функций *data_type* является явно заданный тип данных параметра (одна из констант PDO::PARAM_*):

```
<?php $sth3->bindParam(‘:id’,$id, PDO::PARAM_INT);
$sth3->bindParam(‘:id’,$id, PDO::PARAM_STR); ?>
```

Третьим методом в цепочке при выполнении подготовленных выражений является метод **PDOStatement::execute(),** который запускает подготовленный запрос. Метод execute также прекрасно работает когда вы повторяете запрос несколько раз.

Если в вашем SQL-выражении много параметров, то назначать каждому по переменной весьма неудобно. В таких случаях можно хранить данные в массиве и передавать его:

```
<?php // набор данных, которые мы будем вставлять
$data = array('Cathy', '9 Dark and Twisty Road', 'Cardiff');
$STH = $DBH->prepare("INSERT INTO folks (name, addr, city) values (?, ?, ?)");
$STH->execute($data);  ?>
```

Элемент $data[0] вставится на место первого placeholder’a, элемент $data[1] – на место второго, и т.д.

Выборка строк из запроса.

Для извлечения следующей строки из результирующего набора используйте метод **fetch()**:

```
mixed PDOStatement::fetch ([ int $fetch_style [, int $cursor_orientation = PDO::FETCH_ORI_NEXT [, int $cursor_offset = 0 ]]] )
```

Для извлечения всех строк из результирующего набор можно применить цикл по результатам выборки:

```
<?php
// Выбираем каждую строку на каждой итерации пока не выберем все строки
while ($row = $res->fetch(PDO::FETCH_NUM)) {
      echo  $id = $row[0];
} ?>
```

В случае успешного выполнения функции возвращаемое значение зависит от режима выборки. Например, для запроса указанного в примере ниже можно задать различные режимы.

```
<?php
$sth = $dbh->prepare("SELECT name, colour FROM fruit");
$sth->execute();?>
```

Параметр *fetch_style* определяет, в каком виде PDO вернет эту строку (одна из констант PDO::FETCH_*):

– PDO::FETCH_ASSOC возвращает массив с названиями столбцов в виде ключей

```
<?php print_r($sth->fetch(PDO::FETCH_ASSOC));?>
```

Вывод результата:

Array ([ID_Authors] => 1 [name] => Косминский Е.А.)

Доступ к данным:

```
<?php  while($row = $sth->fetch(PDO::FETCH_ASSOC))
{    echo "<p>" . $row[' ID_Authors '] . " " . $row[' name '] . "</p>";}?>
```

– PDO::FETCH_NUM возвращает массив, с ключами в виде порядковых номеров столбцов (начиная с 0);

```
<?php print_r($result->fetch(PDO::FETCH_NUM));?>
```

Вывод результата:

Array ([0] => 1 [1] => Косминский Е.А.)

Доступ к данным:

```
<?php  while($row ($result->fetch(PDO::FETCH_NUM))
{    echo "<p>" . $row[0] . " " . $row[1] . "</p>";}?>
```

– PDO::FETCH_BOTH (по умолчанию) возвращает массив, индексированный именами столбцов результирующего набора, а также их номерами (начиная с 0);

```
<?php print_r($sth->fetch(PDO::FETCH_BOTH)); ?>
```

Вывод результата:

Array ([ID_Authors] => 1 [0] => 1 [name] => Косминский Е.А. [1] => Косминский Е.А.)

Доступ к данным:

```
<?php  while($row ($result->fetch(PDO::FETCH_ BOTH))
{    echo "<p>" . $row[0] . " " . $row['name'] . "</p>";}?>
```

– PDO::FETCH_OBJ создает анонимный объект со свойствами, соответствующими именам столбцов результирующего набора.

```
<?php print_r($result->fetch(PDO::FETCH_OBJ)); ?>
```

Вывод результата:

stdClass Object ([ID_Authors] => 1 [name] => Косминский Е.А.)

Доступ к данным:

```
while ($row = $result->fetch(PDO::FETCH_OBJ))
{       echo  $row->ID_Authors;
```

```
    echo  $row->name;}
```

– PDO::FETCH_LAZY комбинирует PDO::FETCH_BOTH и PDO::FETCH_OBJ, создавая новый объект со свойствами, соответствующими именам столбцов результирующего набора

```
<?php print_r($result->fetch(PDO::FETCH_LAZY)); ?>
```

Вывод результата:

PDORow Object ([queryString] => select ID_Authors, name from Authors order by ID_Authors [ID_Authors] => 1 [name] => Косминский Е.А.)

Доступ к данным:

```
$number=0;
while ($row = $result->fetch(PDO::FETCH_LAZY))
{   echo  $row->$number; echo "<br>";
    echo  $row->name;   echo "<br>";}
```

– PDO::FETCH_BOUND: возвращает TRUE и присваивает значения столбцов результирующего набора переменным PHP, которые были привязаны к этим столбцам методом PDOStatement::bindColumn()

– PDO::FETCH_CLASS создает и возвращает объект запрошенного класса, присваивая значения столбцов результирующего набора именованным свойствам класса. Если *fetch_style* включает в себя атрибут PDO::FETCH_CLASSTYPE (например, PDO::FETCH_CLASS | PDO::FETCH_CLASSTYPE), то имя класса, от которого нужно создать объект, будет взято из первого столбца.

– PDO::FETCH_INTO обновляет существующий объект запрошенного класса, присваивая значения столбцов результирующего набора именованным свойствам объекта

Значением параметра *cursor_orientation* должна быть одна из констант PDO::FETCH_ORI_*, по умолчанию PDO::FETCH_ORI_NEXT, предписывающая выбрать следующую строку из результирующего набора.

Для объектов PDOStatement, представляющих прокручиваемый курсор, параметр cursor_orientation которых принимает значение PDO::FETCH_ORI_ABS, величина *offset* означает абсолютный номер строки, которую необходимо извлечь из результирующего набора.

Для объектов PDOStatement, представляющих прокручиваемый курсор, параметр cursor_orientation которых принимает значение PDO::FETCH_ORI_RE*L*, эта величина *offset* указывает, какая строка относительно текущего положения курсора будет извлечена функцией PDOStatement::fetch()**.**

Для возврата массива, содержащего все строки результирующего набора используйте метод **fetchAll()**:

array **PDOStatement::fetchAll** ([int $fetch_style [, mixed $fetch_argument [, array $ctor_args = array()]]])

Массив представляет каждую строку либо в виде массива значений одного столбца, либо в виде объекта, имена свойств которого совпадают с именами столбцов.

```
<?php
$sth = $dbh->prepare("select ID_Authors, name from Authors order by ID_Authors");
$sth->execute();
/* Извлечение всех оставшихся строк результирующего набора */
print("Извлечение всех оставшихся строк результирующего набора:\n");
$result = $sth->fetchAll();
print_r($result);
?>
```

Вывод результата:

```
Array ( [0] => Array ( [0] => 1 [1] => Косминский Е.А. )
        [1] => Array ( [0] => 2 [1] => Бочаров В.В. )
        [2] => Array ( [0] => 3 [1] => Уильям Уолкер Аткинсон )
        [3] => Array ( [0] => 4 [1] => Лоис Макмастер Буджолд ) )
```

По умолчанию параметр *fetch_style* принимает значение PDO::ATTR_DEFAULT_FETCH_MODE, которое в свою очередь имеет по умолчанию значение PDO::FETCH_BOTH. Если требуется извлечь только уникальные строки одного столбца, нужно передать побитовое ИЛИ (|) констант PDO::FETCH_COLUMN и PDO::FETCH_UNIQUE.

Чтобы получить ассоциативный массив строк сгруппированный по значениям определенного столбца, нужно передать побитовое ИЛИ (|) констант PDO::FETCH_COLUMN и PDO::FETCH_GROUP.

Смысл аргумента *fetch_argument* зависит от значения параметра fetch_style. Например, если это параметр PDO::FETCH_COLUMN, то будет возвращен указанный столбец (индексация столбцов начинается с 0).

```
<?php
$sth = $dbh->prepare("SELECT name, colour FROM fruit");
$sth->execute();
/* Извлечение всех значений первого столбца */
$result = $sth->fetchAll(PDO::FETCH_COLUMN, 0);
foreach ($result as $row)
{echo $name=$row; echo "<br>";} ?>
```

Вывод результата:

```
Array(3)(   [0] =>    string(5) => apple
            [1] =>    string(4) => pear
            [2] =>    string(10) => watermelon)
```

Для случаев, когда параметру fetch_style присвоено значение PDO::FETCH_CLASS могут быть заданы аргументы конструктора класса *ctor_args*.

Если необходимо возвратить данные одного столбца следующей строки результирующего набора можно воспользоваться функцией **fetchColumn()**, параметром которой является номер столбца, данные которого необходимо извлечь:

string PDOStatement::fetchColumn ([int $column_number = 0])

```
<?php
$sth = $dbh->prepare("SELECT name, colour FROM fruit");
$sth->execute();
print("Получение значения первого столбца следующей строки:\n");
$result = $sth->fetchColumn();
print("Получение значения второго столбца следующей строки:\n");
$result = $sth->fetchColumn(1);
?>
```

Вывод результата:

```
Получение значения первого столбца следующей строки:
name = lemon
Получение значения второго столбца следующей строки:
colour = red
```

Для того, чтобы извлечь следующую строку и получить ее в виде объекта воспользуйтесь функцией **fetchObject()**:

mixed PDOStatement::fetchObject ([string $class_name = "stdClass" [, array $ctor_args]])

Этот метод является альтернативой вызову PDOStatement::fetch() с параметром PDO::FETCH_CLASS или PDO::FETCH_OBJ. В качестве параметров у этого метода

применяется имя класса создаваемого объекта (*class_name*), элементы этого массива будут переданы в конструктор класса (*ctor_args*)

Для задания режима выборки по умолчанию для всех вызовов объекта запроса используется метод **setFetchMode()**, варианты которого представлены ниже:

bool PDOStatement::setFetchMode (int $mode)

bool PDOStatement::setFetchMode (int $PDO::FETCH_COLUMN , int $colno)

bool PDOStatement::setFetchMode (int $PDO::FETCH_CLASS , string $classname , array $ctorargs)

bool PDOStatement::setFetchMode (int $PDO::FETCH_INTO , object $object)

Возвращает 1 в случае успешной установки режима или FALSE в случае возникновения ошибки.

```
<?php
$sql = 'SELECT name, colour, calories FROM fruit';
try {
  $stmt = $dbh->query($sql);
  $result = $stmt->setFetchMode(PDO::FETCH_NUM);
// цикл while() переберет весь результат запроса.
  while ($row = $stmt->fetch()) {
    print $row[0] . "\t" . $row[1] . "\t" . $row[2] . "\n";}
}
catch (PDOException $e) {
  print $e->getMessage();
}
?>
```

Метод **closeCursor**() класса PDOStatement освобождает соединение с сервером, давая возможность запускать другие SQL запросы. Метод оставляет запрос в состоянии готовности к повторному запуску.

Выполнение транзакций данных.

Транзакция – это совокупность запросов базу данных, которые должны быть обязательно выполнены все. Если какой-либо запрос не выполнен или выполнен с ошибкой, то транзакция отменяется и изменений данных в базе не происходит.

Это нужно, чтобы гарантировать сохранение целостности данных при нескольких запросах. например при переводе денежных средств со счета на счет.

Чтобы выполнить транзакцию в PDO необходимо перейти в режим ручного подтверждения запросов.

Кстати говоря, транзакции используются постоянно, но обычно PDO работает в режиме автоподтверждения, потому все транзакции состоят из одного запроса.

Инициализация транзакции осуществляется с помощью метода PDO::beginTransaction(), который выключает режим автоматической фиксации транзакции (режим автоподтверждения). После этого выполняем столько запросов к базе данных сколько необходимо сделать в этой транзакции.

В то время, как режим автоматической фиксации выключен, изменения, внесенные в базу данных через объект экземпляра PDO, не применяются, пока вы не завершите (фиксируете) транзакцию, вызвав PDO::commit(),возвращая соединение с базой данных в режим автоматической фиксации до тех пор, пока следующий вызов PDO::beginTransaction() не начнет новую транзакцию.

```
<?php
/* Начало транзакции, отключение автоматической фиксации */
$dbh->beginTransaction();
/* Вставка множества записей по принципу "все или ничего" */
$sql = 'INSERT INTO fruit    (name, colour, calories)    VALUES (?, ?, ?)';
$sth = $dbh->prepare($sql);
```

```
foreach ($fruits as $fruit) {
   $sth->execute(array( $fruit->name, $fruit->colour, $fruit->calories,   ));
}
/* Фиксация изменений */
$dbh->commit();
/* Соединение с базой данных снова в режиме автоматической фиксации */
?>
```

Вызов PDO::rollBack() откатит все изменения в базе данных, сделанные в рамках текущей транзакции, которая была создана методом PDO::beginTransaction(), и вернет соединение к режиму автоматической фиксации. Если активной транзакции нет, будет выброшено исключение PDOException.

В следущем примере создается транзакция и выполняются два запроса, которые модифицируют данные в базе, а затем база возвращается к исходному состоянию. В MySQL, тем не менее, выражение DROP TABLE автоматически фиксирует изменения, находящиеся внутри транзакции, поэтому ничего откачено не будет.

```
<?php
/* Начинаем транзакцию, выключаем автофиксацию */
$dbh->beginTransaction();

/* Изменяем схему базы данных и данные в таблицах */
$sth = $dbh->exec("DROP TABLE fruit");
$sth = $dbh->exec("UPDATE dessert
   SET name = 'hamburger'");

/* Осознаем свою ошибку и откатываем транзакцию */
$dbh->rollBack();
/* База данных возвращается в режим автофиксации */
?>
```

Для проверки есть ли активные транзакции в настоящее время внутри драйвера используется метод PDO::inTransaction().

2.5 Варианты заданий

Окончательный вариант задания должен быть реализован с использованием подготовленных выражений.

Вариант 1. Создать структуру базы данных для хранения информации об информационных ресурсах библиотеки (рисунок 2.3). Различают 3 вида ресурсов: книги, журналы, газеты. Книги характеризуются названием, уникальным номером (ISBN), издательством, годом издания, количеством страниц. У книги может быть произвольное количество авторов. Журналы характеризуются названием, годом выпуска, номером. Газеты характеризуются названием и датой выхода (день, месяц, год). Книги и журналы могут содержать дополнительные информационные ресурсы (например, компакт-диски), которые учитываются и регистрируются отдельно.

Сформировать запросы, которые будут выводить на экран информацию о:

- книгах, журналах и газетах с указанным именем;
- книгах, журналах и газетах за указанный временной период;
- книгах указанного автора.

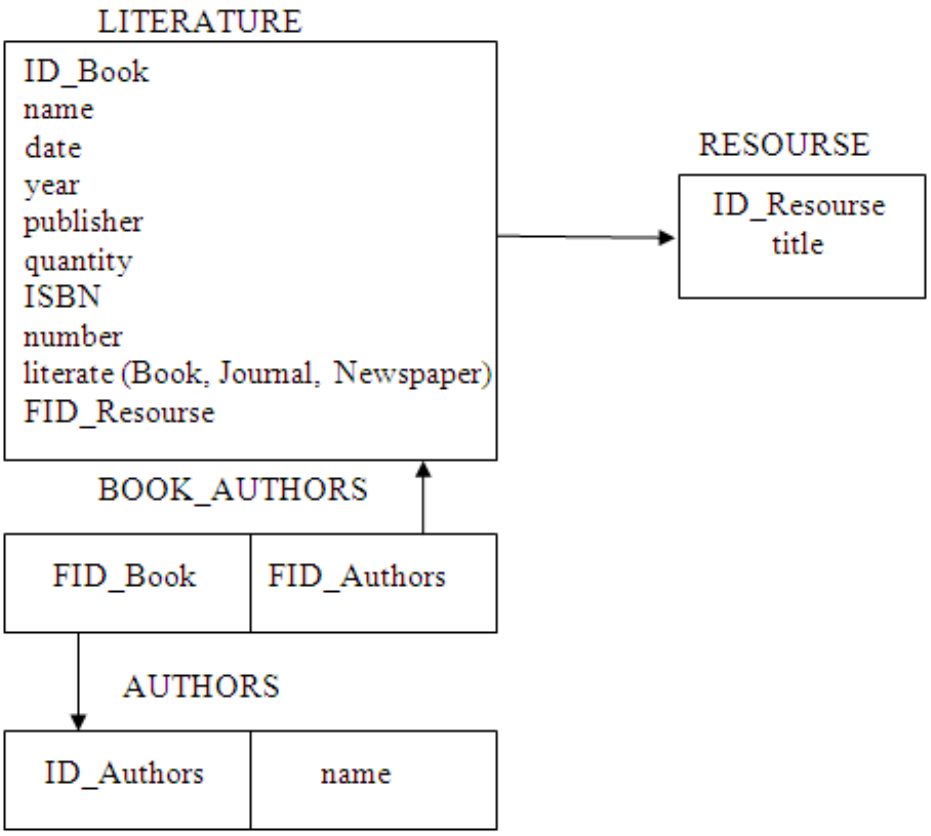

Рисунок 2.3 – Структура базы данных «Библиотека»

Вариант 2. Создать структуру базы данных для хранения информации о расписании проведения занятий (рисунок 2.4). Для каждого занятия задается день недели, номер пары, аудитория, группа, дисциплина, вид занятия. Различают 3 вида занятий – лекции (1 преподаватель ведет занятия с 1 и более группами), практические занятия (1 преподаватель работает с 1 группой), лабораторные занятия (2 преподавателя работают с 1 группой). Сформировать запросы и вывести расписание занятия для:

- произвольной группы из списка;
- произвольного преподавателя из списка;
- произвольной аудитории из списка.

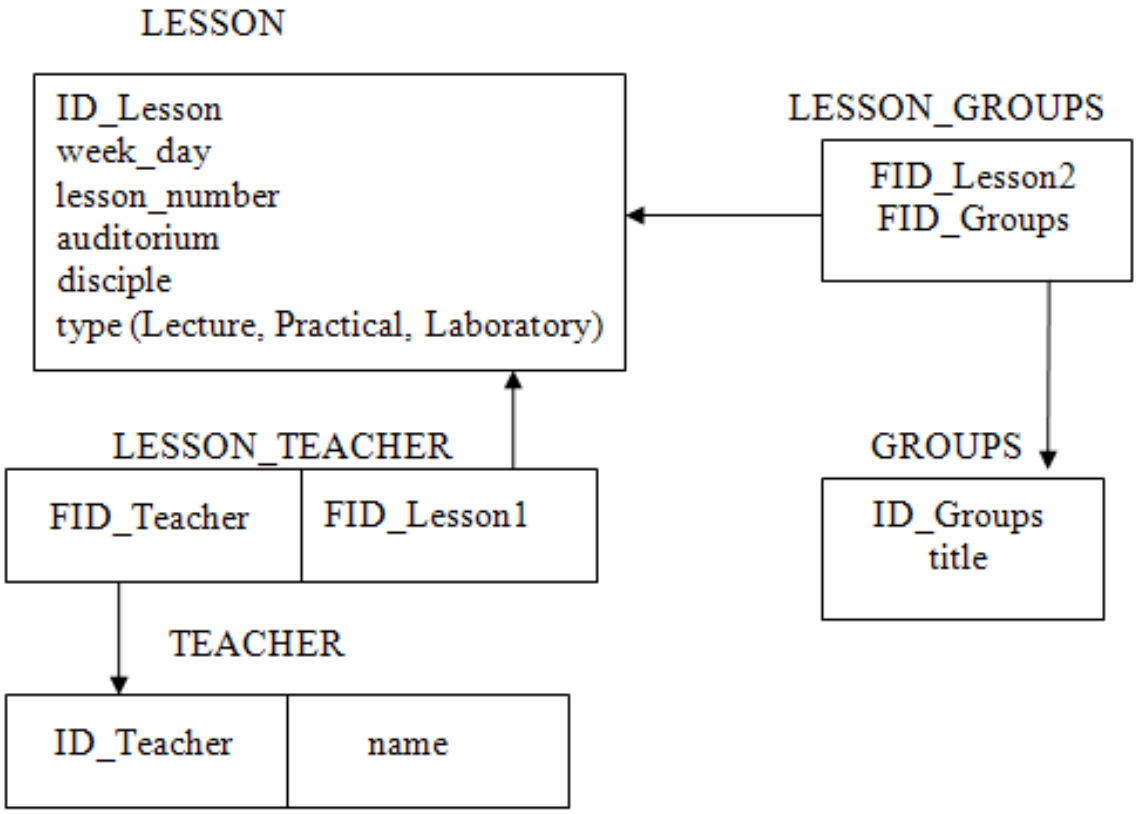

Рисунок 2.4 – Структура базы данных «Расписание занятий»

Вариант 3. Создать структуру базы данных для хранения информации о времени работы над проектом каждого сотрудника (рисунок 2.5). Различают 3 вида сотрудников:

- начальник отдела (может просматривать информацию о времени работы над проектами всех подчиненных);

- менеджер проекта (может просматривать информацию о времени работы над проектом каждого разработчика);

- разработчик – может помещать в базу информацию о времени работы над проектом.

Организация может одновременно вести несколько проектов. Каждый сотрудник может быть вовлечен в работу над несколькими проектами и выполнять разные роли (например, начальник отдела может быть менеджером одного проекта и разработчиком в другом).

Сформировать запросы и вывести результаты:

- время работы любого сотрудника;
- время работы над проектом всех вовлеченных сотрудников;
- время работы сотрудников выбранного отдела.

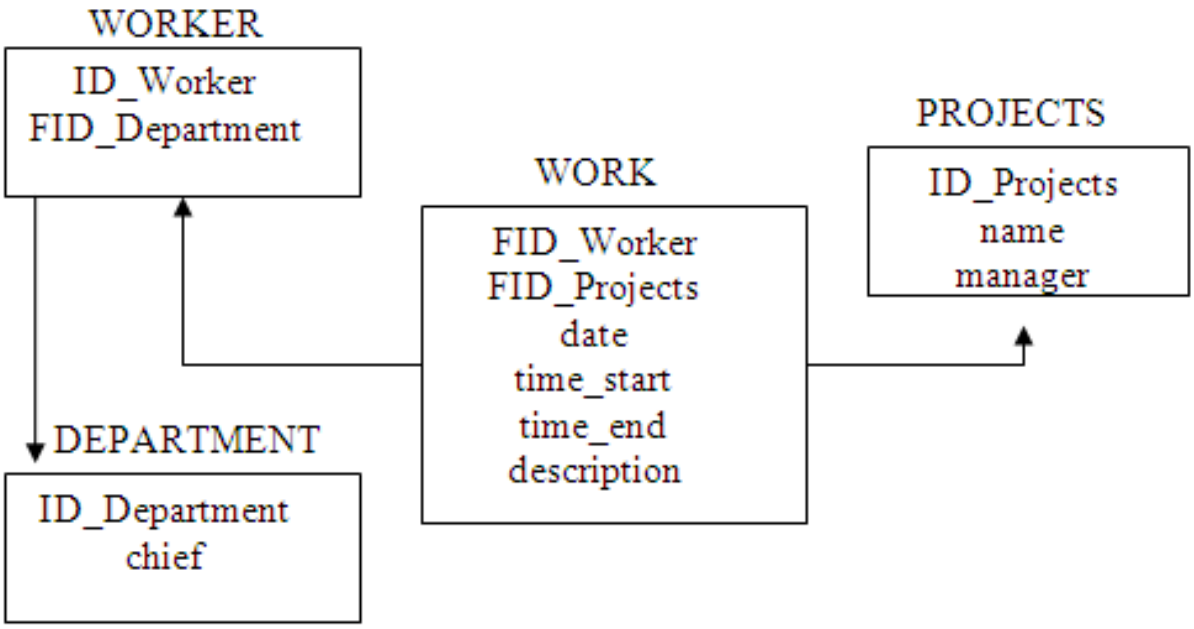

Рисунок 2.5 – Структура базы данных «Работа сотрудников»

Вариант 4. Создать структуру базы данных для хранения информации о сменах дежурств в поликлинике (рисунок 2.6). Для медсестры задается дата дежурства и смена (первая, вторая или третья), отделение, номера палат.

Сформировать запросы и вывести результаты:

- расписание дежурств выбранной медсестры;
- расписание дежурств по выбранному отделению;
- расписание дежурств по выбранной смене.

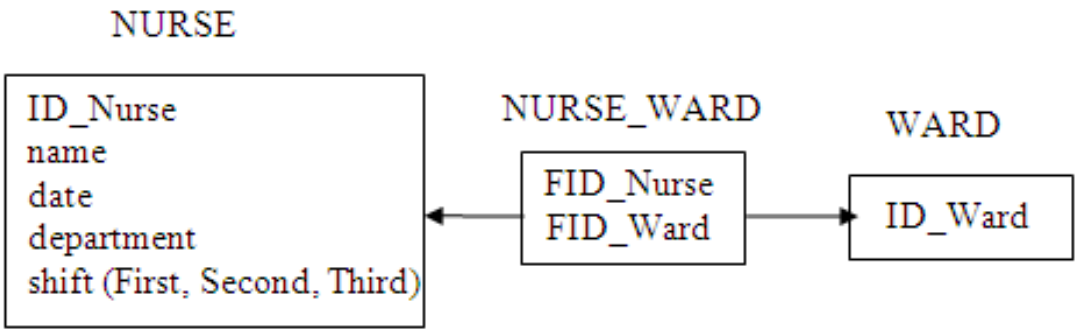

Рисунок 2.6 – Структура базы данных «Смена дежурств»

Вариант 5. Создать структур базы данных для хранения дополнительной информации о фильмотеке (рисунок 2.7). Для фильма задается название, жанр (может быть более чем один), год выхода, страна, качество, тип носителя (видеокассета, CD, DVD, BR). Для цифровых носителей определяется разрешение в пикселях и типы применяемых кодеков. Также для фильма может задаваться дополнительная информация, такая как: продюсер, режиссер и актеры (произвольное количество).

Сформировать запросы и вывести результаты:

- список фильмов по выбранной категории;
- список фильмов с выбранным актером;
- список фильмов за указанный временной интервал.

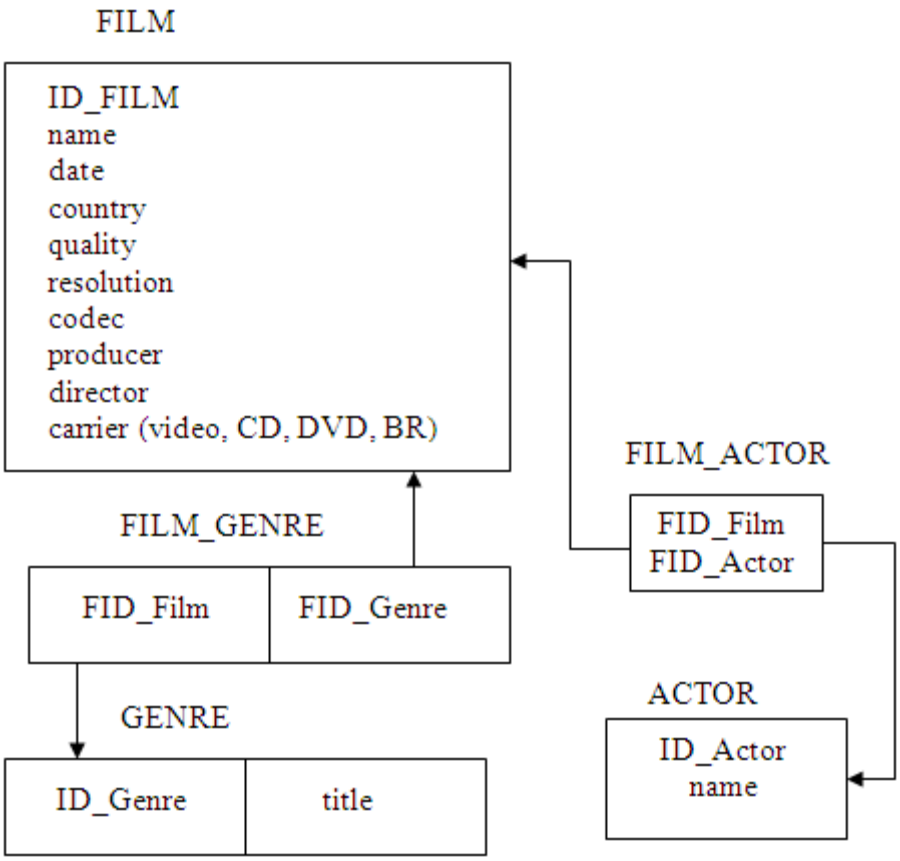

Рисунок 2.7 – Структура базы данных «Фильмотека»

Вариант 6. Создать структуру базы данных для хранения информации о товарах в интернет-магазине (рисунок 2.8). Для товара задается название, фирма-производитель, категория товара (процессоры, материнские платы и т.д.), цена товара, количество единиц на складе.

Сформировать запросы и вывести результаты:

- товары выбранного производителя;
- товары выбранной категории;
- товары в выбранном ценовом диапазоне.

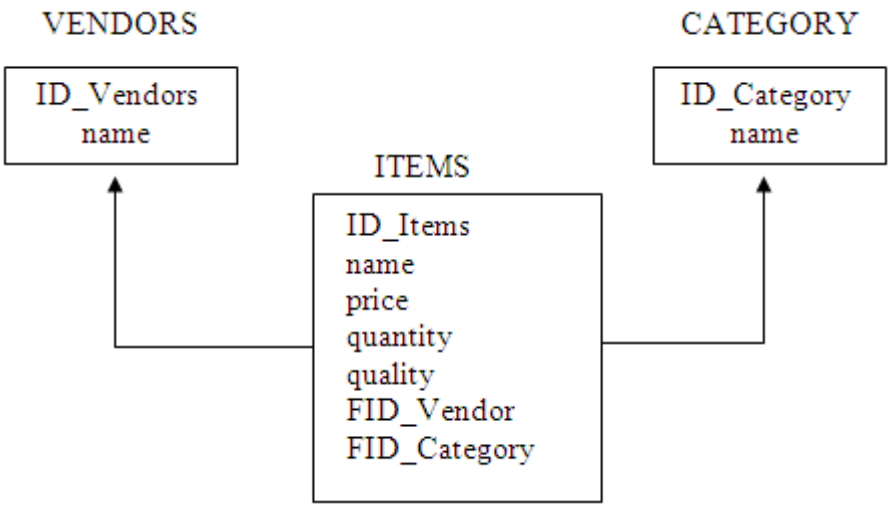

Рисунок 2.8 – Структура базы данных «Товары в магазине»

Вариант 7. Создать структуру базы данных для хранения информации о пункте проката машин (рисунок 2.9). В базе данных задается название машины, производитель, год выпуска, пробег, состояние, стоимость аренды (в час и за сутки), время аренды автомобиля.

Сформировать запросы и вывести результаты:

- автомобили указанного ценового диапазона;
- автомобили выбранного производителя;
- свободные автомобили на выбранную дату.

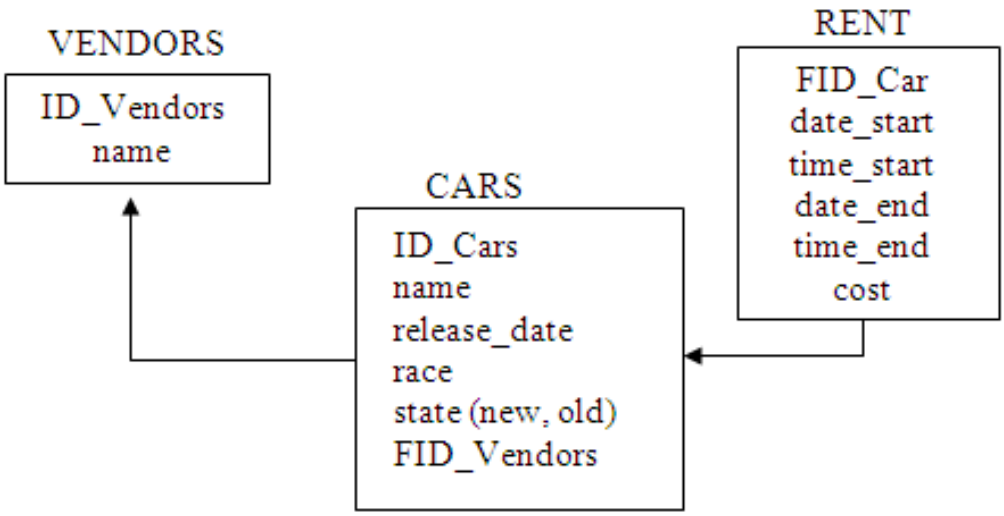

Рисунок 2.9 – Структура базы данных «Прокат машин»

Вариант 8. Создать структуру базы данных для хранения информации о результатах футбольного чемпионата (рисунок 2.10). Для каждой футбольной команды задается такая информация: название, лига, главный тренер. Для каждой игры задается дата проведения, команды участницы, место проведения, финальный счет, участвующие футболисты.

Сформировать запросы и вывести результаты:

- таблица чемпионата выбранной лиги;
- список игр на указанный временной интервал;
- список игр выбранного футболиста.

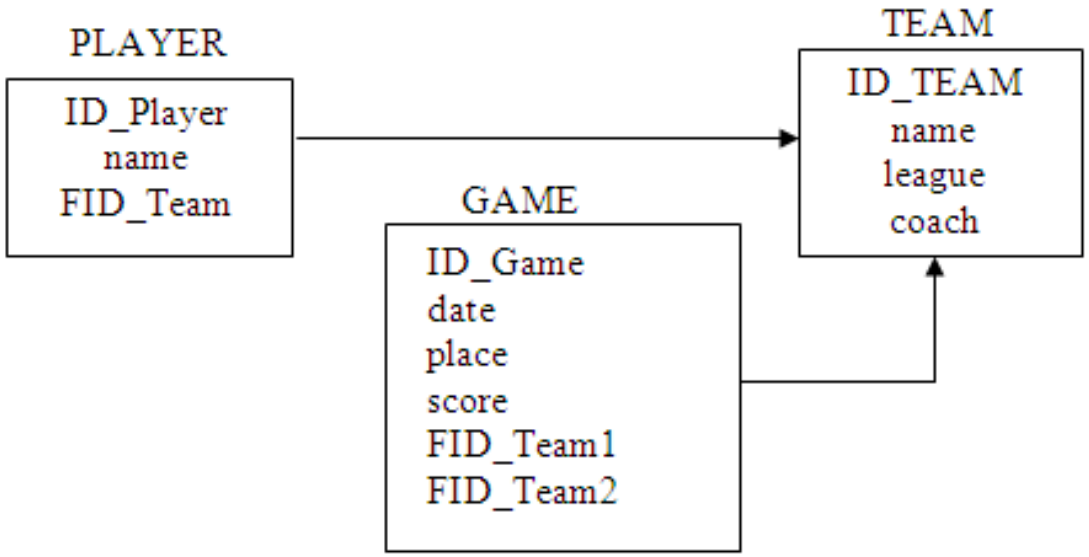

Рисунок 2.10 – Структура базы данных «Футбольный чемпионат»

Вариант 9. Создать структуру базы данных для хранения информации об использовании сетевого трафика (рисунок 2.11). Для каждого клиента задается логин, пароль, статический IP-адрес машины клиента, баланс счета. Для сеансов работы задается время начала и окончания сеанса работы, количество входящего трафика, количество исходящего трафика.

Сформировать запросы и вывести результаты:

- статистику работы в сети выбранного клиента;
- статистику работы в сети за указанный промежуток времени;
- вывести список клиентов с отрицательным балансом счета.

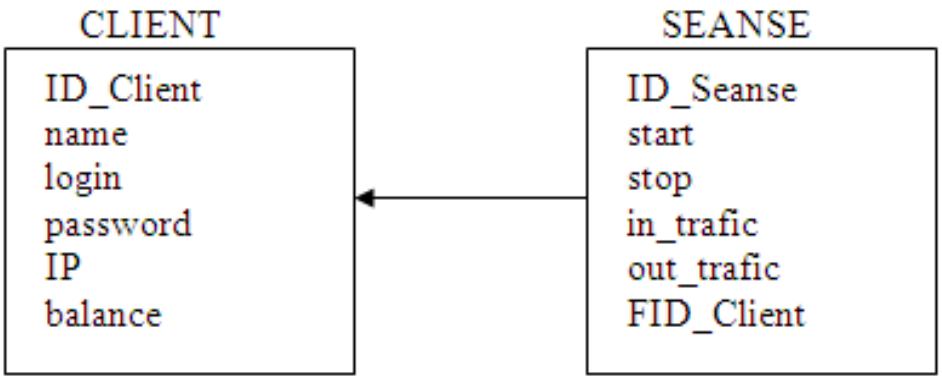

Рисунок 2.11 – Структура базы данных «Сетевой трафик»

Вариант 10. Создать структуру базы данных для хранения информации о компьютерах организации (рисунок 2.12). Для каждого компьютера задается сетевое имя, тип центрального процессора, тип материнской платы, объем ОЗУ и НЖМД, тип монитора, установленное программное обеспечение, фирма-продавец, дата покупки, срок гарантии.

Сформировать запросы и вывести результаты:

- список компьютеров с заданным типом центрального процессора;
- список компьютеров с установленным ПО (название ПО выбирается из перечня);
- список компьютеров с истекшим гарантийным сроком.

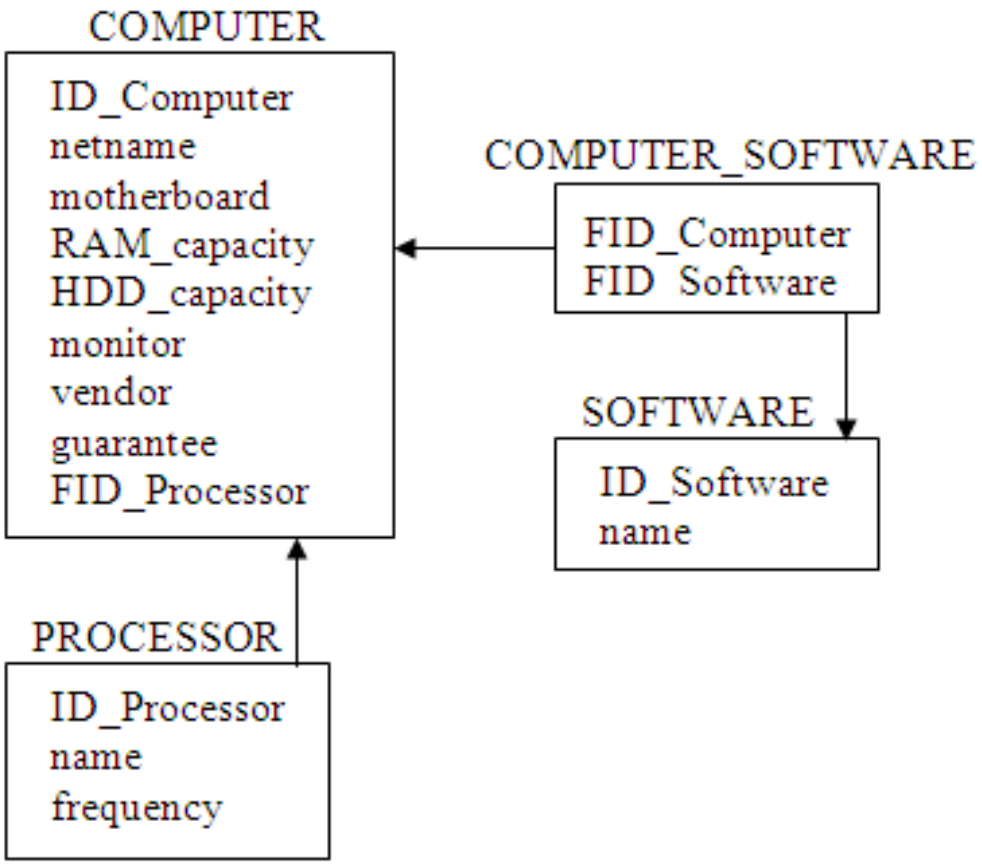

Рисунок 2.12 – Структура базы данных «Компьютеры организации»

2.6 Содержание отчета

- цель лабораторной работы;
- задание на лабораторную работу;
- ER-диаграмма структуры базы данных;
- дамп базы данных, сформированный PHPMyAdmin;
- исходные коды PHP – скриптов;
- результаты выполнения запросов в виде снимков экрана браузера;
- выводы по работе.

2.7 Контрольные вопросы и задания

1) В чем преимущество применения интерфейса доступа к базам данных PDO?
2) Каким образом осуществляется управление подключениями к базе данных?
3) Как выполняется обработка ошибок с помощью расширения PDO?
4) Пояснить принципы использования хранимых процедур и обращения к ним средствами PDO.
5) Для каких целей предназначены подготовленные запросы?
6) Определение транзакции, пример использования?

3 ПРОГРАММНЫЕ МЕХАНИМЫ ШАБЛОНИЗАЦИИ WEB-ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ SMARTY

3.1 Цель работы

Изучение синтаксиса и принципов работы с компилирующим механизмом шаблонов Smarty.

3.2 Методические указания по организации самостоятельной работы студентов

При подготовке к выполнению лабораторной работы следует повторить основные концепции модели объектно-ориентированного программирования, а также шаблоны проектирования приложений [5]. Кроме того, требуется ознакомиться с базовым синтаксисом объявления переменных, условий и циклов в компилирующем механизме шаблонов Smarty, а также программные интерфейсы соответствующего класса [7,8].

3.3 Программное обеспечение ПК

При выполнении лабораторной работы используется ПЭВМ под управлением операционной системы Windows XP и старше, веб-сервер xampp 2.4.4, шаблонизатор Smarty, среда разработки Notepad++ 6.4.5.

3.4 Методические указания по выполнению лабораторной работы

При подготовке к выполнению лабораторной работы следует повторить синтаксис объявления переменных, условий и циклов в компилирующем механизме шаблонов Smarty, а также программные интерфейсы соответствующего класса.

Далее будут рассмотрены основные принципы работы с классом smarty.

Практика использования языка PHP для проектирования сложных и масштабируемых интернет приложений говорит о необходимости отделения данных от представления этих данных. Поскольку язык PHP является встраиваемым и, фактически, смешивает код вывода данных и код отображения данных. Это существенно усложняет задачу как разработчика интернет-приложения как такового, поскольку ему требуется искать вкрапления PHP-кода в достаточно большой HTML-странице, так и дизайнеру страницы. Фактически, даже небольшое изменение дизайна страниц для большого корпоративного сайта, не использующего отделение представления от вывода данных, является существенной проблемой. Для решения данной задачи разработан целый ряд механизмов шаблонов, которые позволяют хранить шаблон HTML-страницы отдельно от логики интернет-приложения, разделяя файлы на группы в соответствии с теми функциями, которые они выполняют в приложении (рисунок 3.1). При этом можно относительно легко изменять и сам код логики приложения и дизайн страницы.

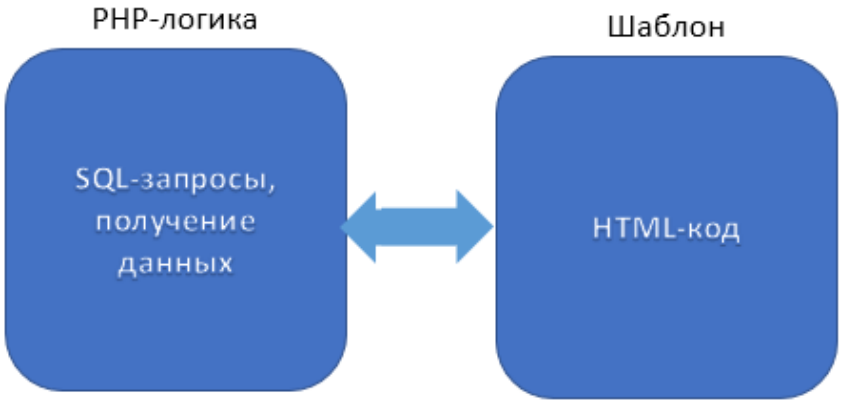

Рисунок 3.1 – Схема разработки приложения с примененеми клиентских шаблонов

Smarty – это компилирующий обработчик шаблонов для PHP. Говоря более четко, он предоставляет один из инструментов, которые позволяет добиться отделения прикладной логики и данных от представления. Это очень удобно в ситуациях, когда программист и верстальщик шаблона - различные люди.

Одно из предназначений Smarty - это отделение логики приложения от представления.

Шаблоны могут содержать в себе логику, но лишь при условии, что эта логика необходима для правильного представления данных. Такие задачи, как подключение других шаблонов, чередующаяся окраска строчек в таблице, приведение букв к верхнему регистру, циклический проход по массиву для его отображения и т.д. - всё это примеры логики представления.

Тем не менее, не следует полагать, что Smarty заставляет вас разделять прикладную логику и логику представления. Smarty не видит разницы между этими вещами, так что переносить прикладную логику в шаблоны вы можете на свой страх и риск.

Одна из уникальных возможностей Smarty – компилирование шаблонов. Это означает, что Smarty читает файлы шаблонов и создает PHP-код на их основе. Код создаётся один раз и потом только выполняется. Поэтому нет необходимости в медленной обработке файл шаблона для каждого запроса. Каждый шаблон может пользоваться всеми преимуществами таких компиляторов PHP и кэширующих решений, как eAccelerator, ionCube, mmCache, Zend Accelerator и прочих.

Некоторые особенности Smarty:

– высокая скорость и эффективность механизма шаблонов;

– никакой лишней обработки шаблонов, они компилируются только один раз;

– перекомпилируются только те шаблоны, которые изменились;

– возможность создания собственных пользовательских функций и модификаторов переменных, что делает язык шаблонов чрезвычайно расширяемым;

– настраиваемые {разделители} тэгов шаблона, то есть возможность использования {$foo}, {{$foo}}, <!--{$foo}--> и т.д.;

– конструкции {if}..{elseif}..{else}..{/if} передаются обработчику PHP, так что синтаксис выражения {if...} может быть настолько простым или сложным, насколько вам угодно;

– допустимо неограниченное вложение секций, условий и т.д;

– существует возможность включения PHP-кода прямо в ваш шаблон, однако обычно в этом нет необходимости (и это не рекомендуется), так как движок весьма гибок и расширяем;

– встроенный механизм кэширования;

– произвольные источники шаблонов;

– пользовательские функции кэширования;

– компонентная архитектура.

Переменные класса Smarty.

В Smarty для установки значений компиляции, кеширования, отладки и др. настроек используются следующие переменные класса Smarty:

– $compile_dir – имя каталога, в котором хранятся компилированные шаблоны;

– $config_dir – каталог для хранения конфигурационных файлов, используемых в шаблонах;

– $debugging – активирует debugging console, порожденное при помощи Javascript окно браузера, содержащее информацию о подключенных шаблонах и загруженных переменных для текущей страницы;

– $compile_check при каждом вызове PHP-приложения Smarty проверяет, изменился или нет текущий шаблон с момента последней компиляции;

– $cashing – сообщает Smarty будет или нет кеширвоаться вывод шаблонов;

– $cashe_dir – имя каталог, в котором хранится кеш шаблонов;

– $error_reporting, если это свойство имеет ненулевое значение, то оно используется в качестве значения error_reporting внутри display() и fetch(). При включенном режиме отладки это значение игнорируется и уровень обработки ошибок не меняется.

Методы класса Smarty.

Методы класса Smarty используются в php скриптах:

– append() - добавляет элемент к назначенному массиву;

– append_by_ref() - добавляет значение по ссылке;
– assign() - назначает значение шаблону;
– assign_by_ref() - назначает переменную по ссылке;
– clear_all_assign() - очищает список назначенных переменных;
– clear_all_cache() - полностью очищает кэш шаблонов;
– clear_assign() - очищает назначенную переменную;
– clear_cache() - очищает кэш определенного шаблона;
– clear_compiled_tpl() - очищает скомпилированную версию указанного шаблона;
– clear_config() - очищает назначенную конфигурационную переменную;
– config_load() - загружает данные из конфигурационного файла и назначает их шаблону;
– display() - отображает шаблон.

Базовый синтаксис шаблонов.

Все тэги шаблонов Smarty располагаются между специальными разделителями (по умолчанию это { и }, но символы-разделители могут быть изменены). Для дальнейших примеров будут использоваться стандартные разделители. Smarty все содержимое вне разделителей отображает как статический контент, без изменений. Когда Smarty встречает тэги шаблона, то пытается интерпретировать их и вывести вместо них соответствующий результат.

Комментарии

Комментарии в шаблонах заключаются в звездочки (*) окруженные разделителями, например: {* это комментарий *}. Комментарии не отображаются в выводе шаблона (не существуют в скомпилированном выводе), в отличие от <!-- комментариев HTML -->, и они используются для внутренних примечаний в шаблонах, которые никто не увидит.

Переменные шаблона начинаются со знака $доллара (таблица 3.1) Они могут состоять из цифр, букв, знаков подчёркивания - как и обычные PHP variable. Вы можете обращаться к массивам по числовым и нечисловым индексам. Вы также можете

обращаться к свойствам и методам объектов. Переменные конфигурационного файла - это исключения из долларового синтаксиса; к ним можно обращаться, окружив их #решетками# или воспользовавшись специальной переменной $smarty.config.

Переменные запроса, такие как $_GET, $_SESSION и др. также доступны, отображение которых показано в коде ниже.

```
{*отображение параметра page  из формы ($_POST[‘page’])*}
{$smarty.post.page}
{*отображение значения cookie “username”($_COOKIE[‘username’])*}
{$smarty.cookie.username}
```

Smarty распознает присвоенные переменные, если они встречаются в строках, заключенных в двойные кавычки, если имена переменных состоят из цифр, букв, знака подчеркивания и квадратных скобок. В случае, если переменная содержит другие символы (точки, ссылки и т.д.) переменные необходимо заключить в обратные кавычки.

Таблица 3.1 – Примеры переменных Smarty

{$foo}	отображение простой переменной (не массив и не объект)
{$foo[4]}	отображает 5-й элемент числового массива
{$foo.bar}	отображает значение ключа "bar" ассоциативного массива, подобно PHP $foo['bar']
{$foo.$bar}	отображает значение переменного ключа массива, подобно PHP $foo[$bar]
{$foo->bar}	отображает свойство "bar" объекта
{$foo->bar()}	отображает возвращаемое значение метода "bar" объекта
{#foo#} {$smarty.config.foo}	отображает переменную "foo" конфигурационного файла

Практические примеры предствлены ниже

{include file=”subdir/$tpl_name.tpl”} {*заменит $tpl_name на ее значение*}

{cycle values=”one, two, ‘smarty.config.myval’”} {*заключение в обратные кавычки*}

Пример полноценного приложения, написанного с применением Smarty и применяющего переменные, назначенные из PHP, состоит из HTML-страницы с тегами Smarty и логики работы в PHP-скрипте.

HTML-страница с тегами Smarty (index.tpl) показана ниже.

```
<!DOCTYPE html >
<html>
<head>
  <title>{$title_text}</title>
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
</head>

<body> {* Комментарий, которого не будет в HTML коде *}
 Hello {$firstname}! <br/>
<p>{$body_text}</p>
{$contacts.email} <br/>
{$details[0]} <br/>
{$details[1][0]} <br/>
</body><!-- Комментарий, который будет в HTML коде -->
</html>
```

Логика работы в PHP-скрипте (index.php) может быть реализована в виде, предложенном ниже.

```
define('SMARTY_DIR', 'smarty-2.6.9/' );
require_once(SMARTY_DIR . 'Smarty.class.php');
```

```
$smarty = new Smarty();
$smarty->template_dir = './templates/';
$smarty->compile_dir = './templates/compile/';
$smarty->cache_dir = './templates/cache/';
$smarty->caching = false;
$smarty->error_reporting = E_ALL; // LEAVE E_ALL DURING DEVELOPMENT
$smarty->debugging = true;

$smarty->assign('title_text', 'TITLE: Пример использования Smarty ...'); // установка значений переменных в шаблоне
$smarty->assign('body_text', 'BODY: Это текст, выведенный ф-цией assign()');
$smarty->assign(‘firstname’, ‘Udjin’);
$smarty->assign(‘contacts’, array(‘email’=>’example@mail.ru’, ‘phone’=>’555-444-33’));
$smarty->assign(‘details’, array(‘address’, array(‘cell’, ‘phone’) ));

$smarty->display('index.tpl'); // вывод готового шаблона HTML-страницы
```

Модификаторы переменных могут быть применены к переменным, пользовательским функциям или строкам. Для их применения надо после модифицируемого значения указать символ | (вертикальная черта) и название модификатора. Так же модификаторы могут принимать параметры, которые влияют на их поведение. Эти параметры следуют за названием модификатора и разделяются двоеточием (:).

Примеры модификаторов показаны ниже.

```
{* применение модификатора к переменной*}
{* переводит строку в верхний регистр, lower – в нижний регистр *}
{$title|upper}
{*преобразует первые буквы каждого в переменной слова в заглавные*}
```

```
{articleTitle|capitalize}
{*удаляет теги разметки*}
{$articleTitle|strip_tags}

{* модификатор с параметрами *}
{*обрезает переменную до определенной длины, по умолчанию – 80 символов*}
{$title|truncate:40:'...'}
{*поиск и замена при помощи регулярного выражения в переменной*}
{$articleTitle|regex_replace:”[a-z]”}
{*простой поиск и замена в переменной*}
{$articleTitle|replace:” ’:’ ”}
{*заменяет все повторяющиеся пробелы, переводы строк и символы табуляции одним пробелом или другой указанной строкой *}
{$articleTitle|strip:”  ”}
{* использование date_format для форматирования текущей даты *}
{$smarty.now|date_format:"%Y/%m/%d"}
{*используется для кодирования/экранировая спецсимовлов*}
{$articleTitle|escape:’htmlall’}

{* применение модификатора к аргументу функции *}
{html_table loop=$myvar|upper}
{* применение модификатора к строке *}
{'foobar'|upper}
{* применение модификатора к функции *}
{mailto|upper address='smarty@example.com'}
{* использование функции PHP str_repeat *}
{'='|str_repeat:80}
```

Встроенные функции.

Конструкция {if} в Smarty такая же гибкая, как и конструкция if в PHP, только с несколькими дополнительными возможностями для шаблонов. Каждый тэг {if} должен иметь пару {/if}. {else} и {elseif} так же допустимы. Досутпны все квалификаторы и функции из PHP, такие как ||, or, &&, and, is_array() и т.д.

```
{if $name eq 'Fred'}
        Welcome Sir.
{elseif $name eq 'Wilma'}
   Welcome Ma'am.
{else}
   Welcome, whatever you are.
{/if}
{* пример с логикой "или" *}
{if $name eq 'Fred' or $name eq 'Wilma'}
{* то же самое, что и выше {if $name == 'Fred' || $name == 'Wilma'}*}
...  {/if}
```

Секции используются для обхода массивов данных (также, как и foreach), аргументы цикла показаны в таблице 3.2.

Каждый тег {section} дожен иметь пару{/section}. Переменная loop (обычно-массив занчений) определяет количество итераций цикла. При печати переменных внутри секции, имя секции должно быть указано рядом с именем переменной внутри квадратных скобок []. Ветвь {sectionelse} выполняется в том случае, если параметр loop не содержит значений.

Таблица 3.2 – Аргументы (параметры) цикла {section}

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
name	string	Да	n/a	Название секции
loop	mixed	Да	n/a	Значение, определящее количество итераций цикла
start	integer	Нет	0	Индекс позиции, с которой будет начинаться цикл. Если значение отрицательное, то начальная позиция вычисляется от конца массива.
step	integer	Нет	1	Значение шага, которое используется для прохода по массиву. Если шаг отрицателен, то обход массива будет производиться в обратном направлении.
max	integer	Нет	1	Максимальное количество итераций
show	boolean	Нет	true	Указывает, показывать или нет эту секцию

Применение цикла {section} в работе показан ниже.

Назначение массива в PHP-скрипте может быть реализовано в виде, предложенном ниже.

```
<?php
$smarty->assign('custid', array(1000,1001,1002));
$smarty->assign('name', array(‘John’, ‘Jack’, ‘Jane’));
?>
```

Шаблон для вывода значений массива показан ниже.

```
{section name=customer loop=$custid}
<p>
```

id: {$custid[customer]}

id: {$name[customer]}

{/section}

Результат выполнения данного примера.

id:1000

name: John

id:1001

name: Jack

id:1002

name: Jane

Применение цикла {foreach} при работе с базой данных показан ниже.

HTML-страница с тегами Smarty для создание выпадающего списка показана ниже.

<!DOCTYPE html >

<html><body>

<select name="post_genre">

{section name=genre loop=$genres}

<option value={$genres[genre].title}>{$genres[genre].title}</option>

{/section}

</body></html>

Логика работы в PHP-скрипте может быть реализована в виде, предложенном ниже.

```
<?php
$sth = $dbh->query('select ID_Genre, title from GENRE');
$smarty->assign('genres', $sth->fetchAll());
?>
```

Секции так же имеют собственные переменные, которые содержат свойства секций. Они обзначаются как {$smarty.section.sectionname.varname}. Переменные цикла section: index, index_prev, index_next, iteration, first, last, rownum, loop, show, total, {strip}.

Например, переменная index используется для отображения текущего индекса массива, начиная с нуля (или с атрибута start) и увеличиваясь на единицу (или на значение атрибута step).

```
{section name=customer loop=$custed}
{$smarty.section.customer.index} id: {$custid[customer]}<br/>
{/section}
```

Циклы {foreach} являются альтернативой циклам {section}. Цикл {foreach} используется для прохода по единственному ассоциативному массиву, аргументы цикла показаны в таблице 3.3. Синтаксис {foreach} намного проще синтаксиса {section}, но с другой стороны его можно использовать только для одного массива. Каждый тэг {foreach} должен иметь пару {/foreach}. Обязательными параметрами являются from и item. Имя цикла {foreach} может быть любым, состоящим из букв, цифр и знаков подчеркивания. Циклы {foreach} могут быть вложенными и имена вложенных {foreach} должны быть уникакльными между собой. Параметр from (обычно - массив значений) определяет количество итераций цикла {foreach}. Ветвь {foreachelse} выполняется в том случае, если параметр from не содержит значений.

Таблица 3.3 – Аргументы (параметры) цикла {foreach}

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
from	array	Да	n/a	Массив, по которому надо пройтись
item	string	Да	n/a	Имя переменной, которая будет выступать в качестве значения текущего элемента
key	string	Нет	n/a	Имя переменной, которая будет выступать в качестве ключа текущего элемента
name	string	Нет	n/a	Название цикла foreach для доступа к его свойствам

Применение цикла {foreach} при работе с контактами показан ниже.

Назначение массива в PHP-скрипте может быть реализовано в виде, предложенном ниже.

```
<?php
$smarty->assign('contacts', array(
                array('phone' => '1', 'fax' => '2', 'cell' => '3'),
                array('phone' => '555-4444',
                        'fax' => '555-3333',
                                'cell' => '760-1234')
                ));
?>
```

Шаблон для вывода значений массива показан ниже.

```
{foreach name=outer item=contact from=$contacts}
   <hr />
   {foreach key=key item= key from=$contact}
```

```
        {$key}: {$key}&lt;br&gt;
        {$key}: {$key}<br />
    {/foreach}
{/foreach}
```

Результат выполнения данного примера.

```
<hr />
    phone: 1<br />
    fax: 2<br />
    cell: 3<br />
<hr />
    phone: 555-4444<br />
    fax: 555-3333<br />
    cell: 760-1234<br />
```

Применение цикла {foreach} при работе с базой данных показан ниже.

HTML-страница с тегами Smarty для создание выпадающего списка показана ниже.

```
<!DOCTYPE html >
<html><body>
<select name="post_genre">
  {foreach item=genre from=$genres}
   <option value={$genre[1]}>{$genre[0]}</option>
  {/foreach}
 </select>
</body></html>
```

Логика работы в PHP-скрипте может быть реализована в виде, предложенном ниже.

```
<?php
$sth = $dbh->query('select ID_Genre, title from GENRE');
```

```
$smarty->assign('genres', $sth->fetchAll());
?>
```

Пользовательские функции.

Smarty поставляется с несколькими пользовательскими функциями, которые вы можете использовать в шаблонах:

- {assign} используется для установки значения переменной в процессе выполнения шаблона:

```
{assign var=”name” value=”Bob”}
```

- {cycle} используется для прохода по циклу набора переменных;
- {fetch} используется для отображения содержимого локальных файлов, http или ftp страниц;
- {html_checkboxes} создает группу флажков в HTML по указанной информации

```
<?php
$smarty->assign (‘cust_ids’, array(1000,1001,1002));
$smarty->assign (‘cust_names’, array(‘Joe’,’Jack’,’Jane’));
$smarty->assign (‘customer_id’, 1001);
?>
{html_checkboxes name=’id’ values=$cust_ids output=$cust_names selected=$customer_id separator=”<br/>”}
```

– {html_image} создает HTML-теги для изображений;

– {html_options} создает группу HTML-тегов option по указанной информации. Традиционный пример с использованием цикла foreach для создания выпадающего списка, как показано ниже, может быть заменен пользовательской функцией.

```
<select name="post_fio">
{html_options values=$fis output=$fis|upper| selected=$fis[0]}
 </select>
```

– {html_radios} создает группу радиокнопок в HTML по указанной информации;

– {html_table} распечатывает массив данных в HTML-тег table;

– {mailto}создает ссылки mailto: и опционально кодирует их;

– {math} позволяет дизайнерам шаблонов проводить математические вычисления в шаблоне;

– {popup} используется для создания всплывающих окон при помощи javascript;

– {textformat} – блоковая функция, используемая для форматирования текста.

3.4 Пример выполнения лабораторной работы

Рассмотрим структуру базы данных для хранения дополнительной информации о фильмотеке. ER-диаграмма базы данных изображена на рисунке 3.1.

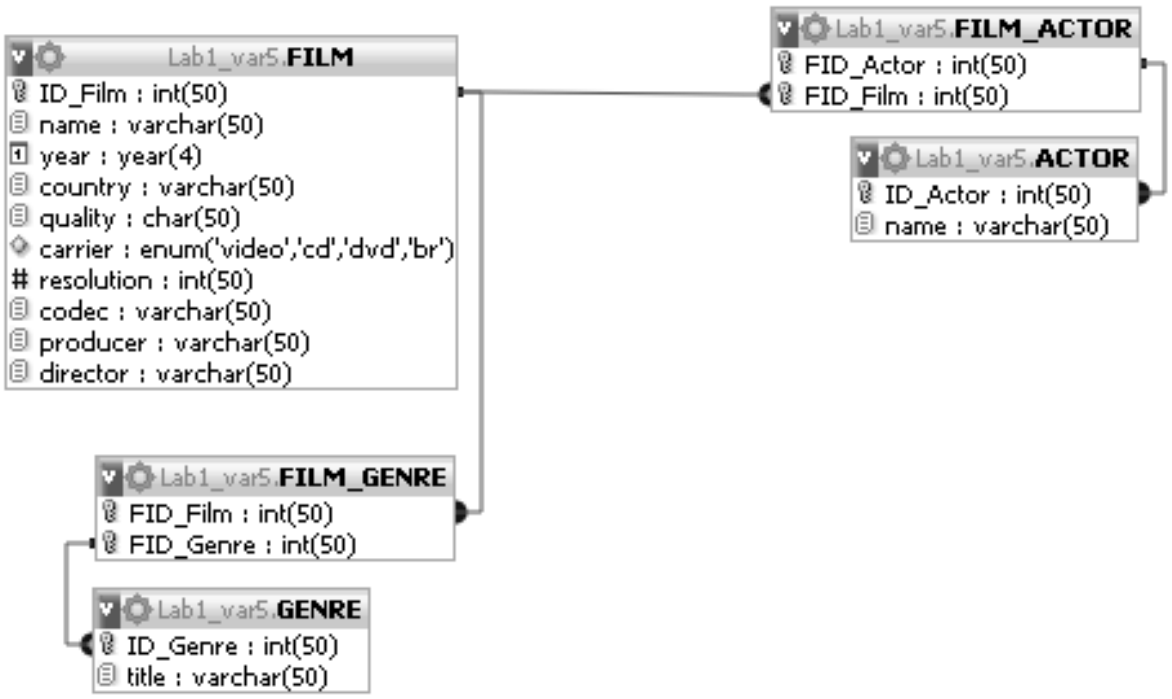

Рисунок 3.1 – ER-диаграмма базы данных автомобилей

Необходимо реализовать упроценный веб-интерфейс к базе данных, показанной на рисунке 3.1 с использованием шаблонов Smarty. Веб-интерфейс содержит заполненное поле ввода, кнопку, по нажатию которой происходит выборка данных по запросу из базы данных, и таблицу для вывода результатов.

На рисунках 3.2, 3.3, 3.4 представлено содержание таблиц FILM, FILM_GENRE, GENRE, они учавствуют в итоговом запросе для создания результирующей таблицы.

ID_Film	name	year	country	quality	carrier	resolution	codec	producer	director
1	Американец	2010	США	высокое	cd	230	кодек№1	Алан Бернон	Элизабет Авеллан
2	Призрак	2009	Канада	среднее	dvd	430	кодек№2	Робер Бенмусса	Роман Полански

Рисунок 3.2 – Содержание таблицы FILM

FID_Film	FID_Genre
1	1
2	2

Рисунок 3.3 – Содержание таблицы FILM_GENRE

ID_Genre	title
1	драма
2	комедия

Рисунок 3.4 – Содержание таблицы GENRE

Исходный код PHP-скрипта, представленный ниже, позволяет изучить основные механизмы передачи данных в шаблон.

```
<?php
require_once './libs/Smarty.class.php';
$smarty = new Smarty;
$smarty->compile_check = true;
$smarty->debugging = true;

 // параметры соединения с базой данных
$dbh = new PDO('mysql:host=localhost;dbname=Lab1_var5', 'root', '');
$dbh->exec("SET CHARACTER SET 'CP1251'");
 error_reporting(E_ERROR ^ E_DEPRECATED);
  if (isset($_POST['f1submit']))
```

```
{       // формирование результата
$rth = $dbh->prepare('SELECT distinct a.name, country, quality, carrier FROM FILM a, FILM_GENRE b, GENRE WHERE ID_Film=b.FID_Film and ID_Genre=FID_Genre and title=:title');
 $rth->bindParam(':title', $_POST['post_genre']);
$rth->execute();
$result= $rth->fetchAll();
}
 // передача значений в шаблон
 $sth = $dbh->query('select ID_Genre, title from GENRE');
 $smarty->assign('result',$result);
// компиляция шаблона
$smarty->display('index1.tpl');
$dbh=null;
?>
```

Исходный код шаблона Smarty является примером реализации концепции разделения файлов на группы в соответствии с теми функциями, которые они выполняют в приложении.

```
{config_load file=test.conf section="setup"}
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
<H1>Film Database</H1>
{*форма поиска по фильмам выбранной категории*}
<form action="index1.php" method="post" name="form1">
 <h2>search by genre</h2>
 <input type="text" size="40"  name="post_genre">
 <input type="submit" name="f1submit" value="search">
</form>
```

```
<h2>search result</h2>
<select name="post_genre">
  {foreach item=result from=$result }
   <option value={$result[0]}> {$result[0]}</option>
  {/foreach}
 </select>
</body>
</html>
```

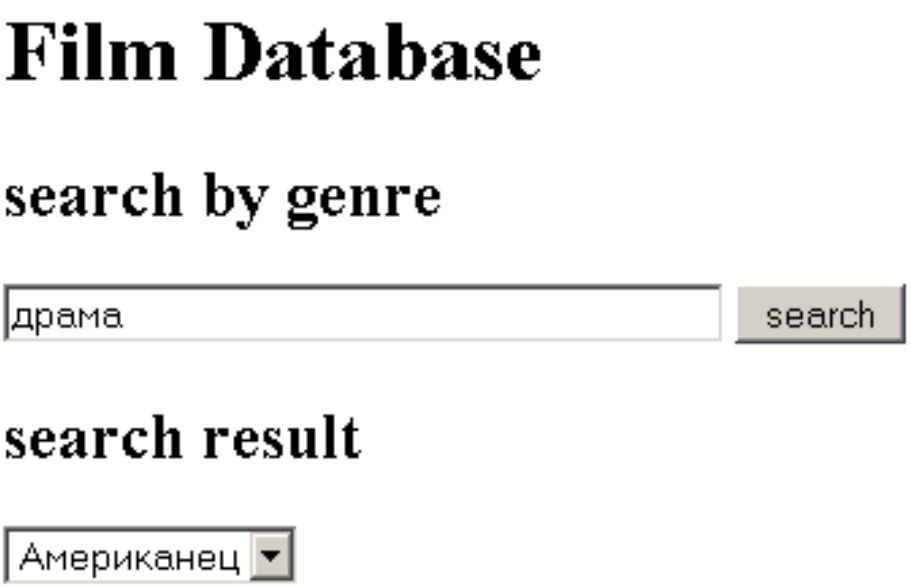

Рисунок 3.5 – Результат выполнения программы

3.6 Варианты заданий

Разработать веб-интерфейс к базе данных, аналогичный реализованному в лабораторной работе №2, с использованием компилирующего механизма шаблонов Smarty. Окончательный вариант задания должен быть реализован с пирменением различных методов выборки данных для веб-интерфейса к базе данных.

3.7 Содержание отчета

Отчет по лабораторной работе должен содержать:

- тему и цель работы;
- задание на лабораторную работу;
- ER-диаграмму структуры базы данных (из лабораторной работы №1);
- исходные коды PHP-скриптов и Smarty-шаблонов;
- результаты выполнения скриптов в виде снимков экрана браузера;
- выводы по работе.

3.8 Контрольные вопросы и задания

1) Какие отличительные особенности имеет механизм шаблонов Smarty по сравнению с HTML_Template_Foxy?
2) Какие возможности для вывода сложного содержимого предоставляет дизайнеру страниц Smarty?
3) Как осуществляется настройка кэширования шаблонов Smarty?
4) В чем отличие пользовательских и конфигурационных переменных?
5) В чем недостаток смешивания кода и представления?
6) Какие функции выполняют элементы архитектуры Model-View-Controller?
7) Какие основные отличия шаблонов и каркасов?
8) В каких случаях оправданным является использование CMS (Content Management System) и CMF (Content Management Framework)?

4 АСИНХРОННЫЙ ОБМЕН ДАННЫМИ С СЕРВЕРОМ НА ОСНОВЕ ТЕХНОЛОГИИ AJAX

4.1 Цель работы

Изучить программные механизмы технологии AJAX для динамической отправки или загрузки данных с сервера.

4.2 Методические указания по организации самостоятельной работы

При подготовке к выполнению лабораторной работы следует ознако-мится с особенностями синхронного и асинхронного методов передачи данных, изучить форматы обмена данными JSON и XML, а также повто-рить методы и свойства класса XMLHttpRequest [9].

4.3 Программное обеспечение ПК

При выполнении лабораторной работы используется ПЭВМ под управлением операционной системы Windows XP и старше, веб-сервер xampp 2.4.4, среда разработки Notepad++ 6.4.5.

4.4 Методические указания по выполнению лабораторной работы

При разработке Web-приложений существует два способа обмена данными клиентских приложений с сервером: синхронный и асинхронный. AJAX –аббревиатура, которая означает Asynchronous Javascript and XML, не является новой технологией, так как и Javascript, и XML существуют уже довольно продолжительное время.

AJAX – подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером.

Асинхронный обмен данных с сервером базируется на синтезе Javascript и XML. В результате, при обновлении данных, веб-страница не перезагружается полностью, и веб-приложения становятся более быстрыми и удобными.

Правда при использовании AJAX, разработчику необходимо разрабатывать средства, при помощи которых пользователь будет в курсе того, что происходит на странице. Это обычно реализуется при помощи использования индикаторов загрузки, текстовых сообщений о том, что идёт обмен данными с сервером.

AJAX базируется на двух основных принципах:

1) использование технологии динамического обращения к серверу «на лету», без перезагрузки всей страницы полностью, например:

– с использованием XMLHttpRequest (основной объект);

– через динамическое создание дочерних фреймов;

– через динамическое создание тега <script>;

2) использование DHTML для динамического изменения содержания страницы.

В качестве формата передачи данных обычно используются JSON (JavaScript Object Notation) или XML.

К преимуществам AJAX можно отнести:

– Экономия трафика - использование AJAX позволяет значительно сократить трафик при работе с веб-приложением благодаря тому, что часто вместо загрузки всей страницы достаточzно загрузить только изменившуюся часть, как правило, довольно небольшую.

– Уменьшение нагрузки на сервер - AJAX позволяет несколько снизить нагрузку на сервер. К примеру, на странице работы с почтой, когда вы отмечаете прочитанные письма, серверу достаточно внести изменения в базу данных и отправить клиентскому скрипту сообщение об успешном выполнении операции без необходимости повторно создавать страницу и передавать её клиенту.

– Ускорение реакции интерфейса — так как нужно загрузить только изменившуюся часть, пользователь видит результат своих действий быстрее.

К недостаткам AJAX можно отнести:

– Отсутствие интеграции со стандартными инструментами браузера. Динамически создаваемые страницы не регистрируются браузером в истории посещения страниц, поэтому не работает кнопка «Назад», предоставляющая пользователям возможность вернуться к просмотренным ранее страницам, но существуют скрипты, которые могут решить эту проблему. Другим недостатком динамического изменения контента страницы при постоянном URL является невозможность сохранения закладки на желаемый материал. Частично решить эти проблемы можно с помощью динамического изменения идентификатора фрагмента (части URL после #), что позволяют многие браузеры.

– Динамически загружаемое содержимое недоступно поисковикам (если не проверять запрос, обычный он или XMLHttpRequest). Поисковые машины не могут выполнять JavaScript, поэтому разработчики должны позаботиться об альтернативных способах доступа к содержимому сайта.

– Старые методы учёта статистики сайтов становятся неактуальными. Многие сервисы статистики ведут учёт просмотров новых страниц сайта. Для сайтов, страницы которых широко используют AJAX, такая статистика теряет актуальность.

– Усложнение проекта. Перераспределяется логика обработки данных — происходит выделение и частичный перенос на сторону клиента процессов первичного форматирования данных. Это усложняет контроль целостности форматов и типов. Конечный эффект технологии может быть нивелирован необоснованным ростом затрат на кодирование и управление проектом, а также риском снижения доступности сервиса для конечных пользователей.

– Требуется включенный JavaScript в браузере.

Технология AJAX использует комбинацию:

– (X)HTML, CSS для представления и стилизации информации

– DOM-модель, операции над которой производятся javascript на стороне клиента, чтобы обеспечить динамическое отображение и взаимодействие с информацией

– XMLHttpRequest для асинхронного обмена данными с веб-сервером. В некоторых AJAX-фреймворках и в некоторых ситуациях, вместо XMLHttpRequest

используется IFrame, SCRIPT-тег или другой аналогичный транспорт.

– Формат обмена данными – любой, не обязательно XML (текст, список, JSON).

Объект XMLHTTPRequest.

Браузеры предоставляют средство для выполнения асинхронных запросов - объект XMLHTTPRequest. Объект XMLHTTPRequest – низкоуровневая основа большинства AJAX-приложений. Использоавние его методов, свойств и особенностей помогает писать приложения на низком уровне с минимумом javascript-кода, а также понять, каким образом осуществляются операции внутри фреймворков.

В таблице 4.1 представлены методы объекта XMLHttpRequest.

Таблица 4.1 - Методы объекта XMLHttpRequest

Метод	Описание
abort()	Вызов этого метода обрывает текущий запрос. Для браузера Internet Explorer вызов abort() может не обрывать соединение, а оставлять его в подвешенном состоянии на некоторый таймаут (20-30 секунд).
getAllResponseHeaders()	Возвращает строку со всеми HTTP-заголовками ответа сервера.
getResponseHeader(headerName)	Возвращает значение заголовка ответа сервера с именем headerName.
open(method, URL, async, userName, password)	Определяет опциональные параметры запроса: – method – HTTP-метод. Как правило, используется GET либо POST; – URL – адрес запроса; – async – определяет режим запроса (при установке значения в true, задается асинхронный режим);

Метод	Описание
	– userName, password – данные для HTTP-авторизации.
send(content)	Отсылает запрос на сервер. Аргумент – тело запроса. Например, для GET запроса тела нет, поэтому используется send(null), а для POST запросов тело содержит параметры запроса.
setRequestHeader(name, value)	Устанавливает заголовок name запроса со значением value. Если заголовок с таким name уже есть – он заменяется. Пример. xmlhttp.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded')

overrideMimeType(mimeType)	Позволяет указать mime-type документа, если сервер его не передал или передал неправильно. Метод отсутствует в Internet Explorer!
	Некоторые версии некоторых браузеров Mozilla не будут корректно работать, если ответ сервера не содержит mime-заголовка XML. Чтобы решить эту проблему, вы можете использовать вызовы дополнительных методов для переопределения заголовка полученного от сервера, в случае, если он отличен от text/xml.
	Пример.
	httpRequest = new XMLHttpRequest();
	httpRequest.overrideMimeType('text/xml');

В таблице 4.2 представлены свойства класса XMLHttpRequest.

Таблица 4.2 - Свойства класса XMLHttpRequest

Свойство	Описание
onreadystatechange	Обработчик события, которое происходит при каждой смене состояния объекта
readyState	Возвращает текущее состояние объекта (0 — неинициализирован, 1 — открыт, 2 — отправка данных, 3 — получение данных и 4 — данные загружены)
responseText	Текст ответа на запрос. Полный текст ответа возможнго получить только при установлено readyState в состоянии 4.
responseXML	Текст ответа на запрос в виде XML (при readyState=4), который затем может быть распарсен посредством DOM. Чтобы браузер корректно разобрал ответ сервера в responseXML, в заголовке должен быть Content-Type: text/xml.
status	Возвращает HTTP-статус в виде числа (404 — «Not Found», 200 — «OK» и т. д.). Запросы по протоколам FTP, FILE:// не возвращают статуса, поэтому нормальным для них является status=0.
statusText	возвращает статус в виде строки (например, «Not Found», «OK» и т.д.)

4.5 Пример использования технологии AJAX

Любое приложение, построенное по технологии AJAX, состоит из двух взаимодействующих между собой частей: клиентской и серверной. Серверная часть – это сценарий, который запускается на сервере в ответ на тот или иной GET/POST запрос пользователя. Клиентская часть системы – некоторый код на языке программирования Javascript, выполняющийся непосредственно в браузере пользователя. Он принимаетт данные, сгенерированные серверной частью, обрабатывает их и отображает в заранее

отведенной для этого области страницы. Общая схема этапов выполнения AJAX-запроса показана на рисунке 4.1

Рисунок 4.1 – Этапы выполнения AJAX-запроса

Основным объектом, при помощи которого идет обращение клиентских запросов к сервер, является XMLHttpRequest. Инициализация данного объекта отличается в различных браузерах. Рассмотрим пример кроссбраузерного создания объекта XMLHttpRequest.

```
<script type="text/javascript">
 // глобальная переменная для хранения обработчика запросов
var ajax;
InitAjax();
  function InitAjax()
     {
  // используем структуру try..catch для попытки создать обработчик запросов XMLHTTP
         try
         {
  // пробуем создать компонент XMLHTTP для IE старых версий
                  ajax = new ActiveXObject("Microsoft.XMLHTTP");
         }
         catch (e)
         {
 // если не получилось создать компонент XMLHTTP для IE пробуем следующий и т.д.
```

```
            try
            {
              // пробуем создать компонент XMLHTTP для IE версий 6 и выше
                ajax = new ActiveXObject("Msxml2.XMLHTTP");
            }
            catch (e)
            {
                    try
                    {
 // пробуем создать компонент XMLHTTP для Mozilla, и остальных
                            ajax = new XMLHttpRequest();
                    }
                    catch (e)
                    {
                            ajax = 0;
                    }
            }
        }
    }
</script>
```

Результат функции createRequestObject должен ссылаться на корректный объект XMLHttpRequest, независимо от используемого пользователем браузера.

В первом примере предалагается задача динамической загрузки данных для формирования выпадающего списка. В данном примере используется свойство responseText класса XMLHttpRequest в качестве доступа к текстовому содержимому.

В клиентской части приложения из базы данных формируются элементы первого списка, второй список остается пустой и заполнятеся на основе асинхронной передачи данных с серверной стороны.

```
<!DOCTYPE html >
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
   <meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
   <meta http-equiv="Pragma" content="no-cache" />
   <title>Ajax - примеры</title>
   </head>
  <body>
        <b>Задание№1</b>
        <form  name="form1" method="get">
        <select id="select1" name="name1">
        <option value="0">SQL: Полное руководство</option>
        <option value="1">Из истории культуры средних веков и Возрождения</option>
        <option value="2">Ежедневный журнал</option>
        </select>
<input type="button" name="form1submit" value="Поиск" onclick="javascript: gets2();"/>
        <br><br>
        <select id="select2" >
        <option>no data</option>
        </select>
        </form>
   </body>
   </html>
```

В скриптовой части определяется метод (функция gets2()), реагирующий на нажатие кнопки. Эта функция выполняет поиск элемента в списке select1 в объектной модели документа и вызвает функцию open, которая создает соединение. Для данного случая в GET-параметре запроса передается идентификатор выбранной книги.

```
   <script type="text/javascript">
        // функция, которая будет запрашивать содержимое новой страницы
        function gets2()
```

```
  {
        if (!ajax)
    {   alert("Ajax не инициализирован");
        return;     }
    var s1val = document.getElementById("select1").value;
    // установка для экземпляра объека ajax функции обработчика события
onreadystatechange
    ajax.onreadystatechange = UpdateSelect2; // определение обработчика
    /*открытие соединения с указанием типа запроса (GET или POST), URL серверной
части, флага асинхронного режима и имени и пароля пользователя (если необходимо)*/
    ajax.open("GET", "get.php?select1="+s1val, true);// формирование запроса
    //непосредственно отправка запроса
    ajax.send(null); // запрос на сервер
 }
    </script>
```

Кроме того определяется функция UpdateSelect2(), которая будет обрабатывать (вставлять возвращенный текст в нужную область страницы).

```
  <script type="text/javascript">
function UpdateSelect2()
{
    if (ajax.readyState == 4) // если статус - выполнено
    {
        if (ajax.status == 200) // если ошибок нет
        {
            var divBody = document.getElementById('select2');
            divBody.innerHTML = ajax.responseText;
        }
        else alert(ajax.status + " - " + ajax.statusText);
        ajax.abort();
    }
}
  </script>
```

В серверной части приложения реализован механизм формирования второго выпадающего списка на основе выбранного значения первого списка.

```
<?php
        $id = $_GET['select1'];
 switch ($id)
  {    case "0":
         echo '<option>BHV</option><option>10000</option><option>10-15-XX-44</option>';
         break;
       case "1":
         echo '<option>Science</option><option>1000</option><option>44-78-106-X
</option>';
         break;
       case "2":
         echo '<option>BHV</option><option>150</option>';
         break;
  } ?>
```

Во втором примере предлагается задача динамической загрузки данных для блочного элемента. Используется свойство responseXML класса XMLHttpRequest для получения ответа на запрос в виде XML.

В клиентской части приложения из базы данных добавлены три пустых блочных тега :

```
<div id="book"></div>
<div id="publisher"></div>
<div id="quantity"></div>
```

Переопределим метод UpdatePage(), указав другое свойство responseXML для обработки ответа от сервера в виде формата XML и распарсив каждый элемент полученного XML ответа.

```
function UpdatePage()
{
```

```
        if(ajax.readyState == 4) // якщо статус – виконано
        {
                if(ajax.status == 200) // якщо немає помилок
                {
                        xmlDoc=ajax.responseXML;
                        document.getElementById("book").innerHTML =
                xmlDoc.getElementsByTagName("book")[0].childNodes[0].nodeValue;
                        document.getElementById("publisher").innerHTML =
                xmlDoc.getElementsByTagName("publisher")[0].childNodes[0].nodeValue;
                        document.getElementById("quantity").innerHTML =
                xmlDoc.getElementsByTagName("quantity")[0].childNodes[0].nodeValue;
                }
                else
                {
                        alert(ajax.status + " - " + ajax.statusText);
                         ajax.abort();
                }
        }
}
```

В серверной части приложения необходимо указать корректрый тип документа и реализовать механизм формирования XML файла, который будет передан клиенту для обработки.

```
<?php
header('Content-Type: text/xml');
header("Cache-Control: no-cache, must-revalidate");
$id = $_GET['select1'];
echo '<?xml version="1.0" encoding="utf8" ?>';
echo "<row>";
echo "<book>BOOK from XML:".$id."</book>";
```

```
echo "<publisher>PUBLISHER from XML: BHV </publisher>";
echo "<quantity>QUANTITY from XML: 10000</quantity>";
echo "</row>";      ?>
```

В третьем примере используется свойство responseText класса XMLHttpRequest для получения ответа на запрос в виде JSON. Переопределим метод UpdatePage(), осуществив парсинг полученного ответа от севера в виде JSON.

```
function UpdatePage()
{
    if(ajax.readyState == 4) // якщо статус - виконано
    {
        if(ajax.status == 200) // якщо немає помилок
        {
            var res = JSON.parse(ajax.responseText);
            document.getElementById("book").innerHTML = res.book;
            document.getElementById("publisher").innerHTML = res.publisher;
            document.getElementById("quantity").innerHTML = res.quantity;
        }
        else
        {
            alert(ajax.status + " - " + ajax.statusText);
            ajax.abort();
        }
    }
}
```

В серверной части приложения необходимо указать корректрый тип документа и реализовать механизм формирования JSON файла, который будет передан клиенту для обработки.

```
<?php
    header('Content-Type: application/json');
    header("Cache-Control: no-cache, must-revalidate");
    $id = $_GET['select1'];
    $data = array('book' => $id, 'publisher' => 'Smit', 'quantity' => 1000);
    echo json_encode($data);
```

Во всех браузерах, кроме IE, это можно сделать с помощью свойства innerHTML. Для достижения кроссбраузерности в IE элементы должны добавляться по одному.

4.6 Варианты заданий

Для вариантов заданий, указанных в лабораторной работе №2-3, добавить динамическую загрузку необходимых данных с сервера по технологии AJAX.

Варианты 1, 4, 7, 10 осуществляют передачу данных на сервер с использованием XML в качестве протокола обмена данными между сервером и клиентом.

Варианты 2, 5, 8 осуществляют передачу данных на сервер с использованием JSON в качестве протокола обмена данными между сервером и клиентом.

Варианты 3, 6, 9 осуществляют передачу данных на сервер с использованием текстового формата.

4.7 Содержание отчета

Отчет по лабораторной работе должен содержать:

- тему и цель работы;
- задание на лабораторную работу;
- исходные коды PHP- и AJAX-скриптов;
- результаты выполнения запросов в виде снимков экрана браузера;
- выводы по работе.

4.8 Контрольные вопросы и задания

1) Опишите ключевые концепции Web2.0, приведите изменения, внесенные Web2.0 по сравнению с Web1.0 (web.com).

2) Перечислите преимущества и недостатки использования технологии AJAX.

3) Укажите возможности, предоставляемые технологией AJAX.

4) Какие изменения обработки данных на клиентской стороне введены для поддержки AJAX?

5) Какими объектами и методами поддерживается методология AJAX в JavaScript?

6) Какие реализации библиотек поддержки AJAX существуют?

5 ПОЛНОДУПЛЕКСНЫЙ ОБМЕН ДАННЫМИ МЕЖДУ БРАУЗЕРОМ И ВЕБ-СЕРВЕРОМ НА ОСНОВЕ ПРОТОКОЛА ПЕРЕДАЧИ ДАННЫХ WEBSOCKET

5.1 Цель работы

Исследование клиентских и серверных программных средств для обмена сообщениями между браузером и веб-сервером в режиме реального времени.

5.2 Методические указания по организации самостоятельной работы

При подготовке к выполнению лабораторной работы следует ознакомится с механизмом обмена сообщениями по протоколу WebSocket, а также событиями и методами коллективного обмена сообщениями на стороне клиента и сервера библиотеки socket.io [10,11].

5.3 Программное обеспечение ПК

При выполнении лабораторной работы используется ПЭВМ под управлением операционной системы Windows XP и старше, серверная платформа NODE.js, пакет socket.io, среда разработки Notepad++ 6.4.5.

5.4 Методические указания по выполнению лабораторной работы

WebSocket – протокол полнодуплексной связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени. Организация такого рода тесного взаимодействия между браузером и веб-сервером дает возможность создавать интернет-приложения нового уровня,

например, игры реального времени и другие веб-приложения с интенсивным обменом данными, требовательные к скорости обмена и каналу.

Стандартный протокол HTTP ограничен моделью запрос-ответ: клиент посылает запрос HTTP и ожидает на него HTTP-ответ. По сути, сервер не может сообщить что-либо клиенту до тех пор, пока клиент его «не попросит». Такого рода приложения ограничены сетевой задержкой и необходимостью пересоздания всего документа в момент перехода по ссылке.

Благодаря предоставлению стандартного способа для отправки содержимого сервером браузеру без дополнительного запроса клиента в привычной схеме «запрос URL — ответ», между браузером и сервером может происходить двусторонний (двунаправленный) обмен сообщениями одновременно, пока соединение открыто.

Протокол WebSocket предоставляет значительно меньшие накладные расходы за счет постоянно открытого канала по сравнению с повторяющимися обращениями к серверу для отслеживания изменений. Раньше такие функции были доступны только с помощью технологии плагинов типа Flash. Кроме того, обмен информацией идёт по TCP-порту 80, что является большим преимуществом для тех сред, которые блокируют нестандартные подключения к интернету с помощью межсетевого экрана.

Рассмотрим схему обмена сообщениями между клиентом и веб-сервером по протоколу WebSocket.

Когда браузер направляет запрос по URL-адресу веб-сокета, сервер отправляет обратно заголовки, завершающие обмен подтверждениями по протоколу WebSocket. Ответное квитирование WebSocket должно начинаться строго со строки HTTP/1.1 101 WebSocket Protocol Handshake. Фактически, порядок и содержание заголовков "рукопожатия" определяются более строго, чем для заголовков HTTP. После завершения квитирования клиент и сервер могут в любой момент начать обмен сообщениями. Каждое соединение представляется на сервере экземпляром объекта WebSocket Connection.

Метод send объекта WebSocket Connection преобразует строку сообщения к виду, соответствующему протоколу WebSocket. Границы кадра обозначаются байтами 0x00 и

OxFF, между которыми помещается строка сообщения в кодировке UTF-8: 0x00, <строка в кодировке UTF-8>, 0xFF. Для отправки в качестве строки можно предоставлять, в том числе данные формата XML и JSON. Получая сообщение от сервера, браузер выполняет функцию callback. Коэффициент полезного действия такого протокола стремится к 95%, поскольку нет необходимости пересылать несколько килобайт заголовков, что особенно заметно если делать частый обмен небольшими блоками данных.

Готовое веб-приложение, реализованное с использованием протокола WebSocket, представляет собой клиент-серверную реализацию.

Поэтому, прежде чем начать использовать протокол WebSocket, требуется создать сервер, поддерживающий веб-сокеты. Наиболее популярным является серверный JavaScript-фреймворк node.js, на основе которого было создано несколько серверов WebSocket. Причем node.js – событийно-ориентированная модель и хорошо развитые функции callback в javascript.

Библиотека Socket.io обеспечивает взаимодействие в режиме реального времени между сервером node.js и клиентами, обеспечивая встроенное мультиплексирование, горизонтальную масштабируемость, автоматическое кодирование/декодирование JSON и больше.

Конфигурация socket.io.

Socket.io можно настроить посредством методов configure, set, enable и disable, а также при установлении соединения.

Следующие опции могут быть настроены **на стороне сервера**:

– heartbeats (по умолчанию включено) – используется ли режим heartbeats для проверки состояния соединения socket.io;

– transports - транспортирует по умолчанию websocket, htmlfile, xhr-polling, jsonp-polling;

– log level (по умолчанию равен 3) – данные, выводимые регистратору: 0 – ошибка, 1 – предупреждения, 2 – информация, 3 – отладка;

– close timeout (по умолчанию 60 секунд) - тайм-аут для клиента, в течении которого в случае закрытия соединение имеется возможность повторного его открытия. Это значение посылают клиенту после успешного рукопожатия;

– heartbeat timeout (по умолчанию 60 секунд) - тайм-аут для клиента, в течении которого он должен отправить новое значение heartbeat на сервер. Это значение посылают клиенту после успешного рукопожатия.

Следующие опции могут быть настроены **на стороне клиента**:

– connect timeout (по умолчанию 10000 мс) - задержка перед попыткой подключения к серверу с использованием другого транспорта;

– reconnect (по умолчанию включена) - повторное подключение в случае, если socket.io обнаружит разрыв связи или тайм-аут.

– reconnection delay (по умолчанию 500 мс) – задержка перед началом восстановления соединения;

– maxReconnectionAttempts - максимальное количество попыток переподключения.

Выбор транспорта передачи информации.

С помощью пакета socket.io возможно с лёгкостью создавать кросбраузерные real-time приложения. Легкий и удобный уровень абстракции в socket.io достигается за счёт использования различных транспортов передачи информации на сервер и с сервера в браузер. Причём технология выбирается совершенно прозрачно и для клиента, и для сервера.

Транспорты выбираются в следующей последовательности:

- WebSocket;
- Adobe Flash Socket;

- AJAX multipart streaming;
- AJAX long polling;
- Iframe(только в IE);
- JSONP Polling.

Если браузер поддерживает WebSockets, будут использоваться именно он. Для других браузеров будет обеспечен fallback до флешовых сокетов, а если и этих нет – до обычного XHR с long polling.

Socket.io поддерживает следующие десктопные браузеры: Internet Explorer 5.5+, Safari 3+, Google Chrome 4+, Firefox 3+, Opera 10.61+. А также следующие мобильные браузеры: iPhone Safari, iPad Safari, iPad Safari, Android WebKit, WebOs WebKit.

Ниже приведены примеры выбора **на стороне сервера** в качестве транспорта протокола WebSocket или нескольких протоколов с использованием метода socket.set(key, value).

```
// подключаем модуль для создания сервера
// и ставим на прослушивание 80-порта
var io = require('socket.io').listen(80);

//Способ 1
/// отключаем вывод полного лога
io.set('log level', 1);
//ограничиваем транспорт только протоколом WebSocket
io.set('transports', ['websocket']);

//Способ 2
io.configure(function () {
  io.set('transports', ['websocket', 'flashsocket', 'xhr-polling']);
});
```

```
//Способ 3
io.configure('development', function () {
  io.set('transports', ['websocket', 'xhr-polling']);
});
```

Получение соединения WebSocket.

Ниже приведены примеры использования клиентской и серверной библиотеки socket.io для получения соединения соответственно.

Серверная сторона.

Подключение WebSocket на веб-сервере осуществляется посредством следующей функции:

```
var socket = require('socket.io').listen(80, {
  // опции можно описать здесь
});
```

Пример использования серверной библиотеки socket.io (server.js) показан ниже.

```
var port = 8080;
//открываем соединение
//с установленной опцией разрешенных методов транспорта
var io = require('socket.io').listen(port,  {'transports': ['websocket']});
io.sockets.on('connection', function (socket) {
socket.emit('news', { hello: 'world' });
socket.on('my other event', function messageReceived(data) {    console.log(data); });
});
```

Клиентская сторона.

Подключение WebSocket на клиенте осуществляется посредством следующей функции:

```
var socket = io.connect('http://server.com', {
  // опции можно описать здесь
});
```

Пример использования клиентской библиотеки socket.io (client.html) показан ниже.

```
<!DOCTYPE html>
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<!-- Подключаем библиотеку, именно таким способом: запрашиваем ее со стороны сервера, где она динамически генерируется.  -->
<script src="http://localhost:8080/socket.io/socket.io.js"></script>
<script>
//адрес сервера
var serverURL = 'http://localhost:8080';
 var socket = io.connect(serverURL, { 'connect timeout': 5000 });
  socket.on('news', function (data) {
 console.log(data);
 socket.emit('my other event', { my: 'data' });
  });
</script>
</head>
</html>
```

События установления подключения, переподключения и закрытия соединения.

Рассмотрим события на стороне клиента и сервера, которые посылаются в случае установления подключения, переподключения и закрытия соединения.

На стороне сервера, предполагая, var io = require('socket.io'):

– io.sockets.on('connection', function(socket) {}) - начальное соединение от клиента. Аргумент socket должен быть использован в дальнейшей коммуникации с клиентом.

– socket.on('disconnect', function() {}) - событие разъединения срабатывает во всех случаях, когда соединение клиент-сервер закрыто. Срабатывает в случаях желательного, нежелательного, мобильного, не мобильного, клиентского и серверного отключения. Не существует события восстановления связи. Вы должны использовать событие "connection " для восстановления управляемости.

На клиенте, предполагая, socket = io.connect (host, options):

1) в случае первого подключения:

– socket.on('connect', function () {}) – событие "connect" посылается, когда сокет успешно подключен;

– socket.on('connecting', function () {}) - событие "connecting" посылается, когда сокет пытается подключиться к серверу;

2) при моментальной потере соединения:

– socket.on('disconnect', function () {}) - событие "disconnect" посылается, когда сокеты отключены;

– socket.on('reconnecting', function () {}) - событие "reconnecting" посылается при попытке сокета восстановить связь с сервером;

– socket.on('connect_failed', function () {}) - событие "connect_failed" посылается, когда socket.io не может установить соединение с сервером и не имеет других вариантов транспорта;

– socket.on('reconnect_failed', function () {}) - событие "reconnect_failed" посылается, когда socket.io не удается восстановить рабочую связь после того как подключение было прекращено;

– socket.on('reconnect', function () {}) - событие "reconnect" посылается, когда socket.io успешно повторно подключился к серверу;

3) при полной потере соединения:

– socket.on('disconnect', function () {}) - событие "disconnect" посылается, когда сокеты отключены;

– socket.on('reconnecting', function () {}) - событие "reconnecting" посылается, когда сокет пытается восстановить связь с сервером.

Закрыть соединение имеется возможность у любой из сторон, как сервера и/или браузера, поскольку в итоге существует только одно соединение.

При возникновении ошибки можно воспользоваться событием на стороне клиента: socket.on('error', function () {}) - событие "error" посылается, когда возникает ошибка, которая не может быть обработана другими типами событий.

Отправка данных к клиенту.

Метод socket.send выполняет передачу сообщения text на основе базового события "message":

socket.send(text).

В socket.io v0.6 метод socket.send будет автоматически конвертировать объект в JSON, например, отправка данных клиенту будет соответствовать socket.send({ a: 'b' }).

Это свойство одновременно является и преимуществом при передаче данных, и в то же время создает проблемы, поскольку JSON не только шифрует объекты, но также и строки, числа, и т.д. Это в свою очередь приводит к потерям производительности,

связанные с кодированием/декодированием JSON. Используемое API станет более прозрачным, если явно будет указано, что будет передаваться JSON.

В socket.io v0.7 используется флаг json для отправки сообщения в формате JSON:

socket.json.send(data[, callback]).

Передача клиентам сообщения в формате JSON позволяет генерировать на клиенте легко изменяемые данные меньшего обьема, представление которых не зависит от сервера.

Отправка сообщения для всех.

На сервере существует возможность выбора «текущего» клиента с помощью socket, так и выбора всех подключенных клиентов с помощью io.sockets.

Таким образом, если вы хотите отправить сообщение *любому* вы можете ссылаться на io.sockets:

```
io.sockets.send('сообщение');
io.sockets.emit('событие');
```

Отправка непостоянных сообщений.

В случае если определенный клиент не будет готов получить сообщения (из-за сетевой медлительности или других проблем, или потому что он соединен посредством долгого опроса и находится в середине цикла ответа запроса), тогда определенные сообщения могут быть отброшены.

В этом случае, имеется возможность отправки непостоянных сообщений на стороне сервера:

```
var io = require('socket.io').listen(80);
io.sockets.on('connection', function (socket) {
setInterval(function () {
```

```
    socket.volatile.emit('bieber tweet', tweet);  }, 100);
});
```

Отправка событий.

Теперь вы можете передавать и получать пользовательские события между браузером и сервером с помощью:

```
socket.emit('custom event'[, arguments][, callback]);
```

Аргументы для события автоматически шифруются в формат данных JSON.

Пример отправки на клиентский сокет текущего запроса:

```
socket.emit('message', "this is a test");
```

Пример отправки на клиентский сокет данных формата JSON:

```
socket.emit('whereami', {'location': loc}))
```

Получение событий.

При возникновении события event метода socket.on(event, callback) выполняется вызов функции callback.

На клиенте, предполагая, socket = io.connect (host, options):

– socket.on('message', function (message, callback) {}) - событие "message" выполняется в случае, когда получено сообщение, посланное с socket.send. Сообщение «message» означает, что отправлено сообщение, и функция обратного вызова является дополнительной функцией подтверждения;

– socket.on('anything', function(data, callback) {}) - событие "anything" может быть любым пользовательским событием, кроме резервных .Аргумент data и функция обратного вызова может быть использованы для отправки ответа.

На стороне сервера, предполагая, var io = require('socket.io'):

– socket.on('message', function(message, callback) {}) - событие "message" выполняется в случае, когда получено сообщение, посланное с socket.send. Сообщение "message" означает, что отправлено сообщение, и функция обратного вызова является дополнительной функцией подтверждения;

– socket.on('anything', function(data) {}) - событие "anything" может быть любым событием, кроме резервных.

Пример получения события:

```
socket.on('whereami', function(loc){ console.log('I\'m in ' + loc + '!'); }))
```

Хранение данных, связанных с клиентом.

Иногда необходимо хранить данные, связанные с клиентом, что необходимо на время сеанса.

В примере ниже показано как реализовать хранение данных **на серверной стороне.**

```
var io = require('socket.io').listen(80);
io.sockets.on('connection', function (socket) {
  socket.on('set nickname', function (name) {
    socket.set('nickname', name, function () { socket.emit('ready'); });
  });

  socket.on('msg', function () {
    socket.get('nickname', function (err, name) {
      console.log('Chat message by ', name);
    });
  });
});
```

И **на клиентской стороне** соответственно.

```
<script>
  var socket = io.connect('http://localhost');

  socket.on('connect', function () {
    socket.emit('set nickname', prompt('What is your nickname?'));
    socket.on('ready', function () {
      console.log('Connected !');
      socket.emit('msg', prompt('What is your message?'));
    });
  });
</script>
```

Широковещательная передача сообщений.

В socket.io v0.7.0 имеется возможность отправить сообщение из отдельного сокета к остальной части сокетов, используя флаг broadcast:

```
socket.broadcast.send('сообщение');
socket.broadcast.emit('событие'[, arguments]);
```

Пример отправка сообщения всем клиентам, кроме текущего:

```
socket.broadcast.emit('message', "this is a test");
```

Отправка и передача сообщений отдельному сокету.

Кроме того, имеется в наличии средства выбора конкретно взятого клиента с заданным идентификатором ID:

```
io.sockets.socket(< id>).send('мое сообщение')
io.sockets.socket(< id>).emit('имя события'[, arguments])
```

Пример отправки сообщения отдельным socketid на стороне сервера:

```
io.sockets.socket(socketid).emit('message', 'for your eyes only');
```

На клиентской стороне socketid получают с помощью:

```
var io = io.connect('localhost');
io.on('connect', function () {
   console.log(this.socket.id);

});
```

5.5 Варианты заданий

Задание №1.

Создать, выполнить и убедиться в работоспособности клиентской и серверной частей приложения «Чат», реализованного с использованием пакета socket.io. Добавить функциональность согласно заданию 2.

Задание №2.

Реализовать систему реального времени, состоящую из клиентской и серверной частей.

Над выполнением вариантов задания работает две бригады, каждая на выбор реализует либо клиентскую, либо серверную часть.

Окончательный вариант задания должен быть протестирован на скорость передачи данных с использованием протокола Websocket с помощью инструмента веб-разработчика браузера Google Chrome.

Вариант 1.

Диспетчер такси.

Клиентская часть предоставляет возможность выбора машины и частоты обновления данных, а также получает от сервера данные о перемещении такси, благодаря расположенному в нем GPS-трекеру (идентификатор машины, водитель, измерения широты и долготы промежуточных точек маршрута, скорость движения, текущие измерения широты и долготы) и отображает их пользователю. Сервер генерирует данные выбранной машины с заданным интервалом.

Вариант 2.

Система производственно-экологического мониторинга.

Клиентская часть предоставляет возможность выбора санитарно-защитной зоны и частоты обновления данных, а также получает от сервера данные замеров стационарных и мобильных постов экологического мониторинга (метеорологического контроля, химического контроля, пылемеров, анализаторов воды и почвы). Сервер генерирует данные дистанционного зондирования выбранной санитарно-защитной зоны с заданным интервалом.

Вариант 3.

Онлайн-игра «Управление марсоходом».

Клиентская часть предоставляет возможность выбора марсохода, количества мин, разбросанных по полю, указания размерности поля (поле квадратное) и текущих координат марсохода, а также получает от сервера данные о местонахождении мины. Сервер генерирует данные о текущем местоположении мины, а также в случае попадания на мину передает данные о разрушении марсохода.

Вариант 4.

Мониторинг погоды.

Клиентская часть предоставляет возможность выбора населенного пункта и частоты обновления данных, а также получает от сервера данные о погодных условиях (текущая температура, осадки, уровень давления, скорость ветра) и отображает их пользователю. Сервер генерирует данные о погодных условиях выбранного города с заданным интервалом.

Вариант 5.

Биржа.

Клиентская часть предоставляет возможность выбора биржевого товара и частоты обновления данных, а также получает от сервера данные о предложенных контрактах, формируемых в течение дня (компания, цена на единицу товара, количество товара, покупка или продажа). Сервер генерирует данные о предложенных контрактах с заданным интервалом.

Вариант 6.

Система поддержания жизнедеятельности.

Клиентская часть предоставляет возможность выбора объекта для исследования и частоты обновления данных, а также получает от сервера данные от датчиков среды (химический состав, температура, влажность, излучение) и отображает их пользователю. Сервер генерирует данные датчиков с заданным интервалом, а также выводит сообщение об угрозе жизни для объекта.

Вариант 7.

Обучающая система «Организм человека».

Клиентская часть предоставляет возможность выбора выбора органа человека и его состояния, а также получает от сервера данные об общих физиологических показателях организма (общее количество лейкоцитов, тромбоцитов, эритроцитов, скорость оседания

эритроцитов, количество белков, выделяемые гормоны) и отображает их пользователю. Сервер генерирует данные о физиологических показателях организма с заданным интервалом.

5.6 Содержание отчета

Отчет по лабораторной работе должен содержать:

– тему и цель работы;
– задание на лабораторную работу;
– исходные коды серверного и клиентского JS-скриптов;
– результаты выполнения запросов в виде снимков экрана браузера;
– выводы по работе.

5.7 Контрольные вопросы и задания

1) Сравните работу соединения посредством HTTP Long polling с протоколом WebSocket.
2) Опишите процесс создания соединения WebSocket на стороне клиента и сервера, а также его закрытия.
3) Перечислите клиентские и серверные события, выполняемые при создании соединения или его переподключении.
4) Каким образом осуществляется обмен данными между клиентом и сервером с применением методов пакета socket.io? Приведите примеры.
5) Каким образом существует возможность отправки ответа всем клиентам кроме отправителя?
6) Перечислите методы для отправки и получения данных формата JSON.
7) Поясните механизм создания комнат.

ПЕРЕЧЕНЬ ССЫЛОК

1. Дронов В.А. HTML5, CSS3 и Web 2.0 Разработка современных Web-приложений / В.А. Дронов. – СПб:БХВ-Петербург, 2011. – 416с.
2. Хоган Б. HTML5 и CSS3 Веб-разработка по стандартам нового поколения/ Б. Хоган. – СПб: Питер, 2012. – 272 с.
3. Зольников Д.С. PHP5 / Д.С. Зольников. – М.:НТ Пресс, 2007. – 256 с.
4. Кузнецов М.В. PHP5 на примерах / М.В. Кузнецов, И.В. Симдянов, С.В. Голышев. – СПб: БХВ-Петербург, 2005. – 576с.
5. Ловэйн П. Объектно-ориентированное программирование на PHP5 / Питер Ловэйн; пер. с англ. А.А. Слинкина. – М.: НТ Пресс, 2007. – 224с.
6. Гутманс Э. PHP5 Профессиональное программирование/ Э. Гутманс, С. Баккен, Д. Ретанс. – СПб: Символ-Плюс, 2006. -704с.
7. Зервас Квентин Web 2.0 создание приложений на PHP / Квентин Зервас. – Москва: ООО «И.Д. Вильямс», 2010. – 544с.
8. Zmievski Andrei Руководство по Smarty / Andrei Zmievski. – New Digital Groups, Inc, 2007. – 214 с.
9. Котеров Д.В. PHP5/ Д.В. Котеров, А.Ф. Костарев. – 2-е изд., перераб. и доп. – СПб:БХВ-Петербург, 2008. – 1104 с.
10. Introducing of Socket.io v.9 [Электронный ресурс]. – Режим доступа: http://socket.io/ – Загл. с экрана.
11. NODE.js v0.10.21 [Электронный ресурс]. – Режим доступа: http://nodejs.org/ – Загл. с экрана.