

# Интернет- технологии

СКИФ



Кафедра «Вычислительные системы  
и информационная безопасность»

**Лекционный курс**

**Авторы**

**Галушка В.В., Панасенко Н.Д.**

## Аннотация

Лекционный курс предназначен для обучающихся по направлению 10.03.01 Информационная безопасность

Обучающиеся, завершившие изучение дисциплины «Интернет - технологии» должны обладать следующими навыками в соответствии с ФГОС ВО 10.03.01

Информационная безопасность:

- способностью понимать значение информации в развитии современного общества, применять информационные технологии для поиска и обработки информации;
- способностью применять программные средства системного, прикладного и специального назначения, инструментальные средства, языки и системы программирования для решения профессиональных задач.

## ОГЛАВЛЕНИЕ

Раздел 1. Введение, основные понятия	4
1.1 История развития интернета	4
1.2 Адресация в сети интернет	5
Раздел 2. Технология WWW	8
2.1 Общие принципы построения глобальной сети	8
2.2 Технология гипертекста	10
2.3. Каскадные листы стилей	18
Глава 3. Технологии динамической генерации страниц	25
3.1. Средства разработки динамических web-страниц	25
3.2. Программирование на языке PHP	32
3.3. Передача данных с web-форм	54
3.4. Динамические страницы с использованием JavaScript	58
Раздел 4. Технологии баз данных.	69
4.1. СУБД MySQL.	69
Раздел 5. Безопасность в сети интернет	83
5.1. Методы аутентификации пользователей	83

## Раздел 1. Введение, основные понятия

### 1.1 История развития интернета

Интернет (Internet, сокращённое от INTERconnected NETworks – объединённые сети; сленговое инёт, нет) – глобальная всемирная телекоммуникационная сеть, обеспечивающая связь для пересылки сообщений электронной почты, передачи файлов, соединения с другими компьютерами и получения доступа к информации, существующей в самых различных формах.

В 1957 году, после запуска СССР первого искусственного спутника земли, правительство США решило, что в случае войны неплохо бы иметь надёжную систему передачи данных. Разработка такой системы была поручена нескольким крупным университетам Америки. Компьютерную сеть в проекте назвали ARPANET (англ. Advanced Research Projects Agency Network) и уже в 1969 году сеть связала четыре университета: Калифорнийский, Стэнфордский, а так же Университеты Калифорнии и Санта-Барбары. Все работы получали финансирование из средств Министерства обороны США. Позже сеть ARPANET была задействована учёными из разных областей науки – сеть росла.

Суть проекта – он должен был, с одной стороны, привести к созданию практически не поддающихся разрушению каналов связи, а с другой — облегчить сотрудничество между разбросанными по всем штатам исследовательским организациям оборонной промышленности.

В основу проекта были положены три основные идеи: каждый узел сети Соединен с другими, так что существует несколько различных путей от узла к узлу; все узлы и связи рассматриваются как ненадежные – существуют автоматически обновляемые таблицы перенаправления пакетов – пакет, предназначенный для не соседнего узла отправляется на ближайший к нему, согласно таблице перенаправления пакетов, при недоступности этого узла - на следующий и т.д.

Эти идеи должны были обеспечить функционирование сети в случае разрушения любого числа ее компонентов. В принципе сеть можно было считать работоспособной даже в случае, когда остается функционировать всего два компьютера. Кроме того созданная по такому принципу система не имела централизованного узла управления, и следовательно безболезненно могла изменять свою конфигурацию.

После первой успешной передачи данных в сети ARPANET следующим значимым этапом стала разработка в 1971 году первой программы для отправки электронной почты по сети. Данная программа мгновенно обрела популярность.

## Интернет-технологии

Общее развитие электронной почты шло через развитие локального взаимодействия пользователей на многопользовательских системах. Пользователи могли, используя программу mail (или её эквивалент), пересылать друг другу сообщения в пределах одного мейнфрейма (большого компьютера). Следующий шаг был в возможности переслать сообщение пользователю на другой машине – для этого использовалось указание имени машины и имени пользователя на машине.

К 1973 году в состав сети были включены первые зарубежные организации из Великобритании и Норвегии через трансатлантический телефонный кабель. С этого момента сеть стала считаться международной.

В 70-х годах прошлого века основным предназначением сети была пересылка электронной почты. В то же время появляются первые почтовые рассылки, различные доски объявлений и новостные группы. Однако во взаимодействии с другими сетями, построенными на других стандартах, были большие проблемы. Бурное развитие различных протоколов передачи данных, а так же их последующая стандартизация в 82-83 годах и переход на «общий», объединяющий протокол TCP/IP решили данную проблему. Этот переход состоялся 1 января 1983 года. Именно в этом году сеть ARPANET закрепила за собой термин «Интернет».

### 1.2 Адресация в сети интернет

Следующим этапом развития была разработка системы доменных имён (англ. Domain Name System, DNS), которая состоялась в 1984 году.

Так же в этом году появляется серьёзный конкурент сети ARPANET – межуниверситетская сеть NSFNet (англ. National Science Foundation Network). Эта сеть была объединением множества мелких сетей, имела пропускную способность гораздо большую, чем у ARPANET, а так же высокую динамику подключения новых пользователей (около 10 тысяч машин в год). Гордое звание «Интернет» перешло к NSFNet.

После появления распределённой глобальной системы имён DNS, для указания адреса стали использоваться доменные имена — user@example.com — пользователь user на машине example.com. Одновременно с этим происходило переосмысление понятия «на машине»: для почты стали использоваться выделенные серверы, на которые не имели доступ обычные пользователи (только администраторы), а пользователи работали на своих машинах, при этом почта приходила не на рабочие машины пользователей, а на почтовый сервер, откуда пользователи забирали свою почту по различным сетевым протоколам (среди распространённых на настоящий момент — POP3, IMAP, MAPI, веб-интерфейсы).

Программы для обмена текстовыми строками, несмотря на простоту самой идеи, появились не сразу. Примерно в 1974 году для мейнфрейма

## Интернет-технологии

PLATO была разработана программа Talkomatic, потенциально позволявшая общаться между тысячей терминалов системы. В 1980-х появилась система 'Freelancing' Round table. Однако по-настоящему популярным стал разработанный в 1988 году анонсированный протокол мгновенной передачи текстовых сообщений Internet Relay Chat (IRC), вследствие этого в Интернете стало возможным «живое» общение в чате в реальном времени. Общение в IRC быстро стало популярным из-за простоты процесса и дружелюбности среды.

Разработчики IRC настолько хорошо продумали его архитектуру, что её с тех пор практически не требовалось изменять. Конечно, у него есть недостатки: короткие сообщения, проблема с кодировками, невозможность посмотреть историю сообщений при подключении. Однако он был и остаётся популярным средством для чата, хотя и в значительной мере потеснен со своих позиций. В частности, в 1998 году был придуман похожего назначения протокол Jabber — даже его название (англ. jabber болтовня, трёп; тарабарщина) отсылало к слову chat. Jabber содержал в себе многие технические новшества и постепенно получил широкое распространение, а также стал основой многих сервисов.

В 1989 году знаменитый британский учёный Тим Бернерс-Ли предлагает глобальный гипертекстовый проект, теперь известный как Всемирная паутина. Проект подразумевал публикацию гипертекстовых документов, связанных между собой гиперссылками, что облегчило бы поиск и консолидацию информации для учёных CERN. Он так же за два последующих года разрабатывает (совместно с его помощниками) протокол HTTP, язык гипертекстовой разметки HTML и идентификаторы URI, которые необходимы для реализации проекта.

В период с 1991 по 1993 год Бернерс-Ли усовершенствовал технические спецификации этих стандартов и опубликовал их. Но, всё же, официально годом рождения Всемирной паутины нужно считать 1989 год.

Всемирная паутина (англ. World Wide Web) — распределённая система, предоставляющая доступ к связанным между собой документам, расположенным на различных компьютерах, подключённых к Интернету. Для обозначения Всемирной паутины также используют слово веб (англ. web «паутина») и аббревиатуру WWW.

Всемирную паутину образуют сотни миллионов веб-серверов. Большинство ресурсов всемирной паутины основаны на технологии гипертекста. Гипертекстовые документы, размещаемые во Всемирной паутине, называются веб-страницами. Несколько веб-страниц, объединённых общей темой, дизайном, а также связанных между собой ссылками и обычно находящихся на одном и том же веб-сервере, называются веб-сайтом. Для загрузки и просмотра веб-страниц используются специальные программы — браузеры (англ. browser).

## Интернет-технологии

В 1990 году сеть ARPANET, проиграв в конкурентной борьбе NSFNet, прекращает своё существование. Так же в этом году состоялось первое подключение к сети Интернет по телефонной линии (Dialup access – «дозвон»).

1991 год ознаменовался общедоступностью Всемирной паутины в Интернете.

1993 год – появление первого распространенного веб-браузера с графическим интерфейсом NCSA Mosaic. Его код был открытым и впоследствии использовался при разработке браузеров Netscape Navigator и Internet Explorer.

В 1995 году появился браузер Opera. Первоначально Opera распространялась как условно-бесплатное ПО (shareware), этим, возможно, объясняется её низкая популярность в большинстве стран мира и высокая популярность в странах СНГ.

Происходит быстрый рост популярности Интернета.

В 1995 году роль маршрутизации всего сетевого трафика Интернета возложили на себя сетевые провайдеры, а суперкомпьютеры NSFNet вернулись к роли исследовательской сети.

В этом же году был образован Консорциум всемирной паутины (W3C), призванный упорядочить веб-стандарты.

С 1996 году Всемирная паутина (WWW) почти полностью подменяет собой понятие интернет, и обгоняет по трафику протокол пересылки файлов FTP.

1990-е годы произошло массовое объединение большинства существовавших сетей под флагом Интернет. Открытость технических стандартов во много способствовало быстрому росту сети.

К 1997 году в Интернете насчитывалось около 10 млн. компьютеров и более 1 млн. доменных имён. Интернет – популярнейшее средство для обмена информацией.

Сейчас получить доступ в интернет можно через телефон, радиоканалы, сотовую связь, спутники связи, кабельное телевидение, специальные оптико-волоконные линии и даже электропровода. А с 22 января 2010 года прямой доступ в Интернет появился и на Международной космической станции.

## Раздел 2. Технология WWW

### 2.1 Общие принципы построения глобальной сети

Интернет технологии это все, связанное с Интернет. Прежде всего, все многообразие сайтов, плюс чаты, форумы, электронная почта, Интернет торговля (включая Интернет магазины), социальные сети и масса всего, что создано для работы в Интернет или с использованием Интернет (см. рисунок 1).

Интернет технологии создаются по определенным методам в соответствии с определенными правилами при помощи специальных технических средств (сетей, серверов и т.п.) и специальных программ.

Обзор истории Интернет технологии, да и рассуждение о глобальном вхождении Интернет технологии в нашу жизнь оставлю за рамками этой статьи. А далее, поговорим, где практически Интернет технологии находят свое применение и главное развитие.



Рисунок 1 – Интернет - технологии

Для чего нужны Интернет технологии

Современные Интернет технологии позволяют создавать в сети:

- веб-сервера;
- сайты, порталы и блоги;
- электронную почту;
- форумы;
- чаты и ICQ подобное;
- видеоконференции, вебсеминары, телеконференции;
- wiki-энциклопедии.

Могли бы мы жить без Интернет технологий?

Вероятнее всего да. Но на сегодня Интернет технологии не только приближают человечество к общей доступности информации или



## Интернет-технологии

информационным ресурсам, но и определяют дальнейшее развитие всего общества.

Простое подключение к Интернет, глобально расширили количество пользователей Интернет информации, от любопытствующих фрилансеров, до расширения Интернет ресурсов правительственных, общественных и социальных органов, включая «компьютеризацию» и «интернетизацию» офисов, включает в обработку и передачу информации основную часть населения стран, беспокоящихся о своем развитии. И здесь не обойтись без новых Интернет технологий, автоматизирующих информационные ресурсы общества, делающих их интуитивно доступными и понятными.

Беспроводная связь WiFi и WiMAX, также, увеличивают число пользователей Интернет. Теперь уже не нужно быть «привязанным» к технологии проводной связи для выхода в net.

Интернет технологии вывели социальное общение людей на новый уровень. Уровень и способы общения меняются с внедрением новых Интернет технологий.

Огромно влияние Интернет технологий и на экономику стран. Говоря про нашу страну, за последние годы, технологии Интернет торговли, в буквальном смысле ломают традиционную торговлю, поворачивая ее в сторону Интернет технологий.

Нельзя не упомянуть, о влиянии Интернет технологии на процессы обучения и переобучения, причем для всех слоев населения от школьников, до среднего класса. Внедрение Интернет технологий в школах, институтах, online обучение на дому, семинары, вебинары, веб - конференции это уже стало общедоступно и становится обязательным в любом обучающем процессе.

Но Интернет технологии это не только прямые инструменты для работы в Интернет, но заставляют развиваться обслуживающие сервисы обслуживания Интернет. Это и безопасность Интернет, безопасность внутренних сетей, инженерное оборудование серверов, дата центров, площадок интернет провайдеров. И везде не обойтись, без постоянно изменяющихся Интернет технологий.

Последними тенденциями развития Интернет технологий, стало сделать доступным Интернет и Интернет технологии, доступными для совершенных «чайников», не имеющих представление и желания изучать, что куда подключается и какую и где программу нужно установить. Интернет технологии становятся невидимыми для простого пользователя, переходя на синхронизацию и автоматизацию процессов, делая интерфейс общения человека с машиной более коммуникативным, приближая нас к картинкам из фантастических фильмов о будущем.

Однако если Интернет и сети становятся более доступными, то Интернет технологии это сложнейшие системы объединяющие, как физические, так и логические компоненты.

## Интернет-технологии

Интернет технологии это физические элементы.

Физические составляющие имеют материальную ценность, что позволяет развивать бизнес Интернет технологий.

Физические элементы Интернет технологии включают в себя:

- Сетевые технологии. Сервера. Дата центры;
- Программное обеспечение Интернет;
- Топология Интернет (взаимодействие компьютеров и серверов в сети);
- Сетевые службы (электронная почта, служба DNS, протокол FTP и т.п.);
- Локальные и домашние сети, маршрутизаторы.

Логические составляющие Интернет технологии

Логические составляющие позволяют создать практически любой Интернет ресурс в сети: веб - сайт, веб - приложение, веб-портал.

Веб технологии:

- Языки разметки (HTML);
- Каскадные таблицы стилей (CSS);
- Скриптовой язык (JavaScript);
- Браузеры;
- Веб-страницы DOM (объектная модель документа (DOM));
- Языкразметки XML (Extensible Markup Language);
- Поисковые системы;
- SEO(поисковая оптимизация).

Разделение на физическую и логическую составляющие, несколько условны, потому-то они могут существовать только во взаимосвязи и не имеют особого назначения друг без друга.

Также нужно понимать, что это, конечно же, неполный список элементов Интернет технологии. Но он дает общее представление о таком объемном понятии, как Интернет технологии.

## 2.2 Технология гипертекста

Особенность электронных документов состоит в том, что посредством ссылки можно перейти прямо к тому тексту, на который она указывает. Такие ссылки могут указывать не только на тексты, но и на графическую, звуковую, видеоинформацию.

Поэтому они получили название гиперссылок, а документы, формируемые с помощью языка гипертекстовой разметки HTML (Hypertext Markup Language), стали называть гипертекстовыми.

Информация во Всемирной паутине хранится в форме Web-сайтов.

Web-сайт состоит из Web-страниц, объединенных гиперссылками.

Язык HTML является нечувствительным к регистру.

Язык HTML предназначен для описания форматированного

Интернет-технологии

электронного документа с помощью специальных управляющих кодов — *тегов* (от англ. tag — ярлык).

Тег показывает браузеру, что он имеет дело с элементом разметки, а не с простым текстом.

Каждый тег предназначен для отображения определённым образом.

Существует два типа тегов – парные и непарные.

Парные теги содержат открывающий и закрывающий тэги.

Одиночные

```
<тег атрибут1="значение" атрибут2="значение">
```

Парные

```
<тег атрибут1="значение" атрибут2="значение">
```

содержимое тега

```
</тег>
```

Атрибут хранит информацию о теге, от которой зависит, как браузер интерпретирует конкретный элемент. Атрибуты и теги могут быть обязательными и нет.

Содержимое тега обычно является текстом, который будет показан пользователю в том или ином виде в зависимости от тегов и значений атрибутов.

Нарушение правил HTML, как и нарушение правил любого другого языка, может привести к непониманию со стороны браузера и неверному отображению документа пользователю.

Пример структуры HTML документа (см. рисунок 2 и 3)

```
<!DOCTYPE html>
<html>
  <head>
    <title> Заголовок окна</title>
  </head>
  <body>
    Содержание страницы
  </body>
</html>
```

Рисунок 2 – Структура HTML

Интернет-технологии



Рисунок 3 – Структура HTML, разъяснения

Основные теги

<http://htmlbook.ru>  
<http://www.w3schools.com>

**<b>...</b>** (**<strong>**, **<i>**, **<em>**, **<u>**)

Устанавливает жирное (полужирное, курсивное, подчёркнутое) начертание шрифта.

**<p>...</p>**

Определяет текстовый абзац, который всегда начинается с новой строки.

Атрибут:

**align = "left | center | right | justify"** — определяет выравнивание текста в абзаце.

**<div>...</div>**

Определяет выделения фрагмента документа.

Атрибут:

**align = "left | center | right | justify"** — определяет выравнивание текста.

**title** — добавляет всплывающую подсказку к содержимому.

**<font>...</font>**

Представляет собой контейнер для изменения характеристик шрифта, таких как размер, цвет и гарнитура.

Атрибуты:

face — служит для задания гарнитуры шрифтов, использующихся для текста. Названий шрифтов можно привести несколько, через запятую. В этом случае, если первый указанный шрифт не будет найден, браузер станет использовать следующий по списку.

color — устанавливает цвет текста.

`<font color = "#ff0000">...</font>` установит красный цвет.

size — задает размер шрифта в условных единицах от 1 до 7. Средний размер, используемый по умолчанию принят 3. Размер шрифта можно указывать как абсолютной величиной (например, `size="4"`), так и относительной (например, `size="+1"`, `size="-1"`).

`<a href="URL">...</a>`                      `<a name="идентификатор">`

`<a href="#идентификатор">...</a>`

Предназначен для создания ссылок. В зависимости от присутствия атрибутов `name` или `href` тег `<a>` устанавливает ссылку или якорь. Якорем называется закладка внутри страницы, которую можно указать в качестве цели ссылки. При использовании ссылки, которая указывает на якорь, происходит переход к закладке внутри веб-страницы. Пример тега см. рисунок 4.

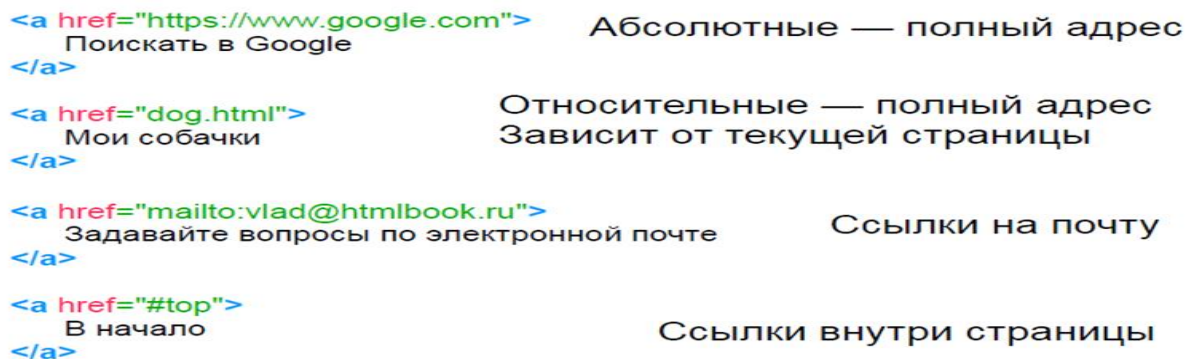


Рисунок 4 – Пример тега `<a>`

``

Предназначен для отображения на веб-странице изображений в графическом формате GIF, JPEG или PNG. Этот тег имеет единственный обязательный атрибут `src`, который определяет адрес файла с картинкой. Если необходимо, то рисунок можно сделать ссылкой на другой файл, поместив тег `<img>` в контейнер `<a>`. При этом вокруг изображения отображается рамка, которую можно убрать, добавив атрибут `border="0"` в тег `<img>`.

Атрибуты:

`src` — адрес изображения (абсолютный или относительный).

`align = "bottom | left | middle | right | top"` — указывает положение рисунка относительно текста или других изображений на веб-странице.

`border` — устанавливает толщину рамки вокруг изображения.

Интернет-технологии

height, width — задаёт высоту и ширину изображения.

`<h1>...</h1>` (`<h2>` ... `<h6>`)

Определяют заголовки (см. рисунок 5).

`<h1>Heading 1</h1>`

`<h2>Heading 2</h2>`

`<h3>Heading 3</h3>`

`<h4>Heading 4</h4>`

`<h5>Heading 5</h5>`

`<h6>Heading 6</h6>`

Рисунок 5 – Пример тега `<h>`

`<center>...</center>`

Выравнивает содержимое по центру относительно родительского элемента.

`<br>`

Устанавливает перевод строки в том месте, где этот тег находится.

`<xmp>...</xmp>`

Отображает содержимое контейнера «как есть» и шрифтом фиксированной ширины. Пока тег `<XMP>` не закрыт, все теги внутри него отображаются как обычный текст.

`<pre>...</pre>`

Определяет блок предварительно форматированного текста. Такой текст отображается обычно моноширинным шрифтом и со всеми пробелами между словами. По умолчанию, любое количество пробелов идущих в коде подряд, на веб-странице показывается как один.

`<!-- текст -->` — комментарий.

`<ul>...</ul>`

Устанавливает маркированный список. Каждый элемент списка должен начинаться с тега `<li>`.

`<ul>`

`<li>элемент маркированного списка</li>`

`<li>элемент маркированного списка</li>`

`</ul>`

Атрибут:

`type="disc | circle | square"` — устанавливает вид маркера списка.

## Интернет-технологии

`<ol>...</ol>`

Устанавливает нумерованный список. Каждый элемент списка должен начинаться с тега `<li>`.

`<table>...</table>`

Служит контейнером для элементов, определяющих содержимое таблицы.

Атрибуты:

`align` — задает выравнивание таблицы по краю окна браузера.

`bgcolor` — устанавливает цвет фона таблицы.

`border` — устанавливает толщину рамки в пикселах.

`bordercolor` — устанавливает цвет рамки таблицы.

`cellpadding` — определяет расстояние между границей ячейки и ее содержимым.

`cellspacing` — задает расстояние между внешними границами ячеек.

`width` — задает ширину таблицы.

`<tr>...</tr>`

Служит контейнером для создания строки таблицы.

Атрибуты:

`align` — определяет выравнивание содержимого ячеек по горизонтали.

`valign` — определяет выравнивание содержимого ячеек по вертикали.

`<td>...</td>`

Предназначен для создания одной ячейки таблицы.

Атрибуты:

`align` — определяет выравнивание содержимого ячейки по горизонтали.

`bgcolor` — цвет фона ячейки.

`colspan` — объединяет горизонтальные ячейки.

`rowspan` — объединяет вертикальные ячейки.

`valign` — выравнивание содержимого ячейки по вертикали.

# Таблицы

Теги

`<table>` - таблица

`<tbody>` - тело таблицы

`<tr>` - строка

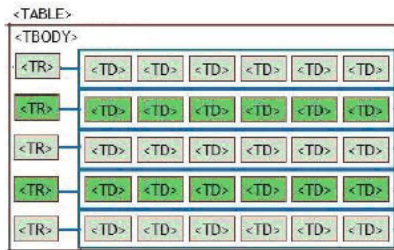
`<td>` - ячейка

Объединение ячеек по горизонтали

`<td colspan="число">...</td>`

Объединение ячеек по вертикали

`<td rowspan="число">...</td>`



	Col 1	Col 2	Col 3	Col 4	Col 5
Row 1	Header				
Row 2	Left Index	text		pic	pic
Row 3		pic	text		
Row 4		text			

Рисунок 6 – Пример тега

`<form></form>` - форма

`<input>` - элемент формы

Твое имя

Твой email

Тема сообщения

Текст \*

Защита от спама \*  [сменить картинку](#)

Рисунок 7 – Пример

**Все данные, заполненные пользователем в поля ввода, передаются web-серверу, который решает, как с ними дальше поступить (сохранить в базу данных, отправить на почту администратору и т. д.).**

Для реализации такой обработки необходимо написание отдельной программы на одном из языков программирования для web (например, PHP).

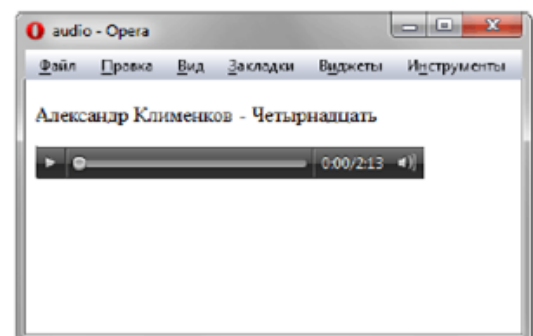


Таблица 1

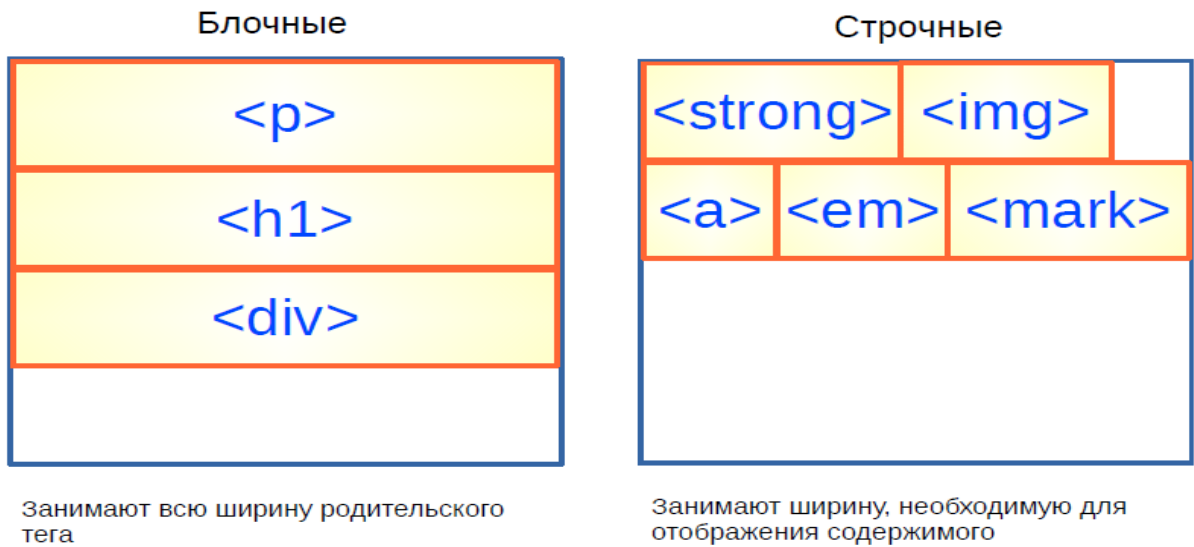
Тип	Описание	Вид
button	Кнопка.	<input type="button" value="Кнопка"/>
checkbox	Флажки. Позволяют выбрать более одного варианта из предложенных.	<input type="checkbox"/> Пиво <input type="checkbox"/> Чай <input type="checkbox"/> Кофе
file	Поле для ввода имени файла, который пересылается на сервер.	<input type="button" value="Обзор..."/> Файл не выбран.
hidden	Скрытое поле. Оно никак не отображается на веб-странице.	
image	Поле с изображением. При нажатии на рисунок данные формы отправляются на сервер.	<input alt="Отправить" type="image"/>
password	Обычное текстовое поле, но отличается от него тем, что все символы показываются звездочками. Предназначено для того, чтобы никто не подглядел вводимый пароль.	<input type="password"/>
radio	Переключатели. Используются, когда следует выбрать один вариант из нескольких предложенных.	<input type="radio"/> Пиво <input type="radio"/> Чай <input type="radio"/> Кофе
reset	Кнопка для возвращения данных формы в первоначальное значение.	<input type="button" value="Сброс"/>
submit	Кнопка для отправки данных формы на сервер.	<input type="button" value="Отправить запрос"/>
text	Текстовое поле. Предназначено для ввода символов с помощью клавиатуры.	<input type="text"/>

```
<video>
<source src="URL">
</video>
```

```
<audio>
<audio src="URL">
</audio>
```



## Типы тегов



Специальные символы в HTML  
Приведем некоторые в таблице 2.  
Таблица 2

Описание символа	Обозначение	Вид
Параграф	<code>&amp;sect;</code>	§
знак авторского права охраняемый знак	<code>&amp;copy;</code>	©
плюс-минус	<code>&amp;plusmn;</code>	±
вторая степень	<code>&amp;sup2;</code>	²
третья степень	<code>&amp;sup3;</code>	³
одна четвертая	<code>&amp;frac14;</code>	¼
одна вторая	<code>&amp;frac12;</code>	½
острое ударение	<code>&amp;acute;</code>	'
А с острым ударением	<code>&amp;Aacute;</code>	Á
амперсанд	<code>&amp;amp;</code>	&
левая угловая скобка	<code>&amp;lt;</code>	<

### 2.3. Каскадные листы стилей

CSS (англ. Cascading Style Sheets — каскадные таблицы стилей) — формальный язык описания внешнего вида документа, написанного с использованием языка разметки.

CSS используется создателями веб-страниц для задания цветов, шрифтов, расположения отдельных блоков и других аспектов представления внешнего вида этих веб-страниц. Основной целью разработки CSS являлось разделение описания логической структуры веб-страницы (которое производится с помощью HTML или других языков разметки) от описания

## Интернет-технологии

внешнего вида этой веб-страницы (которое теперь производится с помощью формального языка CSS). Такое разделение может увеличить доступность документа, предоставить большую гибкость и возможность управления его представлением, а также уменьшить сложность и повторяемость в структурном содержимом.

Способы описания каскадных листов стилей:

### 1. Стилль, встроенный в тег (inline styles)

```
<html>
<head>
  <title>Лабораторная работа № 2</title>
</head>
<body>
  <br><br><br><br><br><br>
<p style="color:red;
font-size:36pt; text-align:center">Привет</p>
</body>
</html>
```

### 2. Лист стилей, встроенный в документ

```
<html>
<head>
  <title>Лабораторная работа № 2</title>
  <style>
    p {color:blue; font-size:36pt; text-align:center}
  </style>
</head>
<body>
  <p>Привет</p>
</body>
</html>
```

### 3. Связанные листы стилей

В отдельном файле «mystyle.css» написать:

```
p {color:blue; font-size:36pt; text-align:center}
```

В HTML-файле подключить CSS-файл.

```
<html>
<head>
  <title>Лабораторная работа № 2</title>
  <link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
<body>
  <p>Привет</p>
</body>
</html>
```

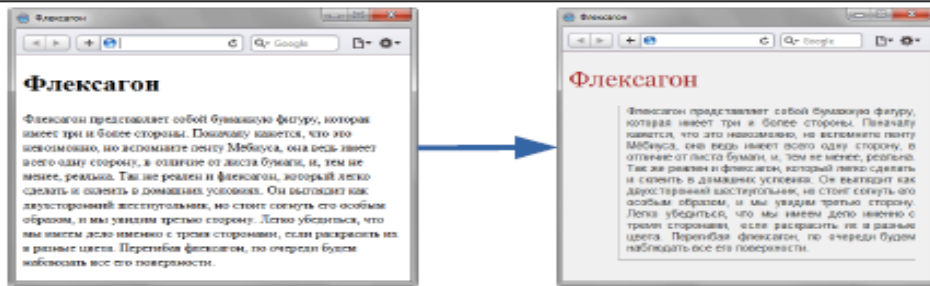
## Интернет-технологии

## Пример содержимого CSS файла

```

h1
{
  color: #a52a2a; /* Цвет заголовка */
}
p
{
  text-align: justify; /* Выравнивание по ширине */
}

```



Стили CSS могут применяться к одному или нескольким тегам. То, к чему применяется правило, называется селектором. Селекторы бывают следующих видов:

Селектор элемента:

p {font-family: arial;} — применяется ко всем тегам <p>.

Селектор класса:

.p\_red {color: red; background: yellow; font-weight: bold;}

Использовать это правило будут теги, для которых атрибут class="p\_red".

Селектор идентификатора:

#p\_red {color: red; background: yellow; font-weight: bold;}

Использовать это правило будут теги, для которых атрибут id="p\_red".


Селектор дочерних элементов:

p > b {color: green;}

Использовать это правило будут теги <b>, находящиеся внутри тега <p></p>

Основные свойства

Интернет-технологии

<p><b>background-color</b></p> <p>Определяет цвет фона элемента</p> <pre>body {   background-color: lightcyan; }</pre>	<p><b>background-image</b></p> <p>Определяет фоновое изображение</p> <pre>body {   background-image: url("paper.gif"); }</pre>
<p><b>background-position</b></p> <p>Положение фона элемента</p> <pre>body {   background-image: url("img_tree.png");   background-repeat: no-repeat;   background-position: right top;   margin-right: 200px; }</pre> 	<p><b>Background-attachment</b></p> <p>прокрутка фона</p> <p>fixed   scroll   local   inherit</p> <p><b>Background-repeat</b></p> <p>повторение фонового изображения</p> <p>no-repeat   repeat   repeat-x   repeat-y</p>

background-repeat — управляет способом циклического повторения фонового изображения:

- repeat-x размножает изображение только по вертикали,
- repeat-y размножает изображение только по горизонтали,
- repeat размножает изображение в обоих направлениях,
- no-repeat не размножает изображение;

color — устанавливает цвет текста элемента;

font-size — позволяет задать размер шрифта:

- xx-small, x-small, small, medium, large, x-large, xx-large,
- 50px размер в пикселях,
- 14pt размер в пунктах,
- 130% размер в процентах;

### Единицы измерения размеров в CSS

**Относительные**

- em относительно размера шрифта
- ex относительно высоты буквы x
- %
- ch относительно ширины символа 0
- gem относительно размера шрифта корневого элемента
- vw относительно 1% ширины окна вывода
- vh относительно 1% высоты окна вывода
- vmin относительно 1% меньшей размерности монитора
- vmax относительно 1% большей размерности монитора
- %
- px в пикселях

**Абсолютные**

- cm сантиметры
- ex миллиметры
- in дюймы (1in = 96px = 2.54cm)
- pt пункты (1pt = 1/72 of 1in)
- pc picas (1pc = 12 pt)

font-family — позволяет выбрать шрифт для отображения текста;

width — ширина.

height — высота.

Интернет-технологии

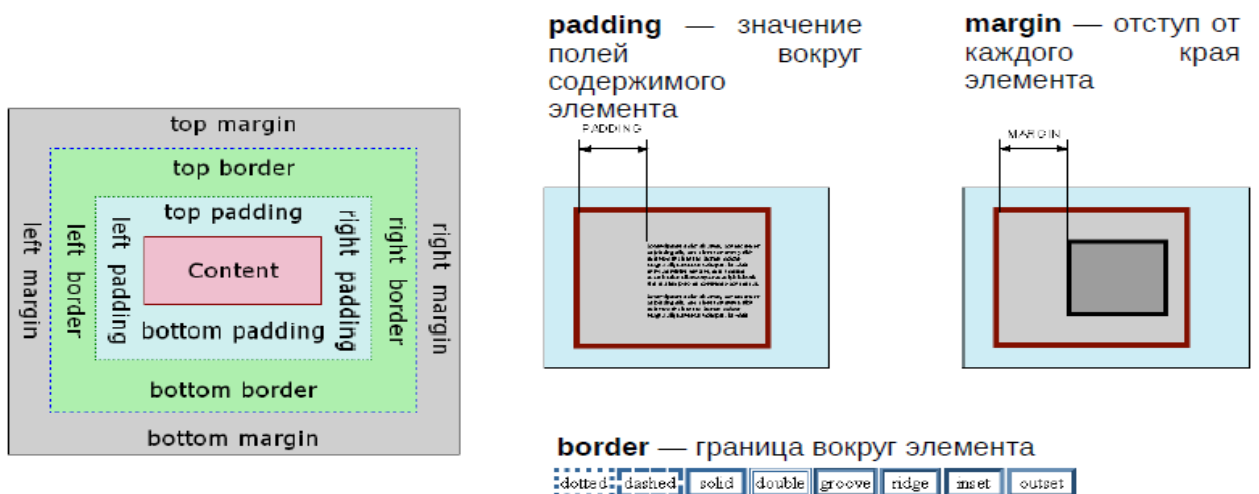
**border** — управляет внешним видом рамки вокруг элемента, позволяя определить в одной строке сразу несколько атрибутов: цвет, стиль и ширину. **border-collapse: collapse | separate** — устанавливает, как отображать границы вокруг ячеек таблицы. Это свойство играет роль, когда для ячеек установлена рамка, тогда в месте стыка ячеек получится линия двойной толщины. Значение **collapse** заставляет браузер анализировать подобные места в таблице и убирать в ней двойные линии. При этом между ячейками остается только одна граница, одновременно принадлежащая обоим ячейкам. То же правило соблюдается и для внешних границ, когда вокруг самой таблицы добавляется рамка.

**padding** — сокращённое свойство, предоставляет быстрый способ задать следующие параметры: **padding-top**, **padding-right**, **padding-bottom** и/или **padding-left**. **Padding** — устанавливает значение полей вокруг содержимого элемента. Поле называется расстояние от внутреннего края рамки элемента до воображаемого прямоугольника, ограничивающего его содержимое.

**margin** — устанавливает величину отступа от каждого края элемента. Отступом является пространство от границы текущего элемента до внутренней границы его родительского элемента.

**vertical-align** — выравнивает элемент по вертикали относительно своего родителя, окружающего текста или ячейки таблицы (может принимать значения **baseline|bottom|middle|sub|super|text-bottom|text-top|top|inherit**).

**text-align** — определяет горизонтальное выравнивание текста в пределах элемента (может принимать значения **center | justify | left | right | inherit**).



# CSS свойство display

```
p.inline
{
  display: inline;
}
```

**Блочные** элементы занимают всю ширину, **Строчные** элементы занимают необходимую для отображения содержимого ширину, **Таблицы** заполняют ширину по содержимому, автоматически вычисляя ширину колонок.

**inline** отобразить элемент как строчный,  
**block** отобразить элемент как блочный,  
**inline-block** отобразить элемент как строчный, а содержимое — как блочный,  
**table** отобразить элемент как таблицу,  
**none** не отображать элемент на странице,  
**initial** установить значение свойства в значение по умолчанию,  
**inherit** унаследовать значение от родительского элемента.

```
text-align: center; /*right, justify, left*/
margin: auto; /*выравнивание таблиц по центру экрана*/
width: 100%; /*указание конкретной ширины элемента*/
vertical-align: top; /*выравнивание содержимого внутри ячейки
таблицы или выравнивание inline-block элемента внутри родителя*/
```

## CSS позиционирование

**position: static;**  
 По умолчанию, элементы позиционируются согласно **нормальному потоку браузера** (слева направо сверху вниз).

**position: relative;**  
 Элемент позиционируется с помощью свойств **top**, **left**, **bottom** и **right** относительно своего положения в **нормальном потоке браузера** (где бы он был при position: static).

**z-index: число;**  
 Так как элементы могут накладываться друг на друга с использованием свойства position, то можно указать их расположение по оси z, чем больше свойство z-index у элемента, тем ближе к пользователю он «всплывёт», т. е. **элементы со значением z-index больше остальных окажутся выше** и перекроют другие.

**position: fixed;**  
 Элемент позиционируется **относительно окна браузера** с помощью свойств **top**, **left**, **bottom** и **right** и занимает фиксированное положение на странице, не изменяющееся даже при прокрутке. Элемент удаляется из **нормального потока браузера**, т. е. остальные элементы ведут себя так, как если бы его не было.

**position: absolute;**  
 Элемент позиционируется с помощью свойств **top**, **left**, **bottom** и **right** **относительно первого родительского элемента с position не равным static**. Элемент удаляется из **нормального потока браузера**, т. е. остальные элементы ведут себя так, как если бы его не было.

## CSS тень

**text-shadow: h-shadow v-shadow blur-radius|none|initial|inherit;**  
**h-shadow** смещение тени по горизонтали относительно текста  
**v-shadow** смещение тени по вертикали относительно текста  
**blur-radius** радиус размытия тени

```
h1
{
  font-family: sans-serif;
  color: white;
  text-shadow: 2px 2px 8px black;
}
```

Text-shadow on white text

**box-shadow: h-shadow v-shadow blur spread color [inset|none|initial|inherit];**  
**spread** растяжение или сужение тени,  
**inset** тень выводится внутри элемента

```
div
{
  width: 150px;
  height: 50px;
  background-color: yellow;
  box-shadow: 10px 10px 5px gray;
}
```



Применение CSS к документам HTML основано на принципах наследования и каскадирования.

## Интернет-технологии

Принцип наследования заключается в том, что свойства CSS, объявленные для элементов-предков, наследуются элементами потомками. Но, естественно, не все свойства CSS наследуются – например, если для тега параграфа `p` средствами CSS задана рамка, то она не будет наследоваться ни одним тегом, содержащимся в данном теге `p`, а вот если для параграфа `p` средствами CSS задан цвет шрифта (например, `color:green;`), то это свойство будет унаследовано каждым элементом-тегом, находящимся в параграфе.

Псевдоклассы определяют динамическое состояние элементов, которое изменяется со временем или с помощью действий пользователя.

**Псевдоклассы** — определяют реакцию или состояние элементов для ссылок

<b>:link</b>	непосещённая ссылка
<b>:visited</b>	посещённая ссылка
<b>:active</b>	стиль для активной ссылки

**для всех элементов**

<b>:hover</b>	элемент при проведении над ним мышью
<b>:active</b>	активный (выбирается по якорю) элемент
<b>:not(селектор)</b>	элементы, которые не соответствуют указанному селектору
<b>:target</b>	целевой элемент (выбирается по якорю)
<b>:first-child</b>	первый элемент своего родителя
<b>:last-child</b>	последний элемент своего родителя
<b>:nth-child(n)</b>	элемент на основе нумерации в дереве элементов
<b>:nth-last-child(n)</b>	элемент на основе нумерации в дереве элементов с конца

**Псевдоэлементы** — элементы, чётко не определённые в структуре документа

<b>::after</b>	добавление содержимого после элемента
<b>::before</b>	добавление содержимого перед элементом
<b>::first-letter</b>	выбор первой буквы элемента
<b>::first-line</b>	выбор первой строки элемента
<b>::selection</b>	выделенный пользователем внутри элемента текст



## Глава 3. Технологии динамической генерации страниц

### 3.1. Средства разработки динамических web-страниц

#### Верстка сайтов

Вёрстка веб-страниц – создание структуры html-кода, размещающего элементы веб-страницы (изображения, текст и т. д.) в окне браузера, согласно разработанному макету, таким образом, чтобы элементы дизайна выглядели аналогично.

Процесс сложен и имеет творческую основу, ни один из способов не является каноничным и принятым как основа. Все подходы к верстке имеют как преимущества, так и недостатки.

У каждого вида дизайна есть свои минусы и плюсы и выбор зависит от решаемой задачи. При этом может использоваться и смешанный дизайн.

Типы макетов:

- жёстко фиксированные (Rigid fixed),
- адаптивные резиновые (Adaptable fluid),
- расширяемые эластичные (Expandable elastic) макеты.

Фиксированный тип макета – дизайн (табличный либо блочный), в котором ширина столбца/рисунка заданы в пикселях, то есть оговорены точно.

Преимущества:

Такой дизайн гораздо легче разрабатывать, можно заранее предугадать, как будет выглядеть сайт.

У дизайнера есть возможность следить за размером строки.

Недостатки:

На различных разрешениях может появиться горизонтальный скроллинг, у сайта существует только одно оптимальное разрешение экрана.

Резиновый тип макета – дизайн, в котором ширина столбца/рисунка задана в процентах от текущего разрешения экрана.

Преимущества:

Сайт будет заполнять всё пространство браузера, что значительно улучшает его вид.

Сайт с таким дизайном будет одинаково смотреться на разных разрешениях.

Недостатки:

Разработка такого дизайна сплошной эксперимент, не знаешь, как поведёт себя вся структура сайта в тот или иной момент.

На больших разрешениях возможно появление слишком длинных строк, что значительно утрудняет читателя.

Адаптивная вёрстка – дизайн, который подстраивается (адаптируется) под размер экрана, в том числе может происходить перестройка блоков с одного места на другое, или их замена блоками отображаемыми только при определённом разрешении. Адаптивная вёрстка пришла на смену идеи создания специальных мобильных версий сайта, "живущих" на отдельных

поддоменах.

Преимущества:

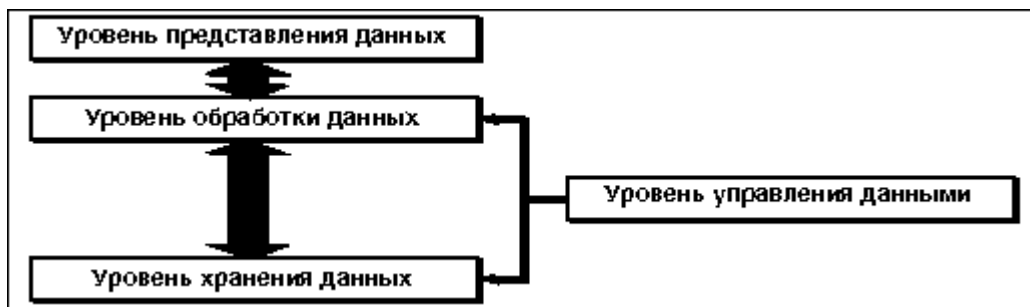
Сайт будет отображаться при разных размерах экрана так, как это наиболее удобно пользователю;

Недостатки:

Требует тщательной проработки нескольких макетов, для различных размеров экранов.

## Архитектура информационных систем

Любая информационная система должна иметь минимум три основные функциональные части – модули хранения данных, их обработки и интерфейса с пользователем. Каждая из этих частей может быть реализована независимо от двух других. Например, не изменяя программ, используемых для хранения и обработки данных, можно изменить интерфейс с пользователем таким образом, что одни и те же данные будут отображаться в виде таблиц, графиков или гистограмм. Не меняя программ представления данных и их хранения, можно изменить программы обработки, например изменив алгоритм полнотекстового поиска. И наконец, не меняя программ представления и обработки данных, можно изменить программное обеспечение для хранения данных, перейдя, например, на другую файловую систему.



Как и любая сложная система ИС может быть представлена в виде совокупности более простых элементов.

Архитектура программного обеспечения программы или вычислительной системы — это структура, которая включает программные компоненты, свойства этих компонентов, а также отношения между ними.

Есть много распространенных способов разработки программных модулей и их связей, в том числе:

- Blackboard
- Клиент-серверная модель (client-server)
- Архитектуры, построенные вокруг базы данных (database-centric architecture)

## Интернет-технологии

- Распределенные вычисления (distributed computing)
- Событийная архитектура (event-driven architecture)
- Front end and back end
- Неявные вызовы (implicit invocations)
- Монолитное приложение (monolithic application)
- Peer-to-peer
- Пайпы и фильтры (pipes and filters)
- Plugin
- Representational State Transfer
- Rule evaluation
- Поиск-ориентированная архитектуры
- Сервис-ориентированная архитектура
- Shared nothing architecture
- Software componentry
- Space based architecture
- Структурированная
- Трех-уровневая

### Архитектура клиент-сервер

Клиент-сервер — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг (сервисов), называемыми серверами, и заказчиками услуг, называемыми клиентами. Клиенты и серверы взаимодействуют через компьютерную сеть и могут быть как различными физическими устройствами, так и программным обеспечением.

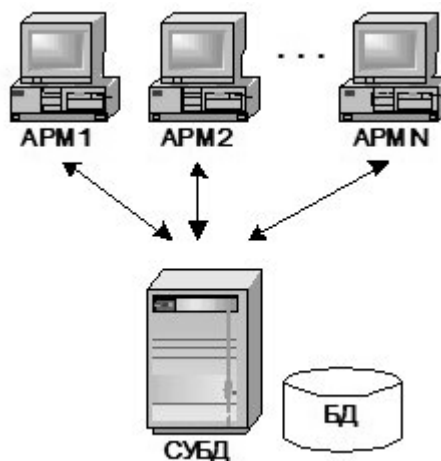


Рисунок – Двухуровневая клиент-серверная архитектура.

Данная архитектура характеризуется наличием двух взаимодействующих самостоятельных модулей — автоматизированного рабочего места (АРМа) и сервера базы данных, в качестве которого может выступать Microsoft SQL Server, Oracle, Sybase и другие.

## Интернет-технологии

Автоматизированное рабочее место (АРМ) — программно-технический комплекс, предназначенный для автоматизации деятельности определенного вида. Оно объединяет программно-аппаратные средства, обеспечивающие взаимодействие человека с компьютером.

Сервер БД отвечает за хранение, управление и целостность данных, а также обеспечивает возможность одновременного доступа нескольких пользователей. Клиентская часть представлена так называемым “толстым” клиентом, то есть приложением на котором сконцентрированы основные правила работы системы и расположен пользовательский интерфейс программы. При всей простоте построения такой архитектуры, она обладает множеством недостатков, наиболее существенные из которых — это высокие требования к сетевым ресурсам и пропускной способности сети компании, а также сложность обновления программного обеспечения из-за “размазанной” бизнес-логики между АРМом и сервером БД. Кроме того, при большом количестве АРМов возрастают требования к аппаратному обеспечению сервера БД, а это, как известно, самый дорогостоящий узел в любой информационной системе.

### Трехуровневая архитектура

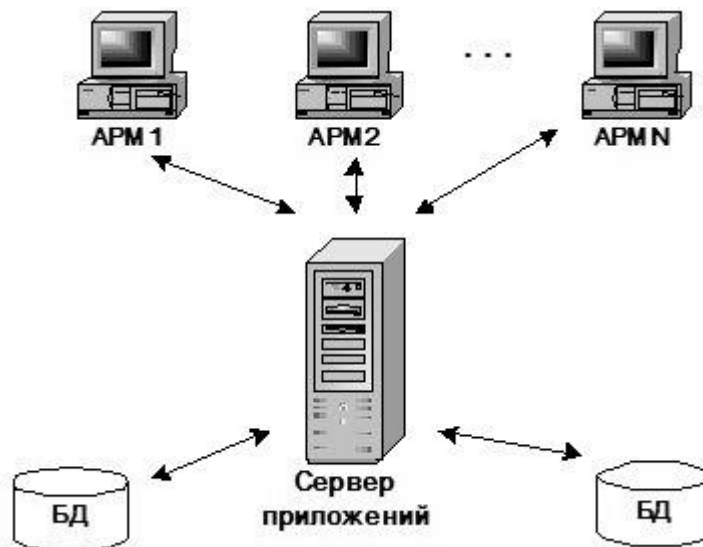


Рисунок – Трехуровневая клиент-серверная архитектура

### Основные понятия данной архитектуры

— Терминал — это интерфейсный (обычно графический) компонент, который представляет первый уровень, собственно приложение для конечного пользователя. Первый уровень не должен иметь прямых связей с базой данных (по требованиям безопасности), быть нагруженным основной бизнес-логикой (по требованиям масштабируемости) и хранить состояние приложения (по требованиям надежности). На первый уровень

## Интернет-технологии

может быть вынесена и обычно выносятся простейшая бизнес-логика: интерфейс авторизации, алгоритмы шифрования, проверка вводимых значений на допустимость и соответствие формату, несложные операции (сортировка, группировка, подсчет значений) с данными, уже загруженными на терминал.

- Сервер приложений располагается на втором уровне. На втором уровне сосредоточена большая часть бизнес-логики. Вне его остаются фрагменты, экспортируемые на терминалы, а также погруженные в третий уровень хранимые процедуры и триггеры.
- Сервер базы данных обеспечивает хранение данных и выносятся на третий уровень. Обычно это стандартная реляционная или объектно-ориентированная СУБД.

Такие архитектуры более разумно распределяют модули обработки данных, которые в этом случае выполняются на одном или нескольких отдельных серверах. Эти программные модули выполняют функции сервера для интерфейсов с пользователями и клиента — для серверов баз данных. Кроме того, различные серверы приложений могут взаимодействовать между собой для более точного разделения системы на функциональные блоки, выполняющие определенные роли. Например, можно выделить сервер управления персоналом, который будет выполнять все необходимые для управления персоналом функции. Связав с ним отдельную базу данных, можно скрыть от пользователей все детали реализации этого сервера, разрешив им обращаться только к его общедоступным функциям. Кроме того, такую систему очень просто адаптировать к Web, поскольку проще разработать html-формы для доступа пользователей к определенным функциям базы данных, чем ко всем данным.

Благодаря концентрации бизнес-логики на сервере приложений, стало возможно подключать различные БД. Теперь, сервер базы данных освобожден от задач распараллеливания работы между различными пользователями, что существенно снижает его аппаратные требования. Также снизились требования к клиентским машинам за счет выполнения ресурсоемких операций сервером приложений и решающих теперь только задачи визуализации данных. Именно поэтому такую схему построения информационных систем часто называют архитектурой «тонкого» клиента.

Но, тем не менее, узким местом, как и в двухуровневой клиент-серверной архитектуре, остаются повышенные требования к пропускной способности сети, что в свою очередь накладывает жесткие ограничения на использование таких систем в сетях с неустойчивой связью и малой пропускной способностью (Internet, GPRS, мобильная связь).

В простейшей конфигурации физически сервер приложений может быть совмещён с сервером базы данных на одном компьютере, к которому по сети подключается один или несколько терминалов. В «правильной» (с точки зрения безопасности, надёжности, масштабирования) конфигурации сервер

Интернет-технологии

базы данных находится на выделенном компьютере (или кластере), к которому по сети подключены один или несколько серверов приложений, к которым, в свою очередь, по сети подключаются терминалы.

WEB-сервер

Веб-сервером называют специальную программу, обеспечивающую работу сайта, а также компьютер, на котором она работает. Веб-сервер обрабатывает запросы, полученные через Интернет от браузера (клиента) и выдает в ответ нужный ресурс: HTML-код страниц, изображения, видеоролики и т.п.

Веб-сервер – сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-поток или другими данными.

Клиент, которым обычно является веб-браузер, передаёт веб-серверу запросы на получение ресурсов, обозначенных URL-адресами.

Ресурсы – это HTML-страницы, изображения, файлы, медиа-поток или другие данные, которые необходимы клиенту. В ответ веб-сервер передаёт клиенту запрошенные данные. Этот обмен происходит по протоколу HTTP.

Примеры веб-серверов: Apache и IIS ().

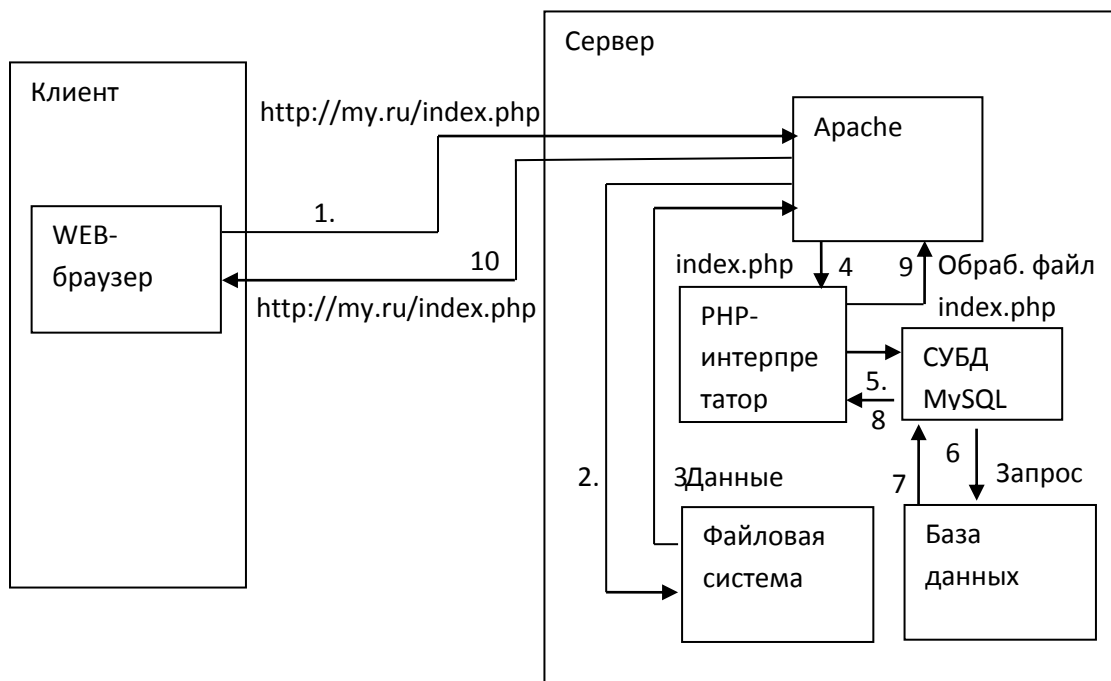


Рисунок – выполнения HTTP-запроса WEB-сервером.

Для веб-сервера, как правило, используется специально выделенный компьютер, работающий круглосуточно и имеющий постоянное подключение к Интернет. Так как к серверу могут одновременно

## Интернет-технологии

обращаться множество клиентов, сервер должен быть быстродействующим и надежным.

На рисунке изображено несколько серверов в специальном шкафу.



Рисунок – Серверный шкаф

Понятие "Веб-сервер" может относиться как к железу так и к программному обеспечению.

1. С точки зрения железа, 'Веб-сервер' это компьютер который хранит ресурсы сайта (HTML документы, CSS стили, JavaScript файлы и другое) и доставляет их на устройство конечного пользователя (веб-браузер и т.д.). Обычно подключен к сети Интернет и может быть доступен через, доменное имя, например mozilla.org. mozilla.org.

2. С точки зрения ПО, Веб-сервер включает в себя некоторые вещи, которые контролируют доступ Веб-пользователей к размещенным на сервере файлам, это минимум HTTP сервера. HTTP сервер это часть ПО которая понимает URL'ы (веб-адреса) и HTTP (протокол который использует ваш браузер для просмотра веб-страниц).

Простыми словами, когда браузеру нужен файл размещенный на веб-сервере, браузер запрашивает его через HTTP. Когда запрос достигает нужного веб-сервера (железо), сервер HTTP (ПО) передает запрашиваемый документ обратно, также через HTTP.

### Функции веб сервера

Главная задача веб сервера принимать HTTP-запросы от пользователей, обрабатывать их, переводить в цифровой компьютерный код. Затем выдавать HTTP-ответы, преобразуя их из миллионов нулей и единичек в изображения, медиа-поток, буквы, HTML страницы.

Любой веб сервер, для удобства его использования пользователями, должен иметь удобный веб-браузер. Он передает веб серверу запросы, преобразованные в URL-адреса интернет - ресурсов.

Наряду со стандартными функциями, некоторые веб серверы имеют дополнительные. Так, к примеру, соответствующее программное

обеспечение может фиксировать число обращений пользователей к тому или иному ресурсу, записывать их в отдельный журнал. А еще они могут поддерживать HTTPS, что не маловажно для защищенного соединения между сайтами и пользователями. Зачастую веб-сервер устанавливается вместе с мейл-сервером. Это позволяет пользователям быстро переходить на страничку почты прямо с сайта, нажав всего лишь на одну гиперссылку.

Часто требуется не просто выдавать пользователю готовые ресурсы, а производить вычисления «на лету» и выдавать их результат. Например, когда один пользователь оставляет другому сообщение на сайте, сервер должен проверить его и сохранить у себя, чтобы потом показать адресату. В таких случаях

возможностей HTML недостаточно и необходимо создавать специальные программы с помощью языков программирования. Одним из таких языков является PHP.

PHP (PHP Hypertext Preprocessor) – серверный язык создания приложений, ориентированный на веб-разработку. PHP код может быть внедрен в HTML-страницу и будет выполняться при каждом ее посещении. Код PHP интерпретируется веб-сервером и генерирует HTML-код или другой вывод (например,

графику), который отсылается браузеру пользователя. Так как PHP интерпретируемый язык, он не требует компиляции (преобразования в машинный код в файле .exe) – программы хранятся на веб-сервере как обычные текстовые файлы.

### 3.2. Программирование на языке PHP

PHP (Hypertext Preprocessor) – наиболее простой скриптовый язык программирования, широко применяющийся при создании динамически генерируемых веб-страниц.

Основная масса Интернет ресурсов, на данный момент, написана с использованием именно этого языка программирования. При всей своей простоте, PHP позволяет разрабатывать профессиональные веб-проекты любой сложности, от небольших сайтов до крупных порталов.

PHP-код программы выполняется на стороне сервера. После того, как пользователь совершил на сайте некое действие, например клик по ссылке в меню, с целью перейти на другую страницу сайта, браузер посылает запрос серверу на соответствующую страницу с PHP-кодом. Далее, PHP-код обрабатывается интерпретатором PHP и генерируется HTML-код, который возвращается серверу. Сервер в свою очередь, передаёт этот HTML-код обратно браузеру. В результате пользователь видит отображение в браузере новой страницы, имеющей свой HTML-код.

PHP-программа запускается при вводе ее адреса в строке браузера или отправке ей данных формы. Выполнение этих действий занимает несколько секунд. Это время зависит от производительности клиентской и серверной



## Интернет-технологии

машины и скорости передачи данных между ними.

Рассмотрим абстрактный пример – типичная последовательность действий при аутентификации (входе в систему) пользователя.

1. Пользователь вводит логин и пароль в HTML-форму и жмет кнопку отправки.
2. Данные через Интернет отправляются на веб-сервер. Браузер начинает ждать ответа от сервера.
3. Веб-сервер запускает PHP-программу и передает ей введенный логин и пароль.
4. PHP-программа:
  - a. Подключается к базе данных.
  - b. Делает запрос к базе данных «существует ли пользователь с таким логином и паролем?».
  - c. Если ответ положительный, программа выводит страницу приветствия. Если отрицательный – сообщение об ошибке.
5. Веб-сервер отправляет ответ PHP-программы назад в браузер в виде HTML-кода.
6. Браузер обрабатывает HTML-код и выводит страницу на экран компьютера пользователя.

Язык PHP может работать внутри HTML-страниц, наделяя его возможностью генерации содержания по требованию.

Web-сайт перестает быть совокупностью статических страниц, а представляет собой набор Web-страниц с внедренными PHP-скриптами, которые генерируют в соответствии с запросом пользователя HTML-страницу.

Язык PHP является языком сценариев, выполняемых на стороне сервера.

Обычно файлы, содержащие php-код имеют расширение php.

Большой плюс языка PHP состоит в том, что PHP-код можно внедрять непосредственно в HTML-файлы. PHP-код встраивается в HTML-страницы при помощи угловых скобок и знака вопроса:

```
<?php ...здесь находится код программы php... ?>
```

Также можно использовать теги <? ... ?>, <% ... %> или

<script language="php"> ... </script>, но наиболее предпочтительным является первый вариант <?php ... ?>, т.к. он является общепринятым и работает при любых настройках интерпретатора PHP.

Для создания веб-проектов на языке php, необходимо программировать, используя либо установленный локальный сервер на своём компьютере, либо работая с помощью удалённого сервера. Удалённый сервер не всегда удобен, да и, как правило, за это надо платить.

Для создания сервера на своём компьютере понадобятся следующие программы: Apache или Denver (сервер), MySQL (базы данных), PHP.

## Интернет-технологии

Все программы актуальных версий можно найти на официальных сайтах разработчиков.

Создание PHP-файлов, написание кода и работа с ним ничем не отличается от того же процесса, что и при работе с HTML.

Можно в принципе обойти процесс установки серверных программ на компьютер и долгой их настройки.

Это возможно при использовании уже готовых сборок в виде одной целой программы. Например, Top Server, phpDesigner7pe и подобные программы. Такой вариант идеален для создания и разработки сайтов на компьютере и проверки их в «живую».

Для использования компьютера в качестве сервера такой вариант, конечно, использовать нельзя.

Для тех, кто только начинает знакомиться с языком программирования PHP и не имеет особого навыка в программировании, кому, нужно лишь только написав некоторый код проверить его в действии, такой вариант будет самым удобным.

### Внедрение PHP-кода в HTML-код

Текст скрипта можно свободно смешивать с текстом обычного форматированного HTML-документа, заключая его при этом в специальные последовательности символов:

`<?php скрипт; ?>`

*Пример:*

`<html>`

`<head>`

`<title>Пример</title>`

`</head>`

`<body>`

`<? php`

`echo "Привет, я - скрипт PHP!";`

`?>`

`</body>`

`</html>`

Как Вы уже поняли, оператор **echo** нужен для вывода данных. Содержимое (в нашем случае пока только текст) берём в **кавычки**, а в конце ставим точку с запятой; это обозначает конец работы оператора.

### Операторы вывода PHP

При работе с любым языком программирования очень часто возникает необходимость вывода информации на экран.

Так, в PHP для того, чтобы вывести информацию на экран предусмотрены операторы «print» и «echo».

Данные операторы практически ничем не отличаются, поэтому можно использовать тот, который больше нравится.

`<?php`

## Интернет-технологии

```
print (" Привет, я - скрипт PHP!"); /*Выводим на экран строку Привет, я -  
скрипт PHP!*/ ?>
```

```
<? php  
echo "Привет, я - скрипт PHP!"; ?>
```

Для вывода большого количества информации на экран (большой объем текста, различных таблиц с применением html кода) существует синтаксис heredoc. Он начинается с символов <<<, после которых указывается идентификатор. В качестве идентификатора можно использовать любое слово, например,

**TEXT** или **HERE**.

Данный идентификатор необходимо указать также в конце кода.

```
<?php  
echo <<<HERE  
<h1>Пример</h1>  
<p>Пример вывода большого объема текста  
с использованием html</p>  
<p>Второй абзац такого же объёмного текста.</p>  
HERE;  
?>
```

## Описание переменных

При программировании на PHP можно не скупиться на объявление новых переменных. Принципы экономии памяти, которые были актуальны несколько лет назад, сегодня в расчет не принимаются. Однако, при хранении в переменных больших объемов памяти, лучше удалять неиспользуемые переменные.

Вообще, переменная - это область оперативной памяти, доступ к которой осуществляется по имени. Все данные, с которыми работает программа, хранятся в виде переменных (исключение — константа, которая, впрочем, может содержать только число или строку). Такого понятия, как указатель (как в Си), в PHP не существует — при присвоении переменная копируется один-в-один, какую бы сложную структуру она ни имела. Тем не менее, в PHP, начиная с версии 4, существует понятие ссылок — жестких и символических.

Имена переменных чувствительны к регистру букв: например, \$var — не то же самое, что \$Var или \$VAR:

Чтобы PHP мог отличать имена переменных от команд или функций, имя переменной должно начинаться со знака доллара "\$".

То же самое "Привет, я - скрипт PHP! " можно получить следующим образом:

```
<?php
```

## Интернет-технологии

```
$message = "Привет, я - скрипт PHP!";  
echo $message;  
?>
```

В PHP нет необходимости описывать типы переменных.

```
<?php  
    $a=2;  
    $b=3;  
    echo $a+$b;  
?>
```

Выведет 5.

## Типы данных в PHP

PHP поддерживает восемь простых типов данных:

Четыре скалярных типа:

- **boolean** (двоичные данные)
- **integer** (целые числа)
- **float** (числа с плавающей точкой или 'double')
- **string** (строки)

Два смешанных типа:

- **array** (массивы)
- **object** (объекты)

И два специальных типа:

- **resource** (ресурсы)
- **NULL** ("пустые")

Существуют также несколько псевдотипов:

- **mixed** (смешанные)
- **number** (числа)
- **callback** (обратного вызова)

## Ветвления и циклы

## Оператор ветвления

В операторе **if** ставится условие, если оно выполняется, то выполняется часть кода следующая в фигурных скобках за ним, если нет, то часть кода идущая после оператора **else**. Также можно создавать сложные условия с помощью **elseif**. Схематично выглядит так:

```
if (условие) {  
    операции, выполняющиеся в случае, если условие верно;  
} else {  
    операции, выполняющиеся в случае, если условие неверно;  
}
```

## Оператор выбора

Этот оператор позволяет делать более сложные ветвления, чем **if**, более

## Интернет-технологии

простым способом. На самом деле, при помощи `if` можно полностью заменить этот оператор, но знать про него полезно.

Итак, этот оператор называется `switch`. В дословном переводе — «переключатель». Схематично выглядит так:

```
switch (переменная или выражение) {  
  case условие:  
    команды ;  
    break;  
  case условие:  
    команды ;  
    break;  
  default: команды;  
}
```

При входе в оператор `switch`, сначала вычисляется выражение, стоящее в скобках. Затем перебираются все значения, указанные при помощи `case`. Когда значение, которое имеет вычисленное в скобках выражение, найдено у какого-либо `case`, начинается исполнение участка кода внутри `case`. И вот здесь важно понимать, что означает "внутри `case`". Когда соответствующее значение найдено, произойдет исполнение всего, что есть дальше того `case`, которому соответствует значение. И это значит, что если после этого `case` есть другие, они тоже будут выполнены. Этот эффект алгоритмически обоснован и часто нужен. Но часто он и не нужен.

В таком случае можно воспользоваться оператором `break`. Он позволяет выйти из оператора `switch`. Поставив его после кода, записанного внутри каждого `case`, мы обеспечим себе исполнение только одного участка кода для каждого значения.

Кроме этого, у оператора `switch` может быть необязательная часть `default`. Код в ней выполнится, если значение выражения не совпало ни с одним из значений в `case`.

Это очень похоже на часть `else` для оператора `if .. elseif .. else`.

Но не стоит забывать, что `switch` и `if` — не одно и то же. Иногда может пригодиться возможность `switch` исполнять весь участок кода, включая другие `case`, после того как найдено совпадение.

В PHP реализованы стандартные алгоритмические конструкции: цикл с предусловием (`while`),

цикл с постусловием (`do...while`),

цикл `for`.

Кроме того, есть некоторые неклассические циклы, например, цикл по элементам не скалярной переменной (`foreach`).

Цикл с предусловием

Цикл `while` — простейший тип цикла. Основная форма оператора `while`:

```
while (условие)
```

```
{  
    тело цикла;  
}
```

Смысл оператора **while** прост. Он предписывает PHP выполнять вложенный(е) оператор(ы) до тех пор пока условие выполняется. Значение выражения проверяется каждый раз при начале цикла, так что если значение выражения изменится внутри цикла, то он не прервется пока не начнется следующий цикл. Иногда, если условие не выполняется с самого начала, цикл не выполняется ни разу. Если в цикле только один оператор, то фигурные скобки можно опустить.

**while** (условие) оператор;

### Цикл с постусловием

Цикл **do..while** очень похож на **while** за исключением того, что значение логического выражения проверяется не до, а после окончания итерации.

Основное отличие в том, что **do..while** гарантировано выполнится хотя бы один раз, что в случае **while** не обязательно.

Для циклов **do..while** существует только один вид синтаксиса:

```
do {  
    тело цикла  
} while (условие);
```

Этот цикл выполнится один раз, так как после окончания условие не выполнится (**условие**), и выполнение цикла завершится.

### Цикл for

Циклы **for** — наиболее мощные циклы в PHP.

Синтаксис цикла **for**:

```
for (инструкция инициализации; условие; шаговая инструкция) {  
    тело цикла;  
}
```

Инструкция инициализации выполняется в начале цикла. В начале каждого шага цикла проверяется *условие*. Если оно выполняется, то цикл продолжается и выполняется *шаговая инструкция*. Если нет, то цикл заканчивается. В конце каждого шага цикла выполняется тело цикла.

### Цикл по элементам

Цикл по элементам (**foreach**) очень удобен для обработки массивов (и других структурированных данных). Выглядит он очень просто:

**foreach** (массив **as** \$ключ=>\$значение)

```
{  
    команды;  
}
```

Здесь команды циклически выполняются для каждого элемента массива, при этом очередная пара **ключ=>значение** оказывается в переменных **\$ключ** и **\$значение**.

```
for (инструкция инициализации; условие; шаговая инструкция)
{
for (инструкция инициализации; условие; шаговая инструкция)
{
if (условие)
{
continue;
}
тело цикла;
}
тело цикла;
}
```

### Немного о прерывании циклов

Можно также использовать еще один интересный оператор в циклах — **break**.

С помощью него можно в любой момент оборвать любой цикл, как **while**, **do..while**, так и **for** или **foreach**.

Кроме **break** существует еще один управляющий циклом оператор — **continue**.

Он позволяет сразу же переходить к следующей итерации цикла, не выполняя в текущей ничего, что стоит после этого оператора:

### Массивы в php

Массивы (arrays) – это упорядоченные наборы данных, представляющие собой список однотипных элементов.

Существует два типа массивов, различающиеся по способу идентификации элементов.

1. В массивах первого типа элемент определяется индексом в последовательности. Такие массивы называются **простыми массивами**.

2. Массивы второго типа имеют ассоциативную природу, и для обращения к элементам используются ключи, логически связанные со значениями. Такие массивы называют **ассоциативными массивами**.

Важной особенностью PHP является то, что PHP, в отличие от других языков, позволяет создавать массивы любой сложности непосредственно в теле программы (скрипта).

Массивы могут быть как одномерными, так и многомерными.

### Простые массивы и списки в PHP

При обращении к элементам простых индексированных массивов используется целочисленный индекс, определяющий позицию заданного элемента.

*Простые одномерные массивы:*

Обобщенный синтаксис элементов простого одномерного массива:

**Симя[индекс];**

Массивы, индексами которых являются числа, начинающиеся с нуля - это списки. С технической точки зрения разницы между простыми массивами и списками нет.

Простые массивы можно создавать, не указывая индекс нового элемента массива, это за вас сделает PHP.

*Простые многомерные массивы:*

Обобщенный синтаксис элементов многомерного простого массива:

**Симя[индекс1][индекс2]..[индексN];**

**Пример:**

```
$fruit[0]="банан";
```

```
$fruit[1]="ананас"; // и так далее
```

или

```
$names=array("Вася", "Маша", "Дима", "Лена");
```

```
$surnames[]="Иванов";
```

```
$surnames[]="Петров";
```

Простейший способ обойти массив в цикле — использование **count()** для определения числа элементов в массиве, а затем организовать цикл **for()**:

```
$m = array("ru", "de", "us");
```

```
$n = count($m);
```

```
for($i=0; $i<$n; ++$i)
```

```
{
```

```
    echo ($m[$i]."<BR>");
```

```
}
```

**Ассоциативные массивы в PHP**

В PHP индексом массива может быть не только число, но и строка. Причем на такую строку не накладываются никакие ограничения: она может содержать пробелы, длина такой строки может быть любой.

Ассоциативные массивы особенно удобны в ситуациях, когда элементы массива удобнее связывать со словами, а не с числами.

Итак, массивы, индексами которых являются строки, называются ассоциативными массивами.

*Одномерные ассоциативные массивы:*

Одномерные ассоциативные массивы содержат только один ключ (элемент), соответствующий конкретному индексу ассоциативного массива.

**array( key => value)**

*Многомерные ассоциативные массивы:*

Многомерные ассоциативные массивы могут содержать несколько ключей, соответствующих конкретному индексу ассоциативного массива.

**array(**

**key => value,**

**key2 => value2,**



```
key3 => value3,
...
)
```

В качестве параметров она принимает любое количество разделенных запятыми пар *key => value* (ключ => значение).

Запятая после последнего элемента массива необязательна и может быть опущена. Для многострочных массивов обычно используется завершающая запятая, так как позволяет легче добавлять новые элементы в конец массива. Начиная с PHP 5.4 возможно использовать короткий синтаксис определения массивов, который заменяет языковую конструкцию *array()* на *[]*.

## Комментарии в php

Комментарии – служат примечанием для людей, читающих код. При выполнении программы комментарии игнорируются.

В PHP существуют следующие виды комментариев:

```
/* Многострочный комментарий в стиле C */
// Комментарий в стиле C++
# Комментарий в стиле Perl
```

## Операции с числами

+ сложение

- вычитание

\* умножение

/ деление

% деление по модулю (возвращает остаток от деления)

Также существуют объединенные операции присваивания:

$\$a += \$b$  аналогично  $\$a = \$a + \$b$

$\$a -= \$b$  аналогично  $\$a = \$a - \$b$

$\$a *= \$b$  аналогично  $\$a = \$a * \$b$

$\$a /= \$b$  аналогично  $\$a = \$a / \$b$

$\$a \% = \$b$  аналогично  $\$a = \$a \% \$b$

## Операторы инкремента и декремента

Операции инкремента (++) и декремента (--) аналогичны операциям +=1 и -=1 – соответственно прибавляют и вычитают единицу из числа.

PRE - сначала значение изменяется, потом передается дальше на обработку

<b>++\$a</b>	Увеличивает \$a на единицу и возвращает значение \$a.
<b>--\$a</b>	Уменьшает \$a на единицу и возвращает значение \$a.

POST - сначала переменная отдается на обработку, а только потом изменяется

## Интернет-технологии

<code>\$a++</code>	Возвращает значение <code>\$a</code> , а затем увеличивает <code>\$a</code> на единицу.
<code>\$a--</code>	Возвращает значение <code>\$a</code> , а затем уменьшает <code>\$a</code> на единицу.

### Булевские величины

В PHP для обозначения булевских величин используются ключевые слова **true (истина)** и **false (ложь)**.

С каждой величиной в PHP связано булево значение истинности:

Для целых чисел и чисел с плавающей точкой, число считается значением **false**, если оно равно 0, и **true** - в остальных случаях.

Строка считается значением **false**, если она пуста, и **true** – в остальных случаях.

Массив без элементов считается значением **false**, и **true** – в остальных случаях.

### Операции сравнения

Используются для сравнения двух значений.

Выражения, в которых используются эти операции, возвращают значения истина (**true**) или ложь (**false**).

`==` равно

`===` равно и относятся к одному типу

`!=` неравно

`<` меньше

`>` больше

`<=` меньше или равно

`>=` больше или равно

### Логические операции

`!` НЕ

`&&` И

`||` ИЛИ

### Строки

Существует 2 типа строк: **разбираемые** и **неразбираемые**.

Разбираемые строки заключаются в двойные кавычки. В таких строках происходит подстановка значений переменных и обработка управляющих последовательностей символов.

В неразбираемых строках, заключенных в одинарные кавычки, этого не происходит.

Пример:

PHP-код: `$age = 2011 - 1936; //74`

`$string1 = "МГДДЮТ $age года<BR>";`

`$string2 = 'МГДДЮТ $age года<BR>';`

```
echo $string1;
```

```
echo $string2;
```

В браузере:

```
МГДДЮТ 74 года  
МГДДЮТ $age года
```

Управляющие последовательности позволяют включить в строку специальные символы:

`\n` – перевод строки

`\r` – возврат каретки

`\\` - обратный слеш

`\$` - знак доллара

`'` – одинарная кавычка

`"` – двойная кавычка

Пример: `echo "1000 \$";`

В браузере: 1000 \$

### Структуры включения *PHP*-файлов (библиотек)

Для включения файлов в страницы PHP имеется две инструкции — **require** и **include**, которые считывают и выполняют код, представленный в указанном файле.

Благодаря этому можно создавать повторно используемые функции, константы и прочий код и хранить их централизованно в файле, который доступен всем остальным сценариям. Инструкция **require** осуществляет замену только в момент запуска сценария, а **include** — в момент выполнения.

**require(имя\_файла)** и **include(имя\_файла)**

Отличие между ними состоит в том, что

Конструкция **require** позволяет включать файлы в сценарий PHP до исполнения сценария и при ошибке останавливает программу,

а **include** позволяет включать файлы в код PHP скрипта во время выполнения сценария и при ошибке только генерирует предупреждение.

Инструкция **include** не является оператором в прямом смысле, а напрямую подставляет содержимое файла. Это приведет к тому, что первая строка файла будет в теле цикла, а остальные — вне его. Поэтому необязательные на первый взгляд фигурные скобки, тем не менее, обязательны.

Таким образом, **include** имеет в функциональном отношении некоторые преимущества, связанные с возможностью его использования в циклах. Тем не менее, он является более ресурсоемким и везде по возможности необходимо использовать **require**.

Также имеются инструкции **require\_once** и **include\_once**, которые осуществляют подключение файла только в том случае, если до этого затребованный файл не был подключен. По возможности везде используйте инструкции с суффиксом **\_once**, что позволит сделать программу более масштабируемой.

## Удаление элементов из массивов

Операция удаления элемента из массива является простой и полностью аналогичной операции удаления значения, присвоенного переменной. Достаточно просто вызвать функцию **unset()**, как показано ниже:

```
$my_array[0] = 'значение';
$my_array[1] = 'это значение нужно удалить';
$my_array[2] = 'еще одно значение';
unset($my_array[1]);
```

После завершения выполнения кода эта переменная `$my_array` будет содержать два значения ('значение', 'еще одно значение'), ассоциированные с двумя индексами (соответственно 0 и 2).

## Простые PHP-функции для получения сведений о массивах

Функция	Описание
<b>is_array()</b>	Принимает единственный параметр любого типа и возвращает истинное значение, если этот параметр является массивом; в противном случае возвращает ложное значение
<b>count()</b>	Принимает в качестве фактического параметра массив и возвращает количество непустых элементов в массиве
<b>sizeof()</b>	Идентична count()
<b>in_array()</b>	Принимает два фактических параметра: элемент (который может быть записан в массив в качестве значения) и массив (который может содержать элемент). Возвращает истинное значение, если элемент содержится в массиве в качестве значения; в противном случае возвращает ложное значение. (Обратите внимание на то, что эта функция не выполняет проверку на наличие в массиве определенных ключей.)
<b>shuffle()</b>	Располагает элементы массива в случайном порядке
<b>sort()</b>	Сортирует массив в порядке возрастания
<b>rsort()</b>	Сортирует массив в порядке убывания
<b>asort()</b>	Сортирует массив по значениям в порядке возрастания
<b>arsort()</b>	Сортирует массив по значениям в порядке убывания
<b>ksort()</b>	Сортирует массив по ключам в порядке возрастания
<b>krsort()</b>	Сортирует массив по ключам в порядке убывания
<b>isset(\$array[\$key])</b>	Принимает форму <code>array[key]</code> и возвращает истинное значение, если часть, обозначенная ключом <code>key</code> , представляет собой допустимый ключ для массива <code>array</code> . (Это — специализированный способ использования более общей функции <code>isset()</code> , который проверяет, является ли переменная связанной.)

Интернет-технологии

Функция	Параметры	Побочный эффект	Возвращаемое значение
<b>current()</b>	Один фактический параметр с обозначением массива	Отсутствует	Значение из пары "ключ-значение", на которую в настоящее время указывает внутренний "текущий" указатель (или ложное значение, если таковое значение отсутствует)
<b>reset()</b>	Один фактический параметр с обозначением массива	Продвигает указатель в обратном направлении так, чтобы он указывал на первую пару "ключ-значение" (или занял позицию "перед началом массива", если массив пуст)	Первое значение, хранящееся в массиве, или ложное значение, в случае пустого массива
<b>next()</b>	Один фактический параметр с обозначением массива	Продвигает указатель на один элемент. Если указатель уже направлен на последний элемент, эта функция продвигает указатель "за пределы массива", поэтому последующий вызов функции <code>current()</code> возвратит ложное значение	Значение, на которое направлен указатель после его продвижения (или ложное значение, если текущее значение отсутствует, т.е. указатель вышел за пределы массива)
<b>prev()</b>	Один фактический параметр с обозначением массива	Продвигает указатель в обратном направлении на один элемент. Если указатель уже направлен на первый элемент, продвигает указатель за пределы массива, устанавливая его "перед началом массива"	Значение, на которое направлен указатель после его продвижения в обратном направлении (или ложное значение, если текущее значение отсутствует, т.е. указатель вышел за пределы массива)
<b>end()</b>	Один фактический параметр с обозначением массива	Перемещает указатель в прямом направлении и устанавливает его на последнюю пару "ключ-значение"	Последнее значение, которое находится в настоящее время в списке пар "ключ-значение"
<b>pos()</b>	Один фактический параметр с обозначением массива	Отсутствует (Эта функция является псевдонимом функции <code>current()</code> .)	Значение пары "ключ-значение", на которую в настоящее время направлен указатель
<b>array_walk</b>	Первый параметр —	Функция <code>array_walk()</code>	Возвращает значение 1

Интернет-технологии

<p><b>k()</b></p>	<p>фактический параметр с обозначением массива, второй параметр — имя функции с двумя (или тремя) фактическими параметрами, которая вызывается применительно к каждой паре, состоящей из ключа и значения, а третий параметр — необязательный фактический параметр</p>	<p>вызывает функцию, указанную в качестве второго фактического параметра, применительно к каждой паре "ключ-значение". Побочные эффекты зависят от побочных эффектов переданной функции</p>	
<p><b>each()</b></p>	<p>Один фактический параметр с обозначением массива</p>	<p>Перемещает указатель вперед, на следующую пару "ключ-значение"</p>	<p>Массив, который содержит данные о ключе и значении из пары "ключ-значение", которая была текущей до перемещения указателя (или ложное значение, если текущая пара отсутствует, т.е. указатель находится за пределами массива). В возвращенном массиве ключ и значение хранятся под собственными ключами массива, соответственно 0 и 1, а также под собственными ключами массива, соответственно 'key' и 'value'</p>

**Выборка значений**

**Выборка с помощью индекса или ключа**

Если по индексу 5 в массив \$my\_array было записано некоторое значение, то вычисление выражения \$my\_array[5] должно привести к выборке хранимого значения.

**Конструкция list()**

Применяется для присваивания нескольких значений подряд идущих элементов массива переменным.

**Конструкция each()**

Также используется совместно с list() для обхода массива

```
<?php
$fruit = array('a' => 'apple', 'b' => 'banana', 'c' => 'cranberry');
reset($fruit);
while (list($key, $val) = each($fruit)) {
    echo "$key => $val";
}
?>
```

Результат выполнения данного примера:

```
a => apple  
b => banana  
c => cranberry
```

Функция **reset()** переводит внутренний указатель на первый элемент массива. А затем для каждого элемента массива название индекса и его значение заносятся соответственно в **\$key** и **\$value**.

```
$fruit = array('Апельсин', 'Яблоко', 'Банан', 'Груша');  
list($fruit1, $fruit2) = $fruit;
```

В результате выполнения этих операторов происходит присваивание строки **'Апельсин'** переменной **\$fruit1** и строки **'Яблоко'** переменной **\$fruit2** (строка **'Банан'** не присваивается какой-либо переменной, поскольку не было задано достаточное количество переменных).

Переменным в конструкции **list()** значения элементов массива присваиваются в том порядке, в каком эти элементы были первоначально сохранены в массиве. Обратите внимание на то, насколько необычной является синтаксическая структура используемой операции присваивания — конструкция **list()** находится слева от знака операции присваивания (**=**), тогда как в этой позиции обычно можно встретить только имена переменных.

### Функция

Это именованная последовательность операторов, которая при необходимости может принимать параметры и возвращать значение. Для определения функции используется следующий синтаксис:

```
function имя_функции($параметр1, $параметр2 ...)  
{  
операторы  
}
```

В отличие от имен переменных, имена функции не чувствительны к регистру: **My\_Function()**, **my\_function()** и **MY\_FUNCTION()** будут обозначать одну и ту же функцию. Т.к. функция подразумевает какое-то действие, в качестве имен лучше всего использовать английские глаголы.

Функция может вернуть значение в программу при помощи оператора **return**. При его вызове выполнение функции прекращается.

На имена функций накладываются следующие ограничения:

Функция не может иметь то же имя, что и существующая функция;

Имя функции может содержать только буквы, цифры и символ подчеркивания;

Имя функции не может начинаться с цифры.

### Встроенные переменные

Язык PHP делает удобной работу с внешними данными, такими как переменные, полученные от клиентов по протоколу HTTP, переменными сессий, переменными окружения и т.д.

В PHP 5 определены следующие суперглобальные переменные (superglobals),

## Интернет-технологии

которые являются ассоциативными массивами:

**\$\_GLOBALS** — глобальные переменные выполняемого скрипта. Ключи элементов массива соответствуют названиям переменных.

**\$\_GET** — переменные, переданные скрипту методом GET. Массив аналогичен устаревшему \$HTTP\_GET\_VARS, который доступен, но его применение не рекомендуется.

**\$\_POST** — переменные, переданные скрипту методом POST.

**\$\_COOKIE** — переменные, переданные скрипту посредством механизма HTTP cookies.

**\$\_FILES** — данные, полученные при загрузке файлов на сервер методом POST.

**\$\_ENV** — переменные, передаваемые скрипту из среды окружения.

**\$\_SESSION** — переменные, зарегистрированные в текущей сессии для выполняемого скрипта.

**\$\_SERVER** — переменные, установленные веб-сервером или напрямую связанные со средой выполнения текущего скрипта.

**Пример содержимого массива:**

**\$\_SERVER["HTTP\_HOST"]** = www.mysite.ru

**\$\_SERVER["HTTP\_USER\_AGENT"]** = Mozilla/5.0 (Windows NT 6.1; WOW64; rv:21.0) Gecko/20100101 Firefox/21.0

**\$\_SERVER["HTTP\_ACCEPT"]** = text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

**\$\_SERVER["HTTP\_ACCEPT\_LANGUAGE"]** = ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3

**\$\_SERVER["SERVER\_SIGNATURE"]** =

Apache/2.2.4 (Win32) mod\_ssl/2.2.4 OpenSSL/0.9.8k PHP/5.3.1 Server at www.mysite.ru Port 80

**\$\_SERVER["SERVER\_SOFTWARE"]** = Apache/2.2.4 (Win32) mod\_ssl/2.2.4 OpenSSL/0.9.8k PHP/5.3.1

**\$\_SERVER["SERVER\_NAME"]** = www.mysite.ru

**\$\_SERVER["SERVER\_ADDR"]** = 127.0.0.1

**\$\_SERVER["SERVER\_PORT"]** = 80

**\$\_SERVER["REMOTE\_ADDR"]** = 127.0.0.1

**\$\_SERVER["DOCUMENT\_ROOT"]** = W:/home/mysite.ru/www

**\$\_SERVER["SERVER\_ADMIN"]** = admin@localhost

**\$\_SERVER["SCRIPT\_FILENAME"]** = W:/home/mysite.ru/www/lab4.php

**\$\_SERVER["REMOTE\_PORT"]** = 6645

**Вывод содержимого всего массива:**

```
reset($_SERVER);
while(list($key,$value) = each($_SERVER))
{
    echo "\$_SERVER[\"$key\"] = \"$value\"<BR>";
}
```

или



Интернет-технологии

```
foreach($_SERVER as $key => $value)
{
    echo "\$_SERVER[\" . $key . "\"] = " . $value . "<BR>";
}
```

Формы в HTML

<form></form> - форма

<input> - элемент формы

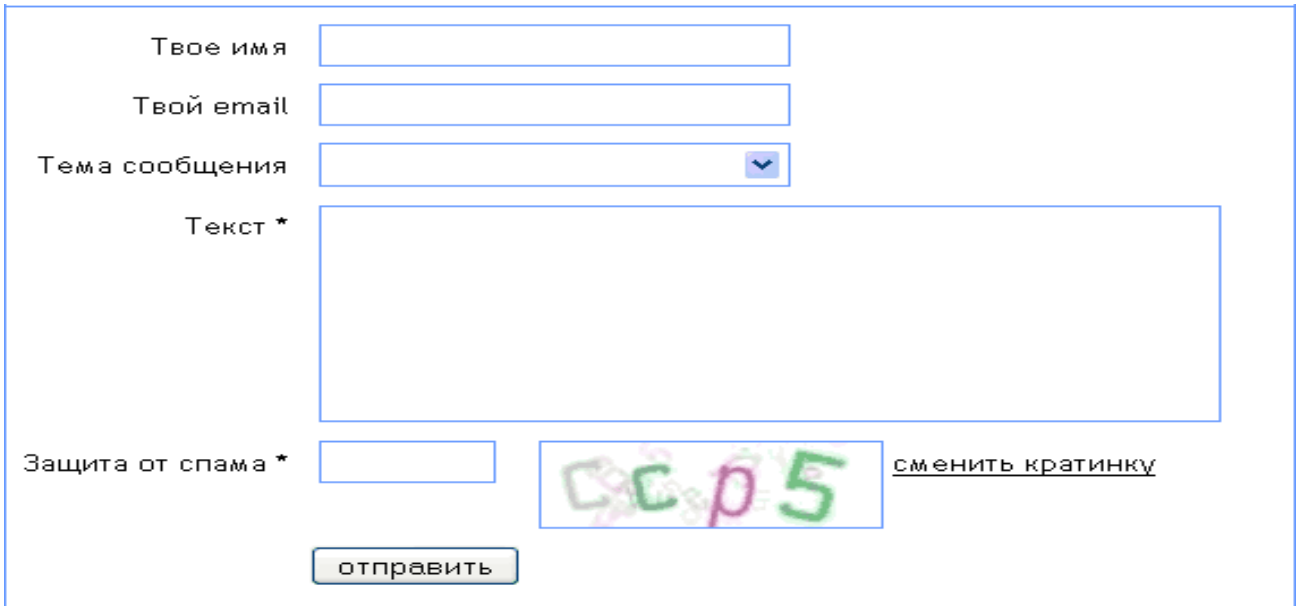


Рисунок 7 – Пример

**Все данные, заполненные пользователем в поля ввода, передаются web-серверу, который решает, как с ними дальше поступить (сохранить в базу данных, отправить на почту администратору и т. д.).**

Для реализации такой обработки необходимо написание отдельной программы на одном из языков программирования для web (например, PHP).

Форма — это средство, позволяющее получать различные данные от пользователя web-страницы и затем обрабатывать их с помощью JavaScript на машине клиента либо передавать их на сервер, который затем может сформировать страницу специально для этого пользователя, занести эти данные в базу данных и т. д.

Формы HTML используются для передачи данных от пользователя PHP-приложениям.

Границы формы определяются тегом <form>...</form>

Тег <form> имеет следующие атрибуты:

— **action** — адрес серверной программы, которая запустится при вызове команды submit;

— **method** — метод передачи данных (“get” или “post”).

Кроме того, как и большинство тегов, тег <form> имеет атрибуты **id**, **name**, **class**, **style**. В теге также можно указать слово атрибут **disabled**, что делает

## Интернет-технологии

все поля формы недоступными для редактирования пользователем.

Атрибуты:

**action**="URL" - адрес приложения, которому будут переданы данные формы (по умолчанию текущий URL)

**method**="..." - метод передачи параметров: get (по умолчанию) или post

Свойство **method** сообщает браузеру, как передавать данные: включив их в URL (метод "get") или поместив их в пакет данных протокола HTTP, по которому и осуществляется в основном передача данных в Интернет (метод "post").

Первый метод удобен тогда, когда нет необходимости передавать большие объемы данных и не нужно скрывать их от глаз пользователей (например, ключевые слова для поисковых серверов или номер товара в базе данных Интернет-магазина).

Метод "post" не имеет ограничений на объем пересылаемых данных и используется для передачи больших массивов данных и файлов.

**Замечание:**

При отсылке параметров методом get данные присоединяются к URL запроса после знака вопроса (?) парами ключ=значение, разделенными символом амперсанда (&).

*Например: <http://www.yandex.ru/yandsearch?rpt=rad&text=HTML>*

Такой способ отсылки небезопасен (например, при передаче паролей), т.к. все данные видны в строке браузера.

При использовании метода post данные передаются в теле запроса и не видны в браузере.

Интернет-технологии

Тип	Описание	Вид
button	Кнопка.	<input type="button" value="Кнопка"/>
checkbox	Флажки. Позволяют выбрать более одного варианта из предложенных.	<input type="checkbox"/> Пиво <input type="checkbox"/> Чай <input type="checkbox"/> Кофе
file	Поле для ввода имени файла, который пересылается на сервер.	<input type="button" value="Обзор..."/> Файл не выбран.
hidden	Скрытое поле. Оно никак не отображается на веб-странице.	
image	Поле с изображением. При нажатии на рисунок данные формы отправляются на сервер.	<input alt="Отправить" type="image"/>
password	Обычное текстовое поле, но отличается от него тем, что все символы показываются звездочками. Предназначено для того, чтобы никто не подглядел вводимый пароль.	<input type="password"/>
radio	Переключатели. Используются, когда следует выбрать один вариант из нескольких предложенных.	<input type="radio"/> Пиво <input type="radio"/> Чай <input type="radio"/> Кофе
reset	Кнопка для возвращения данных формы в первоначальное значение.	<input type="button" value="Сброс"/>
submit	Кнопка для отправки данных формы на сервер.	<input type="button" value="Отправить запрос"/>
text	Текстовое поле. Предназначено для ввода символов с помощью клавиатуры.	<input type="text"/>

Для установки большинства элементов формы используется тег **<input>**.

Обязательными атрибутами являются

**type** – указывает тип элемента

**name** – обозначает имя ключа (переменной), передающегося обрабатывающей программе.

Значение атрибута type для тега **<input>** по умолчанию определено как text, поэтому его можно не указывать явно.

Атрибуты текстового поля:

Атрибут	Описание
Size	Ширина текстового поля, которая определяется числом символов моноширинного шрифта. Иными словами, ширина задается количеством близстоящих букв одинаковой ширины по горизонтали.
maxlength	Устанавливает максимальное число символов, которое может быть введено пользователем в текстовом поле. Когда это количество достигается при наборе, дальнейший ввод становится невозможным. Если этот атрибут опустить, то можно вводить строку длинее самого поля.
name	Имя поля, предназначено для того, чтобы обработчик формы мог его идентифицировать.
value	Начальный текст отображаемый в поле.

`<select>...</select>` - создает выпадающий список.

Обязательным атрибутом является `name="..."` – указывает имя ключа (переменной), передающегося обрабатывающей программе. Варианты выбора в списке задаются вложенными тегами `<option>...</option>`.

Содержимое тега `<option>...</option>` отображается как текст пункта в списке.

Атрибуты `<option>`: **selected** - (без значения) устанавливает данный пункт выбранным по умолчанию `value="..."` - значение передаваемое при отсылке формы, если данный элемент был выбран. Если не задано будет передано содержимое `<option>...</option>`.

#### Синтаксис

```
<select name="spisok">
<option>Пункт 1</option>
<option>Пункт 2</option>
</select>
```

`<textarea>...</textarea>` - создает многостроковое поле ввода текста. Внутри него указывается текст, который будет отображен в этом поле по умолчанию.

Атрибуты: `name="..."` - (обязательный) задает имя ключа (переменной), передающегося обрабатывающей программе

`cols="N"` - количество колонок поля (ширина в символах)

`rows="N"` - количество столбцов поля (высота в символах)

**Примечание:** если вводимый текст не умещается в поле, появляются полосы прокрутки - количество столбцов поля (высота в символах)

Элементы формы можно собирать в группы с помощью контейнера `<fieldset>...</fieldset>`.

Группа будет выделена рамкой, в верхней части которой может быть отображен заголовок, определенный с помощью тега `<legend>...</legend>`.

```
<form action="...">
  <fieldset> <legend>Работа со временем</legend>
  <input type="checkbox"> создание пунктуальности;<br>
  <input type="checkbox"> изменение восприятия времени и часов.
  <p><input type="submit"></p>
</fieldset>
</form>
```

Работа со временем

создание пунктуальности ;

изменение восприятия времени  
и часов.

Подача запроса

Общими для всех типов элементов являются такие атрибуты, как:  
***disabled*** — делает элемент недоступным для пользователя,  
***readonly*** — элемент только для чтения.

<FORM >

Имя пользователя <INPUT type="text" name="username" value="Гость"><BR>

Пароль <INPUT type="password" name="password"><BR>

Пол

<INPUT type="radio" name="sex" value="male" checked> М

<INPUT type="radio" name="sex" value="female"> Ж<BR>

Род занятий

<SELECT name="occupation">

<OPTION value="teacher">Преподаватель</OPTION>

<OPTION value="student"

selected>Студент</OPTION>

</SELECT><BR>

Немного о себе

<TEXTAREA name="about" cols="25" rows="3">свое хобби и  
т.п.</TEXTAREA><BR>

<INPUT type="checkbox" name="mail" value="yes" checked>

Присылать мне новости сайта<BR>

<INPUT type="submit" value="Зарегистрироваться"> <INPUT

type="reset" value="Очистить">

</FORM>

Интернет-технологии

Имя пользователя

Пароль

Пол  М  Ж

Род занятий

Немного о себе

Присылать мне новости сайта

### 3.3 Передача данных с web-форм

#### Методы передачи данных с форм

При отправке форм PHP-сценариям данные формы становятся доступны в программе в массивах `$_GET` или `$_POST` в зависимости от выбранного метода отправки формы.

Массивы `$_GET` и `$_POST` являются суперглобальными, т.е. доступны во всей программе.

Это ассоциативные массивы, содержащие список ключей, представляющих имена элементов формы, указанные в атрибутах `name`, и ассоциированных с ними значений.

	GET	POST
Ограничение на объём	4 Кб	Ограничения задаются сервером.
Передаваемые данные	Видны сразу всем.	Видны только при просмотре через расширения браузера или другими методами.
Кэширование	Страницы с разными запросами считаются различными, их можно кэшировать как отдельные документы.	Страница всегда одна.
Закладки	Страницу с запросом можно добавить в закладки браузера и обратиться к ней позже.	Страницы с разными запросами имеют один адрес, запрос повторить нельзя.

### *Cookie*

Cookies — это механизм хранения данных браузером удаленного компьютера для идентификации возвращающихся посетителей и хранения параметров веб-страниц (например, переменных).

Куки используются для хранения небольшой по объему информации у клиента (посетителя) сайта, например: настройки сайта (цвет фона страниц, язык, оформление таблиц и т.д.), а также другой информации.

Файлы Cookies представляют собой обыкновенные текстовые файлы, которые хранятся на диске у посетителей сайтов. Файлы Cookies и содержат ту информацию, которая была в них записана сервером.

Для установки Cookies используется функция `SetCookie()`. Для этой функции можно указать шесть параметров, один из которых является обязательным:

- `name` — задает имя (строка), закрепленное за Cookie;
- `value` — определяет значение переменной (строка);
- `expire` — время "жизни" переменной (целое число). Если данный параметр не указать, то Cookie будут "жить" до конца сессии, то есть до закрытия браузера. Если время указано, то, когда оно наступит, Cookie самоуничтожится.
- `path` — путь к Cookie (строка);
- `domain` — домен (строка). В качестве значения устанавливается имя хоста, с которого Cookie был установлен;
- `secure` — передача Cookie через защищенное HTTPS-соединение.

Обычно используются только три первые параметра.

Пример установки Cookies:

```
<?php
    // Устанавливаем Cookie до конца сессии:
    SetCookie("Test", "Value");

    // Устанавливаем Cookie на один час после установки:
    SetCookie("My_Cookie", "Value", time()+3600);
?>
```

При использовании Cookies необходимо иметь в виду, что Cookies должны устанавливаться до первого вывода информации в браузер (например, оперетором `echo` или выводом какой-либо функции). Поэтому желательно устанавливать Cookies в самом начале скрипта. Cookies устанавливаются с помощью определенного заголовка сервера, а если скрипт выводит что-либо, то это означает, что начинается тело документа. В результате Cookies не будут установлены и может быть выведено предупреждение.

Функция `SetCookie()` возвращает `TRUE` в случае успешной установки Cookie. В случае, если Cookie установить не удастся `SetCookie()` возвратит `FALSE` и возможно, предупреждение (зависит от настроек PHP).

Получить доступ к Cookies и их значениям достаточно просто. Они хранятся в суперглобальных массивах `$_COOKIE` и `$HTTP_COOKIE_VARS`.

Пример:

```
echo $_COOKIES["Test"];
```

Прочитать значение cookie можно только после того как он был отправлен браузеру. Это означает, что при установке cookie его нельзя прочитать до тех пор, пока браузер не перезагрузит страницу.

При чтении значений Cookies обращайтесь внимание на проверку существования Cookies, например, используя оператор `isset()`.

```
if(isset($_COOKIES["Test"]))
{
    echo ($_COOKIES["Test"]);
}
```

Иногда возникает необходимость удаления Cookies. Сделать это несложно, необходимо лишь вновь установить Cookie с идентичным именем и пустым параметром. Например:

```
<?php
    // Удаляем Cookie 'Test':
    SetCookie("Test","");
?>
```

Можно установить массив Cookies, используя квадратные скобки в именах Cookies [], а затем прочитать массив Cookies и значения этого массива:

```
<?php
    // Устанавливаем массив Cookies:
    setcookie("cookie[1]", "Первый");
    setcookie("cookie[2]", "Второй");
    setcookie("cookie[3]", "Третий");

    // После перезагрузки страницы мы отобразим
    // Состав массива Cookies 'cookie':
    if (isset($_COOKIE['cookie'])) {
        foreach ($_COOKIE['cookie'] as $name => $value) {
            echo "$name : $value <br>";
        }
    }
?>
```

Преимущества использования Cookies неоспоримы. Однако существуют и некоторые проблемы их использования. Первая из них заключается в том, что посетитель может блокировать прием Cookies браузером или попросту удалить все Cookies или их часть. Таким образом, мы можем иметь некоторые проблемы в идентификации таких посетителей.



### Сессии

Очень часто при написании скриптов на php необходимо хранить некоторую информацию о посетителе в течение всего сеанса его работы. Типичным примером такой ситуации является необходимость "помнить" логин и пароль пользователя при его нахождении в закрытой части сайта (например, на форуме). Собственного говоря, именно для этого и существует механизм сессий, реализованный в php.

Сессии представляю собой группы переменных, которые хранятся на сервере и относятся только к текущему пользователю. Для того, чтобы обеспечить обращение нужных пользователей к нужным переменным, т.е. для уникальной идентификации этих пользователей их браузерами сохраняются файлы cookie. Эти файлы имеют значение только для сервера и не могут быть использованы для извлечения информации о пользователе.

Для того, чтобы в скрипте можно было работать с сессиями, необходимо сначала инициализировать механизм сессий. Делается это с помощью функции `session_start()`. При этом будет создана новая сессия или восстановлена уже существующая. Как сервер узнает что ему делать: создавать или восстанавливать? Очень просто. Дело в том, что когда посетитель заходит на сайт, ему присваивается 32-х разрядный идентификатор вида: `abcd1efgh2ijkl3mnop4qrs5tuv6wxyz`, который "следует" за ним при всех перемещениях по сайту. "Следование" обеспечивается либо по средствам cookies, либо, если они отключены у пользователя, добавлением ко всем адресам GET-запроса вида: `PHPSESSID=идентификатор`, т.е. адрес `myscript.php` превратится в `myscript.php?PHPSESSID=идентификатор`. При обработке запроса на сервере движок php ищет идентификатор в переданных ему данных и, если находит и сессия не устарела, то обновляет ее. В противном случае создается новая. Кстати, идентификатор сессии можно получить с помощью функции `session_id()`;

После того, как механизм сессий был инициализирован в скрипте мы можем сохранять любую информацию в ассоциативном массиве `$_SESSION`. Этот массив является глобальной переменной сессии. Таким образом, сохранив в нем, к примеру, значения переменных `login` и `password` на странице авторизации мы можем использовать их на всех остальных страницах защищенной части сайта.

Завершение сессии происходит либо автоматически по истечению определенного промежутка времени, либо принудительно при использовании функции `session_destroy()`.

Другие полезные функции для работы с сессиями:

`session_unregister(string)` — сессия "забывает" значение заданной глобальной переменной;

`session_set_cookie_params(int lifetime [, string path [, string domain]])` — с помощью этой функции можно установить, как долго будет "жить" сессия, задав `unix_timestamp` определяющий время "смерти" сессии. По умолчанию, сессия "живёт" до тех пор, пока клиент не закроет окно браузера.

Пример:

```
<?php
    session_set_cookie_params(3600);
    session_start();

    $_SESSION["a"] = 5;
?>

<?php
    session_start();

    echo $_SESSION["a"];
?>

<?php
    session_start();
    session_destroy();

    setcookie("PHPSESSID", "", time()-1);
?>
```

### 3.4. Динамические страницы с использованием JavaScript

#### *Встраивание JavaScript в HTML код*

По аналогии с CSS, программы на JavaScript можно встраивать непосредственно в теги при помощи обработчиков событий или в качестве значения атрибута *href* тега `<a>`. Однако чаще всего одного-двух операторов для решения задачи недостаточно, и включение программ в теги неудобно.

Для того чтобы отделить код программы на JavaScript (или любом другом языке скриптов) от остальной части страницы в спецификацию HTML был введен *контейнер* `<script>...</script>`. В него заключается текст программы. Все операторы этой программы будут выполняться сразу после загрузки документа в браузер вне зависимости от того, где расположен контейнер — в заголовке или в теле документа.

Тег `<script>` имеет атрибут *language*, который для большинства браузеров по умолчанию имеет значение “JavaScript”, однако явное указание этого атрибута является хорошим тоном. Текст скрипта может также находиться и в отдельном файле. Тогда URL файла указывается с помощью атрибута *src*: `<script src="modul.js"> </script>`.

#### *События*

Обработчик	Описание
onChange	Изменение значений элементов формы. Возникает после потерей элементом фокуса, т.е. после события <b>blur</b>
onClick	Одинарный щелчок (нажата и отпущена кнопка мыши)
onDbClick	Двойной щелчок
onKeyDown	Нажата клавиша на клавиатуре

Интернет-технологии

onKeyPress	Нажата и отпущена клавиша на клавиатуре
onKeyUp	Отпущена клавиша на клавиатуре
onMouseDown	Нажата кнопка мыши в пределах текущего элемента
onMouseMove	Перемещение курсора мыши в пределах текущего элемента
onMouseOut	Курсор мыши выведен за пределы текущего элемента
onMouseOver	Курсор мыши наведен на текущий элемент
onMouseUp	Отпущена кнопка мыши в пределах текущего элемента
onSubmit	Отправка данных формы (щелчок по кнопке)

**Использование JavaScript внутри тега. Пример с картинкой.**

```
onmouseover="this.src='file://localhost/C:/Users/VV/Desktop/Примеры/images/button_hover.png' "
onmouseout="this.src='file://localhost/C:/Users/VV/Desktop/Примеры/images/button.png' "
```

**Программирование на JavaScript**

*Операторы* в JavaScript в большинстве совпадают с операторами языка Си (+, -, \*, /, %, +=, -=, ==, !=, <, >, <= и т. д.). Кроме того, оператор сложения (+) применяется для конкатенации строк. Например: s = "string1"+"string2".

Управление потоком вычислений осуществляется набором следующих операторов:

— *условный оператор*

```
if (условие) { ветвь_1 } else { ветвь_2 };
```

— *оператор множественного выбора:*

```
switch (переменная) {
    case значение_1: оператор_1; break;
    case значение_2: оператор_2; break;
```

```
    [default: оператор_N;]
}
```

— *операторы цикла:*

1) **for** (i=0; i<9; i++) { тело цикла };

2) **for** (i in obj) {str = obj[i]}. Здесь переменная i используется для перебора всех свойств (элементов) объекта (контейнера) obj;

3) **while** (условие) { тело цикла }; — цикл с предусловием;

4) **do** { тело цикла } **while** (условие); — цикл с постусловием;

— *операторы управления выполнением цикла* **break** (досрочный выход из цикла) и **continue** (переход на следующую итерацию цикла, до завершения выполнения всех последующих операторов тела цикла);

— *оператор объявления переменной* **var**: **var a** = "str";. Тип переменной не указывается, а определяется по типу присвоенного ей значения. В данном случае переменная **a** строкового типа.

**Ввод**

```
<input type="text" id="идентификатор">
```

идентификатор.value — получить значение поля

Более правильно:

```
document.getElementById ("идентификатор") ;
```

```
var myName = prompt ("Как тебя зовут?", "");
```

**Вывод**

```
alert("Сообщение");
```

document.write("Тебя зовут ", myName); — Заменит текущее содержимое HTML-документа.

```
<p id="dlya_vivoda"></p>
document.getElementById("dlya_vivoda").innerHTML="Тебя зовут "+myName;
```

Использование тега <script> в разделе <head>. Пример с рисованием таблицы.

```
onclick
```

```
<p id="for_table"></p>
```

```
<script language="JavaScript">
  function PrintTable()
  {
    s="<table>";
    for(i=0; i<15; i++)
      {
        s+="<tr><td>"+i+"</td><td>"+i*i+"</td></tr>";
      }
    s+="</table>";

    for_table.innerHTML=s;
  }
</script>
```

**Массивы в JavaScript.** Массивы в JavaScript определяются при помощи специального литерала массива, представляющего собой заключенный в квадратные скобки список элементов, разделенных запятыми:

```
var array1=[1, 2, , 4].
```

В этом определении массива пропущен элемент с номером 2 (нумерация начинается с нуля), а потому, например, при обращении к этому элементу так:  $a=array1[2]$ ; переменная  $a$  будет пустой.

**Функции в JavaScript.** JavaScript позволяет создавать функции при помощи оператора **function**. Синтаксис описания функции следующий:

```
function <имя функции> (<список формальных аргументов>)
{ тело функции }
```

Функция может возвращать значение с помощью оператора **return**. Тип возвращаемого значения в объявлении функции не указывается.

Переменные, объявленные вне функции, являются *глобальными* и доступны из любого места программы, в том числе и из функций. Все переменные, определенные внутри функции являются *локальными* и не видны из основной программы.

## Интернет-технологии

В JavaScript существуют также следующие *встроенные функции*:

- **escape(строка)** — заменяет недопустимые в URL символы на их шестнадцатеричные коды;
- **eval(строка)** — вычисляет выражение, находящееся в строке, как если бы оно было написано в коде программы. **eval("2+2")** вернет **4**;
- **parseFloat(строка)** — преобразует строку в число с плавающей точкой. Если строка не может быть преобразована, то возвращает **NaN**.

## AJAX

AJAX (Asynchronous Javascript and XML — «асинхронный JavaScript и XML») — подход к построению интерактивных пользовательских веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате, при обновлении данных веб-страница не перезагружается полностью, а на неё добавляется небольшая часть данных.

Технология AJAX реализуется через объект XMLHttpRequest языка JavaScript.

## XMLHttpRequest

Объект XMLHttpRequest (или, сокращенно, XHR) дает возможность браузеру делать HTTP-запросы к серверу без перезагрузки страницы.

Несмотря на слово XML в названии, XMLHttpRequest может работать с данными в любом текстовом формате, и даже с бинарными данными.

Все популярные браузеры (IE7+, Firefox, Chrome, Safari и Opera) имеют встроенный объект XMLHttpRequest.

В них создание объекта выполняется следующим образом:

```
var r = new XMLHttpRequest();
```

Для старых браузеров, например, Internet Explorer (IE5 и IE6) используют ActiveX Object:

```
var r = new ActiveXObject("Microsoft.XMLHTTP");
```

Кроссбраузерная функция создания XMLHttpRequest:

```
function getXmlHttp(){
    var xmlhttp;
    try {
        xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (e) {
        try {
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
        } catch (E) {
            xmlhttp = false;
        }
    }
    if (!xmlhttp && typeof XMLHttpRequest!='undefined') {
        xmlhttp = new XMLHttpRequest();
    }
    return xmlhttp;
}
```

}

**Методы объекта XMLHttpRequest**

**open(метод, URL, синхронный/асинхронный, имя\_пользователя, пароль)** — запрос к серверу

*метод* — метод протокола HTTP. Как правило, используется GET либо POST, хотя доступны и более экзотические, вроде TRACE/DELETE/PUT и т.п.

*URL* — адрес запроса. Можно использовать не только HTTP/HTTPS, но и другие протоколы, например FTP и FILE://.

*синхронный/асинхронный* (логическое значение: true или false) — true для использования асинхронных запросов;

*имя\_пользователя, пароль* — данные HTTP-аутентификации.

*имя\_пользователя и пароль небезопасно хранить в скрипте, в особенности на клиентской части, так как данные аутентификации будут доступны сторонним пользователям.*

*Обязательны только первые 2 параметра.*

**send(содержимое)** — отправка HTTP запроса на сервер и получение ответа. В качестве аргумента указывается тело запроса. Поскольку для GET-запроса тела нет, надо писать send(null), для POST-запросов тело содержит параметры запроса, например список переменных.

**abort()** — отменяет текущий запрос. У браузера есть лимит — два одновременных подключения к домену/порту. Третье будет открыто только после завершения сеанса одно из предыдущих двух (по таймауту).

**setRequestHeader(название, значение)** — устанавливает заголовок запроса с указанным значением. Если заголовок с таким именем уже есть — он заменяется. Например,

r.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded')

**getAllResponseHeaders()** — возвращает строку со всеми HTTP-заголовками ответа сервера.

**getResponseHeader(headerName)** — возвращает значение заголовка ответа сервера с именем headerName.

**Свойства объекта XMLHttpRequest**

**readyState** — номер состояния запроса от 0 до 4.

0 — не инициализирован (перед началом работы объекта),

1 — загрузка (однажды, когда идет инициализация объекта),

2 — загружен (однажды, когда получен ответ от сервера),

3 — доступен (пока объект соединен с сервером),

4 — завершен (после того, как объект выполнил все задачи).

Используется только 4. Состояния 0-2 вообще не используются.

## Интернет-технологии

**onreadystatechange** — ссылается на функцию-обработчик состояний запроса, которая вызывается каждый раз при смене статуса объекта. (одно из основных свойств объекта XMLHttpRequest).

**status** — состояние ответа от сервера. Например, для HTTP-запросов статусный код ответа сервера: 200 — OK, 404 — Not Found, и так далее. Запросы по протоколам FTP, FILE:// не возвращают статуса, поэтому нормальным для них является status=0.

**statusText** — текстовое описание состояния ответа от сервера. Например, Not Found или OK

**responseText** — текст ответа сервера. Полный текст есть только при readyState=4, ряд браузеров дают доступ к полученной части ответа сервера при readyState=3.

**responseXML** — ответ сервера в формате XML. Это свойство хранит объект типа XML document, с которым можно обращаться так же, как с обычным HTML-документом. Например, использовать функцию getElementById.

```
var authorElem = xmlhttp.responseXML.getElementById('author')
```

Чтобы браузер распознал ответ сервера в формате XML, в ответе должен быть заголовок Content-Type: text/xml. Иначе свойство responseXML будет равно null.

```
header('Content-type: text/xml');
```

### Синхронные и асинхронные запросы

Различают два использования XMLHttpRequest. Первое — самое простое, синхронное.

При синхронном запросе браузер "подвисает" и ждет, пока сервер не ответит на запрос.

При асинхронном запросе, функция send() не останавливает выполнение скрипта, а просто отправляет запрос, который регулярно отчитывается о своём состоянии через вызов обработчика onreadystatechange.

Состояние под номером 4 означает конец выполнения, поэтому функция-обработчик при каждом вызове проверяет — не настало ли это состояние.

Пример 1. Вывод информации о пользователе на ту же страницу, где находится список пользователей.

Файл edit\_user.php (дописать в самое начало)

```
session_start();
```

Файл show\_users.php

изменить ссылку

## Интернет-технологии

```
<a href='index.php?action=edituser&id=$id'  
onclick='GetUser($id)' class='edit'>редактировать</a>  
  
    добавить в конец  
<script type="text/javascript">  
    var list = document.getElementsByClassName("edit");  
  
    for(i=0; i<list.length; i++)  
    {  
        list[i].href="#";  
    }  
  
    var request = new XMLHttpRequest();  
  
    function GetUser(id)  
    {  
        request.onreadystatechange = StateChange;  
  
        request.open("GET", "edit_user.php?id="+id, true);  
        request.send(null);  
    }  
  
    function StateChange()  
    {  
        if (request.readyState == 4)  
        {  
            if (request.status == 200)  
            {  
                // обработка ответа  
                document.getElementById("info").innerHTML =  
request.responseText;  
            }  
        }  
    }  
</script>
```



Интернет-технологии

Пример 2. Чат

Вариант 1 — полная загрузка страницы с сообщениями.

1) добавить ссылку на чат в файл user\_info.php.

```
echo "<br><a href='index.php?action=chat'>Чат</a>";
```

2) создать сам файл chat.php

```
<?php
    session_start();
    if(empty($_SESSION["user_id"]) )
    {
        echo "Зарегистрируйтесь чтобы войти в чат";
    }
    else
    {
        ?>
        <table width=100%>
            <tr>
                <td id="messages">
                    Сообщения
                </td>
            </tr>
            <tr>
                <td>
                    <textarea id="txt"></textarea>
                    <br>
                    <button
onclick="Send()">Отправить</button>
                </td>
            </tr>
        </table>

        <script>
            var request = new XMLHttpRequest();

            ShowMessages();

            function Send()
            {
                var req = new XMLHttpRequest();

                req.open("POST", "add_message.php", true);
                req.setRequestHeader('Content-Type',
'application/x-www-form-urlencoded');

                req.send("text="+encodeURIComponent(document.getElementById(
"txt").value));

                ShowMessages();
            }

            function ShowMessages()
            {
                request.onreadystatechange = StateChange;
```

## Интернет-технологии

```

true);
        request.open("GET", "get_messages.php",
        request.send(null);
        update_timer();
    }

    function StateChange()
    {
        if (request.readyState == 4)
        {
            if (request.status == 200)
            {
                document.getElementById("messages").innerHTML =
                request.responseText;
            }
        }
    }

    var timer;
    function update_timer() {
        if (timer) // если таймер уже был, сбрасываем
            clearTimeout(timer);
        timer = setTimeout(function () {
            ShowMessages();
        }, 1000);
    }
</script>
<?php
}
?>

```

3) создать файл `add_message.php`, принимающий сообщение и добавляющий его в БД:

```

session_start();

$id=$_SESSION["user_id"];
$text=$_POST["text"];

require_once("db.php");

$query="INSERT INTO messages (user, text) VALUES ($id,
'$text')";
mysql_query($query);
?>

```

4) создать файл `get_messages.php`, возвращающий все сообщения в формате HTML.

## Интернет-технологии

Вариант 2 — загрузка каждого сообщения отдельно

1) изменить формат выдаваемого сообщения на XML в файле

get\_messages.php:

```
<?php
    require_once ("db.php");

    $last_id=$_GET["last_id"];

    $query="SELECT      messages.id,      users.login      as      user,
messages.text      FROM      messages      INNER      JOIN      users      ON
users.id=messages.user WHERE messages.id>$last_id LIMIT 1";
    $res=mysql_query($query);

    header('Content-type: text/xml');
    echo "<?xml version=\"1.0\" encoding=\"UTF-8\"?>";
    echo "<document>";

    while($row = mysql_fetch_assoc($res))
    {
        $id=$row["id"];
        $user=$row["user"];
        $text=$row["text"];

        echo      "<element      id='id'>$id</element><element
id='user'>$user</element><element id='text'>$text</element>";
    }
    echo "</document>";
?>
```

2) в файле chat.php изменить функцию отображения сообщений:

```
<script>
    var request = new XMLHttpRequest();
    var last_id=0;
    var interval = 100;

    ShowMessages();

    function Send()
    {
        var req = new XMLHttpRequest();
        req.open("POST", "add_message.php", true);
        req.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
        req.send("text="+encodeURIComponent(document.getElementById("txt").value));
        ShowMessages();
    }

    function ShowMessages()
    {
        request.onreadystatechange = StateChange;
```

## Интернет-технологии

```
        request.open("GET",
"get_messages.php?last_id="+last_id, true);
        request.send(null);

        update_timer();
    }

    function StateChange()
    {
        if (request.readyState == 4)
        {
            if (request.status == 200)
            {
                if(
request.responseXML.getElementById("user") != null)
                {

                    document.getElementById("messages").innerHTML +=
"<b>"+request.responseXML.getElementById("user").innerHTML+"</b>
";

                    document.getElementById("messages").innerHTML += " :<br>";

                    document.getElementById("messages").innerHTML +=
request.responseXML.getElementById("text").innerHTML;

                    document.getElementById("messages").innerHTML += "<br><br>";

                    last_id=parseFloat(request.responseXML.getElementById("id").
innerHTML);
                }
                else
                {
                    interval = 1000;
                }
            }
        }
    }

    var timer;
    function update_timer() {
        if (timer)
            clearTimeout(timer);
        timer = setTimeout(function () {
            ShowMessages();
        }, interval);
    }
</script>
```

## Раздел 4. Технологии баз данных

### 4.1. СУБД MySQL

**База данных** – ядро любой операционной системы.

**База данных** – совместно используемый набор логически связанных данных и описание этих данных предназначен для удовлетворения информационных потребностей организации.

**База данных** – совокупность данных, отражающая состояние объектов и их отношений в конкретной предметной области.

**Система управления базами данных (СУБД)** – программное обеспечение (совокупность языковых и программных средств), предназначенных для создания, ведения и совместного использования БД многими пользователями и осуществляющий к ней контролируемый доступ.

**MySQL** (произносится «май-эс-кью-эль») – бесплатная свободно распространяемая СУБД. Данные в базе MySQL хранятся в форме таблиц.

При создании таблицы задаются ее столбцы, дальнейшие манипуляции (добавление, изменение, удаление) производятся со строками. Для управления базой данных используется язык SQL.

На одном сервере MySQL может быть расположено несколько баз данных.

Это позволяет выделить отдельную базу данных для каждого пользователя.

Чтобы не возникало путаницы можно использовать отдельные базы данных, например, для сайта и форума, хотя обычно они могут работать вместе на одной базе.

База данных содержит в себе таблицы.

Таблицы базы данных состоят, как и обычные таблицы, из строк и столбцов.

Столбцы имеют заранее определенное название и тип данных, а строки хранят непосредственно сами данные.

Каждая строка в таблице представляет собой один уникальный объект в реальном мире, а каждый атрибут в ней относится именно к этому объекту.

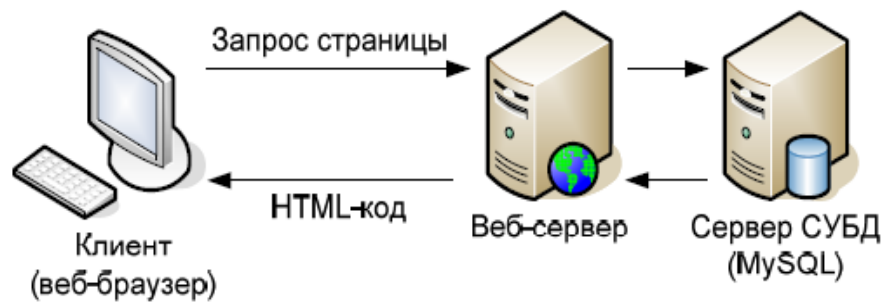
Так же как не могут существовать два идентичных самолета, две строки не могут представлять один и тот же самолет.

Для логической гарантии уникальности каждой из строк один из атрибутов назначается первичным ключом — именно по нему обращаются к строкам таблицы команды модификации данных. Если вы обеспечили, чтобы каждому объекту в реальности соответствовал всего один первичный ключ и наоборот, значит, вы сделали свою работу хорошо.

Язык PHP имеет встроенные функции для работы с MySQL и многими другими базами данных. Это позволяет хранить в базе данных пользователей,

Интернет-технологии

сообщения, содержание страниц, статистику сайта и т.п. При запросе страницы РНР обращается к базе данных, получает необходимые данные и на основе их формирует ответ пользователю (чаще всего это HTML страница).



Перед построением (созданием) базы данных необходимо проанализировать информационные потребности всех пользователей организации для которых создается база данных. При таком анализе выполняется структурирование информации о составляемой предметной области и строится так называемая **концептуальная модель данных**.

При этом выделяют **сущность, атрибуты и связи**.

**Сущность** – отдельный тип объекта предметной области, о котором нужно хранить информацию в базе данных.

**Атрибут** – свойство, которое описывает некоторую характеристику объекта.

**Связь** – объединение нескольких сущностей. Все связи реализуются с помощью внешних ключей.

**Внешний ключ** – столбец таблицы, ссылающийся на уникальный идентификатор другой таблицы.

**Уникальная идентификация** – первичный ключ, выбранный в качестве основного ключа (или ключа по умолчанию).

**Связи и отношения**

База данных включает, как правило, несколько таблиц, которые могут быть связаны друг с другом различными отношениями.

В отношениях "**один к одному**" каждая строка таблицы связана с единственной строкой в другой таблице.

В отношении "**один ко многим**" одной записи первой таблицы ставится в соответствие несколько записей второй таблицы, однако, каждая запись второй таблицы не может иметь более одной соответствующей записи в первой таблице.

В отношениях типа "**многие ко многим**" обе стороны могут быть связаны с множеством элементов противоположной стороны.

**Отношение "один к одному"**

В отношениях "один к одному" каждая строка таблицы связана с единственной строкой в другой таблице.

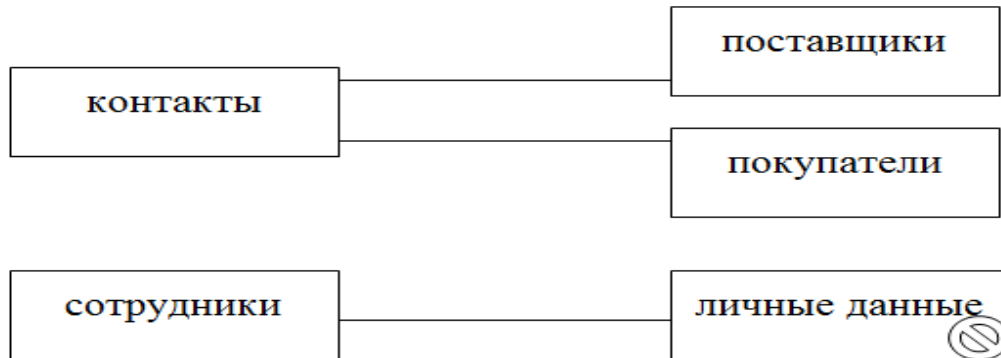
Отношения "один к одному" иногда используют для расширения одной

Интернет-технологии

сущности за счет создания дополнительной со своими атрибутами.

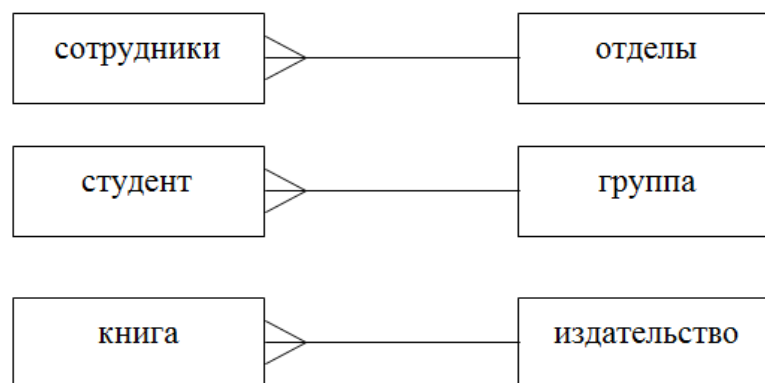
Например, сущность Employee (сотрудники) может хранить основную информацию о сотрудниках компании.

В то же время более конфиденциальная информация может храниться в отдельной сущности. В этом случае защита может применяться на основе отдельных атрибутов; представление может отображать только избранные атрибуты.



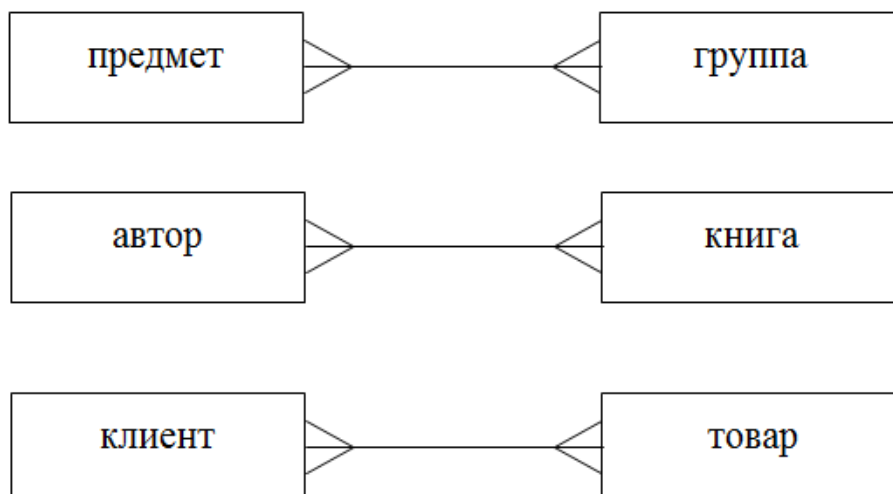
**Отношение “один ко многим”**

В таком отношении одной записи первой таблицы ставится в соответствие несколько записей второй таблицы, однако, каждая запись второй таблицы не может иметь более одной соответствующей записи в первой таблице.



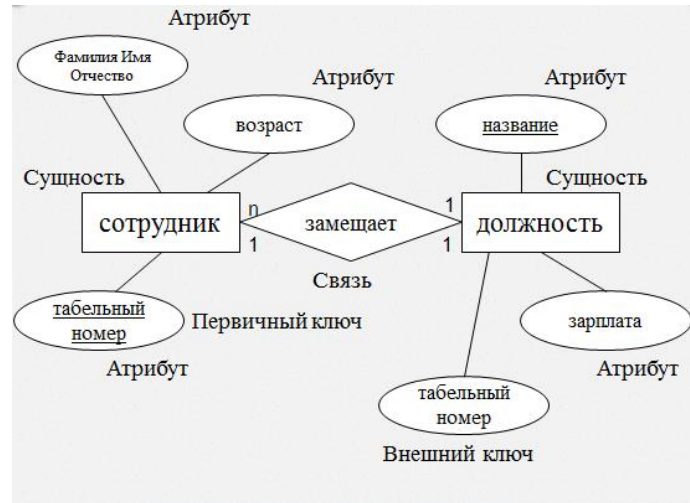
**Отношение “многие ко многим”**

В отношениях типа “многие ко многим” обе стороны могут быть связаны с множеством элементов противоположной стороны.



Интернет-технологии

Пример



При разработке базы данных обычно выделяется несколько уровней моделирования, при помощи которых происходит переход от предметной области к конкретной реализации базы данных средствами конкретной СУБД. Можно выделить следующие уровни:

- ✓ Сама предметная область;
- ✓ Модель предметной области;
- ✓ Логическая модель данных;
- ✓ Физическая модель данных;
- ✓ Собственно база данных и приложения.

**Предметная область** – это часть реального мира, данные о которой мы хотим отразить в базе данных.

Например, в качестве предметной области можно выбрать бухгалтерию какого-либо предприятия, отдел кадров, банк, магазин и т.д. Предметная область бесконечна и содержит как существенно важные понятия и данные, так и малозначащие или вообще не значащие данные. Так, если в качестве предметной области выбрать учет товаров на складе, то понятия "накладная" и "счет-фактура" являются существенно важными понятиями, а то, что сотрудница, принимающая накладные, имеет двоих детей – это для учета товаров неважно. Однако с точки зрения отдела кадров данные о наличии детей являются существенно важными. Таким образом, важность данных зависит от выбора предметной области.

**Модель предметной области** – это наши знания о предметной области. Знания могут быть как в виде неформальных знаний в мозгу эксперта, так и выражены формально при помощи каких-либо средств. В качестве таких средств могут выступать текстовые описания предметной области, наборы должностных инструкций, правила ведения дел в компании и т.п. Опыт показывает, что текстовый способ представления модели предметной области крайне неэффективен. Гораздо более информативными и полезными при разработке баз данных являются описания предметной области, выполненные в виде схем при помощи специализированных графических



нотаций.

**Логическая модель данных** описывает понятия предметной области, их взаимосвязь, а также ограничения на данные, налагаемые предметной областью.

Примеры понятий – "сотрудник", "отдел", "проект", "зарплата". Примеры взаимосвязей между понятиями – "сотрудник числится ровно в одном отделе", "сотрудник может выполнять несколько проектов", "над одним проектом может работать несколько сотрудников". Примеры ограничений – "возраст сотрудника не менее 16 и не более 60 лет".

**Логическая модель данных** является начальным прототипом будущей базы данных. Логическая модель строится в терминах информационных единиц, но без привязки к конкретной СУБД. Более того, логическая модель данных необязательно должна быть выражена средствами именно реляционной модели данных.

**Физическая модель данных** описывает данные средствами конкретной СУБД. Отношения, разработанные на стадии формирования логической модели данных, преобразуются в таблицы, атрибуты становятся столбцами таблиц, для ключевых атрибутов создаются уникальные индексы, домены преобразуются в типы данных, принятые в конкретной СУБД.

Ограничения, имеющиеся в логической модели данных, реализуются различными средствами СУБД, например, при помощи индексов, декларативных ограничений целостности, триггеров, хранимых процедур.

### **Реализация связей**

Связь – понятие из концептуального уровня.

Все связи реализуются с помощью внешних ключей. Внешний ключ – столбец таблицы, ссылающийся на первичный ключ другой таблицы.

Ссылочная целостность (англ. referential integrity) – необходимое качество реляционной базы данных, заключающееся в отсутствии в любом её отношении внешних ключей, ссылающихся на несуществующие кортежи.

Связь многие-ко-многим реализуется с помощью дополнительной таблицы.

Исходной точкой любого проектирования базы данных является представление предметной области в виде одного или нескольких отношений, и на каждом шаге проектирования производится некоторый набор схем отношений, обладающих «улучшенными» свойствами. Таким образом, процесс проектирования представляет собой процесс нормализации схем отношений, причем каждая следующая нормальная форма обладает свойствами, в некотором смысле лучшими, чем предыдущая.

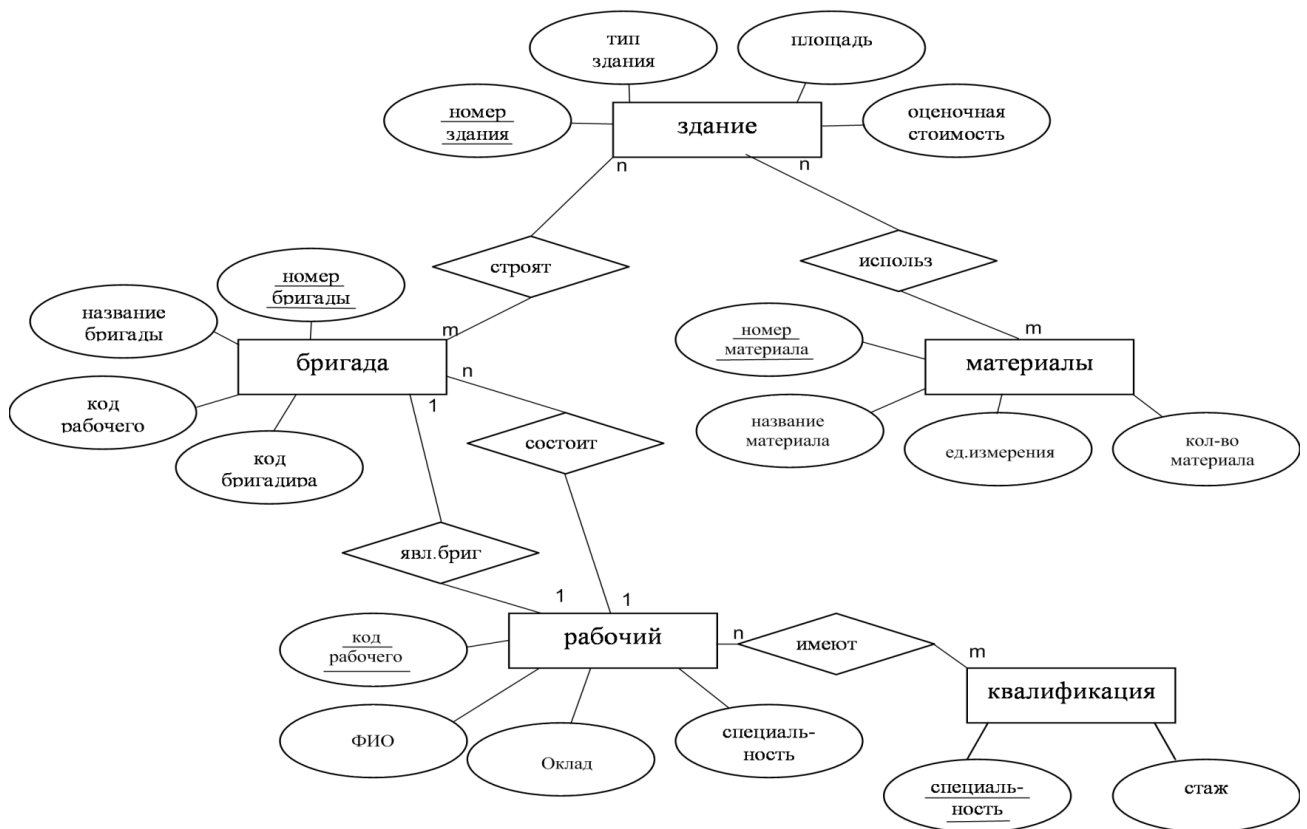
Каждой нормальной форме соответствует определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений.

Основные свойства нормальных форм состоят в следующем:

- 1) каждая следующая нормальная форма в некотором смысле лучше предыдущей нормальной формы;
- 2) при переходе к следующей нормальной форме свойства предыдущих нормальных форм сохраняются.

Построить модель «сущность-связь» для следующей предметной области:

Строительная компания «Премьер» возводит различные здания. Для всех зданий требуются разнообразные материалы в различных количествах. На разных этапах проекта работают разные бригады. Например, есть бригады арматурщиков, каменщиков, штукатуров и т.д. Составляя график работ, фирма «Премьер» варьирует состав бригад. Рабочие назначаются в разные бригады в соответствии с квалификацией. Так, Петров может выполнять работу как плотника, так и каменщика, поэтому его иногда включают в бригаду арматурщиков, иногда каменщиков. Численность бригад меняется в зависимости от размера здания и предъявляемых к нему требований. Для каждой бригады выбираются дни работы. Например, бригаде штукатуров требуется несколько дней для того, чтобы оштукатурить здание. Для каждой бригады, работающей на строительстве данного здания, назначается бригадир. Рабочий может возглавлять одну бригаду и работать в другой простым рабочим. Владелец компании «Премьер» хочет знать, кто из его рабочих в каждую бригаду назначен на разных зданиях, какие материалы используются при возведении разных зданий, а также график работ по каждому зданию.



### Язык запросов SQL

SQL (Structured Query Language) – структурированный язык запросов — универсальный компьютерный язык, применяемый для создания,

## Интернет-технологии

модификации и управления данными в реляционных базах данных.

В язык SQL в качестве составных частей входят:

- язык манипулирования данными (Data Manipulation Language, DML);
- язык определения данных (Data Definition Language, DDL);
- язык управления данными (Data Control Language, DCL).

Это не отдельные языки, а различные команды одного языка. Такое деление проведено только лишь с точки зрения различного функционального назначения этих команд.

Язык манипулирования данными используется, как это следует из его названия, для манипулирования данными в таблицах баз данных.

Он состоит из 4 основных команд:

SELECT (выбрать)  
INSERT (вставить)  
UPDATE (обновить)  
DELETE (удалить)

Язык управления данными используется для управления правами доступа к данным и выполнением процедур в многопользовательской среде. Более точно его можно назвать "язык управления доступом". Он состоит из двух основных команд:

GRANT (дать права)  
REVOKE (забрать права)

Язык определения данных используется для создания и изменения структуры базы данных и ее составных частей — таблиц, индексов, представлений (виртуальных таблиц), а также триггеров и сохраненных процедур. Основными его командами являются:

CREATE DATABASE (создать базу данных)  
CREATE TABLE (создать таблицу)  
CREATE VIEW (создать виртуальную таблицу)  
CREATE INDEX (создать индекс)  
CREATE TRIGGER (создать триггер)  
CREATE PROCEDURE (создать сохраненную процедуру)  
ALTER DATABASE (модифицировать базу данных)  
ALTER TABLE (модифицировать таблицу)  
ALTER VIEW (модифицировать виртуальную таблицу)  
ALTER INDEX (модифицировать индекс)  
ALTER TRIGGER (модифицировать триггер)  
ALTER PROCEDURE (модифицировать сохраненную процедуру)  
DROP DATABASE (удалить базу данных)  
DROP TABLE (удалить таблицу)  
DROP VIEW (удалить виртуальную таблицу)  
DROP INDEX (удалить индекс)

DROP TRIGGER (удалить триггер)

DROP PROCEDURE (удалить сохраненную процедуру)

### Синтаксис команды SELECT

#### Выбора строки

**SELECT** [**ALL** | **DISTINCT**] {<список столбцов результата>| \*}

**FROM** <список таблиц>

[**WHERE** <условное выражение>] условие выбора строк результата

[**GROUP BY** <список столбцов группировки>] задает группировку по указанным столбцам

[**HAVING** <условное выражение для группы>] указать условие выбора групп

[**ORDER BY** <список столбцов по которым упорядочить вывод>] задает столбцы по которым сортируются строки результата

**SELECT** [**ALL** | **DISTINCT**] {<список столбцов результата>| \*}

Ключевое слово **ALL** означает, что результат набора слов включает все строки удовлетворяющие условию запроса, в том числе одинаковые строки.

Ключевое слово **DISTINCT** означает, что результирующий набор включены только различные строки.

По умолчанию действует **ALL**

Символ \* означает, что в результат набора включаются все столбцы из исходных таблиц запроса.

#### **WHERE** <условное выражение>

В условном выражении используются:

- ✓ простые сравнения {=, <, >, <=, >=};
  - ✓ предикат Between A and B (предикат истин, если сравниваемые значения находятся в диапазоне от A до B ) и противоположный предикат Not Between A and B ;
  - ✓ предикаты вложения в множество IN (множество) и Not IN (множество) (множество значений может быть задано простым перечислением или подзапросом. IN истинно, если значения входит в указанное множество);
  - ✓ предикаты сравнения с образцом LIKE и Not LIKE (требуется создания шаблона или образца, с которым сравнивается значение. В шаблон могут включаться следующие символы:
    - ? – любой одиночный символ;
    - \* – произвольное (включая нулевое) число символов;
    - # – любая цифра.);
  - ✓ предикат сравнения с неопределенным значением;
- <имя столбца> Is NULL      <имя столбца> Is Not NULL
- ✓ Предикат существования EXISTS и не существования Not EXISTS используется с подзапросом.

### Синтаксис команды INSERT

Интернет-технологии

**Позволяет в указанную таблицу добавить одну или множество строк**

*Для добавления группы строк*

**INSERT INTO** таблица [(поле1[, поле2, [ ...]])]

**SELECT** [источник] поле1[, поле2, [ ...]]

**FROM** выражение;

*Для добавления одной строки*

**INSERT INTO** таблица [(поле1[, поле2, [ ...]])]

**VALUES** (значение1 [, значение2, [ ...]])

**Синтаксис команды UPDATE**

**Позволяет изменить значение столбцов указанной таблицы в строках удовлетворяющих заданному условию**

**UPDATE** таблица

**SET** поле1 = новое значение1 [, поле2= новое значение2 [,...]]

**WHERE** условие отбора

**Синтаксис команды DELETE**

**Позволяет удалять из одной или нескольких таблиц**

**DELETE** [таблица \*]

**FROM** таблица

**WHERE** условие отбора

Пример

select \*

from student — показывает всех студентов

select familiya, imya, otchestvo

from student — показывает только фамилию, имя и отчество студентов

select \*

from student

inner join группа on student.gruppa=gruppa.id — показывает всех студентов с указанием группы

select \*

from (student inner join группа on student.gruppa=gruppa.id)

inner join facultet on группа.facultet=facultet.id — показывает всех студентов с указанием группы и факультета, так же всех ключей.

Интернет-технологии

БАЗА ДАННЫХ (NEW\_BASE)

Таблица (TABLICA)		
id	infa	infa2
1	Слово 1	Слово 11
2	Слово 2	Слово 12
3	Слово 3	Слово 13
4	Слово 4	Слово 14
5	Слово 5	Слово 15
6	Слово 6	Слово 16
СТРОКА		
8	Слово 8	Слово 18
9	Слово 9	Слово 19
10	Слово 10	Слово 20
11	Слово 11	Слово 21
12	Слово 12	Слово 22
13	Слово 13	Слово 23
14	Слово 14	Слово 24

Еще одна таблица		
	Ячейка	
ПОЛЕ1	ПОЛЕ2	ПОЛЕ3

В любой таблице обязательно должно присутствовать служебное поле называемое *полем первичного ключа*. Это поле позволяет нам нумеровать строки в таблице и потом обращаться к определенной строке по ее значению в поле первичного ключа (в нашем случае роль *поля первичного ключа* играет поле *id*)

При создании таблицы для столбца задается имя, тип хранимых данных и дополнительные атрибуты.

*Основные типы данных в MySQL:*

Целые числа – для хранения чисел без дробной части.

При указании

атрибута **UNSIGNED**

диапазон содержит только

положительные значения

(указаны в скобках).

Интернет-технологии

Тип	Диапазон значений
TINYINT	от -128 до 127 (от 0 до 255)
SMALLINT	от -32768 до 32767 (от 0 до 65535)
MEDIUMINT	от -8388608 до 8388607 (от 0 до 16777215)
INT	от -2147483648 до 2147483647 (от 0 до 4294967295)
BIGINT	от -9223372036854775808 до 9223372036854775807 (от 0 до 18446744073709551615)

Числа с плавающей точкой – для хранения чисел с дробной частью. При указании атрибута **UNSIGNED** можно использовать только неотрицательные значения.

Тип	Диапазон значений
FLOAT	от $3,402823466 \times 10^{38}$ до $-1,175494351 \times 10^{-38}$ , 0, и от $1,175494351 \times 10^{-38}$ до $3,402823466 \times 10^{38}$
DOUBLE	от $-1,7976931348623157 \times 10^{308}$ до $-2,2250738585072014 \times 10^{-308}$ , 0, и от $2,2250738585072014 \times 10^{-308}$ до $1,7976931348623157 \times 10^{308}$

Дата и время

Тип	Значение
DATE	Дата в формате ГГГГ-ММ-ДД
TIME	Время в формате ЧЧ-ММ-СС
DATETIME	Дата и время в формате ГГГГ-ММ-ДД ЧЧ-ММ-СС

В кодировке UTF-8, из расчета использования 3-х байт для хранения одного символа. При разработке русскоязычного сайта для хранения кириллицы

## Интернет-технологии

достаточно 2-х байт на символ.

Тип	Максимальная длина строки
<b>VARCHAR</b>	Задается пользователем , до 21844 символов
<b>TINYTEXT</b>	85
<b>TEXT</b>	21844
<b>MEDIUMTEXT</b>	5592405
<b>LONGTEXT</b>	1431655765

При выборе типа данных столбца нужно стремиться к экономному использованию памяти. Например, для хранения возраста человека в годах не нужно использовать тип INT, - достаточно TINYINT. В большинстве случаев для хранения чисел с дробной частью достаточно типа FLOAT, а DOUBLE использует в 2 раза больше памяти. Имя пользователя уместится в столбец типа VARCHAR(32), номер телефона - в VARCHAR(20), новость на сайте – в TEXT, роман А.Н. Толстого «Война и мир» - в MEDIUMTEXT.

## Функции PHP для работы с СУБД MySQL

### Подключение к БД

По аналогии с HeidiSQL и другими программами перед использованием БД необходимо выполнить 2 предварительных этапа:

#### 1. Подключение к серверу базы данных

`resource mysql_connect ([string server [,string username [,string password [,bool new_link [,int client_flags]]]])` — открывает соединение с сервером MySQL. В случае успеха возвращает идентификатор соединения, иначе — False.

Если второй вызов функции произошёл с теми же аргументами `mysql_connect()`, новое соединение не будет установлено. Вместо этого функция вернёт ссылку на уже установленное соединение. Параметр `new_link` может заставить функцию `mysql_connect()` открыть ещё одно соединение, даже если соединение с аналогичными параметрами уже открыто. Параметр `client_flags` должен быть комбинацией из следующих констант: `MYSQL_CLIENT_COMPRESS`, `MYSQL_CLIENT_IGNORE_SPACE`, `MYSQL_CLIENT_INTERACTIVE`.

Соединение с сервером будет закрыто при завершении исполнения скрипта, если до этого оно не будет закрыто с помощью функции `mysql_close()`.

#### 2. Выбор базы данных из всех доступных на сервере

`bool mysql_select_db (string database_name [,resource link_identifier])` выбирает для работы указанную базу данных на сервере, на который



## Интернет-технологии

ссылается переданный указатель. Если параметр указателя опущен, используется последнее открытое соединение. Если нет ни одного открытого соединения, функция попытается соединиться с сервером аналогично функции `mysql_connect()`, вызванной без параметров.

Возвращает `TRUE` в случае успешного завершения или `FALSE` в случае возникновения ошибки.

**Функции обработки ошибок**

Если произойдет ошибка соединения с MySQL, то вы получите соответствующее сообщение и скрипт завершит свою работу. Это не всегда бывает удобно, прежде всего, при отладке скриптов. Поэтому, в PHP есть следующие две функции:

```
mysql_errno(int $id);  
mysql_error(int $id);
```

Первая функция возвращает номер ошибки, а вторая - сообщение об ошибке. В результате мы можем использовать следующее:

```
echo "ERROR ".mysql_errno()." ".mysql_error()."\n";
```

Теперь вы будете знать, из-за чего произошла ошибка - вы увидите соответствующим образом оформленное сообщение.

**Извлечение данных**

`resource mysql_query (string query [,resource link_identifier])` — отправляет запрос к базе данных, которая определяется идентификатором `link_identifier`. Если он не указан, запрос отправляется через последнее установленное соединение. Если установленных соединений нет, функция пытается установить соединение так же, как и функция `mysql_connect()` без параметров. Функция возвращает ресурс только для операторов `SELECT`, `SHOW`, `EXPLAIN`, `DESCRIBE`. Для остальных запросов — `True`; в случае неудачи для всех видов запросов — `False`.

`mysql_result (resource result, int row [, mixed field])` — возвращает значение одной ячейки результата запроса

`array mysql_fetch_array(resource result [, int result_type])` — возвращает массив из значений ячеек строки результирующей таблицы. Таблица является результатом SQL-запроса и хранится в ресурсе `result`. Тип возвращаемого массива определяется указанием одной из констант `MYSQL_ASSOC` (ассоциативный), `MYSQL_NUM` (индексированный), `MYSQL_BOTH` (ассоциативный и индексированный одновременно, по умолчанию). Имена полей таблицы чувствительны к регистру.

`array mysql_fetch_assoc (resource result)` — возвращает ассоциативный массив с названиями индексов, соответствующими названиям столбцов или `FALSE` если строк больше нет.

array mysql\_fetch\_row (resource result) — возвращает массив, содержащий данные строки.

object mysql\_fetch\_object (resource result) — возвращает объект с полями, соответствующими столбцам в строке таблицы.

#### Пример

```
$conn = mysql_connect("127.0.0.1", "root", "");  
mysql_select_db("mydb", $conn);
```

```
$query="SELECT * FROM users";  
$results=mysql_query($query,$conn);
```

```
while($row=mysql_fetch_array($results))  
{  
    echo $row[0]."|".$row[1]."<br>";  
}
```

Убрать \$conn, вынести подключение в отдельный файл.

#### *Дополнительные функции*

int mysql\_affected\_rows [resource link\_identifier]) — возвращает число измененных записей после выполнения запросов DELETE, INSERT, UPDATE или REPLACE.

int mysql\_num\_rows (resource result) — возвращает количество строк результата запроса. Эта команда работает только с запросами SELECT.

string mysql\_error() — возвращает текст сообщения об ошибке, произошедшей после последней операции MySQL.

## Раздел 5. Безопасность в сети интернет.

### 5.1. Методы аутентификации пользователей

#### SQL-инъекция (внедрение SQL-кода)

Смена пароля:

```
$newpassword=$_POST["password1"];
```

```
$query="UPDATE users SET password='$newpassword' WHERE id=$id";
```

Пример запроса:

```
UPDATE users SET password='123' WHERE id=10
```

Запрос меняющий тип пользователя на администратора:

```
UPDATE users SET password='123', type='0' WHERE id=10
```

Если вместо пароля (\$newpassword) написать часть этого запроса  
12345', type='0

то получим запрос, изменяющий тип пользователя.

Если вместо пароля (\$newpassword) написать  
123'-- [пробел]

То оставшаяся часть запроса с условием WHERE id=\$id проигнорируется и  
пароль будет изменён для всех пользователей в таблице

---

В БД имеется значение, которое нигде на сайте не отображается — это  
столбец type из таблицы users. В файле edit\_user.php есть запрос извлекающий  
из БД информацию о пользователе и программный код выводящий часть её  
на экран:

```
$id=$_GET["id"];
```

```
$query="SELECT * FROM users WHERE id=$id";
```

```
$results=mysql_query($query);
```

```
$username=mysql_result($results, 0, "username");
```

```
$password=mysql_result($results, 0, "password");
```

```
echo "Информация о пользователе<br>";
```

```
echo "id: $id<br>";
```

```
echo "имя пользователя: $username<br>";
```

```
echo "текущий пароль: $password<br>";
```

Чтобы выбрать информацию о другом пользователе можно написать  
следующий запрос:

```
SELECT * FROM users WHERE id=0 UNION SELECT id, username, type as  
password, username FROM users WHERE id=1
```

То есть переменной \$id нужно передать часть этого запроса:

```
0 UNION SELECT id, username, type as password, username FROM users  
WHERE id=1
```

```
0 UNION SELECT id, type as login, id, id, id, id FROM users WHERE id=1
```

Удаление нескольких пользователей:

```
1 OR id > 1
```

Удаление таблицы:

```
5; DROP TABLE users
```

---

Для защиты от внедрения SQL-кода необходимо фильтровать входные параметры, значения которых будут использованы для построения SQL-запроса.

#### 1. Проверка типов значений

Для проверки переменной на числовое значение используется функция `is_numeric(n)`, которая вернёт `true`, если параметр `n` — число, и `false` в противном случае.

Так же можно не проверять значение на число, а вручную переопределить тип. Вот пример, переопределяющий значение `$id`, полученное от `$_GET['id']` в значение целочисленного типа (в целое число):  
`$id=(int)$_GET['id'];`

#### 2. Экранирование ввода

Большинство взломов через SQL происходят по причине нахождения в строках «необезвреженных» кавычек, апострофов и других специальных символов. Для такого обезвреживания нужно использовать функцию `addslashes($str)`, которая возвращает строку `$str` с добавленным обратным слешем перед каждым специальным символом. Данный процесс называется экранизацией.

```
$a="пример текста с апострофом ' ";  
echo addslashes($a); //будет выведено: пример текста с апострофом '
```

Кроме этого существуют две функции, созданные именно для экранизации строк, используемых в SQL выражениях.

```
mysql_escape_string($str);  
mysql_real_escape_string($str);
```

Первая не учитывает кодировку соединения с БД и может быть обойдена, а вот вторая её учитывает и абсолютно безопасна.

## Интернет-технологии

`mysql_real_escape_string($str)`; возвращает строку `$str` с добавленным обратным слешем к следующим символам: `x00`, `n`, `r`, `,`, `'`, `"` и `x1a`.

## Магические кавычки

Магические кавычки — эффект автоматической замены кавычки на обратный слэш (`\`) и кавычку при операциях ввода/вывода. В некоторых конфигурациях PHP этот параметр включён, а в некоторых нет. Для того, чтобы избежать двойного экранирования символов и заэкранировать данные по-нормальному через `mysql_real_escape_string($str)`, необходимо убрать автоматические поставленные обратные слешы (если магические кавычки включены).

Проверка включённости магических кавычек для данных получаемых из GET, POST или Куков организуется через функцию `get_magic_quotes_gpc()`; (возвращает 1 – если магические кавычки включены, 0 – если отключены).

Если магические кавычки включены (т.е. обратные слешы добавляются) и такое встречается чаще, то их нужно убрать. Это делается через функцию `stripslashes($str)`; (возвращает строку `$str` без обратных слешей у кавычек и прямых апострофов).

Код с полной экранизацией строк для записи в БД:

```
if(get_magic_quotes_gpc()==1)
{
$element_title=stripslashes(trim($_POST["element_title"]));
$element_text=stripslashes(trim($_POST["element_text"]));
$element_date=stripslashes(trim($_POST["element_date"]));
}
else
{
$element_title=trim($_POST["element_title"]);
$element_text=trim($_POST["element_text"]);
$element_date=trim($_POST["element_date"]);
}

$element_title=mysql_real_escape_string($element_title);
$element_text=mysql_real_escape_string($element_text);
$element_date=mysql_real_escape_string($element_date);
```

***Хранение паролей в БД***

1 способ. Хранение паролей в исходном виде. Например, если ввели пароль "qwerty", то он так и сохранится. Минус в том, что при взломе БД злоумышленник получает все пароли в явном виде.

## Интернет-технологии

2 способ. Хранение хеш-функций паролей md5, sha1 и т.д.

Допустим у пользователя пароль «123456». Придумаем свою хеш-функцию. Сложим все цифры получим число «21» — это и будет являться результатом нашей хеш функции, хеш-суммой или хешем. Конечно наш алгоритм отвратителен (да и хеш суммой его назвать нельзя, это скорее контрольная сумма), так как один и тот же хеш может соответствовать большому количеству различных паролей. То есть пароли «654321», «555222», «100299» и т.д. будут давать такой же хеш, или по-научному будут являться коллизией.

Хеш функция должна обладать следующими параметрами:

- Необратимость — хеш сумма не должна «расшифровываться» подобно обычным алгоритмам шифрования.
- Отсутствие коллизий — для каждого данных проходящих через хеш-функцию должен получиться уникальный хеш.

Если первый параметр практически достигнут в современных алгоритмах хеш-функций. То второй параметр не достигим для хешей с фиксированной длиной (а таких алгоритмов сейчас большинство) даже чисто теоретически (я надеюсь вы понимаете почему).

И теперь при регистрации пользователя, указанный им пароль проходит через хеш-функцию и вместо пароля в БД будет занесен полученный хеш. При каждой попытке авторизации указанный пароль будет каждый раз проходить через хеш-функцию и полученный хеш будет сравниваться с хешем хранимым в БД, и если хеши будут соответствовать значит пароль был указан верный.

Существуют сервисы, которые хранят очень большие базы хешей различных слов и позволяют осуществлять быстрый поиск исходной информации по её хешу.

3 способ. Хранение паролей в зашифрованном виде, с добавлением нескольких случайных символов, уникальных для каждого пользователя (так называемая соль).

```
$salt = "";
$length = rand(5,10); // длина соли (от 5 до 10 символов)
for($i=0; $i<$length; $i++) {
    $salt .= chr(rand(33,126)); // символ из ASCII-table
}
```

Различные варианты солениия в известных движках:

md5(\$pass.\$salt) — применяется в Joomla

md5(md5(\$pass).\$salt) — применяется в vBulletin

md5(md5(\$salt).md5(\$pass)) — применяется в новых IP.Board