**Как работает интернет**

**Домен** – имя сайта.

**Например**: facebook.com, ua-football.com

**Сервер** – компьютер, на котором расположен сайт

Домен связан со своим сайтом с помощью **IP адреса**. По факту **домен** - это есть **IP** **адрес** компьютера, на котором располагается сайт, так как у каждого компьютера есть свой IP-адрес.

Тоесть, когда вы вводим в браузере название сайта facebook.com, это аналогично тому, если бы мы вводили номер IP-компьютера, на котором находиться этот сайт 156.143.189.89

**Алгоритм** – определенная последовательность действий

**Синтаксис** – набор правил, с помощью которых формируется язык программирования команды к нему, чтобы он нас понял

**Web-сервер** –программа, которая позволяет принимать команды из вне от других компьютеров, чтобы передать им свой сайт. Но перед этим **наш Сервер** передает команду языку **ПХП**, чтобы он ее обработал и передал ее **Web-серверу**, а тот уже ее передает **Браузеру** через который пользователь видит содержимое нашего сайта.

**СУБД (Система Управления Базами Данных)** – программа, которая запускается на том же компьютере, что и Web-сервер, необходимая для хранения большого количества информации в доступном виде, а также необходима для быстрого доступа к ней пользователями. Например, чтобы хранить статьи, категории, комментарии на сайте facebook.com , а также иметь быстрый доступ к ним используется программа СУБД – **MySQL**

**SQL (Structure Query Language) –** структурированный язык запросов, который держится на четырёх видах базовых команд:

1. Create – создание записи в Базе данных (БД). Используется при создании аккаунта пользователя с вводом имени, логина, пароля и тд
2. Read – чтение записи из БД. Используется для проверки зарегистрирован ли данный пользователь на сайте или нет
3. Update – обновление записей БД. Используется при обновлении аватарки, логина пароля или других данных
4. Delete – удаление записей в БД

**Скрипт** – это алгоритм команд в коде языка PHP

**Локальный доступ** – означает, что сайт доступен только мне.

**Итерация** – повтор (прохождение) цикла

**Основы PHP**

**<?php** данный тег указывает интерпретатору, что код написан на языке **PHP?>**

**echo “Автобус Богдан”;** PHP воспринимает содержимое в **скобках**, как **текст** и дает команду интерпретатору **вывести его на экран браузера**

Разница между кавычками **‘ ’** и **“ ”** заключается в том, что переменная, которая вписывается в **апостроф ‘ ’** обрабатывается **как обычный текст**, а переменную, которая вписывается в **скобки “ ”** PHP понимает, что она существует и что нужно **вывести ее содержимое**.

**Например:**

**$a = ‘Автобус Богдан’**

**echo ‘$a’;**

**echo “$a”;**

**Выведет:**

**$a  
Автобус Богдан**

А, если написано вот так **echo “ ‘$a’ ”;**

**то выведет:**

**‘Автобус Богдан’**

**Еще пример:**

**$a = 50;**

**echo 'Городу Сумы необходимо закупить $a автобусов для замены маршруток <br>';**

**echo "Городу Сумы необходимо закупить $a автобусов для замены маршруток <br>";**

**Выведет:**

Городу Сумы необходимо закупить **$a** автобусов для замены маршруток   
Городу Сумы необходимо закупить **50** автобусов для замены маршруток

**;** означает **завершение команды**

**$bus=1;** PHP воспринимает содержимое после знака **$** как **цифру 1**, которая прикреплена к **переменной bus**

**echo $bus;** PHP **выводит** на экран браузера **цифру 1**

**echo date (‘d.m.Y h:i’); PHP** воспринимает содержимое как **команду вывода текущей даты и времени на экран браузера**

**echo “Автобус Богдан <br>”;** тег **<br> переносит текст на новую строку** после себя. Также его можно ставить перед значением и тем самым тег сделает **отступ для строки перед собой**.

**Конкатенация строк:**

**$Bogdan = 'В городе Сумы появились';**

**$amount = 4;**

**echo $Bogdan . ' '. $amount . ' '. "больших автобуса <br>"; точка** осуществляет **конкатенацию (соединение)** строк и символов

Выведет: В городе Сумы появились 4 больших автобуса

**//** данный символ означает комментирование, и все, что находиться в нем PHP не видит

**/\*\*+** **Enter** комментирование на несколько рядов вниз

**==** сравнение   
**=** присвоение

**/n** перенос на новую строку

**@** подавляет все ошибки, которые могут появиться при чтении значения переменной. В случае, если он предшествует какому-либо выражению в PHP-коде, любые сообщения об ошибках, генерируемые этим выражением, будут проигнорированы.

На сегодняшний день оператор **"@"** подавляет вывод сообщений даже о критических ошибках, прерывающих работу скрипта. Помимо всего прочего, это означает, что если вы использовали "@" для подавления ошибок, возникающих при работе какой-либо функции, в случае если она недоступна или написана неправильно, дальнейшая работа скрипта будет остановлена без каких-либо уведомлений.

**Типы данных**

1. **$a=10;**

**integer (int)** – переменная, которая имеет тип данных: **целое число**

1. **$b=10.12;**

**float** – переменная, которая имеет тип данных: **дробное число**

1. **$c=’Bus Bogdan’;**

**string, str** – переменная, которая имеет тип данных: **строка.**

1. **$d=array(1,2,3,4,5);**

**массив (array)**–переменная, которая хранит в себе **несколько значений.**

1. **$k = true;**

**boolean (bool)** – переменная **булевого** типа, состоящая из двух значений **истинна (true)** или **ложь (false).** PHP читает **константы true** и **false,** как **1** или **0(пустота)**, а мы должны понимать это как **да** или **нет.**

Например,

**$k = true;**

**echo $k;** на экран выведет значение **1**

**$l = false;**

**echo $l;** на экран не выведет **ничего**

**Математические операции с переменными**

**+ оператор сложения**

Например: **echo 5 + 10 , '<br>';**

Выведет: 15

Пример с переменными:

**$a = 10;**

**$b = 5;**

**echo ‘Ответ: ’ . ($a + $b) , ‘<br>’;**

Выведет: Ответ: 15

**- оператор вычитания**

Например: **echo 15 - 10** **, '<br>';**

Выведет: 5

**\* оператор умножения**

Например: **echo 5 \* 10** **, '<br>';**

Выведет: 50

**/ оператор деления**

Например: **echo 20 / 10** **, '<br>';**

Выведет: 2

**% оператор деления по модулю**

Проверяет, является ли число четным или нет. Если при делении на **2** выведет **0**, значит, что число является **четным**, если **1** – **не четное**.

Например: **echo 2010 % 2** **, '<br>';**

Выведет: **0**

**echo 2013 % 2** **, '<br>';**

Выведет: **1**

**Массив**

Нужен для того, чтобы хранить в себе большое количество значений. У каждого значения **в массиве** есть свой **индекс, который начинается с ноля.** Чтобы вывести содержимое массива, нужно указать какое именно значение мы хотим вывести на экран. Например, чтобы вывести на экран **первое число** из нашего массива **$d=array(1,2,3,4,5);** нужно ввести **индекс в квадратные скобки.**

**echo $d[0]; где $d[0]** – **первый элемент массива**, который выводиться на экран с помощью **команды** **echo.**

Браузер выведет цифру: **1**

В **массиве** также можно хранить **имена**. Например,

**$e= array ('Bogdan', 'MAZ', 'LAZ', 'Solaris', 'Korosa');**

**echo $e[0],'<br>';** на экран выведет слово **Bogdan**

**Ассоциативный массив** – отличается от обычного массива тем, что в нем **можно самостоятельно задать имя индекса**. Например:

**$f = array (**

**‘name of bus’ =>”Bogdan”,**

**‘year of production’ => 2012,**

**‘manufacturer country’ => “Ukraine”,**

**‘prize’ => 155000**

**);**

**echo $f[‘prize’]; выведет цифру 155000**

**Многомерный массив** – массив, в который вложен другой массив.

Например:

**$f = array (**

**'details' => array (**

**'type of engine' => 'diesel',**

**'capacity' => “106 passengers”,**

**'type of gear' => "automatic"**

**)**

**);**

**echo $f['details']['type of engine'];** выведет слово **diesel**, так как мы указали эта строка находиться под именем индекса **'type of gear'** в массиве с именем индекса **'details'**

Еще пример использования **многомерного массива:**

**$g = array(**

**'details' => array(**

**'electronic system - Cegelik',**

**'capacity - 110',**

**'floor type - low floor')**

**);**

**echo $g['details']['2'] . '<br>';**  выведет строку **floor type - low floor**, так как она идет в индексе под номером **2**

**Еще пример:**

**Создаем два массива (вложенный в один многомерный), в котором будут указанны номера кабинетов и название предметов, которые будут в них проводится.**$lessons = array(  
 array("room" => 1, "lesson" => 'Алгебра'),  
 array('room' => 2, 'lesson' => 'Геометрия'),  
 array('room' => 3, 'lesson' => 'Физика'),  
 array('room' => 4, 'lesson' => 'География'),  
 array('room' => 5, 'lesson' => 'Химия'),  
 array('room' => 6, 'lesson' => 'ДПЮ'),  
 array('room' => 7, 'lesson' => 'Украинский язык'),  
 array('room' => 8, 'lesson' => 'Украинская литература'),  
 array('room' => 9, 'lesson' => 'Зарубежная литература'),  
 array('room' => 10, 'lesson' => 'Физкультура'),  
 array('room' => 11, 'lesson' => 'Биология'),  
 array('room' => 12, 'lesson' => 'История Украины'),  
 array('room' => 13, 'lesson' => 'Английский язык'),  
 array('room' => 14, 'lesson' => 'Астрономия'),  
 );

**Но, для того, чтобы вывести содержимое многомерного массива гораздо проще, используется цикл FOREACH.**

**Например:**

**Задача.** У учительницы **Бабены** есть 7 учеников. Эти данные мы внесли в **многомерный массив** **$teachers.** Целью задачи является внести в Базу Данных их **id**, для выполнения дальнейших манипуляций с этими данными.

$teachers = array('id' => '1', "Name" => 'Бабена', "pupils" => array(  
 array('id' => '1', 'Name' => 'Бабченко'),  
 array('id' => '2', 'Name' => 'Волобуев'),  
 array('id' => '3', 'Name' => 'Давыденко'),  
 array('id' => '4', 'Name' => 'Горячев'),  
 array('id' => '5', 'Name' => 'Горовой'),  
 array('id' => '6', 'Name' => 'Гендина'),  
 array('id' => '7', 'Name' => 'Жижа'),  
)  
);

А вот для этого нам и пригодиться использования двойного цикла **FOREACH.**

Для того, чтобы понять как работает двойной **FOREACH** с многомерным массивом мы полностью продебажим каждый наш шаг. Поехали!

1. Начнем с работы **первого цикла** **foreach**

foreach($teachers as $pupils){  
  
 echo '<pre>';  
 print\_r($teachers);  
 echo '</pre>';  
 die;

Его функция вывести из массива **$teachers** все данные. На первом этапе посмотрим, что лежит:

a) внутри массива **$teachers**:

Array

(

[id] => 1

[Name] => Бабена

[pupils] => Array

(

[0] => Array

(

[id] => 1

[Name] => Бабченко

)

[1] => Array

(

[id] => 2

[Name] => Волобуев

)

[2] => Array

(

[id] => 3

[Name] => Давыденко

)

[3] => Array

(

[id] => 4

[Name] => Горячев

)

[4] => Array

(

[id] => 5

[Name] => Горовой

)

[5] => Array

(

[id] => 6

[Name] => Гендина

)

[6] => Array

(

[id] => 7

[Name] => Жижа

)

)

)

б) внутри ключа, например **$teachers[Name] или $teachers[id] для выполнения задачи:**

foreach($teachers as $pupils){  
  
 echo '<pre>';  
 print\_r($teachers[*Name*]);  
 echo '</pre>';  
 die;

Бабена

1. А вот для того, чтобы посмотреть, что лежит в переменной **$pupils**, мы должны использовать **второй цикл** **foreach с ключами.** Там находитьсянаш второй массив, который лежит в ключе [pupils]. (смотри в таблице выше, там где **“pupils” => array**)

а) второй **foreach** автоматом перенаправляет нас вовнутрь второго массива, который вложен в первый:

foreach($teachers as $keys => $pupils) {  
 foreach ($pupils as $data\_pupils) {  
 echo '<pre>';  
 print\_r($pupils);  
 echo '</pre>';  
 die;

Array

(

[0] => Array

(

[id] => 1

[Name] => Бабченко

)

[1] => Array

(

[id] => 2

[Name] => Волобуев

)

[2] => Array

(

[id] => 3

[Name] => Давыденко

)

[3] => Array

(

[id] => 4

[Name] => Горячев

)

[4] => Array

(

[id] => 5

[Name] => Горовой

)

[5] => Array

(

[id] => 6

[Name] => Гендина

)

[6] => Array

(

[id] => 7

[Name] => Жижа

)

)

б) А для того, чтобы достать содержимое второго массива, нужно обратиться к переменной **$data\_pupils**:

foreach($teachers as $keys => $pupils) {  
 foreach ($pupils as $data\_pupils) {  
 echo '<pre>';  
 print\_r($data\_pupils);  
 echo '</pre>';  
 die;

Array

(

[id] => 1

[Name] => Бабченко

)

в) чтобы достать содежимое номера [id] нужно указать его, как ключ внутри массива **$data\_pupils**

foreach($teachers as $keys => $pupils) {  
 foreach ($pupils as $data\_pupils) {  
 echo '<pre>';  
 print\_r($data\_pupils[*id*]);  
 echo '</pre>';  
 die;

1

**Пример использование значений булевого типа в массиве:**

**$f = array (**

**'name of bus'=> "Man",**

**'year of production the model' => 1991,**

**'manufacturer country' => "Germany",**

**'prize' => 155000,**

**'details' => array(**

**'type of engine' => 'diesel',**

**'capacity' => "106 passengers",**

**'type of gear' => "automatic",**

**'was in used' => true**

**)**

**);**

**echo $f['details']['was in used'],'<br>';** выведет значение **1**

**Использование PHP внутри html**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>Пример использования PHP в html</title>**

**</head>**

**<body>**

**Сегодня <?php echo date(‘d.m.Y h:i’);?>**

**</body>**

**<html>**

**Еще один более сложный пример:**

**<!DOCTYPE HTML>**

**<html>**

**<head>**

**<title>Практикуем html и php</title>**

**<meta charset="UTF-8">**

**</head>**

**<body style="background:grey;">**

**<p style="color: whitesmoke">**

**<?php**

**$a = 50;**

**echo 'Городу Сумы необходимо закупить $a автобусов для замены маршруток <br>';**

**echo "Городу Сумы необходимо закупить $a автобусов для замены маршруток <br>";**

**любой другой код**

**?>**

**</p>**

**</body>**

**</html>**

Выведет весь интерпретируемый код **php**, который находиться внутри тегов **<p>**в виде параграфа**<p**>, с заданными параметрами **html**, а именно **белый текст** на **сером** бекграунде:

Городу Сумы необходимо закупить $a автобусов для замены маршруток   
Городу Сумы необходимо закупить 50 автобусов для замены маршруток

**Операторы сравнения**

**>** больше

**<** меньше

**>=** больше или равно

**<=** меньше или равно

**==** равно

**!=** не равно

**Логические операторы**

**and (и) &&**

**or (или) ||**

**Операторы условия и их выполнение в языке PHP**

1. **Условие сравнения if:**

**$buses\_for\_buying = 'Bogdan'; //**создаем переменную, которую будем заменять на любую другую (**MAZ, LAZ, Solaris**), которая будет выполнять условие в коде.

**if($buses\_for\_buying == 'Bogdan'){**

**echo 'Хороший вариант для покупки - цена соответствует качеству';**

**}**

**//**задаем условие, **если** переменная **$buses\_for\_buying будет соответствовать** значению **'Bogdan', то выполниться код echo 'Хороший вариант для покупки - цена соответствует качеству' и оно выведется на экран.**

**if($buses\_for\_buying == 'MAZ'){**

**echo 'Хороший вариант для покупки - но производитель не украинский';**

**}**

**if($buses\_for\_buying == 'LAZ'){**

**echo 'Завод закрыл вое производство';**

**}**

**if($buses\_for\_buying == 'Solaris'){**

**echo 'Очень высокая цена на автобусы';**

**}**

1. **Оператор If else:**

**if($buses\_for\_buying == 'Bogdan'){**

**echo 'Хороший вариант для покупки - цена соответствует качеству';**

**}else ($buses\_for\_buying == 'MAZ'){**

**echo 'Хороший вариант для покупки - но производитель не украинский';**

**}**

1. **Оператор if elseif**

Разница между **else if** и **else** – после **elseif(**мы сюда пишем условие**){**сюда выполнение кода**}**,  
а после **else{**пишем выполнение кода без какого-либо условия вообще**}**.

Читаем код с **else if**: если не выполнилось первое условие, тогда проверим второе условие!   
Используем, если нам нужно после **if** проверять еще какое - то условие. А, если использовать else, то при успешном выполнении первого условия в if – второе уже проверяться не будет.

Тоесть, после того как будет истинно первое условие и на другие написано **else**, то оно после **else** не заходит! А, если просто if () if () if () без **else**, то будет заходить в каждый...

Например:

if ($b == 0) {  
 $arrRandBalls[$b] = $randBalls;  
} elseif (!in\_array($randBalls, $arrRandBalls)) {  
 $arrRandBalls[$b] = $randBalls;  
}  
$b++;

**Оператор Switch:**

Иногда даже использование конструкции операторов [if](http://php720.com/lesson/14) .. [elseif](http://php720.com/lesson/18) несколько утомляет.   
Чтобы исправить эту ситуацию есть оператор switch. Синтаксис:

Сначала записывается ключевое слово switch, после которого в скобках записывается некоторое выражение.   
Далее, после слова case нужно перечислить возможные варианты значений, если значение истина, то выполняется группа операторов, которые записаны до оператора break. Если ни одно условие не подходит, то выполняется оператор default (если оператор default не записывать, то при не выполнении никаких других условий - ничего не произойдет).

**<?php**

**$speed = 55;**

**switch($speed)**

**{**

**case 30 :**

**echo "Ваша скорость 30 км/час";**

**break;**

**case 58 :**

**echo "Ваша скорость 50 км/час";**

**break;**

**case 70 :**

**echo "Превышение скорости !";**

**break;**

**default :**

**echo "Скорость в пределах нормы";**

**break;**

**}**

**?>**

1. **Условие a>b:**

**$prices\_of\_buses = "150000 'euros' <br>";**

**if($buses\_for\_buying < "150000 'euros'"){**

**echo 'Дешевый вариант';**

**}**

**if($buses\_for\_buying = "150000-170000 'euros'"){**

**echo 'Приемлемый вариант';**

**}**

**if($buses\_for\_buying > "170000 'euros'"){**

**echo 'Дорого';**

**}**

1. **Условие != (не равно), ==(равно):**

**$winner = 778;**

**if($winner!=777){**

**echo 'Вы проиграли!';**

**}**

**if($winner == 777){**

**echo 'Вы выиграли!';**

**}**

Выведет: **Вы проиграли!**

1. **Условие**, когда к нему прикрепляется еще одно условие - **if else** (если не выполняется **первое условие**, тогда выполняется **второе** после **else:**

**$winner = 777;**

**if($winner!=777){**

**echo 'Вы проиграли!';**

**} else {**

**echo 'Вы выиграли!';**

**}**

Выведет: **Вы выиграли!**

**Логические операторы**

1. **Условие**, когда прикрепляется условие **and (&&) - и:**

**$bus\_for\_Sumy = "Man";**

**$year\_of\_producing = 10;**

**if($year\_of\_producing >=10 and $bus\_for\_Sumy != "Man"){**

**echo 'Старый автобус';}else{**

**echo 'Хороший б/у автобус из Европы <br>';}**

Тоесть, мы задали условие, что если автобусу старше десяти лет, но (**и -and**) этот автобус **не** «Ман» , тогда выведет фраза **Старый автобус**. **Иначе** выведет **Хороший б/у автобус из Европы**

1. **Условие**, когда прикрепляется условие **or (||) - или:**

**$amount\_of\_trolleybuses = 45;**

**$amount\_of\_buses = 4;**

**if($amount\_of\_trolleybuses >= 113 or $amount\_of\_buses >= 68){**

**echo '<br> В городе достаточное количество общественного транспорта';**

**}else{**

**echo '<br> Не достаточное количество общественного транспорта';**

**}**

Выведет: **Не достаточное количество общественного транспорта**, так как условие того, что количество троллейбусов должно быть **не меньше 113** штук **или** автобусов **не меньше 68** штук –не выполнилось.

**Тернарный оператор**

Тернарный оператор работает почти также как и [оператор **if**](http://php720.com/lesson/14), но при использовании тернарного оператора, мы вместо ключевых слов пишем **?** и **:**.

Синтаксис:

**$var = condition ? exp1 : exp2;**

Если условие выполняется, то [переменной](http://php720.com/lesson/3) $var присваивается результат вычисления exp1, иначе exp2.

Пример:

**<?php**

**$speed = 55;**

**echo ($speed <= 60) ? "Скорость в пределах нормы" : "Превышение скорости !";**

**?>**

В результате мы увидим строку - "Скорость в пределах нормы".

Еще примеры:

1. **$buses = 45;**

**echo ($buses >= 50) ? 'Достаточное количество автобусов' : 'Нехватка автобусов';**

Выведет: **Нехватка автобусов**

// конструкцию читаем так: если выполняется условие $buses >= 50 - тогда выводим на экран ‘'Достаточное количество автобусов’, иначе – ‘Нехватка автобусов’.

1. **for($i = 1; $i <= 50; $i++){**

**echo $i%2 === 0 ? $i . '<br />' : ''; //с помощью тернарного оператора ?: выводим четные числа от 1 до 50**

**}**

**Выведет:**

2  
4  
6  
…  
50

//конструкцию читаем так: если выполняется условие i - четное число, тогда выводим его на экран, иначе - не выводим

## А также был внедрен альтернативный синтаксис  управляющих структур

(PHP 4, PHP 5, PHP 7)

PHP предлагает альтернативный синтаксис для некоторых его управляющих структур, а именно: if, while, for,foreach и switch. В каждом случае основной формой альтернативного синтаксиса является изменение открывающей фигурной скобки на двоеточие (:), а закрывающей скобки на endif;, endwhile;, endfor;, endforeach; или endswitch; соответственно.

<?php if ($a == 5): ?>  
A равно 5  
<?php endif; ?>

|  |
| --- |
| if..endif |

|  |  |
| --- | --- |
| **Пример C-6. Миграция: новый синтаксис if..endif**   |  | | --- | | if ($foo):  echo "yep\n";  elseif ($bar):  echo "almost\n";  else:  echo "nope\n";  endif; | |

Обратите внимание, что точки с запятой заменены на двоеточия во всех операторах, кроме одного, заканчивающего выражение (endif).

**Циклы**

Циклы позволяют повторять определенное (и даже неопределенное - когда работа цикла зависит от условия) количество раз различные [операторы](http://www.php.su/learnphp/operators/). Данные операторы называются телом цикла. Проход цикла называется итерацией.

Цикл - циклическое повторение кода, внутри которого ты можешь писать любую конструкцию языка

PHP поддерживает три вида циклов:

* Цикл с предусловием (**[while](http://www.php.su/learnphp/cs/?cycles" \l "while)**);
* Цикл с постусловием (**[do-while](http://www.php.su/learnphp/cs/?cycles" \l "do-while)**);
* Цикл со счетчиком (**[for](http://www.php.su/learnphp/cs/?cycles" \l "for)**);
* Специальный цикл перебора массивов ([**foreach**](http://www.php.su/learnphp/cs/?cycles#foreach)).

При использовании циклов есть возможность использования операторов **[break](http://www.php.su/learnphp/cs/?cycles" \l "break)** и **[continue](http://www.php.su/learnphp/cs/?cycles" \l "continue).**Первый из них прерывает работу всего цикла, а второй - только текущей итерации.

1. **For:** нужен для того, чтобы повторить наш код заданное количество раз. Тоесть **вместо** того, чтобы **5 раз** прописывать команду **echo**, мы можем **один раз** прописать команду **for**

**Например:** нам нужно **5 раз вывести на экран** фразу **Сумчанам сучасні автобуси 100+ пасажирів**

**for($i), где $i** переменная цикла, в которой будет храниться текущая итерация (повтор цикла)

**for($slogan=1; $slogan<=5; $slogan++){**

**echo 'Сумчанам сучасні автобуси 100+ пасажирів <br>';**

**}**

где **$slogan=1** - секция объявления; **$slogan<=5** - секция условия; **$slogan++** - секция изменения итерируемой переменной $sloganна одну единицу больше

**Выполнения цикла** происходит в такой последовательности:

1. Цикл считывает объявленную переменную, которая равна - 1
2. Проверяет условие, согласно которому переменная должна быть меньше или равна цифре - 5
3. При выполнение условия выше – выводит на экран **Сумчанам сучасні автобуси 100+ пасажирів**, а после этого увеличивает переменную на единицу.
4. Цикл снова проверяет – равна ли переменная 5 или меньше, и если условие выполняется, то выводит на экран **Сумчанам сучасні автобуси 100+ пасажирів.** После этого увеличивает переменную на единицу. И так до тех пор, пока переменная наконец-то не будет больше цифры 5. После этого цикл перестает работать и выводить фразу на экран **Сумчанам сучасні автобуси 100+ пасажирів**

Другими словами, слоган **Сумчанам сучасні автобуси 100+ пасажирів** будет выведен, столько раз, пока переменная **$slogan** не превысит **условие меньше или равно 5**.

**Бесконечность цикла** можно задать путем не указывания секции увеличения переменной на единицу.

**Например:**

**for($slogan=1; $slogan <= 5;){**

**echo 'Сумчанам сучасні автобуси 100+ пасажирів <br>';**

**}**

**Пример сочетания условия if и цикла for:**

**for($o=1;$o<10;$o++){**

**echo "<br> $o";**

**if($o%2==0){**

**echo ' - четное число';**

**}**

**if($o%2!==0){**

**echo ' - нечетное число';**

**}**

**}**

**Пример сочетания условия if else и цикла for:**

**for( $p = 1; $p < 10; $p++){**

**echo "<br> $p";**

**if($p % 2 == 0){**

**echo " - четное число";**

**}else{**

**echo " - нечетное число";**

**}**

**}**

Данный цикл также используется, чтобы сделать блог

1. **While:** выполняет тоже самое, что и цикл for, но отличается тем, что из цикла for используется лишь **секция условия**, а остальные части используются уже в самом блоке коде за пределами цикла.

Например:

**$q = 0;**

**while($q < 5){**

**echo '<br>Сумчанам сучасні автобуси 100+ пасажирів';**

**$q++;**

**}**

**Читаем код так: если условие - истина($q<5),** **то выполняется тело цикла** - **echo '<br>Сумчанам сучасні автобуси 100+ пасажирів';**

Данный цикл используется, чтобы выводить статьи в нашем блоге

1. **Foreach:** необходим для того, чтобы **работать с массивом**.

**Цикл перебора массивов foreach**

В PHP4 появился еще один специальный тип цикла - **foreach**. Данный цикл предназначен специально для перебора массивов.

Синтаксис цикла **foreach** выглядит следующим образом:

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**. Приведем пример работы цикла **foreach**:

<?php  
$names["Иванов"] = "Андрей";  
$names["Петров"] = "Борис";  
$names["Волков"] = "Сергей";  
$names["Макаров"] = "Федор";  
foreach ($names as $key => $value) {  
echo "<b>$value $key</b><br>";  
}  
?>

Рассмотренный сценарий выводит:

**Андрей Иванов**  
**Борис Петров**  
**Сергей Волков**  
**Федор Макаров**

У цикла **foreach** имеется и другая форма записи, которую следует применять, когда нас не интересует значение ключа очередного элемента. Выглядит она так:

foreach (массив as $значение)  
команды;

В этом случае доступно лишь значение очередного элемента массива, но не его ключ. Это может быть полезно, например, для работы с массивами-списками:

<?php  
$names[] = "Андрей";  
$names[] = "Борис";  
$names[] = "Сергей";  
$names[] = "Федор";  
foreach ($names as $value) {  
echo "<b>$value</b><br>";  
}  
?>

**3.1)** Например, нам нужно **вывести на экран содержимое массива (его элементов) полностью:**

**$bus\_Bogdan = array (**

**'Type of engine:' => 'diesel (Euro-5)',**

**'Transmission type:' => 'automatic',**

**'Max speed at full load, km / h:' => 60,**

**'Type of floor:' => 'low-floor',**

**'Air conditioner availability:' => false**

**);**

**foreach($bus\_Bogdan as $full\_content\_array){**

**echo '<br>'. $full\_content\_array;**

**}**

Выведет:

diesel (Euro-5)  
automatic  
60  
low-floor  
1

Тоесть, с новой строки будут выводиться все значения массива по порядку благодаря конструкции:

**foreach ($bus\_Bogdan as $full\_content\_array){**

**echo '<br>'. $full\_content\_array;**

где **$bus\_Bogdan** – значение индекса (ключа[0]), а **$full\_content\_array –** значение массива(10).

**3.2)** Также с помощью цикла **foreach** можно осуществлять математические операции:

**$digits = array(5,6,7,8,9,10);**

**foreach ($digits as $sum\_of\_digits){**

**echo '<br> Сума чисел:' . ($sum\_of\_digits + $sum\_of\_digits);**

**}**

**Выведет:**

Сума чисел:10  
Сума чисел:12  
Сума чисел:14  
Сума чисел:16  
Сума чисел:18  
Сума чисел:20

**3.3) Foreach** также необходим **для вывода** не только содержимого массива(его элементов), но и самих **названий ключей(индексов) массива :**

Осуществляется благодаря конструкции

**$bus\_Bogdan = array (**

**'Type of engine:' => 'diesel (Euro-5)',**

**'Transmission type:' => 'automatic',**

**'Max speed at full load, km / h:' => 60,**

**'Type of floor:' => 'low-floor',**

**'Air conditioner availability:' => true,**

**foreach($bus\_Bogdan as $key => $full\_content\_array){**

**echo '<br>'. $key . ' '. $full\_content\_array;**

**}**

Выведет:

Type of engine: diesel (Euro-5)  
Transmission type: automatic  
Max speed at full load, km / h: 60  
Type of floor: low-floor  
Air conditioner availability: 1

**3.4) А для того, чтобы вывести содержание многомерного массива используется вот такая конструкция с использованием функций print\_r и var\_dump:**

1. **echo '<pre>';**

**print\_r($bus\_details);**

**echo '</pre>';**

С помощью функции **print\_r** на экран выведем полное содержание многомерного массива **с их ключами и значениями:**

Array

(

[Capacity, persons:] => Array

(

[Seats:] => 30

[Full capacity:] => 106

)

[Dimensions:] => Array

(

[Length,] => mm11960

[Width,] => mm2550

[Height,] => mm3140-3200

[Wheel base,] => mm5860

)

)

1. **echo '<pre>';**

**var\_dump($bus\_details);**

**echo '</pre>';**

array(2) {

["Capacity, persons:"]=>

array(2) {

["Seats:"]=>

int(30)

["Full capacity:"]=>

int(106)

}

["Dimensions:"]=>

array(4) {

["Length,"]=>

string(7) "mm11960"

["Width,"]=>

string(6) "mm2550"

["Height,"]=>

string(11) "mm3140-3200"

["Wheel base,"]=>

string(6) "mm5860"

}

}

**Вывести данные из массива и завести в новый массив обратно:**

**В случае, когда работаем с одномерным массивом**

если тебя устраивает что при каждом вызове $statistic [ ] = $megaSuperMAssiv; ключ $var будет идти от 0 до .... с шагом 1, то можно индекс не писать внутри [ ].

*//исключаем из массива $arr\_1000\_it рандомных шаров строки 12, 30, 45*

$statistic = array();  
 foreach ($arr\_1000\_it as $stringOfTheFile => $valueBalls) {*// где, $arr\_1000\_it - массив, $stringOfTheFile - номер строки и  
 // $valueBalls - сама строка с 6 числами* if ($stringOfTheFile !== 12 || $stringOfTheFile !== 30 ||  
 $stringOfTheFile !== 45) {  
 foreach ($valueBalls as $digit) {  
 $statistic[] = $digit;  
 }  
 }  
 }

**В случае, когда работаем с многомерным массивом**

Например:

**foreach( $f AS $k1 => $g ){**

**foreach ( $ g AS $k2 => $c){**

**foreach ( $c AS $k3 => $hui ){**

**$var[$k1][$k2][$k3] = $huil;**

**}}}**

то здесь если ты не напишешь ключи, то выйдет, что каждый раз у тебя каждый ключ увеличивается на один

Если идет два три цикла...и каждый цикл отвечает за свой ключ в многомерном массиве, если в многомерном массиве ключ не указать во время обхода в цикле то у тебя всегда будут элементы [0][0][0], [1][1][1], [2][2][2]а не например [0][1][1], [0][1][2], [0][1][3]

C помощью **функции var\_dump** выводятся не только **ключи и значения** многомерного массива, но и **тип данных в виде: int, string, bool**

**Функции**

Нужны, чтобы блок кода(циклы, массивы, переменные, данные и тд), который в них содержится, использовать в любой другой части остального кода. Функция в PHP состоит из **блока кода**, которому мы сами задаем **имя**.

Например:

**function get\_bigger\_digit($a, $b){**

**if ($a>$b){**

**echo '<br>'.$a;**

**}else**

**{**

**echo '<br>'.$b;**

**}**

**}**

**get\_bigger\_digit (68, 113);**

**get\_bigger\_digit (75, 100);**

**get\_bigger\_digit (186, 113);**

где, **get\_bigger\_digit – придуманое нами имя функции**, в которую записываются определенные аргументы **($a, $b**), а им, в свою очередь, потом просто привязываються любые значения **(68, 113), (75, 100) или (186, 113)**. Все что в скобках **{}** – блок кода, а словом **function** – мы просто объявляем все, что идет посленее - **функцией.** В конце конструкции мы должны вызвать **имя** функции еще столько раз, сколько нам нужно для вывода значений на экран.

На экран выведет:

113  
120  
186

Это аналог конструкции **if else** или **if**, но главное **отличие состоит в том**, что используя **лишь операторы условия**, нам бы **пришлось прописывать код и переменные со значениями трижды**:

**$a1 = 68;**

**$b1 = 113;**

**if ($a1>$b1){**

**echo'<br>' . $a1;**

**}else{**

**echo '<br>' . $b1;**

**}**

**$a2 = 75;**

**$b2 = 100;**

**if ($a>$b){**

**echo'<br>' . $a2;**

**}else{**

**echo '<br>' . $b2;**

**}**

**$a3 = 186;**

**$b3 = 113;**

**if ($a>$b){**

**echo'<br>' . $a3;**

**}else{**

**echo '<br>' . $b3;**

**}**

**И выведет тоже самое:**

113  
120  
186

Более простой пример использовании **функции:**

1. **function new\_buses(){**

**echo '<br> Сумчанам сучасні автобуси 100+';**

**}**

**new\_buses ();**

**new\_buses ();**

**new\_buses ();**

**Выведет:**

Сумчанам сучасні автобуси 100+  
Сумчанам сучасні автобуси 100+  
Сумчанам сучасні автобуси 100+

Аналогом **функции** также могут быть **цикли for или while**, но главное отличие заключается в том, что **echo** выведет строку на экран **разово лишь в самом блоке кода** **for**. А вот **имя функции** **new\_buses** мы можем прописать **где захотим,** и также можем вывести строку в любой момент **без прописывания самой функции и его блока** кода еще раз.

**for** ($q=1; $q<4; $q++){

echo '<br> Сумчанам сучасні автобуси 100+';

}

выведет тоже самое:

Сумчанам сучасні автобуси 100+  
Сумчанам сучасні автобуси 100+  
Сумчанам сучасні автобуси 100+

Главная **цель функции** привязывать блоки кода и **вызывать их заданное нами количество раз** в любых частях кода, а не прописывать большое количество повторений конструкций **if** или **if else.**

Лучше в 10-ти разных местах написать **get\_bigger\_digit();**, чем в 10-ть разных мест копировать одни и те же огромные коды.

Другими словами, сравнивая функцию и цикл ты сравнивашь картошку с гаражем -

в гараже может лежать картошка, как и цикл быть в функции, но сама картошка не есть гаражём

1. **Функция return**
   1. Функция может **не только выводить** переменные на экран, но и **возвращать** собственное значение. ФУНКЦИЯ - это "обертка" кода который нужно часто выполнять или "обертка" большого куска кода или просто обертка кода по желания программиста. Функция на вход получила данные, и вернула что-то. То что она вернула (return $summa; ) пишется, когда мы вызываем функцию sum balls. Функцию просто можно заюзать в любом месте, где мы захотим с разными входящими данными $num\_of\_balls.

**Конструкция работы функции:**

- данные для входа (создаем входящий параметр $num\_of\_balls, которому будут переданы значения при вызове функции )

- после return функция возвращает, что ей передали внутри блока кода

- вызываем функцию, чтобы увидеть что она вернула

Пример 1:

**function sum\_balls($num\_of\_balls){**

**$summa = array\_sum($num\_of\_balls);**

**return $summa;**

**}**

**$b = array(2,4,10);**

**echo sum\_balls($b);**

Выведет: 16

**Объяснение кода:**

Объявляем функцию **function** под названием **sum\_balls (**внутри пишем переменную функции **$num\_of\_balls**, которая является входящим параметром. Тоесть, при дебаге после вызова функции sum\_balls, входящий параметр $num\_balls будет иметь значение равное тому, которое мы передали при вызове функции sum\_balls( $b )**){**

Пишем, что конкретно мы будем с переменной делать

**$summa = array\_sum($num\_of\_balls);**

После команды return функция возвращает то, что мы в ней насчитали. Это значение будет зависить от входящего параметра $num\_of\_balls  
**return $summa;**

**}**

В нашу функцию вкладываем значения массива, сумма которых будет посчитана кодом, который вложен у нее внутри. $num\_balls - это входящий параметр ( переменная ) функции. В функцию запихиваем свой массив неявно когда вызываем функцию **sum\_balls**( $b );

**$b = array(2,4,10);**

**echo sum\_balls($b);**

**Пример 2:**

**function what\_means\_return($a4,$b4){**

**if ($a4>$b4){**

**return $a4;**

**}else{**

**return $b4;**

**}**

**}**

**$what\_means\_return = what\_means\_return(68,113);**

**echo '<br>' . $what\_means\_return;**

**Выведет: 113**

**Пример 3:**

**function what\_means\_return($a4,$b4){**

**if ($a4>$b4){**

**return $a4;**

**}else{**

**return $b4;**

**}**

**}**

**$d = 10;**

**$f = 4;**

**$what\_means\_return = what\_means\_return($d, $f);**

**echo '<br>' . $what\_means\_return;**

**Функция** **может** не только **выводить** свои значения на экран с помощью команды **echo** в браузере, но и может **возвращать** значения с помощью команды **return** . Тоесть, должна быть связка **echo + return**, где команда **echo** отвечает за **вывод** на экран значений, **возвращенных** командой **return** от кода.

Конструкция с использованием **return** удобна тем, что переменную **$what\_means\_return;** мы можем использовать в любой части кода.

* 1. Также **return** останавливает выполнение кода после себя

**function return\_stop\_code(){**

**echo 'Нам нужны большие автобусы';**

**return;**

**echo 'А маршрутчики пошли в жопу';**

**}**

**return\_stop\_code();**

**Выведет: Нам нужны большие автобусы** – так как после **return** выполнение кода останавливаеться и код не выполнется.

**3) Встроенные функции**

**- функции с уже готовым кодом, которые возвращают определенные параметры**

**3.1) abs:** Возвращает абсолютное значение number. Если number имеет тип [**float**](http://www.php.su/learnphp/datatypes/?float), возвращаемое значение также будет иметь тип [**float**](http://www.php.su/learnphp/datatypes/?float), иначе - [**integer**](http://www.php.su/learnphp/datatypes/?integer).

|  |  |
| --- | --- |
| **Пример 1. Пример**   |  | | --- | | <?php  $abs = abs(-4.2); // $abs = 4.2; (double/float) $abs2 = abs(5);   // $abs2 = 5; (integer) $abs3 = abs(-5);  // $abs3 = 5; (integer) ?> | |

**3.2)** **round:** округляет в сторону меньшего числа, если число после запятой меньше 5, и в сторону большего – если соответственно больше 5 или равно.

**round** -- Округляет число типа float

**Описание**

float **round** ( float val [, int precision] )

Возвращает округлённое значение *val* с указанной точностью *precision* (количество цифр после запятой). Последняя может быть отрицательной или нулём (по умолчанию).

|  |  |
| --- | --- |
| **Пример 1. Примеры функции round()**   |  | | --- | | <?php echo round(3.4);         // 3 echo round(3.5);         // 4 echo round(3.6);         // 4 echo round(3.6, 0);      // 4 echo round(1.95583, 2);  // 1.96 echo round(1241757, -3); // 1242000 echo round(5.045, 2);    // 5.05 echo round(5.055, 2);    // 5.06 ?> | |

**3.3)** **ceil**- Округляет дробь в большую сторону

## 

## Описание

float **ceil** ( float value )

Возвращает ближайшее большее целое от value. Тип возвращаемого значения остаётся [**float**](http://www.php.su/learnphp/datatypes/?float) т.к. диапазон [**float**](http://www.php.su/learnphp/datatypes/?float) больше [**integer**](http://www.php.su/learnphp/datatypes/?integer).

|  |  |
| --- | --- |
| **Пример 1. Пример функции ceil()**   |  | | --- | | <?php echo ceil(4.3);    // 5 echo ceil(9.999);  // 10 echo ceil(3.0);    // 3 ?> | |

**3.4)** **floor** – округляет дробь в меньшую сторону

**3.5) rand** –функция генерирует случайное число в диапазоне от минимального и до максимального

При вызове без параметров *min* и *max*, возвращает псевдослучайное целое в диапазоне от 0 до **RAND\_MAX**. Например, если вам нужно случайное число между 5 и 15 (включительно), вызовите *rand (5, 15)*.

|  |  |  |
| --- | --- | --- |
| **Пример 1. Пример**   |  | | --- | | <?php echo rand() . "\n"; echo rand() . "\n";  echo rand(5, 15); ?> |   Пример выше выведет что-то наподобие этого:   |  | | --- | | 7771  22264  11 | |

**3.6)** **min** -- Находит наименьшее значение

**Описание**

mixed **min** ( number arg1, number arg2 [, number ...] )  
  
mixed **min** ( array numbers )

Возвращает наименьшее из предложенных чисел.

Если в качестве аргументов передан только один - массив чисел, **min()** возвращает наименьшее из них. Если первый аргумент - integer или float, то обязательно должен быть хотя бы ещё один. В этом случае функция вернёт наименьшее из них. Можно сравнивать неограниченное количество чисел.

**Замечание:**PHP считает строку, которую не удалось преобразовать в число, как *0*, но вернёт саму строку, в случае, если все остальные числа положительные. Если указано несколько элементов, эквивалентных *0*, **min()** вернёт первый из них (самый левый).

|  |  |
| --- | --- |
| **Пример 1. Примеры использования min()**   |  | | --- | | <?php echo min(2, 3, 1, 6, 7);  // 1 echo min(array(2, 4, 5)); // 2  echo min(0, 'hello');     // 0 echo min('hello', 0);     // hello echo min('hello', -1);    // -1  // При указании нескольких массивов, они сравниваются полностью // в нашем примере: 2 == 2, но 4 < 5 $val = min(array(2, 4, 8), array(2, 5, 1)); // array(2, 4, 8)  // При указании массива и отдельных чисел,  // первый никогда не возвращается, т.к. считается самым большим $val = min('string', array(2, 5, 7), 42);   // string ?> | |

**3.7) count -** посчитает количество элементов массива или количество свойств объекта

Например:

**$a = array (2, 4, 89);**

**echo count ($a);**

Выведет: **3**

|  |  |
| --- | --- |
| **Пример 1. Пример использования count()**   |  | | --- | | <?php $a[0] = 1; $a[1] = 3; $a[2] = 5; $result = count($a); // $result == 3  $b[0]  = 7; $b[5]  = 9; $b[10] = 11; $result = count($b); // $result == 3;  $result = count(null); // $result == 0;  $result = count(false); // $result == 1; ?> | |

|  |  |
| --- | --- |
| **Пример 2. Пример рекурсивного использования count() (PHP >= 4.2.0)**   |  | | --- | | <?php $food = array('fruits' => array('orange', 'banana', 'apple'),               'veggie' => array('carrot', 'collard', 'pea'));  // рекурсивный count echo count($food, COUNT\_RECURSIVE);  // output 8  // обычный count echo count($food);                  // output 2  ?> | |

**Пример использование return и функции в сочетании с циклами и массивами:**

**// 1. Вставляем команду, которая снимает ограничение времени в исполнении кода в 30 секунд**

**ini\_set('max\_execution\_time', 9000);**

**// 2. Создаем массив с выиграшными шарами в лоттереи**

**$winning\_balls = array(1, 3, 7, 13, 21, 27);**

**// 3. Создаем функцию, которая вернет массив с выпавшими случайными шарами в лоттереи**

**function array\_random\_balls(){**

**// 3.1. Создаем пустой массив**

**$result\_lottery = array();**

**// 3.2. Заполняем пустой массив шестью рандомными значениями шаров с помощью цикла for**

**for($balls=1; $balls=<6; $balls++){**

**$result\_lottery[$balls] = rand(1, 36);**

**}**

**//3.3. Возвращаем значение заполненого массива рандомными шарами**

**return $result\_lottery;**

**}**

**// 4. Сравниваем значения двух массивов до тех пор, пока случайные шары второго массива не совпадут с выигрышными первого**

**// 4.1. Циклом while задаем условие: пока не совпадут массивы - будет выполнятся функция,**

**// в которой массив заполняется шестью случайными шарами**

**$counter = 0;**

**while ($winning\_balls != $result\_lottery){**

**//4.1.1. Переменной присваиваем значение функции, чтобы мы могли использовать ее значение в этом циикле**

**$value\_array\_random\_balls = array\_random\_balls();**

**$counter ++;**

**}**

**//5. Выведем количество итераций(повторений кода), которое было необходимым для выполнения условия задачи**

**echo $counter;**

**Повторить код, который не заполняет массив новыми рандомными значениями**

$average\_numb = array();  
for ($i = 0; $i < 3; $i++) {  
  
 $c = 0;  
 $second\_arr = array();  
 while ($win\_balls != $second\_arr) {  
 $second\_arr = secondArr();  
 $c++;  
 }  
 $average\_numb[$i] = $c;  
}  
  
echo '<br>Количество тиражей, чтобы выиграть в лоттерею в первый раз: ' . $average\_numb[0] . ' штук';  
echo '<br>Количество тиражей, чтобы выиграть в лоттерею в первый раз: ' . $average\_numb[1] . ' штук';  
echo '<br>Количество тиражей, чтобы выиграть в лоттерею в первый раз: ' . $average\_numb[2] . ' штук';  
  
echo '<br>Среднеарифметическое тиражей из трех лоттерей для выиграша: ' . round(array\_sum($average\_numb) / count($average\_numb)) . 'штук';

Сравниваем массивы с выиграшными и рандомными шарами

\* - Для того, чтобы посчитать количество итераций ставим условие, что сравнение массивов будет просходить в цикле while

\* до тех пор, пока шары не совпадут в двух массивах.

\* - Также для использования кода, который генерирует заполнение второго массива рандомными шарами, вызываем функцию secondArr():

\* $second\_arr = secondArr();

\* 5. Узнаем среднее арифметическое 3-тиражей для выиграша 3 шаров с 15:

\* - для этого нам нужно повторить код сравнения 3 раза и записать результат 3 итераций в новый пустой массив:

\* $average\_numb = array();

\* Для этого используем цикл for, где указываем, что код должен повториться 3 раза: for($i = 0; $i < 3; $i++)

\* 6. Для того, чтобы код повторился три раза, а не выводил число итерации лишь раз, нам нужно обнулить это число.

\* Иначе, следующие 2 итерации не будут заполняться и выведет ноль. Поэтому пишем перед условием цикла while переменную

\* $second\_arr еще раз, которой присваиваем пустой массив, чтобы удалить из памяти переменной $second\_arr уже

\* заполненые данные после генерации while(!$win\_balls = $second\_arr), и дать возможность в эту переменную внести новые

\* данные. Пустой массив $second\_arr = array() заполняться ничем не будет, и тем самым удаляет предидушие данные.

**Удалить пустые значения (нулевые) из массива:**

*//Create new array $returned\_array which will be without empty keys unlike the previous array $show\_arr example that have: $show\_arr[0] = 5, $show\_arr[1] = , $show\_arr[2] = 31, $show\_arr[3] = , $show\_arr[4] = 6, $show\_arr[5] = 14, $show\_arr[6] = 18, $show\_arr[7] = 10. This happen when $show\_arr[0] = 5 and $show\_arr[1] = 5.*function convertArray($array4convert)  
{  
 $returned\_array = array();  
 $k = 0;  
 foreach ($array4convert as $value) {  
 $returned\_array[$k] = $value;  
 $k++;  
 }  
 return $returned\_array;  
}  
  
*/\*\*Put $string in function that create new array that clear empty keys and return with good keys [0] - [5];  
и потом уже его берешь в запросе $array[1], $array[2]... у тебя всегда будут.  
 \* В цикле каждый массив $string обрабатывай функцией моей  
 \*/*foreach ($show\_arr as $key => $string) {  
  
 $clear\_str = convertArray($string);  
  
 $result = "INSERT INTO combinations (digit1, digit2, digit3, digit4, digit5, digit6)  
 VALUES ('$clear\_str[0]', '$clear\_str[1]', '$clear\_str[2]', '$clear\_str[3]', '$clear\_str[4]', '$clear\_str[5]')";  
  
 if (mysqli\_query($connection, $result)) {  
 echo "Records inserted successfully.";  
 } else {  
 echo "ERROR: Could not able to execute $result. " . mysqli\_error($connection);  
 }  
  
}  
mysqli\_close($connection);

# Функции для работы с файловой системой

**fopen** -- Открывает файл или URL

**$file = ‘file.csv’;**

**fopen($file, w);** где **$file** – переменная которой присоили имя создаваемого файла с раширением csv, а мод **w** – параметр, который указывает на тип доступа, который мы запрашиваем у потока. Он может быть одним из следующих вариантов:

| **Список возможных режимов для fopen(), используя mode** | |
| --- | --- |
| **mode** | **Описание** |
| 'r' | Открывает файл только для чтения; помещает указатель в начало файла. |
| 'r+' | Открывает файл для чтения и записи; помещает указатель в начало файла. |
| 'w' | Открывает файл только для записи; помещает указатель в начало файла и обрезает файл до нулевой длины. Если файл не существует - пробует его создать. |
| 'w+' | Открывает файл для чтения и записи; помещает указатель в начало файла и обрезает файл до нулевой длины. Если файл не существует - пытается его создать. |
| 'a' | Открывает файл только для записи; помещает указатель в конец файла. Если файл не существует - пытается его создать. В данном режиме функция [fseek()](http://php.net/manual/ru/function.fseek.php) не применима, записи всегда добавляются. |
| 'a+' | Открывает файл для чтения и записи; помещает указатель в конец файла. Если файл не существует - пытается его создать. В данном режиме функция [fseek()](http://php.net/manual/ru/function.fseek.php) влияет только на позицию чтения, записи всегда добавляются. |
| 'x' | Создаёт и открывает только для записи; помещает указатель в начало файла. Если файл уже существует, вызов **fopen()** закончится неудачей, вернёт **FALSE** и выдаст ошибку уровня **E\_WARNING**. Если файл не существует, попытается его создать. Это эквивалентно указанию флагов O\_EXCL|O\_CREAT для внутреннего системного вызова open(2). |
| 'x+' | Создаёт и открывает для чтения и записи; в остальном имеет то же поведение что и'x'. |
| 'c' | Открывает файл только для записи. Если файл не существует, то он создается. Если же файл существует, то он не обрезается (в отличие от 'w'), и вызов к этой функции не вызывает ошибку (также как и в случае с 'x'). Указатель на файл будет установлен на начало файла. Это может быть полезно при желании заблокировать файл (см. [flock()](http://php.net/manual/ru/function.flock.php)) перед изменением, так как использование 'w' может обрезать файл еще до того как была получена блокировка (если вы желаете обрезать файл, можно использовать функцию [ftruncate()](http://php.net/manual/ru/function.ftruncate.php) после запроса на блокировку). |
| 'c+' | Открывает файл для чтения и записи; в остальном имеет то же поведение, что и 'c'. |
| 'e' | Установить флаг close-on-exec (закрыть при запуске) на открытый файловый дескриптор. Доступно только в PHP скомпилированном на системах поддерживающих POSIX.1-2008. |

**fputcsv** - форматирует строку (переданную в виде массива) в расширение CSV и записывает её в файловый указатель. Возвращает длину записанной строки или **FALSE** в случае неудачи.

Пример:

**$arr\_1000\_count = arr1000count();**

**$file = 'balls1000.csv';**

**$fp = fopen($file, w);**

**foreach($arr\_1000\_count as $rows\_excel) {**

**fputcsv($fp, $rows\_excel);**

**}**

**fclose($fp);**

В функции **arr1000count()** вложен массив на 1000 значений. Мы вызываем его с помощью присваивания к переменной **$arr\_1000\_count**. Далее мы присваиваем к переменной **$file** файл **balls1000** с раширением **csv**. С помощью функции **fopen** создаем этот файл **balls1000.csv** и присваиваем функцию переменной **$fp**. С помощью цикла **foreach** перебираем массив **$arr\_1000\_count**, и его строки присваиваем переменной **$rows\_excel**.

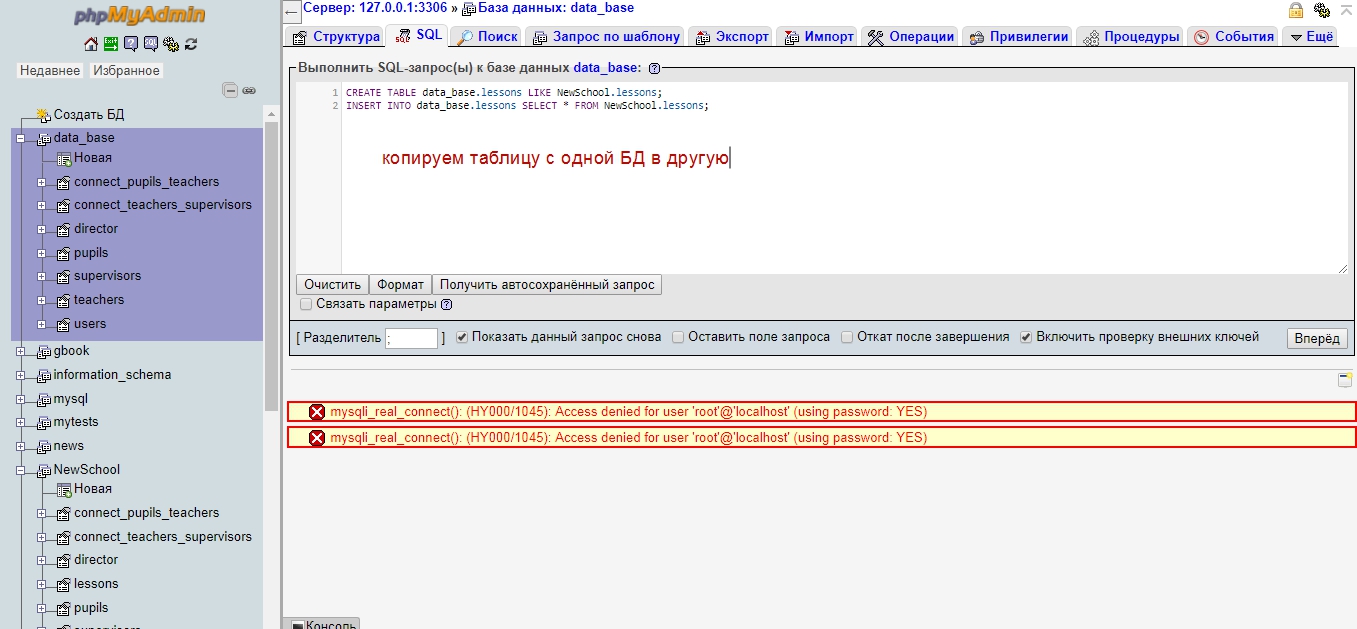
\* Функция **fputcsv()** - форматирует строку **$rows\_excel** (переданную в виде массива ) в файл с раширением CSV и записывает её в указанный файл, который находиться в переменной **$fp**

Функция **fclose** - возвращает длинну записанной строки или FALSE в случае неудачи.

Таким образом в папке возле нашего скрипта создастся файлик **balls1000.csv**, который можно открыть экселем в который мы записал построчно массивы .

Работа с PHP MyAdmin

Переместить таблицу с одной Базы Данных(БД) в другую:



CREATE TABLE db2.table LIKE db1.table;  
INSERT INTO db2.table SELECT \* FROM db1.table;

db2 база в которую копируем

db1 база из которой копируем

table название таблицы

Вставка данных в таблицу Базы данных MySQL

Вставка данных в таблицу базы данных MySQL

Теперь вы поняли, как создавать базу данных и таблицы в MySQL. В этом уроке вы узнаете, как выполнять SQL-запрос для вставки записей в таблицу.

Инструкция **INSERT INTO** используется для вставки новых строк в таблицу базы данных.

Давайте сделаем SQL-запрос с помощью инструкции **INSERT INTO** с соответствующими значениями, после чего мы выполним этот запрос вставки, передав его функции PHP mysqli\_query () для вставки данных в таблицу. Вот пример, который вставляет новую строку в таблицу **persons**, задавая значения для полей **first\_name, last\_name** и **email**.

пример

Процедурный объектно-ориентированный PDO

**<?php**

**/\* Attempt MySQL server connection. Assuming you are running MySQL**

**server with default setting (user 'root' with no password) \*/**

**$link = mysqli\_connect("localhost", "root", "", "data\_base");**

**// Check connection**

**if($link === false){**

**die("ERROR: Could not connect. " . mysqli\_connect\_error());**

**}**

**// Attempt insert query execution**

**$sql = "INSERT INTO persons (first\_name, last\_name, email) VALUES ('Peter', 'Parker', 'peterparker@mail.com')";**

**if(mysqli\_query($link, $sql)){**

**echo "Records inserted successfully.";**

**} else{**

**echo "ERROR: Could not able to execute $sql. " . mysqli\_error($link);**

**}**

**// Close connection**

**mysqli\_close($link);**

**?>**

**Еще пример:**

//Вот вызвал функцию, которая выбирает числа с одного массива и вставляет их в другой, чтобы отфильтровать нулевые значения чисел в массиве. После я использовал эту функцию в цикле.

**$show\_arr = genArray();**//тут массив с 1000 рандомных чисел

**function convertArray($array4convert){**

**$returned\_array = array();**

**$k = 0;**

**foreach ($array4convert as $value) {**

**$returned\_array[$k] = $value;**

**$k++;**

**}**

**return $returned\_array;**

**}**

**$without\_empty\_keys = convertArray($show\_arr);**

**foreach ($without\_empty\_keys as $key => $string) {**

**$result = "INSERT INTO combinations (digit1, digit2, digit3, digit4, digit5, digit6)**

**VALUES ('$string[0]', '$string[1]', '$string[2]', '$string[3]', '$string[4]', '$string[5]')";**

**if (mysqli\_query($connection, $result)) {**

**echo "Records inserted successfully.";**

**} else {**

**echo "ERROR: Could not able to execute $result. " . mysqli\_error($connection);**

**}**

**}**

**mysqli\_close($connection);**

Вставка чисел (int) в таблицу Базы данных MySQL

Отличается от вставки данных в виде строк лишь тем, что в **VALUES** мы не берем значения в скобки. В скобки нужно брать только строки **'Peter', 'Parker', 'peterparker@mail.com',** но никак не - **$id,** в который вложен массив с числами.

$id\_pupils = array(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19),

foreach($id\_pupils as $id) {  
  
 $sql\_insert\_connect = "INSERT INTO connect\_pupils\_teachers (id\_pupils) VALUES ($id)";  
 if (mysqli\_query($link, $sql\_insert\_connect)) {  
 echo "<br> Records inserted successfully.";  
 } else {  
 echo "<br> ERROR: Could not able to execute $sql\_insert. " . mysqli\_error($link);  
 }  
}

**Не забываем, что в MySQL при создании БД строку id заполняем AUTO\_INKREMENT –тоесть цыфрами 123..10. Поэтому нам не нужно прописывать строку id в функции INSERT или SELECT, так она заполниться автоматически.**

Если вы помните из предыдущей главы, поле id было отмечено значком AUTO\_INCREMENT. Этот модификатор сообщает MySQL автоматически присваивать значение этому полю, если он оставлен неуказанным, увеличивая предыдущее значение на 1.

Вставка id в таблицу должна происходить только при выключеных связях, так как выдаст ошибку!

Пример!

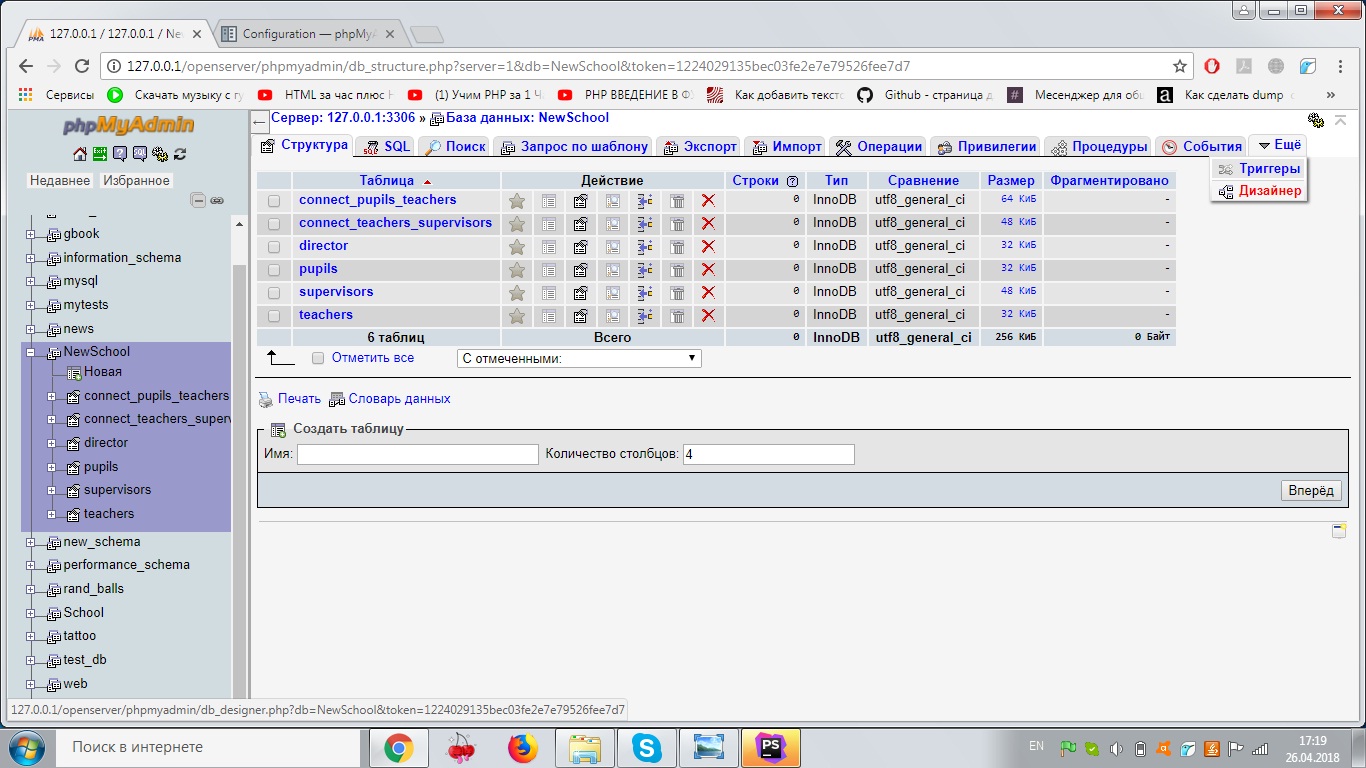
У нас есть код, который вставляет номера учеников **($pupils [id]** и номера учителей $data\_teachers[**[**id**]** в одну таблицу connect\_pupils\_teachers.

$link = mysqli\_connect('localhost', 'root', '', 'NewSchool');  
  
if ($link == false) {  
 die("ERROR: Could not connect. " . mysqli\_connect\_error());  
}  
  
foreach ($teachers\_pupils as $num\_teachers) {  
 foreach ($num\_teachers as $data\_teachers) {  
 foreach ($data\_teachers as $data\_pupils) {  
 foreach ($data\_pupils as $pupils) {  
  
 $connect = "INSERT INTO (pupils\_id, teachers\_id) VALUES ($pupils[id], $data\_teachers[id])";  
 if (mysqli\_query($link, $connect)) {  
 echo "<br> Records inserted successfully.";  
 } else {  
 echo "<br> ERROR: Could not able to execute $connect. " . mysqli\_error($link);  
 }  
 }  
 }  
 }  
}

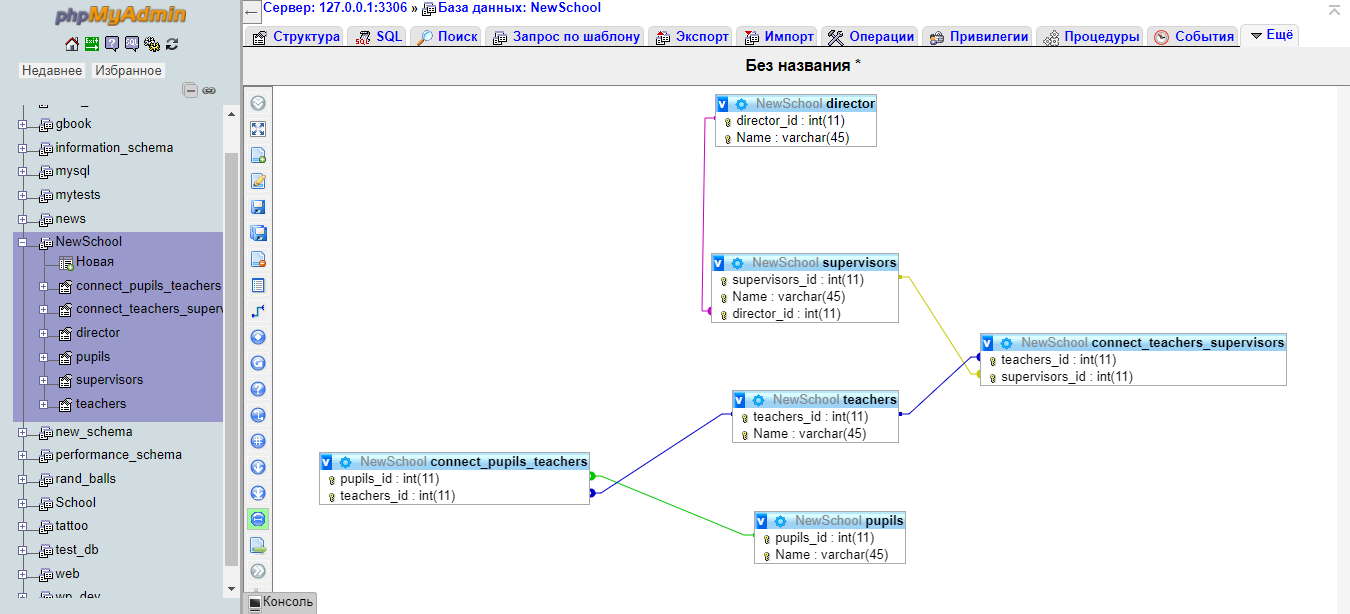
Чтобы не выдало ошибку и данные спокойно вставились, нужно отключить связи между созданными таблицами в MySQL:

Для этого заходим в закладку «Eщё» -> «Дизайнер» .

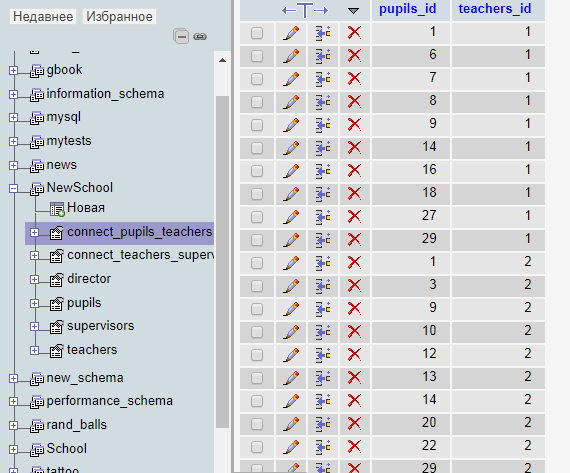
Потом нажимаем на кончики стрелок, которые связвают таблицы и жмем «Удалить». После того, как нам выдаст ОШИБКУ, мы просто перезагружаем таблицу и снова заходим в «Дизайнер». Мы увидим, что связи были разорванны.



Потом нажимаем на кончики стрелок, которые связвают таблицы и жмем «Удалить». После того, как нам выдаст ОШИБКУ, мы просто перезагружаем таблицу и снова заходим в «Дизайнер». Мы увидим, что связи были разорванны.



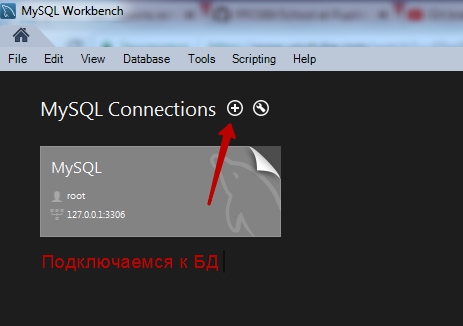
Потом запускаем наш код в PHP и увидим, что наши данные спокойно записались в таблицу MySQL.



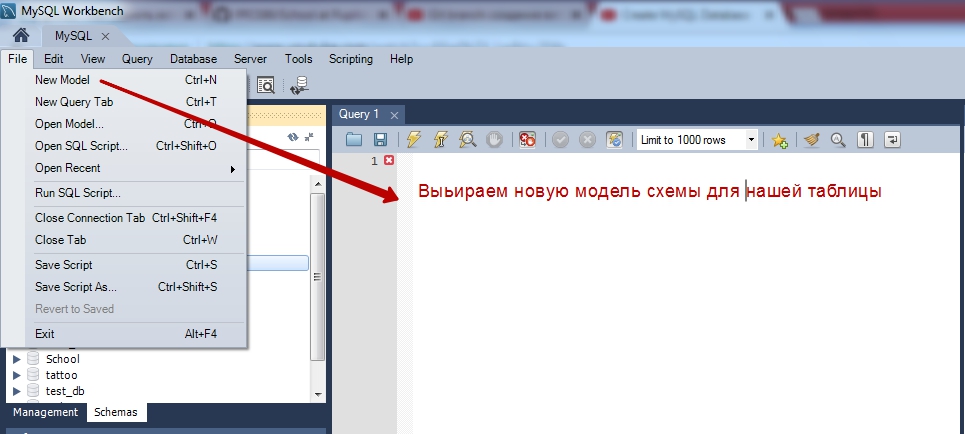
Создание таблиц с помощью MySQL Workbench и отправка их на PHPMyAdmin

**Ссылка на обучающее видео: https://www.youtube.com/watch?v=K6w0bZjl\_Lw&t=204s**

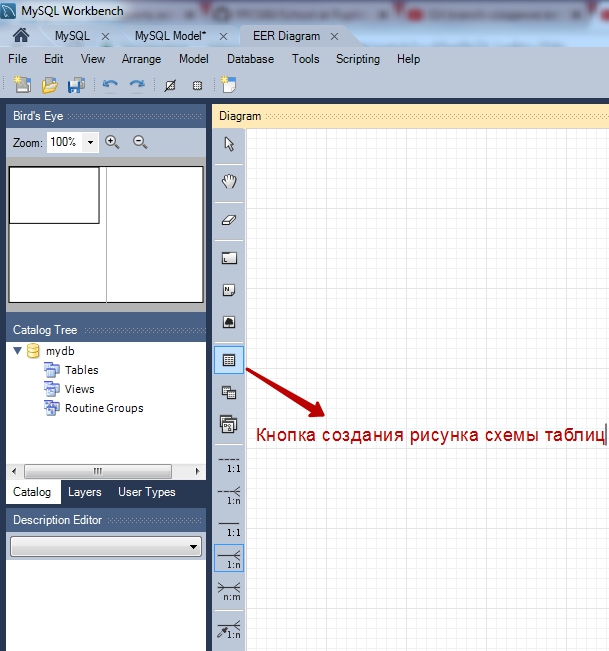
**Подключаемся к БД через MySQL Workbench:**



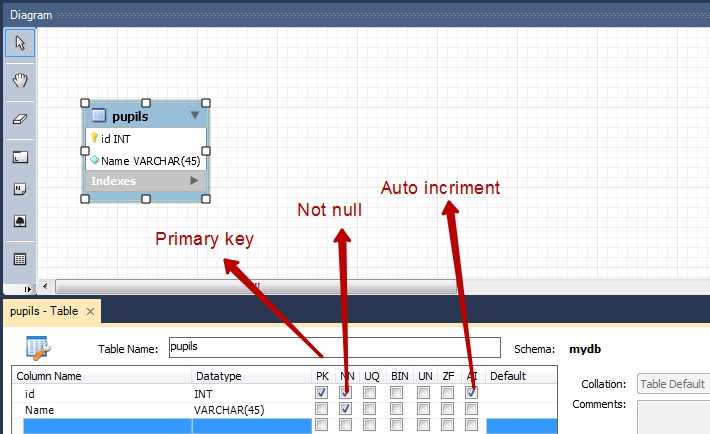
**Нажимаем File 🡪 New model**



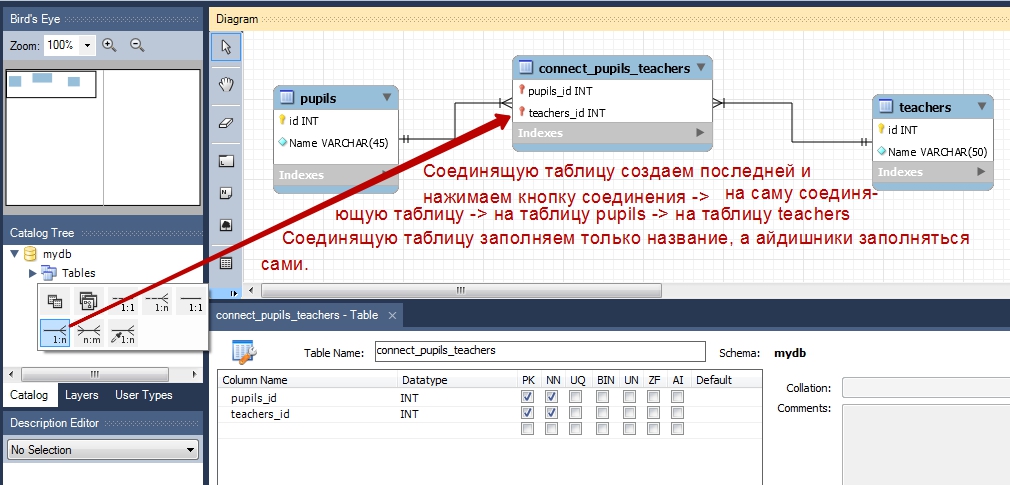
**Потом нажимаем Add diagram и у нас открываеться окно для рисованиея схемы таблицы. Там мы вибираем кнопку Place a New Table:**

****

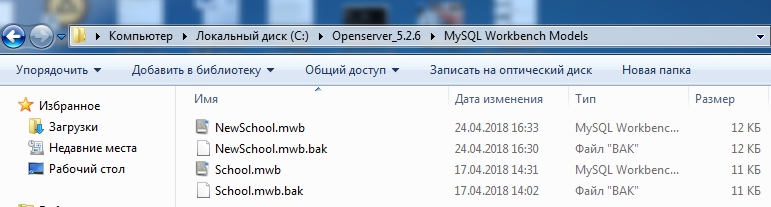
**На экране появилась новая ячейка-таблица. Клацнем по ней мишкой и откроться окно, в котором нужно дать таблице необходимое название. Далее клацаем по другим полям и задаем параметры полей таблицы: имя, тип и ставим галочки в нужных чекбоксах.**



**Создаем еще одну таблицу teachers, а уже после – соединяющую их таблицу connect\_pupils\_teachers. в ней поменять нужно лишь название, а остальные ее поля заполняться самостоятельно, после соединящих функций.**



**Далее мы сохраняем таблицу с помощью кнопки File -> Save Model в папку Openservera:**



**Завершающим шагом будет добавление проэкта таблиц в Базу Данных.**

**Но перед этим шагом пррэкт в БД уже должен быть создан путем или создание его в PHPMyAdmin или через консоль.**

**Нажимаем на панеле инструментов Database -> Forward Engineer и в сплывающим окнам вс время нажимать Next. Если не появилась, то в консоле прописать: create database NewSchool.**

**Или глянуть в конце видоса!**

Выборка данных из таблицы Базы данных MySQL

<?php  
  
*//Примеры всех возможных выборок из БД в MySQL:*$link = mysqli\_connect('localhost', 'root', '', 'web');*//подключаемся к базе данных web на сервере localhost с именем root без пароля  
  
// 1. Выведет список всех баз данных*$result = mysqli\_query($link,'SHOW databases');  
  
*// 2. Выведет список всех таблиц в БД web*$result = mysqli\_query($link, 'SHOW tables in web');  
  
*// 3. Выбирает все данные в таблице lessons*$result = mysqli\_query($link, 'SELECT \* FROM lessons');  
  
*// 4. Выбирает (SELECT) из(FROM) таблицы lessons лимит (LIMIT) 3 записи, начиная с 2.*$result = mysqli\_query($link, 'SELECT \* FROM lessons LIMIT 2,3');  
  
*// 5. Выбирает (SELECT) ВСЕ (\*) записи из (FROM) таблицы tbl\_name и сортирует их (ORDER BY) по полю id по порядку.*$result = mysqli\_query($link,'SELECT \* FROM lessons ORDER BY id');  
  
*// 6. Выбирает (SELECT) ВСЕ записи из (FROM) таблицы tbl\_name и сортирует их (ORDER BY) по полю id в ОБРАТНОМ порядке.*$result = mysqli\_query($link,'SELECT \* FROM lessons ORDER BY id DESC');  
  
*// 7. Выбирает (SELECT) ВСЕ (\*) записи из (FROM) таблицы users и сортирует их (ORDER BY) по полю id в порядке возрастания, лимит (LIMIT) первые 5 записей.*$result = mysqli\_query ($link,'SELECT \* FROM lessons ORDER BY id LIMIT 5');  
  
*// 8. Выбирает (SELECT) ВСЕ (\*) записи из (FROM) таблицы teachers, в которой (WHERE)поле name соответсвует значению Сидоров Сидор Сидорович*$result = mysqli\_query($link, 'SELECT \* FROM teachers WHERE name = "Сидоров Сидор Сидорович"');  
  
*// 9. Выбирает все записи из таблицы teachers, где (WHERE) значение поля name начинается с Сид.*$result = mysqli\_query ($link,'SELECT \* FROM teachers WHERE name LIKE "Сид%"');  
  
*// 10. Соединяем таблицы teachers и courses по полям name, addr и title, и выбираем с них соответствующие записи. Для этого прописуем их id.*$result = mysqli\_query($link,'SELECT teachers.name, teachers.addr, courses.title  
FROM teachers  
INNER JOIN courses ON teachers.id = courses.id');  
  
*// 11. Получить данные с трех таблиц: teachers, courses и lessons( можно и более), в которых совпадают айдишники*$result = mysqli\_query($link,'SELECT teachers.name, courses.title, lessons.room  
FROM ((teachers  
INNER JOIN courses ON teachers.id = courses.id)  
INNER JOIN lessons ON teachers.id = lessons.teacher)');  
  
  
*// 12. Получить данные с таблиц: teachers, lessons и с их полей name, addr и room. Но благодаря функции RIGHT JOIN выбираем все значения из поля room  
// таблицы lessons полностью, не смотря на то совпали id или - нет.*$result = mysqli\_query( $link, 'SELECT teachers.name, teachers. addr, lessons.room   
FROM teachers  
RIGHT JOIN lessons ON teachers.id = lessons.teacher');

**А вот таким макаром я считал все данные из БД в массив $rows и вывел на экран в виде таблицы:**

**$result = mysqli\_query($connection, 'SELECT \* FROM combinations');**

**$rows = mysqli\_fetch\_all($result, MYSQLI\_ASSOC);**

**echo "<table>";**

**foreach ($rows as $key => $value){**

**echo "<tr>";**

**foreach ( $value as $columnName => $columnValue ){**

**echo "<td>" . $columnValue . "</td>";**

**}**

**echo "</tr>";**

**}**

**echo "</table>";**

**mysqli\_close($connection);**

**Пример синтаксиса. ‘SELECT’ обязательно берем в апострофы, а данные в ней - в кавычки “Сидоров Сидор Сидорович”**

$result = mysqli\_query($link, 'SELECT \* FROM teachers WHERE name = "Сидоров Сидор Сидорович"');

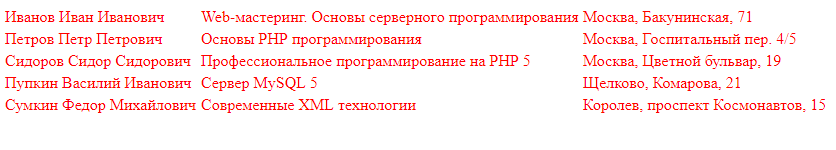
**А вот пример того, как вывести из БД - ‘web’, из таблицы – ‘teachers’, где в поле ‘addr’ введена запись Москва.**

$link = mysqli\_connect('localhost', 'root', '', 'web');*//подключаемся к базе данных web на сервере localhost с именем root без пароля*$result = mysqli\_query($link, 'SELECT \* FROM `teachers` WHERE `addr` LIKE "%москв%"');*//получить все записи(\*) с таблицы teachers*if( !$result){*//если не будет соединения с базой данных (таблицей teachers), то выведет Ошибка*echo 'Ошибка: '  
 . mysqli\_connect\_errno()  
 . ':'*//это тернарный оператор означающий "ИЛИ ТОГДА(ИНАЧЕ)"* . mysqli\_connect\_error();  
}  
$rows = mysqli\_fetch\_all($result, *MYSQLI\_ASSOC*);  
   
  
*//ДЗ: сделать таблицу с 4 строками*echo "<table>";  
foreach ($rows as $key => $value) {*//через цикл foreach проверяем массив с именем rows, где переменной key из массива присваиваем значение value* echo "<tr style='color:red'>";  
 foreach ( $value AS $columnName => $columnValue ){  
 echo "<td>".$columnValue."</td>";  
 }  
 echo "</tr>";  
}  
echo "</table>";

**Соединение данных их двух таблиц:**

$result = mysqli\_query($link,'SELECT teachers.name, teachers.addr, courses.title  
FROM teachers  
INNER JOIN courses ON teachers.id = courses.id');

Читаю код так: выбираем поля **name** с таблицы **teachers**, **title** с таблицы **courses** и **addr** с таблицы **teachers**. Таблицу **teachers** соединяем с таблицей **courses** на основе совпадания айдишников **id**.

Вывело:  
****

**Соединение данных их трех и более таблиц:**

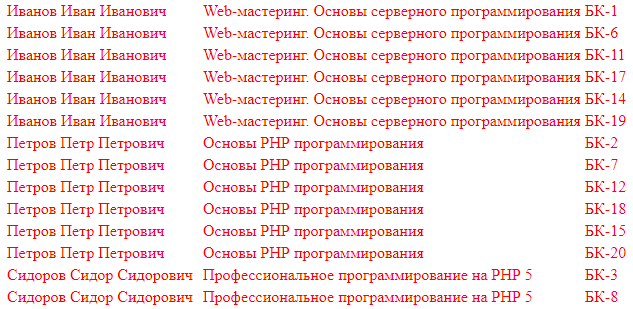
$result = mysqli\_query($link,'SELECT teachers.name, courses.title, lessons.room  
FROM ((teachers  
INNER JOIN courses ON teachers.id = courses.id)  
INNER JOIN lessons ON teachers.id = lessons.teacher)');

Читаю код так: выбираем поля **name** с таблицы **teachers**, **title** с таблицы **courses** и **room** с таблицы **lessons**. Таблицу **teachers** соединяем с таблицей **courses** на основе совпадания айдишников **id**, а после – соединяем уже с другой таблицей **lessons** на основе совпадения айдишников **id** и **teachers.**

**Соединение данных из двух таблиц, но когда данные из первой записываются полностью, независимо совпали id или нет:**

$result = mysqli\_query( $link, 'SELECT teachers.name, teachers. addr, lessons.room   
FROM teachers  
LEFT JOIN lessons ON teachers.id = lessons.teacher');

Можно код читать проще: соединяем таблицы **teachers** и **lessons** по полям name, addr и room, и выбираем с них соответствующие записи. Для этого прописуем их id

Вывело:  


……. и тд

**Нормализация БД:**

Нормализация баз данных заключается в приведении структуры хранения данных к **нормальным формам (NF)**. Всего таких форм существует 8, но часто достаточным является соблюдение первых трех. Рассмотрим их более подробно на примере учебной базы данных.  
**Нормализация в двух словах** - это когда ты данные хранишь не в одной таблице, где у тебя каша повторений много в строках, а каждая сущность находится в своей таблице.

По факту, **нормализация может быть утрированно названа** процессом создания все новых и новых таблиц до тех пор, пока избыточность и повторения не будут полностью уничтожены.

- при нормализации мы не имеем права обращаться к строкам по номерам  
- при перестановке столбцов ничего не должно поломаться, так не должно быть привязки к столбцам.

Например, нельзя говорить, что в столбце номер А – возьмем значение Сидоров Сидор, а правильно сказать – в столбце со значением ИМЯ, ФАМИЛИЯ содержится значение Сидоров Сидор.  
Таким образом, соблюдаются **реляционные принципы**.

В целом суть этих ограничений весьма проста: каждый факт, хранимый в БД, должен храниться один-единственный раз, поскольку дублирование может привести к несогласованности между копиями одной и той же информации. Следует избегать любых неоднозначностей, а также избыточности хранимой информации.

* *Схемой базы данных называется структура связей между полями и таблицами.*
* *Нормализацией схемы базы данных называется процедура, производимая над базой данных с целью удаления в ней избыточности.*
* выделяются шесть нормальных форм, пять из которых так и называются: первая, вторая, третья, четвертая, пятая нормальная форма, а также нормальная форма Бойса-Кодда, лежащая между третьей и четвертой.

Для реляционной модели данных разработано несколько нормализованных форм, три из которых являются основными.

**База данных считается нормализованной, если ее таблицы представлены как минимум в третьей нормальной форме.**

**Приводить ее к 3 нормальной форме нужно по порядку начиная с первой формы:**

## Первая нормальная форма

## - нет ключа id

- запрещает повторяющиеся столбцы и строки , содержащие одинаковую по смыслу информацию

## - каждая строка содержит только одно значение для каждого атрибута (столбца), то есть все атрибуты отношения имеют единственное значение (являются атомарными)

- строки таблиц не должны зависеть друг от друга, т.е. первая запись не должна влиять на вторую и наоборот, вторая на третью и т.д. Размещение записей в таблице не имеет никакого значения.

- аналогичная ситуация со **значением столбцов**. Их порядок не должен влиять на понимание информации.

- требует определить первичный ключ для таблицы, то есть тот столбец или комбинацию столбцов, которые однозначно определяют каждую строку - отсутствуют массивы и списки в любом виде

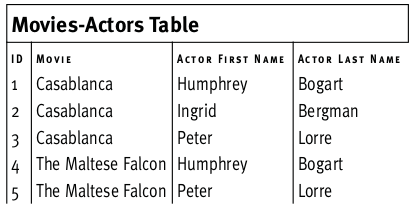
**Вторая нормальная форма:**

- значения столбцов зависят от первичного ключа **id**

- **значения столбцов** зависят друг от друга

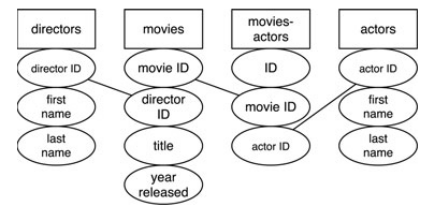
- если значения столбцов могут существовать без ключа **id**, значит нужно перекинуть их в новую таблицу

Например:



На рисунке выше и названия фильмов и имена актеров нарушают правила 2НФ (сами не являются ключами и не зависят от первичного ключа).

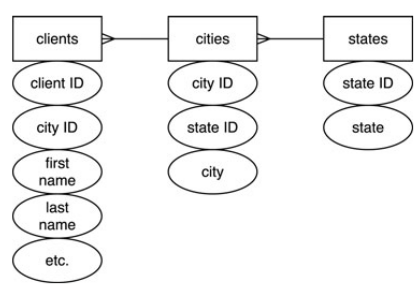
После всех преобразований, база данных с фильмами будет иметь минимум 4 таблицы.



Отношение находится в **третьей нормальной форме (3НФ):**

- если оно находится во второй нормальной форме, но каждый не ключевой атрибут зависит только от первичного ключа **id**  
- **значения столбцов** не зависят друг от друга.

- 3-я НФ нарушается,  если изменив значение в одном столбце, потребуется изменение и в другом столбце.

Например:  


Атрибуты **city ID, state ID, city** – не зависят друг от друга, а лишь от первичного ключа **cities.**

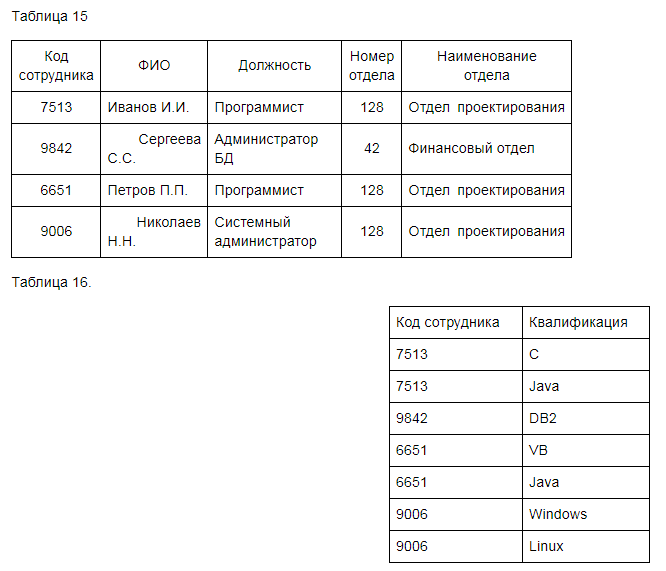
**Усиленная 3-я НФ или нормальная форма Бойса-Кодда (БКНФ)** - рассмотрим на примере ниже;

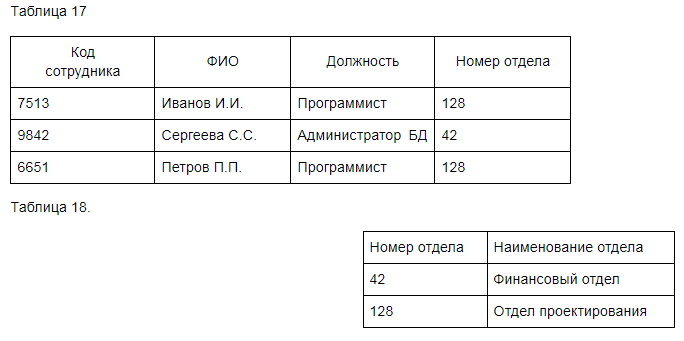
**Пример нормализации таблицы.**

**Своими словами главной задачей нормализации является привидение таблицы к такому виду, когда каждый атрибут таблицы (столбец) станет потенциальным ключом для других столбцов.**

****

В данной таблице есть несвязанные данные, которые можно разбить на составляющие и переместить их в новые таблицы. Например, между столбцами **Должность**, **Номер отдела и Квалификация** нет никакой связи, поэтому нет смысла хранить их в одной таблице.





Таким вот образом атрибуты **Код сотрудника** стал ключом для атрибута **Квалификация**, а **Номер отдела** - для **Наименования отдела**.

**Запомнить правила нормализации помогает изречение: «Нормализация – это ключ, целый ключ и ничего, кроме ключа».**

**Теория нормализации основана на наличии зависимостей между атрибутами отношения. Основными видами зависимостей являются:**

* функциональные;
* многозначные;
* транзитивные.

Базовым является понятие ***функциональной зависимости***, поскольку на его основе формируются определения всех остальных видов зависимостей. Атрибут В функционально зависит от атрибута А, если каждому значению А соответствует в точности одно значение В. Математически функциональную зависимость В от А обозначают А ’ В. Это означает, что во всех кортежах с одинаковым значением атрибута А атрибут В будет иметь также одно и то же значение. При этом А и В могут быть составными, то есть состоять из двух и более атрибутов.

**- Зависимость, при которой каждый не ключевой атрибут (значение столбца) зависит от всего составного ключа id и не зависит от его частей, называется *полной функциональной зависимостью.***

**- Зависимость, при которой атрибут А зависит от атрибута В, а атрибут В зависит от атрибута С (С ’ В ’ А), но обратная зависимость отсутствует, то зависимость А от С называется *транзитивной.***

***Многозначная зависимость – это когда,* один атрибут отношения многозначно определяет другой атрибут того же отношения, если для каждого значения первого атрибута существует множество соответствующих значений второго атрибута. Многозначные зависимости могут быть:**

* один-ко-многим (1:М);
* многие-к-одному (М:1);
* многие-ко-многим (М:М).

**Один ко многим (1:М)** – эта зависимость реализуется тогда, когда объекту А может принадлежать несколько объектов Б, но объекту Б – может принадлежать лишь один объект А.  
Например:

**Таблица 1. Связь с клиентом**

|  |  |  |  |
| --- | --- | --- | --- |
| Клиент | Мобильный телефон | Skype | e-mail |
| Сидоров | 0994563312 | sid1489 | sid\_1489$gmail.com |

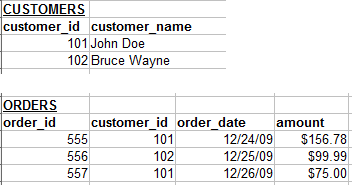
Данный клиент может иметь несколько способов для связи. Но данный мобильный номер, страница в скайпе и мыло может быть закреплен лишь к одному конкретному владельцу(клиенту).

Тоесть номер 099456312 или мыло [sid\_1489@gmail.com](mailto:sid_1489@gmail.com) не может быть у клиента Вася, Петя и тд.

**Еще пример:**  
Рассмотрим такой сайт интернет магазина:

* У покупателей может быть несколько заказов.
* Заказ может содержать несколько товаров.
* Товары могут иметь описание на нескольких языках.

В этих случаях нам потребуется создать связь "Один ко многим". Пример:

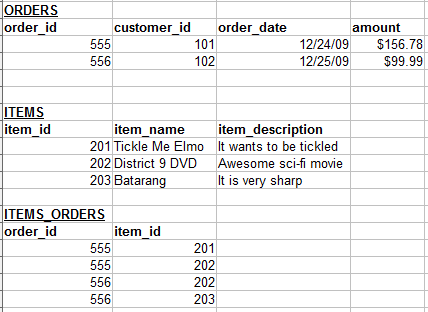


Каждый покупатель может иметь 0 или более заказов. Но каждый заказ может принадлежать только одному покупателю.

**Многие ко многим (1:М) –** эта зависимость реализуется тогда, когда объектам А может принадлежать несколько объектов Б, но и объектам Б – может принадлежать несколько объектов А.

Например, каждый заказ может содержать множество товаров. И каждый товар может присутствовать во многих заказах.

Для такой связи нам потребуется создать дополнительную таблицу:



Назначение таблицы **"Items\_Orders"** только одно - создать связь "Многие ко многим" между товарами и заказами.

Например:

|  |  |  |
| --- | --- | --- |
| Мобильные телефоны | Id Моб телефонов +  Id Производителей | Производители |
| 🡪  🡪 |  | 🡨  🡨 |

Мобильный телефон может иметь много разных производителей, но при этом у производителей может быть много разных моделей мобильных телефонов.

Для установления связи между этими двумя таблицами создается третья связующая таблица, в которую отправляем айдишники с первой таблицы и со второй.

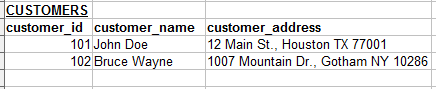
Еще пример: ученики в таблице учеников и по айдишнику связываются с кабинетом. Кабинеты в отдельной таблице... и т.д....Но если ученик может посещать разные кабинеты, то тут уже надо Таблица: Ученик к кабинетам.... а в одном кабинете может быть много учеников, то уже таблица строится с двумя колонками: Ученик\_id, Кабинет\_id. Запросом JOIN можешь вытянуть ученика какого - то и его кабинеты или кабинет и кто его посещает.  
Но, если взять сущность такую, что будет только у одного ученика. Например его ...трусы... то здесь будет соотношение 1 ко многим. т.к. никто не будет носить трусы другого ученика... Ну пример хуевый просто пример...

и тогда связующая таблица не нужна. а в таблице трусов будет просто **id хозяина** ( **id ученика**). таблица: трусы\_id, трусы\_name, хозяин\_id.

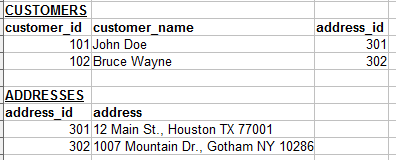
короче база должна строится по бизнес логике задачи.

### Связь один к одному (1:1) – эта зависимость реализуется тогда, когда объекту А может принадлежать лишь один объект Б.

Допустим, есть таблица покупателей (customers):



Мы можем расположить информацию о адресе покупателя в другой таблице:

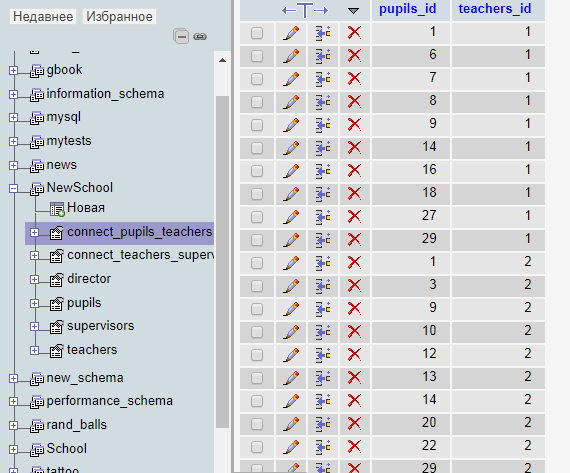


Теперь у нас есть связь между таблицами покупателей (Customers) и адресами (Addresses). Если каждый адрес может принадлежать только одному покупателю, то такая связь называется "Один к одному". Имейте ввиду, что такой тип отношений не очень распостранен. Наша первоначальная таблица, в которой информация о покупателе и его адресе хранилась вместе, в большинстве случаев работает нормально.

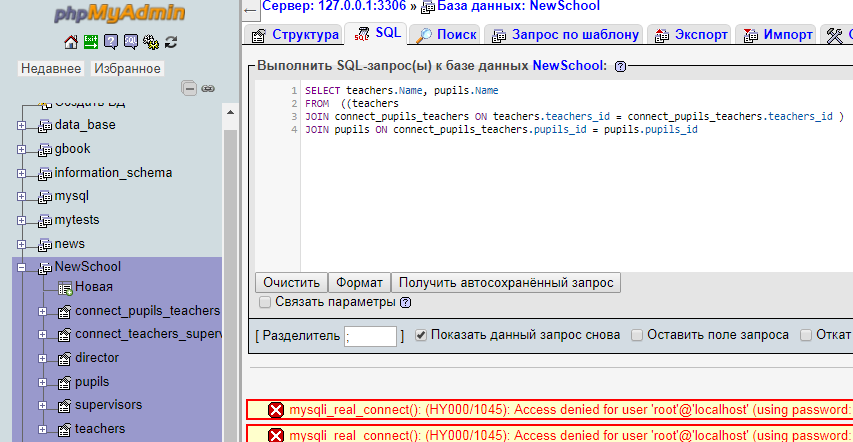
Обратите внимание, что теперь поле с названием "address\_id", в таблице покупателей, ссылается на соответствующую запись в таблице адресов. Оно называется внешним ключом (Foreign Key) и используется во всех видах связей в базе. Мы рассмотрим этот вопрос позже в этой статье.

**Соединение данных из двух таблиц в случае их нормализации, когда есть отношение многие ко многим.**

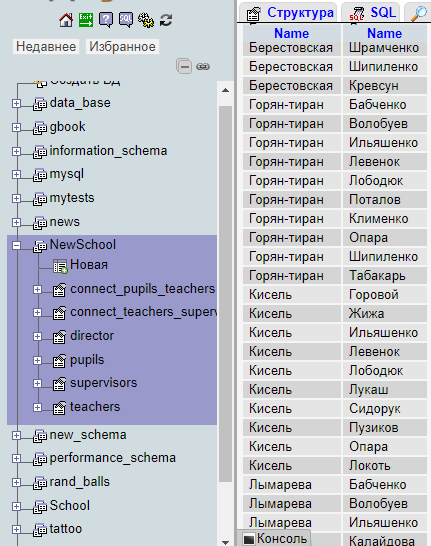
Например, мы создали связующую таблицу **connect\_pupils\_teachers**, которая содержит **id** других таблиц **pupils** и **teachers**.

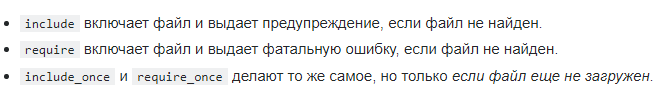


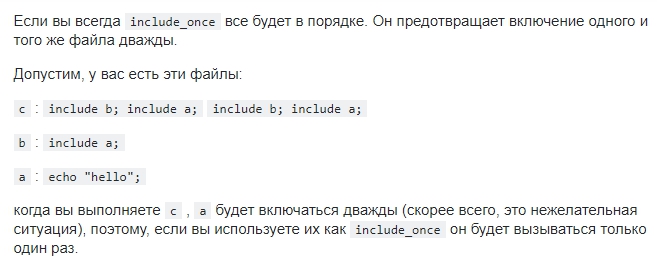
Чтобы сделать запрос на вывод данных из двух таблиц пишем в самом **PHP MyAdmin** в закладке **SQL**



Жмем кнопку Вперед и получаем выборку отношения Учителей к Ученикам. Тоесть, мы увидели, у каких учителей какие ученики. Слава Богу к тиранше Горян я не попал)лол



**Подключение других файлов:  
**

****

А разница между include\_once и require\_once заключается в том, что если функция include\_once получает какую-либо ошибку, она будет предупреждать вас и выполнять код, но функция require\_once обнаруживает любую ошибку, которая остановит сценарий и не выполнит код.

**Регистрация**

**Теория от меня**

Создали форму "Регистрации"---> Клиент ее заполнил и отправил нам на сервер методом ПОСТ ---> Мы проверили, что клиент ее заполнил правильно и если не возникло ошибок при заполнении, то заносим всю информацию в БД, вкючая хешированный пароль с солью ---> Во втором файле клиент уже заполняет форму "Авторизации" ---> После того, как он ее отправил нам на сервер, ме ее снова проверили, и , в случае првильного заполнения выводим из БД логин и пароль, которые проверяем на сопадение ---> Если данные совпали, то мы записываем клиента в Сессию в глобальный массив $\_SESSION. Но как только клиент покинул сайт, то сессия прпадает, и именно для того, чтобы сервер его запомнил, мы даем клиенту COOKIES, которая храниться у него(как я понял) и при повторном посещения нашего сайта мы ее видем

**Теория от Ивана**

1. проверил что всё ок и залогинился

2. сгенерил уникальный ключ после того как залогинился и засунул его или в базу ( возле юзера колонку токен создай ) или в сессию.

3. так же сгенеренный ключ держишь на стороен клиента в бразуере

4. по ключу с кукисом понимаешь кто именно залогинен ( если нпример в базе висит этот ключ)

или же просто с сессиии вытягиваешь ключ и сраниваешь его с ключем с кукисов

суть в том просто

что на стороне клиента ( браузер ) и на стороне сервера ( база или сессия ) висит токен ( ключ хеш набор случайных символов ). И если они идентичны то юзер авторизирован.

user\_id | name | surname | token\_auth | password| колонки

дальше $user = select \* from users where token = $\_COOKIES[token];

echo "Hello, ".$user["name"];

в общем сделай чтобы работало хоть как - то

такое грубо говоря делается в любом месте где тебе надо понять сейчас авторизирован человек или непонятно кто он.

или же как вариант чтобы каждый раз не дергать запрос в базу

можешь и самого пользователя уже записать в кукисы.

но в реальной жизни создается обьект контейнера и там потом где надо проверить кто сейчас будет по типу $this->container->getUserRole();

об этом уже скоро

**Создание гостевой книги:**Гостевая книга должна содержать следующие функицональные страницы:

1. Страница регистрации, содержащая форму для заполнение следующих полей:

- логин;

- пароль;

- подтверждение пароля;

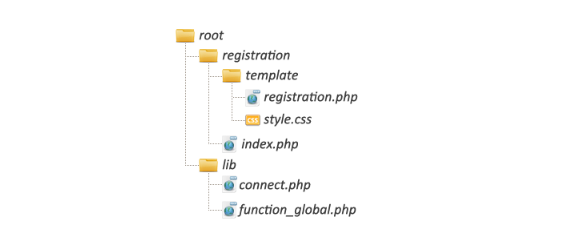
- e-mail

- согласие с правилами

2. Страница авторизации. На нее можно попасть лишь после прохождения регистрации

## Подготовка

Для начала, создадим необходимую иерархию папок и файлов. Выглядеть она должна так:



Сейчас мы будем работать исключительно с папкой **registration.** Внутри неё расположен подкаталог **template**, в котором будут находиться все файлы, отвечающие за внешний вид модуля Регистрации (html шаблоны, **css** файлы и различные изображения). В этой папке создаём файл **registration.php**. Он будет содержать саму форму регистрации. Вкратце, она должна выглядеть так:

**<form** method="post" action="index.php">

Логин: <**input** id="login" type="text" name="login" /><**br** />

Пароль: <**input** id="pass" type="password" name="password" /><**br** />

Подтверждение: <**input** id="re\_pass" type="password" name="password2" /><**br** />

Email: <**input** id="mail" type="text" name="mail" /><**br** />

<**label**><**input** id="no\_xyz" type="checkbox" name="lic" value="ok" /> Обязуюсь не творить хуйни!<**br** /></**label**><**br** />

<**input** type="submit" name="GO" value="Регистрация">

</**form**>

В этом блоке кода есть атрибут **id.** Он означает:

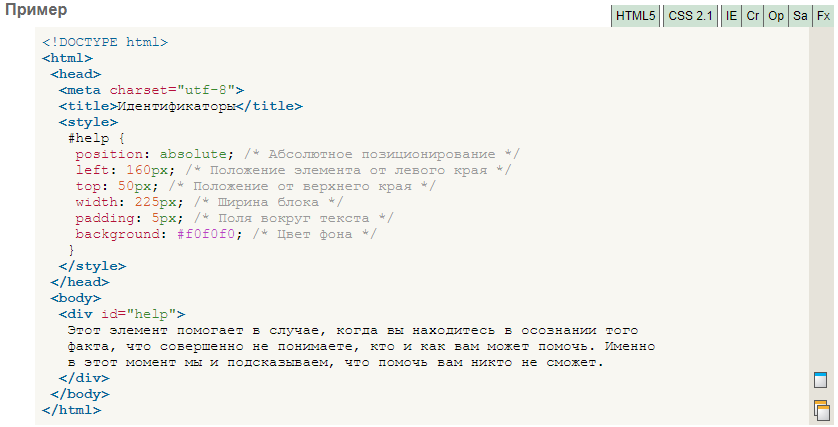
**Идентификатор (называемый также «ID селектор»)** определяет уникальное имя элемента, которое используется для изменения его стиля и обращения к нему через скрипты.

### Синтаксис

E#Имя\_идентификатора { Описание правил стиля }  
#Имя\_идентификатора { Описание правил стиля }

Здесь E — обозначает любой тег. Сначало идентификатор описуется внутри **<head></head>**. При описании идентификатора вначале указывается символ решётки (#), затем идёт имя идентификатора. Оно должно начинаться с латинского символа и может содержать в себе символ дефиса (-) и подчеркивания (\_). Использование русских букв в именах идентификатора недопустимо. В отличие от классов идентификаторы должны быть уникальны, иными словами, встречаться в коде документа только один раз.

Обращение к идентификатору происходит с помощью атрибута id, значением которого выступает имя идентификатора (id="Имя\_идентификатора"). Символ решётки при этом уже не указывается.



Вся форма передаётся методом POST, что необходимо для безопасной передачи данных. Кнопка отправки имеет name=”GO”, по которому мы и будем отлавливать отправку данных пользователем. В этом файле также присутствует некоторый php-код. Но об этом позже.

В каталоге registration создаём корневой файл index.php, который и будет запускаться при входе на страницу регистрации. Он является контроллером, обрабатывающим действия пользователя и решающий, как будет выглядеть страница (в этом файле будет подключаться шаблон из подкаталога template, в зависимости от различных ситуаций). Смысл такого разделения подробно расписан в статье про [MVC](http://true-coder.ru/php/razdelenie-koda-i-shablona-mvc-chast-1.html). Сейчас стоит отметить, что в файле index.php не должно быть никакого вывода, то есть в коде не должно быть ни строки html-кода и ни одного оператора echo или print.

Все данные о зарегистрированных пользователях мы будем хранить в базе данных MySQL. Для этого нам необходимо создать, например, через phpMyAdmin базу данных (БД), а затем таблицу с необходимыми полями. В данной статье, для примера, я буду использовать название БД следующее: *db\_name*. Название таблицы – users. Далее добавим в таблицу следующие поля:

* **id** (при добавлении записи в таблицу данное поле у новой записи будет проставляться автоматически, увеличиваясь на единицу по сравнению с предыдущей добавленной записью; для этого необходимо поствить галочку в поле AI (AUTO\_INCREMENT)). Тип данных – int.
* **login** – собственно, сам логин юзера, varchar(25).
* **password** – хэшированный пароль (об этом позже), varchar(32).
* **salt** – «соль», используемая для «примеси» к паролю, varchar(3).
* **mail\_reg** – регистрационный e-mail, varchar(50).
* **mail** – email, которые позже можно будет изменить в профиле пользователя, varchar(50).
* **last\_act** – время последней активности пользователя, int(11).
* **reg\_date** – дата регистрации, int(11).

Для удобства работы в будущем создадим в корне сайта папку *lib*, в которую поместим 2 файла: ***connect.php*** и ***function\_global.php*.** В дальнейшем в этой папке будут храниться библиотеки различных функций.

Первый файл придётся подключать очень часто в будущем (он отвечает за подключение к БД), а во втором будут храниться часто используемые функции на сайте.

Как только таблица и иерархия файлов созданы, можно приступить к написанию самого функционала.

## ****Реализация****

Код файла **connect.php:**

|  |  |
| --- | --- |
| 1  2  3  4  5 | **<?php**  mysql\_connect("localhost", "login", "password")  or die ("Ошибка подключения к базе данных");  mysql\_select\_db("db\_name");  **?>** |

Файл отвечает за подключение и выбор БД, с которой будем работать. Вписываем в этот файл свои данные. Файл *index.php* выглядит следующим образом:

<?php  
ini\_set ("session.use\_trans\_sid", true);  
session\_start();  
include ('../lib/connect.php'); *//подключаемся к БД*include ('../lib/function\_global.php'); *//подключаем библиотеку функций  
  
//проверим, быть может пользователь уже авторизирован. Если это так, перенаправим его на главную страницу сайта*if (isset($\_SESSION['id']) || (isset($\_COOKIE['login']) && isset($\_COOKIE['password'])))  
{  
 header('Location: http://ваш\_сайт/');  
}  
else  
{  
 if (isset($\_POST['GO'])) *//если была нажата кнопка регистрации, проверим данные на корректность и, если данные введены и введены правильно, добавим запись с новым пользователем в БД* {  
 $correct = registrationCorrect(); *//записываем в переменную результат работы функции registrationCorrect(), которая возвращает true, если введённые данные верны и false в противном случае* if ($correct) *//если данные верны, запишем их в базу данных* {  
 $login = htmlspecialchars($\_POST['login']);  
 $password = $\_POST['password'];  
 $mail = htmlspecialchars($\_POST['mail']);  
 $salt = mt\_rand(100, 999);  
 $tm = time();  
 $password = md5(md5($password).$salt);  
 if (mysql\_query("INSERT INTO users (login,password,salt,mail\_reg,mail,reg\_date,last\_act) VALUES ('".$login."','".$password."','".$salt."','".$mail."','".$mail."','".$tm."','".$tm."')")) *//пишем данные в БД и авторизовываем пользователя* {  
 setcookie ("login", $login, time() + 50000, '/');  
 setcookie ("password", md5($login.$password), time() + 50000, '/');  
 $rez = mysql\_query("SELECT \* FROM users WHERE login=".$login);  
 @$row = mysql\_fetch\_assoc($rez);  
 $\_SESSION['id'] = $row['id'];  
 $regged = true;  
 include ("template/registration.php"); *//подключаем шаблон* }  
 }  
 else  
 {  
 include\_once ("template/registration.php"); *//подключаем шаблон в случае некорректности данных* }  
 }  
 else  
 {  
 include\_once ("template/registration.php"); *//подключаем шаблон в случае если кнопка регистрации нажата не была, то есть, пользователь только перешёл на страницу регистрации* }  
}  
?>

Теперь разберём весь код чуть подробнее.

Вначале мы подключили два файла:

|  |  |
| --- | --- |
| 1  2 | include ('../lib/connect.php');  include ('../lib/function\_global.php'); |

О них говорилось выше. Во втором из них находится функция, которую мы рассмотрим чуть позже.

В следующих строках мы проверяем, не авторизирован ли пользователь:

|  |  |
| --- | --- |
| 1  2  3  4 | if (isset($\_SESSION['id']) || (isset($\_COOKIE['login']) && isset($\_COOKIE['password'])))  {  header('Location: [http://ваш\_сайт/](http://xn--_-7sbbf2b7bj7b/)');  } |

Дело в том, что в будущем, когда я буду рассказывать об авторизации на сайте, авторизация будет происходить следующим образом: в сессию будет писаться ваш id (уникальный для любого пользователя номер), а также будет создаваться два cookie, в одном будет храниться логин, во втором – пароль.

В данных строках, мы проверяем, есть ли у данного пользователя переменные **$\_SESSION[‘id’], $\_COOKIE[‘login’] и $\_COOKIE[‘password’]**. Если такие переменные есть, то перенаправим пользователя на главную страницу сайта:

|  |
| --- |
| header('Location: [http://ваш\_сайт/](http://xn--_-7sbbf2b7bj7b/)'); |

Если же таких данных не оказалось, то идём дальше, где проверяем, была ли нажата кнопка с *name=”GO”*. Если это так, проверяем данные и, если всё хорошо, пишем их в БД. В противном случае, подключаем шаблон с формой:

|  |  |
| --- | --- |
| 1  2  3  4 | else  {  include\_once ("template/registration.php");  } |

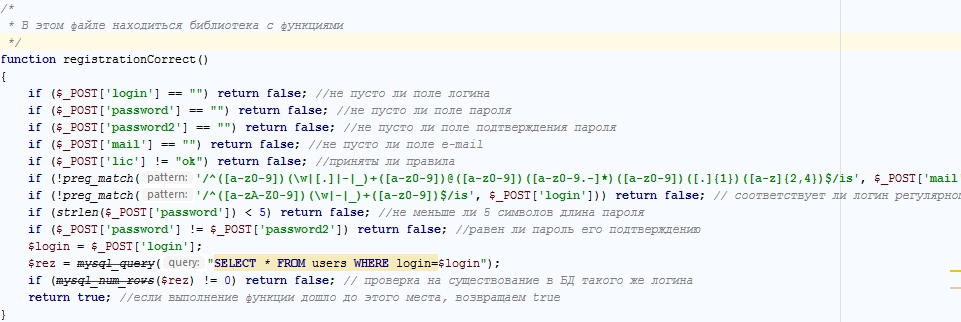
Если же кнопка была нажата, в первую очередь необходимо проверить введённые пользователем данные на корректность. Делается это строкой:

|  |
| --- |
| $correct = registrationCorrect(); |

Наша функция **registrationCorrect** в себе содержит прописанный нами код в подключенном ранее файле **function\_global.php** , который проверяет данные пользователя на корректность заполнения.

**include ('../lib/function\_global.php');**

В файле **function\_global.php** мы заранее написали следующую функцию:



При первом же несоответствии функция вернёт **false**и прекратит свою работу (данные некорректны), если же все данные качественны, выполнение функции дойдёт до конца и вернёт **true (**данные корректны).

Продолжим дальше разбирать файл *index.php*.

|  |  |
| --- | --- |
| 1  2  3  4 | if ($correct)  {  *//этот блок выполнится, если функция registrationCorrect вернула true. В противном случае будет опять подключён шаблон с формой*  } |

Внутри этого блока расположен следующий код, в нём данные непосредственно подготавливаются для записи в БД:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | $login = htmlspecialchars($\_POST['login']);  $password = $\_POST['password'];  $mail = htmlspecialchars($\_POST['mail']);  $salt = mt\_rand(100, 999);  $tm = time();  $password = md5(md5($password).$salt); |

В переменную **$salt** запишем случайное трёхзначное число. Оно нам пригодится для «примешивания» к паролю. Дело в том, что во избежание кражи паролей, они в открытом виде в БД не хранятся. Там хранятся лишь их хэши – 32-символьные строки, уникальные для определённой строки. В php существует функция md5(), которая преобразует любую переданную ей строку, в строку, состоящую из 32 символов. Например, если выполнить следующий код:

**<?** echo md5(“true-coder”); **?>**

, то на экран выведется строка: cfcd208495d565ef66e7dff9f98764da

Еще пример:

$a = "ultras86";  
  
echo md5 ($a);

Выведет: b26ae268bb2fa3ffc52ab203ffefbb08

Понятия «расхэшировать» в природе не существует. Вы можете благополучно захэшировать огромную статью и на выходе получить те же 32 символа. **Хэширование – не сжатие, а получение уникальной контрольной суммы** для данной строки. Народные умельцы давно уже составили большие базы хэшей различных строк, но какими бы большими они ни были, если ваш пароль сколь либо сложен, узнать его будет всё также тяжело. Так вот, в БД не хранятся пароли в открытом виде, они хранятся в виде хэш-кода. А когда вы авторизируетесь на сайте, введённый вами пароль приводится скриптом к хэшу и сравнивается с хэшем в БД. Также поступим и мы, но ещё немного мудрее. Есть такое понятие как salt, так называемая «соль», которая «приклеивается» к паролю, а лишь потом применяется функция md5. Эту соль мы и генерируем ($salt = mt\_rand(100, 999);). Её мы также запишем в БД.

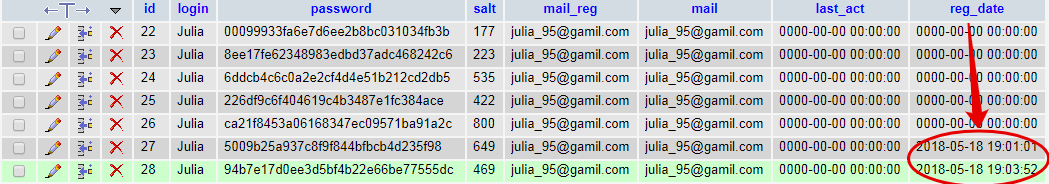
В следующей строке мы создаём хэш пароля:

|  |  |
| --- | --- |
| 1 | $password = md5(md5($password).$salt); |

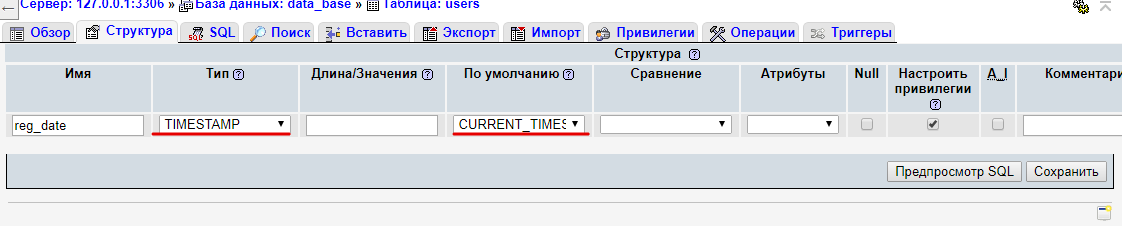
Поясню… Вначале мы применяем к паролю функцию md5, результат – 32-символьная строка. Потом «лепим» справа к этой строке ещё 3 случайных цифры (наша «соль») и применяем опять md5. Результат – опять 32-символьная строка. Смысл этих манипуляций состоит в том, что мы сильно усложняем жизнь злоумышленнику. И ему остаются лишь подбирать брутфорсом ваш пароль, что займёт у него, возможно, многие годы

|  |  |
| --- | --- |
| 1 | $tm = time(); *//в эту переменную пишем время регистрации, необходимое для записи в БД. Также это время будет использовано для сохранения в БД времени последней активности пользователя.* |

Далее идёт собственно запись данных в БД. Если она будет удачной, то скрипт авторизирует пользователя, записав его id в сессию и создав нужные куки. Следует обратить внимание на то, как формируется пароль в куках. Это тот же md5, но применённый по другому принципу. Это нам пригодится при авторизации пользователя в будущем.

Чтобы установилось время регистрации и дата (**reg\_date**) 

можно просто установить опции в самой БД (тип TIMESTAMP и CURRENT\_TIMES):

  
И после перезагрузки формы все последующие данные будут заноситься с текущим временем.

if (mysql\_query("INSERT INTO users (login,password,salt,mail\_reg,mail,reg\_date,last\_act) VALUES ('".$login."','".$password."','".$salt."','".$mail."','".$mail."','".$tm."','".$tm."')")) *//пишем данные в БД и авторизовываем пользователя*

{

setcookie ("login", $login, time() + 50000, '/');

setcookie ("password", md5($login.$password), time() + 50000, '/');

$rez = mysql\_query("SELECT \* FROM users WHERE login=".$login);

@$row = mysql\_fetch\_assoc($rez);

$\_SESSION['id'] = $row['id'];

$regged = **true**;

include ("template/registration.php"); *//подключаем шаблон*

}

После этого у пользователя у появится два cookie: **$\_COOKIE[‘login’] и $\_COOKIE[‘password’]**, а также переменная сессии — **$\_SESSION[‘id’]**, равная id пользователя, который мы узнаём с помощью запроса к БД.

Переменная **$regged** нужна нам в качестве флага, говорящего, что регистрация прошла успешно. Дело в том, что внутри подключаемого файла include («template/registration.php»); присутствует условие, которое, при *$regged == true* выведет текст, что регистрация прошла успешно и не выведет форму регистрации.

**Авторизация**

Для начала необходимо сверстать главную страницу сайта и поместить её в корне сайта в папку **template**. Для данного урока нам достаточно, чтобы в этом файле была форма ввода логина и пароля, а также кнопка «Вход». Далее приведён код этой формы:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <**form** action="/" method="post">  Логин: <**input** type="text" name="login" />  Пароль: <**input** type="password" name="password" />  <**input** type="submit" value="Войти" name="log\_in" />  </**form**> |

Файл назовём index.html.

Метод передачи **post**необходим. Ведь мы не хотим, чтобы при авторизации логин и пароль светились в адресной строке.

Как только форма готова, создадим самый важный файл будущего сайта — главный контроллер, т. е. файл, лежащий в корне сайта — index.php. Именно он и будет запускаться при входе на сайт. На момент написания статьи на нашем проекте код этого файла занимает 92 строки, нам же понадобится пока лишь около 25 строк. Вот его код:

**Cookie**

Мы устанавливаем в Cookie переменную, которая будет храниться на диске удаленного пользователя. Эта переменная и будет хранить информацию о посещениях. Она будет считываться скриптом при обращении посетителя к серверу. Выгода такого метода идентификации очевидна. Во-первых, нам не нужно хранить множество ненужной информации о IP-адресах. Во-вторых, нас не интересуют динамические IP-адреса, поскольку данные о своих посещениях хранятся конкретно у каждого посетителя сайта.

Теперь понятно, для чего мы можем использовать Cookie - 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.

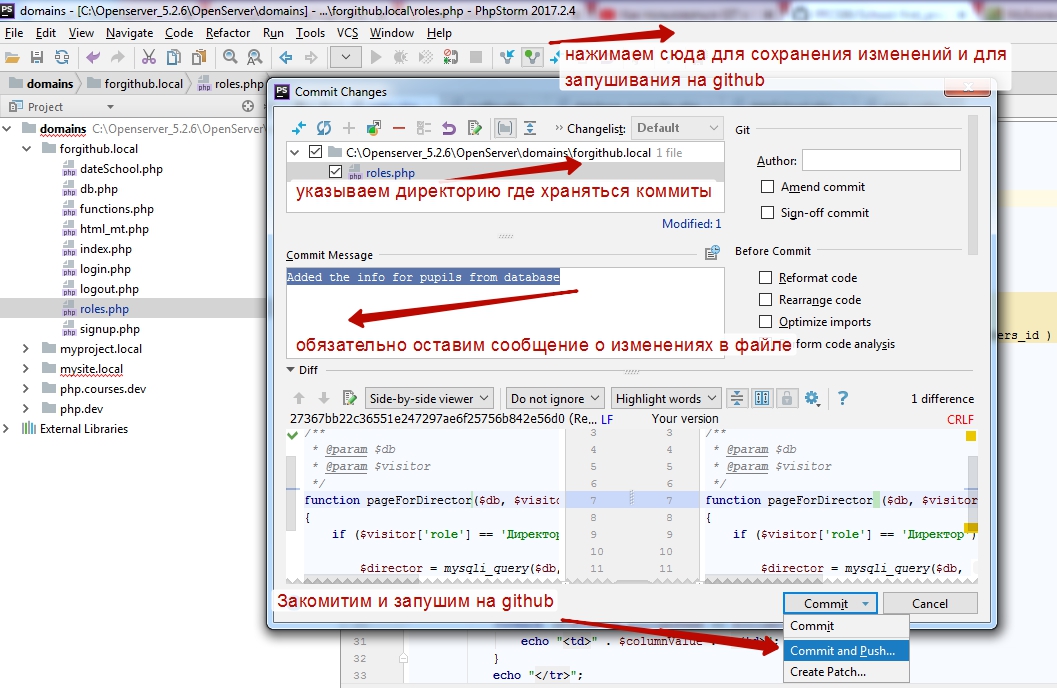
Файлы Cookies представляют собой обыкновенные текстовые файлы, которые хранятся на диске у посетителей сайтов. Файлы Cookies и содержат ту информацию, которая была в них записана сервером.

Кукисы должны быть установленны ПЕРЕД html файлом или командой echo.

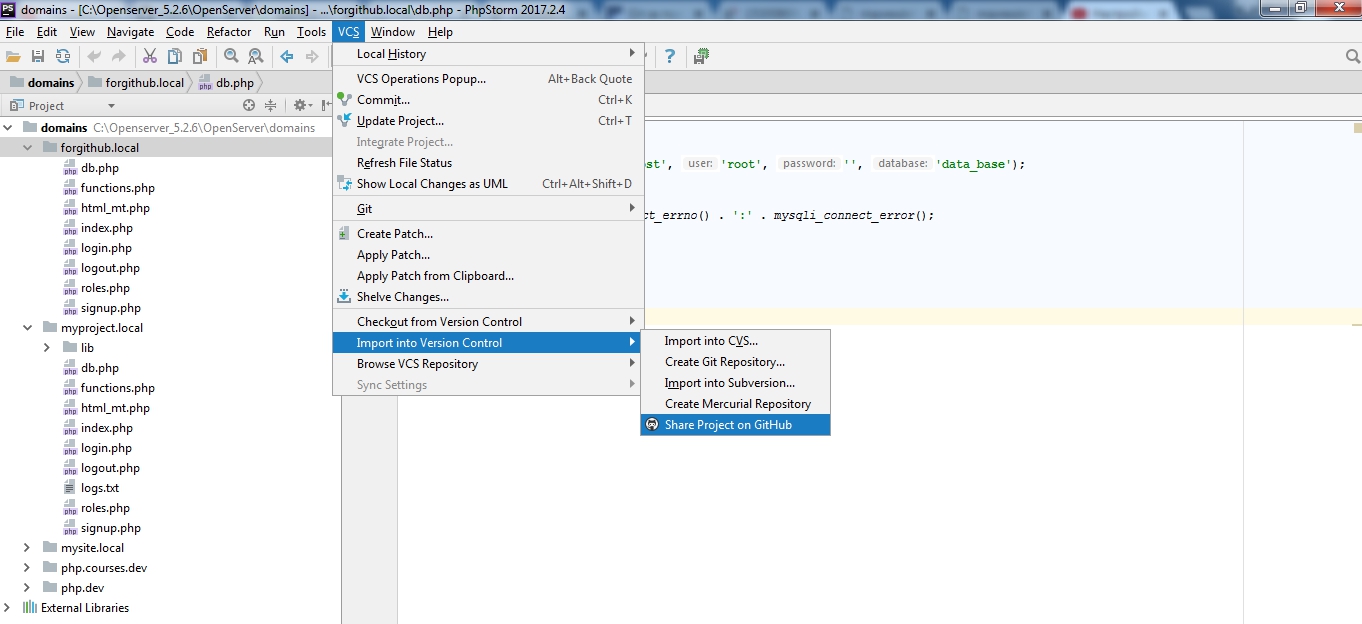
**Работа с Github**

**Отправить проэкт на GitHub:**

**Видоурок:** [**https://www.youtube.com/watch?v=iQqDce\_9y3k**](https://www.youtube.com/watch?v=iQqDce_9y3k)

****

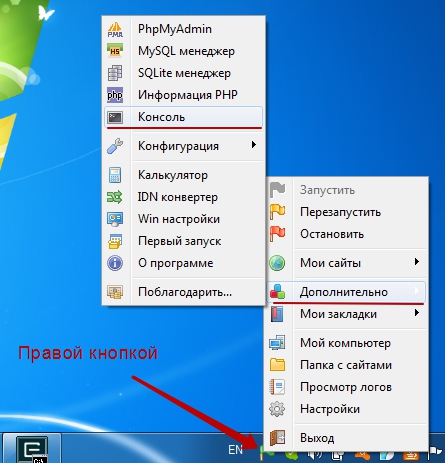
Создаем новый репозиторий на Github.com, привязанный к нашему проэкту. И отправляем его с помощью кнопки VCS -> Import into Version Control -> Share Project on GitHub

****

**Работа с командной строкой:**

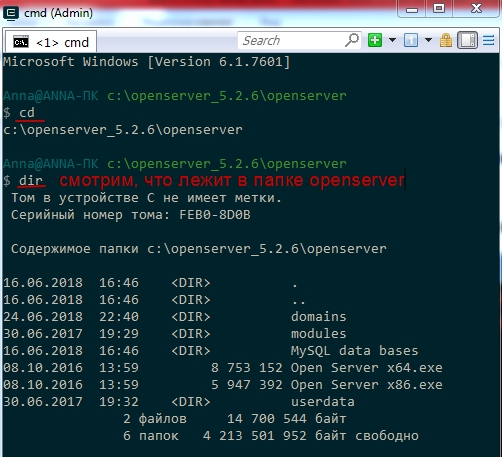
**Чтобы запустить командную строку** – в панеле Пуск прописываем cmd.exe

или открываем командную строку:

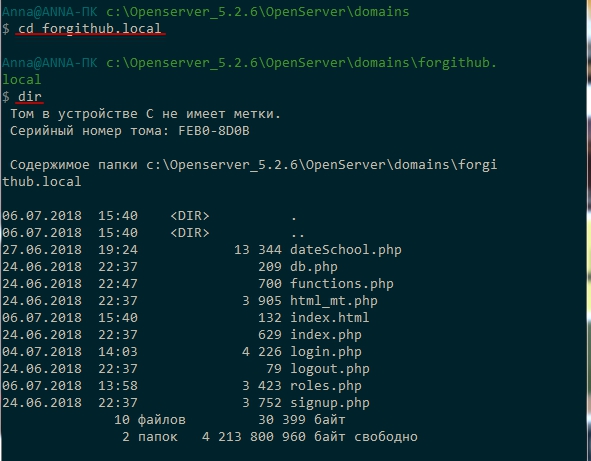


**Чтобы открыть папку** – набираем команду **cd**

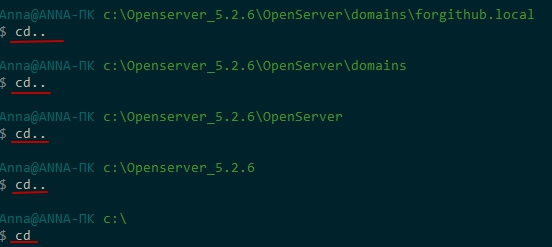
**Чтобы просмотреть содержимое папки** – набираем команду **dir(ls в линуксе)**

****

**Нам нужно добраться до папки с нашим проэктом School, поэтому открываем столько папок, сколь нам необходимо:**

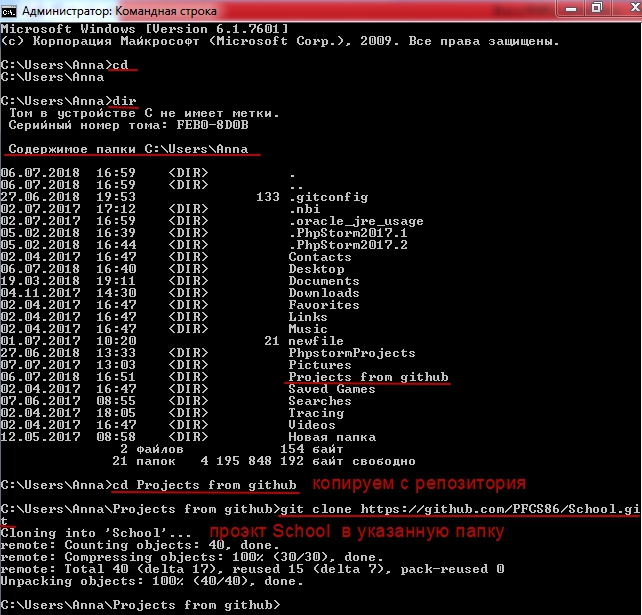
****

**Чтобы вернуться назад в предидущую папку -** набираем команду **cd..**

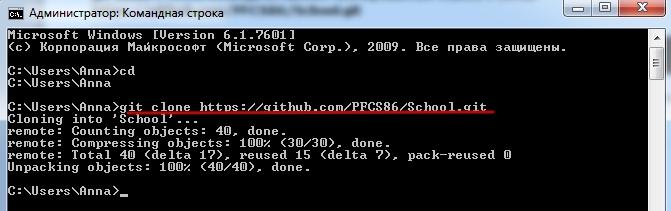


**Для скачивания других проэктов репозитория я создал папку, куда будут храниться чужие проэкты: C:\users\Anna\Projects from github**

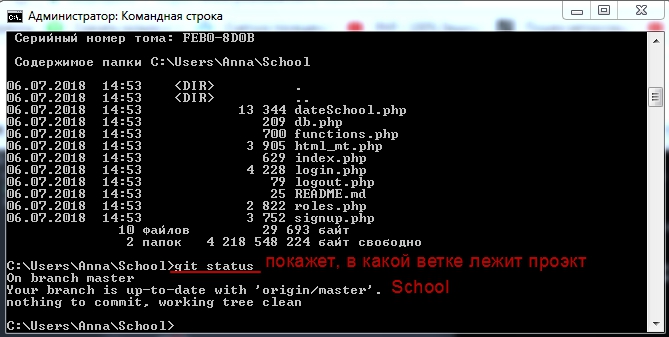
**Чтобы** **сделать клонирование репозитория с нашим проэктом на комп** прописуем в командной строке:  
**git clone** и после команды вставляем скопированную ссылку с Github **https://github.com/PFCS86/School.git**





****

**Чтобы** **просмотреть состояние репозитория или с какой ветки репозитория был скачан проэкт**  прописуем в в командной строке: **git status**

****

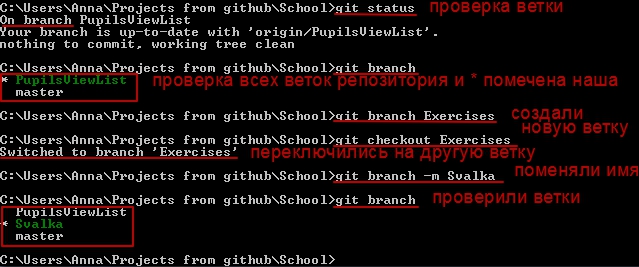
**Еще один видеоурок по созданию, переименованию и удалению ветки:** **https://www.youtube.com/watch?v=r9O-zYjzVmk&t=50s**

**Чтобы посмотреть какие ветки существуют в репозитории и на какой мы находимся: git branch**

**Чтобы создать новую ветку в локальном репозитории: git branch Exercises (где Exercises – название новой ветки).**

**Чтобы переключиться на новую ветку в репозитории: git checkout Exercises**

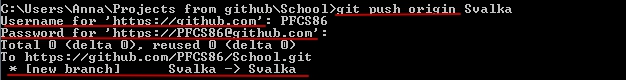
**Чтобы переименовать название ветки в репозитории: git branch –m Svalka**

**Проверим: git branch**

**Чтобы создать новую ветку и сразу же на нее переключиться: git checkout –b Exercises**

**Чтобы удалить ветку: git branch –d Exercises**

**Чтобы опубликовать новую ветку на репозиторий на github: git push origin Svalka**

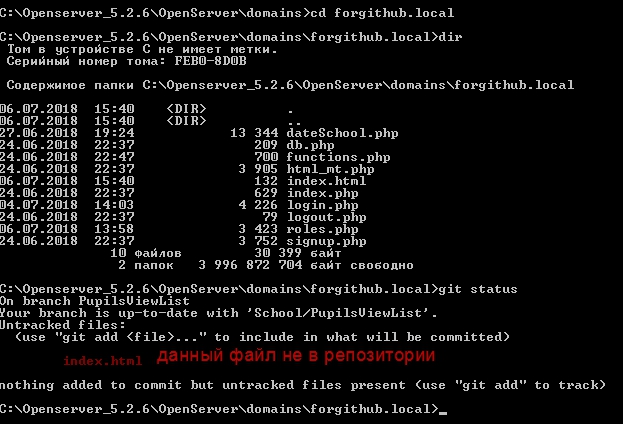
****

**Проверяем результат на нашем профайле на github видем, что ветка Svalka успешно запушена:**

****

**После этого ввести логин PFCS86 и пароль Supporter86**

**Если мы создали новый файл (например, index.html) на нашем компе, то команда git status покажет,** что этого файла нет в нашем репозитории.

****

**Чтобы** **добавить новый файл в проэкт на репозиторий (запушить)**  прописуем в командной строке следующий ряд команд:

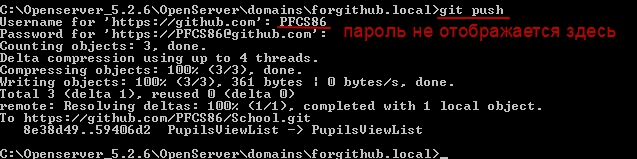
**git add index.html**, где index.html - название файла, который мы хотим добавить.

**git commit –m “Add index.html”** , где в Add index.html – комментарий к добавляемому файлу. Таким образом мы коммитим файл и потом уже из буфера пушим на github

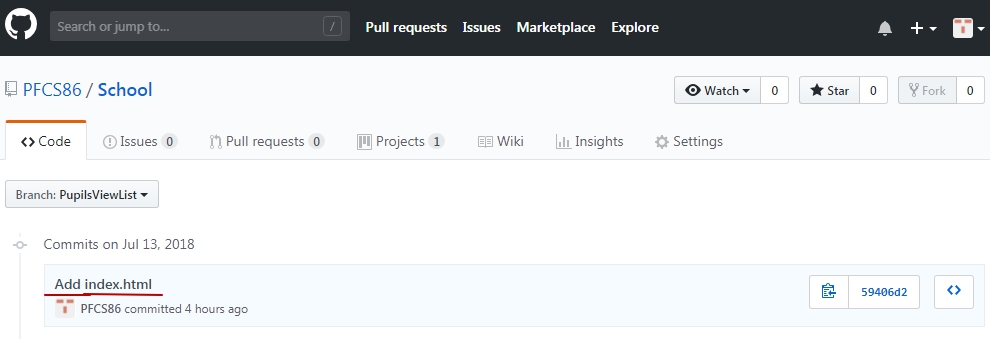
**git push**

После этого необходимо ввести логин на сатйе github.com: **PFCS86**

и пароль вслепую, так как в консоле он не отображается: **Supporter86**

****

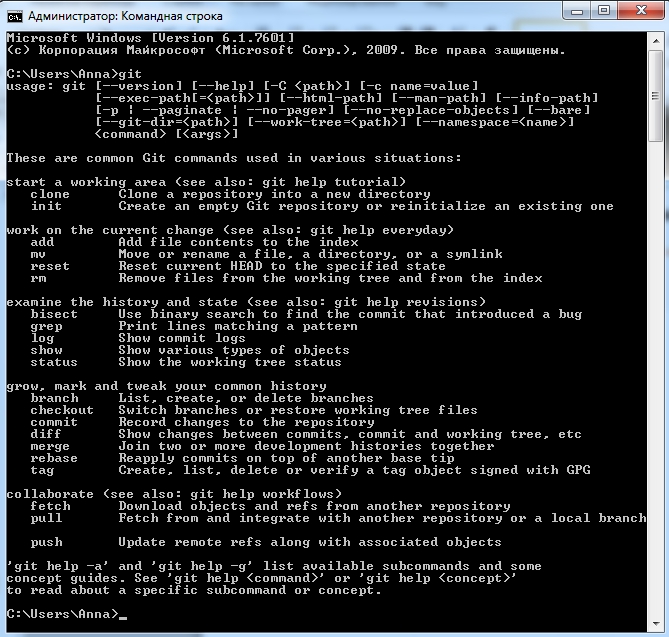
**Проверяем отправки нашего файла на репозиторий. Вуаля, файл запушен успешно!!!**

****

**Чтобы удалить файл github: git rm index.html**

**git commit –m “Удаляем этот уебанский файл**”, где в скобках пишем наш комментарий. Например, чтобы было понятно другим дивам нахуя я удалил это уебанский файл.

**git push** – отправляем коммит на github. В данном случае даем git push удалит файл index.html с репозитория и отправит комментирование.

**Чтобы прочитать о функциях всех команд, которые можно прописать в адресной строке для git –** набираем команду **git**

Больше команд для работ в консоле в Windows и Linux:  
https://gist.github.com/codedokode/10539568

**Проверка кода:**

**echo "<pre>";**

**print\_r($**переменная**);**

**echo "</pre>";**

**die;**

- убивает код после него