Набережночелнинский институт (филиал)
федерального государственного автономного
образовательного учреждения высшего образования
«Казанский (Приволжский) федеральный университет»

Кафедра «Бизнес-информатики и
математических методов в экономике»

Д. М. ЛЫСАНОВ

ЯЗЫК РАЗРАБОТКИ СЦЕНАРИЕВ PHP И СУБД MYSQL

Электронный образовательный ресурс по дисциплине
«Web-программирование»

Набережные Челны
2018

Печатается по решению кафедры «Бизнес-информатики и математических методов в экономике».

Рецензенты:

кандидат педагогических наук, доцент каф. БИММЭ Еремина И.И.,

заместитель директора по развитию ООО «Глобал ИТ Центр» Фрикк В.С.

Язык разработки сценариев PHP и СУБД MySQL: электронный образовательный ресурс по дисциплине «Web-программирование» / Д.М. Лысанов. - Набережные Челны: Изд-полигр. центр НЧИ КФУ, 2018. - 80 с.

В ЭОР рассмотрены основы языка разработки сценариев PHP: синтаксис, типы данных, работа с перменными, константами, операторами, использование управляющих конструкций, функций. Приводятся особенности работы с СУБД MySQL при создании таблиц, добавлении, обновлении и удалении данных, организации связей между таблицами, отборе, группировке и сортировке записей, описываются основные функции PHP для работы с MySQL.

Предназначено для студентов очной и заочной форм обучения направлений подготовки 09.03.03 «Прикладная информатика», 38.03.05 «Бизнес-информатика».

Содержание

I. Язык разработки сценариев PHP

1. Основы синтаксиса

1.1. Теги PHP

Когда PHP обрабатывает файл, он ищет открывающие и закрывающие теги, такие как <?php и ?>, которые указывают PHP, когда начинать и заканчивать обработку кода между ними. Подобный способ обработки позволяет PHP внедряться во все виды различных документов, так как всё, что находится вне пары открывающих и закрывающих тегов, будет проигнорировано парсером PHP.

Если файл содержит только код PHP, предпочтительно опустить закрывающий тег в конце файла. Это помогает избежать добавления случайных символов пробела или перевода строки после закрывающего тега PHP, которые могут послужить причиной нежелательных эффектов.

```
<?php
echo "Hello world"; // ... еще код
echo "Последнее выражение"; // Скрипт заканчивается тут без закрывающего тега PHP
```

1.2. Изолирование от HTML

Все, что находится вне пары открывающегося и закрывающегося тегов, игнорируется интерпретатором PHP, у которого есть возможность обрабатывать файлы со смешанным содержимым. Это позволяет PHP-коду быть встроенным в документы HTML, к примеру, для создания шаблонов.

```
<p>Это будет проигнорировано PHP и отображено браузером.</p>
<?php echo 'А это будет обработано.'; ?>
<p>Это тоже будет проигнорировано PHP и отображено браузером.</p>
```

1.3. Разделение инструкций

Как в C или Perl, PHP требует окончания инструкций точкой запятой в конце каждой инструкции. Закрывающий тег блока PHP-кода автоматически применяет точку с запятой; т.е. нет необходимости ставить точку с запятой в конце последней строки блока с PHP-кодом.

```
<?php
    echo 'Это тест';
?>

<?php echo 'Это тест' ?>

<?php echo 'Мы опустили последний закрывающий тег';
```

1.4. Комментарии

PHP поддерживает комментарии в стиле 'C', 'C++' и оболочки Unix (стиль Perl).

```
<?php
    echo "Это тест"; // Это однострочный комментарий в стиле c++
    /* Это многострочный комментарий
       еще одна строка комментария */
    echo "Это еще один тест";
    echo "Последний тест"; # Это комментарий в стиле оболочки Unix
?>
```

Однострочные комментарии идут только до конца строки или текущего блока PHP-кода, в зависимости от того, что идет перед ними. Это означает, что HTML-код после // ... ?> или # ... ?> БУДЕТ напечатан: ?> завершает режим PHP и возвращает режим HTML, а // или # не могут повлиять на это.

```
<h1>Это <?php # echo "простой";?> пример</h1>
<p>Заголовок вверху выведет 'Это пример'.</p>
```

2. Типы данных

PHP поддерживает восемь простых типов. Четыре скалярных типа: boolean, integer, float, string. Два смешанных типа: array, object. И, наконец, два специальных типа: resource, NULL

Если вы желаете проверить тип и значение определённого выражения, используйте var_dump(). Если же вам для отладки необходимо просто удобочитаемое представление типа, используйте gettype(). Чтобы проверить на определенный тип, применяйте is_type функции.

```
<?php
$a_bool = TRUE;   // логический
$a_str  = "foo";  // строковый
$a_str2 = 'foo';  // строковый
$an_int = 12;     // целочисленный

echo gettype($a_bool); // выводит: boolean
echo gettype($a_str);  // выводит: string

// Если это целое, увеличить на четыре
if (is_int($an_int)) {
    $an_int += 4;
}

// Если $a_bool - это строка, вывести ее
// (ничего не выводит)
if (is_string($a_bool)) {
    echo "Строка: $a_bool";
}
?>
```

2.1. Булев тип

Это простейший тип. boolean выражает истинность значения. Он может быть либо TRUE либо FALSE. Для указания boolean, используйте ключевое слово TRUE или FALSE. Оба регистронезависимы.

```
<?php
$foo = True; // присвоить $foo значение TRUE
?>
```

2.2. Целые числа

Integer - это число из множества Z = {..., -2, -1, 0, 1, 2, ...}.

Целые числа могут быть указаны в десятичной (основание 10), шестнадцатеричной (основание 16), восьмеричной (основание 8) или двоичной (основание 2) системе счисления, с необязательным предшествующим знаком (- или +).

Для записи в восьмеричной системе счисления, необходимо поставить пред числом *0* (ноль). Для записи в шестнадцатеричной системе счисления, необходимо поставить перед числом *0x*. Для записи в двоичной системе счисления, необходимо поставить перед числом *0b*

```
<?php
$a = 1234; // десятичное число
$a = -123; // отрицательное число
$a = 0123; // восьмеричное число (эквивалентно 83 в десятичной системе)
$a = 0x1A; // шестнадцатеричное число (эквивалентно 26 в десятичной системе)
?>
```

Формально, структуру целых чисел можно записать так:

```
десятичные         : [1-9][0-9]*
                   | 0
шестнадцатеричные  : 0[xX][0-9a-fA-F]+
восьмеричные       : 0[0-7]+
двоичные           : 0b[01]+
целые              : [+-]?десятичные
                   | [+-]?шестнадцатеричные
                   | [+-]?восьмеричные
                   | [+-]?двоичные
```

Для явного преобразования в integer, используйте приведение *(int)* или *(integer)*. Однако, в большинстве случаев, в приведении типа нет необходимости, так как значение будет автоматически преобразовано, если оператор, функция или управляющая структура требует аргумент типа integer. Значение также может быть преобразовано в integer с помощью функции intval().

2.3. Числа с плавающей точкой

Числа с плавающей точкой (также известные как "float", "double", или "real") могут быть определены следующими синтаксисами:

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
?>
```

Формально:

```
LNUM          [0-9]+
DNUM          ([0-9]*[\.]{LNUM}) | ({LNUM}[\.][0-9]*)
EXPONENT_DNUM [+-]?(({LNUM} | {DNUM}) [eE][+-]? {LNUM})
```

Для сравнения чисел с плавающей точкой используется верхняя граница относительной ошибки при округлении. Эта величина называется машинной эпсилон или единица округления(unit roundoff) и представляет собой самую маленькую допустимую разницу при расчетах.

$a и *$b* равны до 5-ти знаков после запятой.

```
<?php
$a = 1.23456789;
$b = 1.23456780;
$epsilon = 0.00001;
if(abs($a-$b) < $epsilon) {
    echo "true";
}
?>
```

Некоторые числовые операции могут возвращать значение, представляемое константой NAN. Данный результат означает неопределенное или непредставимое значение в операциях с плавающей точкой. Любое строгое или нестрогое сравнение данного значения с другим значением, включая его самого, возвратит FALSE.

Так как NAN представляет собой неограниченное количество различных значений, то NAN не следует сравнивать с другими значениями, включая ее саму. Вместо этого, для определения ее наличия необходимо использовать функцию is_nan().

2.4. Строки

Строка - это набор символов, где символ - это то же самое, что и байт. Это значит, что PHP поддерживает ровно 256 различных символов, а также то, что в PHP нет встроенной поддержки Unicode.

Строка может быть определена четырьмя различными способами:

- одинарными кавычками
- двойными кавычками
- heredoc-синтаксисом
- nowdoc-синтаксисом

Одинарные кавычки

Простейший способ определить строку - это заключить ее в одинарные кавычки (символ ').

Чтобы использовать одинарную кавычку внутри строки, проэкранируйте ее обратной косой чертой (\). Если необходимо написать саму обратную косую черту, продублируйте ее (\\). Все остальные случаи применения обратной косой черты будут интерпретированы как обычные символы: это означает, что если вы попытаетесь использовать другие управляющие

последовательности, такие как \r или \n, они будут выведены как есть вместо какого-либо особого поведения.

```
<?php
echo 'это простая строка';

echo 'Также вы можете вставлять в строки
символ новой строки вот так,
это нормально';

// Выводит: Однажды Арнольд сказал: "I'll be back"
echo 'Однажды Арнольд сказал: "I\'ll be back"';

// Выводит: Вы удалили C:\*.*?
echo 'Вы удалили C:\\*.*?';

// Выводит: Вы удалили C:\*.*?
echo 'Вы удалили C:\*.*?';

// Выводит: Это не будет развернуто: \n новая строка
echo 'Это не будет развернуто: \n новая строка';

// Выводит: Переменные $expand также $either не разворачиваются
echo 'Переменные $expand также $either не разворачиваются';
?>
```

Двойные кавычки

Если строка заключена в двойные кавычки ("), PHP распознает большее количество управляющих последовательностей для специальных символов:

Последовательность	Значение
\n	новая строка (LF или 0x0A (10) в ASCII)
\r	возврат каретки (CR или 0x0D (13) в ASCII)
\t	горизонтальная табуляция (HT или 0x09 (9) в ASCII)
\v	вертикальная табуляция (VT или 0x0B (11) в ASCII) (с версии PHP 5.2.5)
\e	escape-знак (ESC или 0x1B (27) в ASCII) (с версии PHP 5.4.0)
\f	подача страницы (FF или 0x0C (12) в ASCII) (с версии PHP 5.2.5)
\\	обратная косая черта
\$	знак доллара
\"	двойная кавычка
\[0-7]{1,3}	последовательность символов, соответствующая регулярному выражению символа в восьмеричной системе счисления
\x[0-9A-Fa-f]{1,2}	последовательность символов, соответствующая регулярному выражению символа в шестнадцатеричной системе счисления

Heredoc

Третий способ определения строк - это использование heredoc-синтаксиса: <<<. После этого оператора необходимо указать идентификатор, затем перевод строки. После этого идет сама строка, а потом этот же идентификатор, закрывающий вставку.

Строка *должна* начинаться с закрывающего идентификатора, т.е. он должен стоять в первом столбце строки. Кроме того, идентификатор должен соответствовать тем же правилам именования, что и все остальные метки в PHP: содержать только буквенно-цифровые символы и знак подчеркивания, и не должен начинаться с цифры (знак подчеркивания разрешается).

Heredoc-текст ведет себя так же, как и строка в двойных кавычках, при этом их не имея. Это означает, что вам нет необходимости экранировать кавычки в heredoc, но вы по-прежнему можете использовать вышеперечисленные управляющие последовательности.

```
<?php
$str = <<<EOD
Пример строки,
охватывающей несколько строчек,
с использованием heredoc-синтаксиса.
EOD;
```

Nowdoc

Nowdoc - это то же самое для строк в одинарных кавычках, что и heredoc для строк в двойных кавычках. Nowdoc похож на heredoc, но внутри него *не осуществляется никаких подстановок*. Эта конструкция идеальна для внедрения PHP-кода или других больших блоков текста без необходимости его экранирования.

Nowdoc указывается той же последовательностью <<<, что используется в heredoc, но последующий за ней идентификатор заключается в одинарные кавычки, например, *<<<'EOT'*. Все условия, действующие для heredoc идентификаторов также действительны и для nowdoc, особенно те, что относятся к закрывающему идентификатору.

```
<?php
$str = <<<'EOD'
Пример текста,
занимающего несколько строк,
с помощью синтаксиса nowdoc.
EOD;
```

Доступ к символу в строке и его изменение

Символы в строках можно использовать и модифицировать, определив их смещение относительно начала строки, начиная с нуля, в квадратных скобках после строки, например, *$str[42]*. Думайте о строке для этой цели, как о массиве символов. Если нужно получить или заменить более 1 символа, можно использовать функции substr() и substr_replace().

```
<?php
// Получение первого символа строки
$str = 'This is a test.';
$first = $str[0];

// Получение третьего символа строки
$third = $str[2];

// Получение последнего символа строки
$str = 'This is still a test.';
$last = $str[strlen($str)-1];

// Изменение последнего символа строки
$str = 'Look at the sea';
$str[strlen($str)-1] = 'e';
?>
```

Функции для работы со строками

- addcslashes — Экранирует строку слэшами в стиле языка C
- addslashes — Экранирует строку с помощью слэшей
- bin2hex — Преобразует бинарные данные в шестнадцатиричное представление
- chop — Псевдоним rtrim
- chr — Возвращает символ по его коду
- chunk_split — Разбивает строку на фрагменты
- convert_cyr_string — Преобразует строку из одной кириллической кодировки в другую
- convert_uudecode — Декодирует строку из формата uuencode в обычный вид
- convert_uuencode — Кодирует строку в формат uuencode
- count_chars — Возвращает информацию о символах, входящих в строку
- crc32 — Вычисляет полином CRC32 для строки
- crypt — Необратимое хэширование строки
- echo — Выводит одну или более строк
- explode — Разбивает строку с помощью разделителя
- fprintf — Записывает отформатированную строку в поток
- get_html_translation_table — Возвращает таблицу преобразований, используемую функциями htmlspecialchars и htmlentities
- hebrev — Преобразует текст на иврите из логической кодировки в визуальную
- hebrevc — Преобразует текст на иврите из логической кодировки в визуальную с преобразованием перевода строки
- hex2bin — Преобразует шестнадцатеричные данные в двоичные

- html_entity_decode — Преобразует все HTML-сущности в соответствующие символы
- htmlentities — Преобразует все возможные символы в соответствующие HTML-сущности
- htmlspecialchars_decode — Преобразует специальные HTML-сущности обратно в соответствующие символы
- htmlspecialchars — Преобразует специальные символы в HTML-сущности
- implode — Объединяет элементы массива в строку
- join — Псевдоним implode
- lcfirst — Преобразует первый символ строки в нижний регистр
- levenshtein — Вычисляет расстояние Левенштейна между двумя строками
- localeconv — Возвращает информацию о числовых форматах
- ltrim — Удаляет пробелы (или другие символы) из начала строки
- md5_file — Возвращает MD5-хэш файла
- md5 — Возвращает MD5-хэш строки
- metaphone — Возвращает ключ metaphone для строки
- money_format — Форматирует число как денежную величину
- nl_langinfo — Возвращает информацию о языке и локали
- nl2br — Вставляет HTML-код разрыва строки перед каждым переводом строки
- number_format — Форматирует число с разделением групп
- ord — Возвращает ASCII-код символа
- parse_str — Разбирает строку в переменные
- print — Выводит строку
- printf — Выводит отформатированную строку
- quoted_printable_decode — Преобразует строку, закодированную методом quoted-printable в 8-битовую строку
- quoted_printable_encode — Кодирует 8-битную строку в с помощью метода quoted-printable
- quotemeta — Экранирует специальные символы
- rtrim — Удаляет пробелы (или другие символы) из конца строки
- setlocale — Устанавливает настройки локали
- sha1_file — Возвращает SHA1-хэш файла
- sha1 — Возвращает SHA1-хэш строки
- similar_text — Вычисляет степень похожести двух строк
- soundex — Возвращает ключ soundex для строки
- sprintf — Возвращает отформатированную строку
- sscanf — Разбирает строку в соответствии с заданным форматом
- str_getcsv — Выполняет разбор CSV-строки в массив
- str_ireplace — Регистро-независимый вариант функции str_replace
- str_pad — Дополняет строку другой строкой до заданной длины
- str_repeat — Возвращает повторяющуюся строку
- str_replace — Заменяет все вхождения строки поиска на строку замены
- str_rot13 — Выполняет преобразование ROT13 над строкой
- str_shuffle — Переставляет символы в строке случайным образом
- str_split — Преобразует строку в массив
- str_word_count — Возвращает информацию о словах, входящих в строку
- strcasecmp — Бинарно-безопасное сравнение строк без учета регистра
- strchr — Псевдоним strstr
- strcmp — Бинарно-безопасное сравнение строк
- strcoll — Сравнение строк с учетом текущей локали
- strcspn — Возвращает длину участка в начале строки, не соответствующего маске
- strip_tags — Удаляет HTML и PHP-теги из строки
- stripcslashes — Удаляет экранирование символов, произведенное функцией addcslashes
- stripos — Возвращает позицию первого вхождения подстроки без учета регистра
- stripslashes — Удаляет экранирование символов
- stristr — Регистро-независимый вариант функции strstr
- strlen — Возвращает длину строки
- strnatcasecmp — Сравнение строк без учета регистра с использованием алгоритма "natural order"

- strnatcmp — Сравнение строк с использованием алгоритма "natural order"
- strncasecmp — Бинарно-безопасное сравнение первых n символов строк без учета регистра
- strncmp — Бинарно-безопасное сравнение первых n символов строк без учета регистра
- strpbrk — Ищет в строке любой символ из заданного набора
- strpos — Возвращает позицию первого вхождения подстроки
- strrchr — Находит последнее вхождение символа в строке
- strrev — Переворачивает строку задом наперед
- strripos — Возвращает позицию последнего вхождения подстроки без учета регистра
- strrpos — Возвращает позицию последнего вхождения подстроки в строке
- strspn — Возвращает длину участка в начале строки, полностью соответствующего маске
- strstr — Находит первое вхождение подстроки
- strtok — Разбивает строку на токены
- strtolower — Преобразует строку в нижний регистр
- strtoupper — Преобразует строку в верхний регистр
- strtr — Преобразует заданные символы или заменяет подстроки
- substr_compare — Бинарно-безопасное сравнение 2 строк со смещением, с учетом или без учета регистра
- substr_count — Возвращает число вхождений подстроки
- substr_replace — Заменяет часть строки
- substr — Возвращает подстроку
- trim — Удаляет пробелы (или другие символы) из начала и конца строки
- ucfirst — Преобразует первый символ строки в верхний регистр
- ucwords — Преобразует в верхний регистр первый символ каждого слова в строке
- vfprintf — Записывает отформатированную строку в поток
- vprintf — Выводит отформатированную строку
- vsprintf — Возвращает отформатированную строку
- wordwrap — Переносит строку по указанному количеству символов

2.5. Массивы

На самом деле массив в PHP - это упорядоченное отображение, которое устанавливает соответствие между *значением* и *ключом*. Этот тип оптимизирован в нескольких направлениях, поэтому вы можете использовать его как собственно массив, список (вектор), хэш-таблицу (являющуюся реализацией карты), словарь, коллекцию, стэк, очередь и, возможно, что-то еще. Так как значением массива может быть другой массив PHP, можно также создавать деревья и многомерные массивы.

Массив (тип array) может быть создан языковой конструкцией array(). language construct. В качестве параметров она принимает любое количество разделенных запятыми пар key => value (ключ => значение).

```
array(
    key  => value,
    key2 => value2,
    key3 => value3,
    ...
)
```

Запятая после последнего элемента массива необязательна и может быть опущена. Обычно это делается для однострочных массивов, т.е. *array(1, 2)* предпочтительней *array(1, 2,)*. Для многострочных массивов с другой стороны обычно используется завершающая запятая, так как позволяет легче добавлять новые элементы в конец массива.

Пример # Простой массив

```
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
);
?>
```

key может быть либо типа integer, либо типа string. value может быть любого типа.

Пример # Смешанные ключи типов integer и string

```
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
    100   => -100,
    -100  => 100,
);
var_dump($array);
?>
```

Результат выполнения данного примера:

```
array(4) {
  ["foo"]=>
  string(3) "bar"
  ["bar"]=>
  string(3) "foo"
  [100]=>
  int(-100)
  [-100]=>
  int(100)
}
```

Параметр key является необязательным. Если он не указан, PHP будет использовать предыдущее наибольшее значение ключа типа integer, увеличенное на 1.

Пример # Индексированные массивы без ключа

```
<?php
$array = array("foo", "bar", "hallo", "world");
var_dump($array);
?>
```

Результат выполнения данного примера:

```
array(4) {
  [0]=>
  string(3) "foo"
  [1]=>
  string(3) "bar"
  [2]=>
  string(5) "hallo"
  [3]=>
  string(5) "world"
}
```

Возможно указать ключ только для некоторых элементов и пропустить для других:

Доступ к элементам массива может быть осуществлен с помощью синтаксиса *array[key]*.

Пример # Доступ к элементам массива

```
<?php
$array = array(
    "foo" => "bar",
    42    => 24,
    "multi" => array(
         "dimensional" => array(
             "array" => "foo"
         )
    )
);

var_dump($array["foo"]);
var_dump($array[42]);
var_dump($array["multi"]["dimensional"]["array"]);
?>
```

Результат выполнения данного примера:

```
string(3) "bar"
int(24)
string(3) "foo"
```

Существующий массив может быть изменен явной установкой значений в нем. Это выполняется присвоением значений массиву array с указанием в скобках ключа. Кроме того, вы можете опустить ключ. В этом случае добавьте к имени переменной пустую пару скобок (*[]*).

```
$arr[key] = value;
$arr[] = value;
// key может быть integer или string
// value может быть любым значением любого типа
```

Если массив *$arr* еще не существует, он будет создан. Таким образом, это еще один способ определить массив array. Однако такой способ применять не рекомендуется, так как если переменная *$arr* уже содержит некоторое значение (например, значение типа string из переменной запроса), то это значение останется на месте и *[]* может на самом деле означать доступ к символу в строке. Лучше инициализировать переменную путем явного присваивания значения.

Для изменения определенного значения просто присвойте новое значение элементу, используя его ключ. Если вы хотите удалить пару ключ/значение, вам необходимо использовать функцию unset().

```
<?php
$arr = array(5 => 1, 12 => 2);

$arr[] = 56;    // В этом месте скрипта это
                // то же самое, что и $arr[13] = 56;

$arr["x"] = 42; // Это добавляет к массиву новый
                // элемент с ключом "x"

unset($arr[5]); // Это удаляет элемент из массива

unset($arr);    // Это удаляет массив полностью
?>
```

Функции для работы с массивами

- array_change_key_case — Меняет регистр ключей в массиве
- array_chunk — Разбивает массив на части
- array_combine — Создает новый массив, используя один массив в качестве ключей, а другой в качестве соответствующих значений
- array_count_values — Подсчитывает количество всех значений массива
- array_diff_assoc — Вычисляет расхождение массивов с дополнительной проверкой индекса
- array_diff_key — Вычисляет расхождение массивов, сравнивая ключи
- array_diff_uassoc — Вычисляет расхождение массивов с дополнительной проверкой индекса, осуществляемой при помощи callback-функции
- array_diff_ukey — Вычисляет расхождение массивов, используя callback-функцию для сравнения ключей
- array_diff — Вычислить расхождение массивов
- array_fill_keys — Создает массив и заполняет его значениями, с определенными ключами
- array_fill — Заполняет массив значениями
- array_filter — Фильтрует элементы массива с помощью callback-функции
- array_flip — Меняет местами ключи с их значениями в массиве
- array_intersect_assoc — Вычисляет схождение массивов с дополнительной проверкой индекса
- array_intersect_key — Вычислить пересечение массивов, сравнивая ключи
- array_intersect_uassoc — Вычисляет схождение массивов с дополнительной проверкой индекса, осуществляемой при помощи callback-функции
- array_intersect_ukey — Вычисляет схождение массивов, используя callback-функцию для сравнения ключей
- array_intersect — Вычисляет схождение массивов
- array_key_exists — Проверяет, присутствует ли в массиве указанный ключ или индекс
- array_keys — Возвращает все или некоторое подмножество ключей массива
- array_map — Применяет callback-функцию ко всем элементам указанных массивов
- array_merge_recursive — Рекурсивное слияние двух или более массивов

- array_merge — Сливает один или большее количество массивов
- array_multisort — Сортирует несколько массивов или многомерные массивы
- array_pad — Дополнить размер массива определенным значением до заданной величины
- array_pop — Извлекает последний элемент массива
- array_product — Вычислить произведение значений массива
- array_push — Добавляет один или несколько элеметов в конец массива
- array_rand — Выбирает одно или несколько случайных значений из массива
- array_reduce — Итеративно уменьшает массив к единственному значению, используя callback-функцию
- array_replace_recursive — Рекурсивно заменяет элементы первого массива элементами переданных массивов
- array_replace — Замена элементов массива элементами других переданных массивов
- array_reverse — Возвращает массив с элементами в обратном порядке
- array_search — Осуществляет поиск данного значения в массиве и возвращает соответствующий ключ в случае удачи
- array_shift — Извлекает первый элемент массива
- array_slice — Выбирает срез массива
- array_splice — Удаляет часть массива и заменяет её чем-нибудь ещё
- array_sum — Вычисляет сумму значений массива
- array_udiff_assoc — Вычисляет расхождение в массивах с дополнительной проверкой индексов, используя для сравнения значений callback-функцию
- array_udiff_uassoc — Вычисляет расхождение в массивах с дополнительной проверкой индексов, используя для сравнения значений и индексов callback-функцию
- array_udiff — Вычисляет расхождение массивов, используя для сравнения callback-функцию
- array_uintersect_assoc — Вычисляет пересечение массивов с дополнительной проверкой индексов, используя для сравнения значений callback-функцию
- array_uintersect_uassoc — Вычисляет пересечение массивов с дополнительной проверкой индекса, используя для сравнения индексов и значений callback-функцию
- array_uintersect — Вычисляет пересечение массивов, используя для сравнения значений callback-функцию
- array_unique — Убирает повторяющиеся значения из массива
- array_unshift — Добавляет один или несколько элементов в начало массива
- array_values — Выбирает все значения массива
- array_walk_recursive — Рекурсивно применяет пользовательскую функцию к каждому элементу массива
- array_walk — Применяет пользовательскую функцию к каждому элементу массива
- array — Создает массив
- arsort — Сортирует массив в обратном порядке, сохраняя ключи
- asort — Сортирует массив, сохраняя ключи
- compact — Создает массив, содержащий названия переменных и их значения
- count — Подсчитывает количество элементов массива или что-то в объекте
- current — Возвращает текущий элемент массива
- each — Возвращает текущую пару ключ/значение из массива и смещает его указатель
- end — Устанавливает внутренний указатель массива на его последний элемент
- extract — Импортирует переменные из массива в текущую таблицу символов
- in_array — Проверяет, присутствует ли в массиве значение
- key — Выбирает ключ из массива
- krsort — Сортирует массив по ключам в обратном порядке
- ksort — Сортирует массив по ключам
- list — Присваивает переменным из списка значения подобно массиву
- natcasesort — Сортирует массив, используя алгоритм "natural order" без учета регистра символов
- natsort — Сортирует массив, используя алгоритм "natural order"
- next — Передвигает внутренний указатель массива на одну позицию вперёд
- pos — Псевдоним current

- prev — Передвигает внутренний указатель массива на одну позицию назад
- range — Создает массив, содержащий диапазон элементов
- reset — Устанавливает внутренний указатель массива на его первый элемент
- rsort — Сортирует массив в обратном порядке
- shuffle — Перемешивает массив
- sizeof — Псевдоним count
- sort — Сортирует массив
- uasort — Сортирует массив, используя пользовательскую функцию для сравнения элементов с сохранением ключей
- uksort — Сортирует массив по ключам, используя пользовательскую функцию для сравнения ключей
- usort — Сортирует массив по значениям используя пользовательскую функцию для сравнения элементов

Пример # Коллекция

```
<?php
$colors = array('red', 'blue', 'green', 'yellow');
foreach ($colors as $color) {
    echo "Вам нравится $color?\n";
}
?>
```

Результат выполнения данного примера:

```
Вам нравится red?
Вам нравится blue?
Вам нравится green?
Вам нравится yellow?
```

Пример # Изменение элемента в цикле

```
<?php
// PHP 5
foreach ($colors as &$color) {
    $color = strtoupper($color);
}
unset($color); /* это нужно для того, чтобы последующие записи в
$color не меняли последний элемент массива */

// Обходной прием для старых версий
foreach ($colors as $key => $color) {
    $colors[$key] = strtoupper($color);
}
print_r($colors);
?>
```

Результат выполнения данного примера:

```
Array
(
    [0] => RED
    [1] => BLUE
    [2] => GREEN
    [3] => YELLOW
)
```

Пример # Индекс, начинающийся с единицы

```
<?php
$firstquarter  = array(1 => 'Январь', 'Февраль', 'Март');
print_r($firstquarter);
?>
```

Результат выполнения данного примера:

```
Array
(
    [1] => 'Январь'
    [2] => 'Февраль'
    [3] => 'Март'
)
```

2.6. Объекты

Для создания нового объекта, используйте выражение *new*, создающее в переменной экземпляр класса:

```
<?php
class foo
{
    function do_foo()
    {
        echo "Doing foo.";
    }
}
$bar = new foo;
$bar->do_foo();
?>
```

2.7. Ресурсы

Resource это специальная переменная, содержащая ссылку на внешний ресурс. Ресурсы создаются и используются специальными функциями.

Благодаря системе подсчета ссылок, введенной в PHP, определение отсутствия ссылок на ресурс происходит автоматически, после чего он освобождается сборщиком мусора. Поэтому, очень редко требуется освобождать память вручную.

2.8. NULL

Специальное значение NULL представляет собой переменную без значения. NULL - это единственно возможное значение типа null.

Переменная считается null, если:

- ей была присвоена константа NULL.
- ей еще не было присвоено никакого значения.
- она была удалена с помощью unset().

Существует только одно значение типа null - регистро-независимая константа NULL.

```
<?php
$var = NULL;
?>
```

Приведение переменной к null с использованием *(unset) $var не* удаляет переменную и ее значение. Данное выражение только возвращает NULL

2.9. Манипуляции с типами

PHP не требует (и не поддерживает) явного типа при определении переменной; тип переменной определяется по контексту, в котором она используется. То есть, если вы присвоите значение типа string переменной $var, то $var станет строкой. Если вы затем присвоите $var целочисленное значение, она станет целым числом.

Примером автоматического преобразования типа является оператор сложения '+'. Если какой-либо из операндов является float, то все операнды интерпретируются как float, и результатом также будет float. В противном случае операнды будут интерпретироваться как целые числа и результат также будет целочисленным. Обратите внимание, что это НЕ меняет типы самих операндов; меняется только то, как они вычисляются и сам тип выражения.

```
<?php
$foo = "0";  // $foo это строка (ASCII 48)
$foo += 2;   // $foo теперь целое число (2)
$foo = $foo + 1.3;  // $foo теперь число с плавающей точкой (3.3)
$foo = 5 + "10 Little Piggies"; // $foo это целое число (15)
$foo = 5 + "10 Small Pigs";     // $foo это целое число (15)
?>
```

Приведение типов в PHP работает так же, как и в C: имя требуемого типа записывается в круглых скобках перед приводимой переменной.

```
<?php
```

```
$foo = 10;   // $foo это целое число
$bar = (boolean) $foo;   // $bar это булев тип
?>
```

Допускаются следующие приведения типов:

- (int), (integer) - приведение к integer
- (bool), (boolean) - приведение к boolean
- (float), (double), (real) - приведение к float
- (string) - приведение к string
- (array) - приведение к array
- (object) - приведение к object
- (unset) - приведение к NULL

3. Переменные

Переменные в PHP представлены знаком доллара с последующим именем переменной. Имя переменной чувствительно к регистру.

Имена переменных соответствуют тем же правилам, что и остальные наименования в PHP. Правильное имя переменной должно начинаться с буквы или символа подчеркивания и состоять из букв, цифр и символов подчеркивания в любом количестве. Это можно отобразить регулярным выражением: '[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'

```
<?php
$var = 'Bob';
$Var = 'Joe';
echo "$var, $Var";      // выведет "Bob, Joe"
$4site = 'not yet';     // неверно; начинается с цифры
$_4site = 'not yet';    // верно; начинается с символа подчеркивания
$täyte = 'mansikka';    // верно; 'ä' это (Расширенный) ASCII 228.
?>
```

По умолчанию, переменные всегда присваиваются по значению. То есть, когда вы присваиваете выражение переменной, все значение оригинального выражения копируется в эту переменную. Это означает, к примеру, что, после того как одной переменной присвоено значение другой, изменение одной из них не влияет на другую.

PHP также предлагает иной способ присвоения значений переменным: присвоение по ссылке. Это означает, что новая переменная просто ссылается (иначе говоря, "становится псевдонимом" или "указывает") на оригинальную переменную. Изменения в новой переменной отражаются на оригинале, и наоборот.

Для присвоения по ссылке, просто добавьте амперсанд (&) к началу имени присваиваемой (исходной) переменной. Например, следующий фрагмент кода дважды выводит 'Меня зовут Боб':

```
<?php
$foo = 'Боб';              // Присваивает $foo значение 'Боб'
$bar = &$foo;              // Ссылка на $foo через $bar.
$bar = "Меня зовут $bar";  // Изменение $bar...
echo $bar;
echo $foo;                 // меняет и $foo.
?>
```

Хотя в PHP и нет необходимости инициализировать переменные, это считается очень хорошей практикой. Неинициализированные переменные принимают значение по умолчанию в зависимости от их типа, который определяется из контекста их первого использования: булевы принимают значение FALSE, целые и числа с плавающей точкой - ноль, строки (например, при использовании в echo) - пустую строку, а массивы становятся пустыми массивами.

3.1. Область видимости переменной

Область видимости переменной - это контекст, в котором эта переменная определена. В большинстве случаев все переменные PHP имеют только одну область видимости. Эта единая область видимости охватывает также включаемые (include) и требуемые (require) файлы.

```
<?php
$a = 1;
include 'b.inc';
?>
```

Здесь переменная *$a* будет доступна внутри включенного скрипта *b.inc*. Однако определение (тело) пользовательской функции задает локальную область видимости данной функции. Любая используемая внутри функции переменная по умолчанию ограничена локальной областью видимости функции.

```
<?php
$a = 1; /* глобальная область видимости */
function test()
{
    echo $a; /* ссылка на переменную локальной области видимости */
}
test();
```

```
?>
```

Этот скрипт не сгенерирует никакого вывода, поскольку выражение echo указывает на локальную версию переменной *$a*, а в пределах этой области видимости ей не было присвоено значение. В PHP, если глобальная переменная будет использоваться внутри функции, она должна быть объявлена глобальной внутри определения функции.

Пример # Использование *global*

```
<?php
$a = 1;
$b = 2;
function Sum()
{
    global $a, $b;
    $b = $a + $b;
}
Sum();
echo $b;
?>
```

Вышеприведенный скрипт выведет *3*. После определения *$a* и *$b* внутри функции как global все ссылки на любую из этих переменных будут указывать на их глобальную версию. Не существует никаких ограничений на количество глобальных переменных, которые могут обрабатываться функцией.

Другой важной особенностью области видимости переменной является *статическая* переменная. Статическая переменная существует только в локальной области видимости функции, но не теряет своего значения, когда выполнение программы выходит из этой области видимости.

Пример # Пример использования статических переменных

```
<?php
function test()
{
    static $a = 0;
    echo $a;
    $a++;
}
?>
```

Переменная *$a* будет проинициализирована только при первом вызове функции, а каждый вызов функции *test()* будет выводить значение *$a* и инкрементировать его.

3.2. Переменные переменных

Иногда бывает удобно иметь переменными имена переменных. То есть, имя переменной, которое может быть определено и изменено динамически. Обычная переменная определяется примерно таким выражением:

```
<?php
$a = 'hello';
?>
```

Переменная переменной берет значение переменной и рассматривает его как имя переменной. В вышеприведенном примере hello может быть использовано как имя переменной при помощи двух знаков доллара.

```
<?php
$$a = 'world';
?>
```

Теперь в дереве символов PHP определены и содержатся две переменные: $a, содержащая "hello", и $hello, содержащая "world".

Для того чтобы использовать переменные переменных с массивами, вы должны решить проблему двусмысленности. То есть, если вы напишете $$a[1], обработчику необходимо знать, хотите ли вы использовать $a[1] в качестве переменной, либо вам нужна как переменная $$a, а затем ее индекс [1]. Синтаксис для разрешения этой двусмысленности таков: ${$a[1]} для первого случая и ${$a}[1] для второго.

3.3. Переменные извне PHP

Когда происходит отправка данных формы PHP-скрипту, информация из этой формы автоматически становится доступной ему. Существует много способов получения этой информации.

Пример # Простая HTML-форма

```
<form action="foo.php" method="post">
    Имя:  <input type="text" name="username" /><br />
    Email: <input type="text" name="email" /><br />
    <input type="submit" name="submit" value="Отправь меня!" />
</form>
```

Пример # Доступ к данным из простой HTML POST-формы

```
<?php
// Доступно, начиная с PHP 4.1.0
   echo $_POST['username'];
   echo $_REQUEST['username'];
   import_request_variables('p', 'p_');
   echo $p_username;

// Начиная с PHP 5.0.0, эти длинные предопределенные
// переменные могут быть отключены директивой register_long_arrays.
   echo $HTTP_POST_VARS['username'];

// Доступно, если директива PHP register_globals = on. Начиная
// с PHP 4.2.0, значение по умолчанию register_globals = off.
// Использование/доверие этому методу не рекомендуется.
   echo $username;
?>
```

GET-форма используется аналогично, за исключением того, что вместо POST, вам нужно будет использовать соответствующую предопределенную переменную GET. GET относится также к *QUERY_STRING* (информация в URL после '?'). Так, например,

http://www.example.com/test.php?id=3 содержит GET-данные, доступные как $_GET['id'].

3.4. Функции для работы с переменными

- debug_zval_dump — Выводит строковое представление внутреннего значения zend
- doubleval — Псевдоним floatval
- empty — Проверяет, пуста ли переменная
- floatval — Возвращает значение переменной в виде числа с плавающей точкой
- get_defined_vars — Возвращает массив всех определенных переменных
- get_resource_type — Возвращает тип ресурса
- gettype — Возвращает тип переменной
- import_request_variables — Импортирует переменные GET/POST/Cookie в глобальную область видимости
- intval — Возвращает целое значение переменной
- is_array — Определяет, является ли переменная массивом
- is_bool — Проверяет, является ли переменная булевой
- is_callable — Проверяет, может ли значение переменной быть вызвано в качестве функции
- is_double — Псевдоним is_float
- is_float — Проверяет, является ли переменная числом с плавающей точкой
- is_int — Проверяет, является ли переменная переменной целочисленного типа
- is_integer — Псевдоним is_int
- is_long — Псевдоним is_int
- is_null — Проверяет, является ли значение переменной равным NULL
- is_numeric — Проверяет, является ли переменная числом или строкой, содержащей число
- is_object — Проверяет, является ли переменная объектом
- is_real — Псевдоним is_float
- is_resource — Проверяет, является ли переменная ресурсом
- is_scalar — Проверяет, является ли переменная скалярным значением
- is_string — Проверяет, является ли переменная строкой
- isset — Определяет, была ли установлена переменная значением отличным от NULL

- print_r — Выводит удобочитаемую информацию о переменной
- serialize — Генерирует пригодное для хранения представление переменной
- settype — Присваивает переменной новый тип
- strval — Возвращает строковое значение переменной
- unserialize — Создает PHP-значение из хранимого представления
- unset — Удаляет переменную
- var_dump — Выводит информацию о переменной
- var_export — Выводит в браузер или возвращает интерпретируемое строковое представление переменной

4. Константы

Константы - это идентификаторы (имена) простых значений. Исходя из их названия, нетрудно понять, что их значение не может изменяться в ходе выполнения скрипта (исключения представляют "волшебные" константы, которые на самом деле не являются константами в полном смысле этого слова). Имена констант чувствительны к регистру. По принятому соглашению, имена констант всегда пишутся в верхнем регистре.

Имя константы должно соответствовать тем же правилам, что и другие имена в PHP. Правильное имя начинается с буквы или символа подчеркивания и состоит из букв, цифр и подчеркиваний. Регулярное выражение для проверки правильности имени константы выглядит так: [a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*

```
<?php
// Правильные имена констант
define("FOO",     "something");
define("FOO2",    "something else");
define("FOO_BAR", "something more");

// Неправильные имена констант
define("2FOO",    "something");

// Это корректное объявление, но лучше его не использовать:
// PHP однажды может зарегистрировать "волшебную" константу,
// которая сломает ваш скрипт
define("__FOO__", "something");
?>
```

4.1. Синтаксис

Вы можете определить константу с помощью функции define() или с помощью ключевого слова const вне объявления класса. После того, как константа определена, ее значение не может быть изменено или аннулировано. Константы могут содержать только скалярные данные (boolean, integer, float и string типов).

Получить значение константы можно, указав ее имя. В отличие от переменных, вам не нужно предварять имя константы символом $. Также вы можете использовать функцию constant() для получения значения константы, если вы формируете имя константы динамически. Используйте функцию get_defined_constants() для получения списка всех объявленных констант.

Различия между константами и переменными:

- У констант нет приставки в виде знака доллара ($);
- Константы можно определить только с помощью функции define(), а не присваиванием значения;
- Константы могут быть определены и доступны в любом месте без учета области видимости;
- Константы не могут быть переопределены или аннулированы после первоначального объявления;
- Константы могут иметь только скалярные значения.

```
<?php
define("CONSTANT", "Здравствуй, мир.");
echo CONSTANT; // выводит "Здравствуй, мир."
echo Constant; // выводит "Constant" и предупреждение.
// Работает, начиная с версии PHP 5.3.0
const CONSTANT = 'Здравствуй, мир.';
echo CONSTANT;
?>
```

4.2. "Волшебные" константы

PHP предоставляет большой список предопределенных констант для каждого выполняемого скрипта. Многие из этих констант определяются различными модулями и будут присутствовать только в том случае, если эти модули доступны в результате динамической загрузки или в результате статической сборки.

Есть восемь волшебных констант, которые меняют свое значение в зависимости от контекста, в котором они используются. Например, значение __LINE__ зависит от строки в скрипте, на которой эта константа указана. Специальные константы нечувствительны к регистру и их список приведен ниже:

Имя	**Описание**
__LINE__	Текущий номер строки в файле.
__FILE__	Полный путь и имя текущего файла. Если используется внутри подключаемого файла, то возвращается имя данного файла. **__FILE__** всегда содержит абсолютный путь с разрешенными символическими ссылками, тогда как в старых версиях в некоторых обстоятельствах возвращался относительный путь.
__DIR__	Директория файла. Если используется внутри подключаемого файла, то возвращается директория этого файла. Это эквивалентно вызову *dirname(__FILE__)*. Возвращаемое имя директории не оканчивается на слэш, за исключением корневой директории
__FUNCTION__	Имя функции. Возвращает имя функции точно так, как оно было объявлено (с учетом регистра).
__CLASS__	Имя класса. Возвращает имя класса точно так, как оно было объявлено (с учетом регистра). Имя класса содержит название пространства имен, в котором класс был объявлен (например, *Foo\Bar*).
__TRAIT__	Имя трейта. Возвращает имя трейта точно так, как оно было объявлено (с учетом регистра). Это имя содержит название пространства имен, в котором трейт был объявлен (например, *Foo\Bar*).
__METHOD__	Имя метода класса. Возвращается так, как оно было объявлено (с учетом регистра).
__NAMESPACE__	Имя текущего пространства имен (с учетом регистра). Эта константа определяется во время компиляции

5. Операторы

Оператором называется нечто, принимающее одно или более значений (или выражений, если говорить на жаргоне программирования), и вычисляющее новое значение (таким образом, вся конструкция может рассматриваться как выражение).

Операторы можно сгруппировать по количеству принимаемых ими значений. Унарные операторы принимают только одно значение, например, *!* (оператор логического отрицания) или ++ (инкремент). Бинарные операторы принимают два значения; это, например, знакомые всем арифметические операторы + (плюс) и - (минус), большинство поддерживаемых в PHP операторов входят именно в эту категорию. Ну и, наконец, есть всего один тернарный оператор, *? :*, принимающий три значения (хотя, возможно, более точным названием было бы "условный оператор").

Приоритет оператора определяет, насколько "тесно" он связывает между собой два выражения. Например, выражение 1 + 5 * 3 вычисляется как 16, а не 18, поскольку оператор умножения ("*") имеет более высокий приоритет, чем оператор сложения ("+"). Круглые скобки могут использоваться для принудительного указания порядка выполнения операторов. Например, выражение (1 + 5) * 3 вычисляется как 18.

5.1. Арифметические операторы

Пример	Название	Результат
-$a	Отрицание	Смена знака *$a*.
$a + $b	Сложение	Сумма *$a* и *$b*.
$a - $b	Вычитание	Разность *$a* и *$b*.
$a * $b	Умножение	Произведение *$a* и *$b*.
$a / $b	Деление	Частное от деления *$a* на *$b*.
$a % $b	Остаток от деления	Целочисленный остаток от деления *$a* на *$b*.

Операция деления ("/") возвращает число с плавающей точкой, кроме случая, когда оба значения являются целыми числами (или строками, которые преобразуются в целые числа), которые делятся нацело - в этом случае возвращается целое значение.

При делении по модулю операнды преобразуются в целые числа (удалением дробной части) до начала операции.

Результат операции остатка от деления % будет иметь тот же знак, что и делимое — то есть, результат $a % $b будет иметь тот же знак, что и $a.

```
<?php
echo (5 % 3)."\n";          // выводит 2
echo (5 % -3)."\n";         // выводит 2
echo (-5 % 3)."\n";         // выводит -2
echo (-5 % -3)."\n";        // выводит -2
?>
```

5.2. Оператор присваивания

Базовый оператор присваивания обозначается как "=". На первый взгляд может показаться, что это оператор "равно". На самом деле это не так. В действительности, оператор присваивания означает, что левый операнд получает значение правого выражения, (т.е. устанавливается значением).

Результатом выполнения оператора присваивания является само присвоенное значение. Таким образом, результат выполнения "*$a = 3*" будет равен 3.

Для массивов (array), присвоение значения именованному ключу происходит с помощью оператора "=>".

В дополнение к базовому оператору присваивания имеются "комбинированные операторы" для всех бинарных арифметическихопераций, операций объединения массивов и строковых операций, которые позволяют использовать некоторое значение в выражении, а затем установить его как результат данного выражения.

```
<?php
$a = 3;
$a += 5; // устанавливает $a в 8, как если бы мы написали: $a = $a + 5;
$b = "Hello ";
$b .= "There!"; // устанавливает $b в "Hello There!",  как и $b = $b . "There!";
?>
```

Присваивание по ссылке также поддерживается, для него используется синтаксис $var = &$othervar;. 'Присваивание по ссылке' означает, что обе переменные указывают на одни и те же данные и никакого копирования не происходит.

Пример # Присваивание по ссылке

```
<?php
$a = 3;
$b = &$a; // $b - это ссылка на $a

print "$a\n"; // печатает 3
print "$b\n"; // печатает 3

$a = 4; // меняем $a

print "$a\n"; // печатает 4
print "$b\n"; // также печатает 4, так как $b является ссылкой на $a,
              // а значение переменной $a успело измениться
?>
```

5.3. Операторы сравнения

Операторы сравнения, как это видно из их названия, позволяют сравнивать между собой два значения.

Пример	Название	Результат
$a == $b	Равно	TRUE если *$a* равно *$b* после преобразования типов.
$a === $b	Тождественно равно	TRUE если *$a* равно *$b* и имеет тот же тип.
$a != $b	Не равно	TRUE если *$a* не равно *$b* после преобразования типов.
$a <> $b	Не равно	TRUE если *$a* не равно *$b* после преобразования типов.
$a !== $b	Тождественно не равно	TRUE если *$a* не равно *$b* или они разных типов.
$a < $b	Меньше	TRUE если *$a* строго меньше *$b*.
$a > $b	Больше	TRUE если *$a* строго больше *$b*.
$a <= $b	Меньше или равно	TRUE если *$a* меньше или равно *$b*.
$a >= $b	Больше или равно	TRUE если *$a* больше или равно *$b*.

В случае, если вы сравниваете число со строкой или две строки, содержащие числа, каждая строка будет преобразована в число, и сравниваться они будут как числа. Эти правила также распространяются на оператор switch. Преобразование типов не происходит при использовании === или !== так как в этом случае кроме самих значений сравниваются еще и типы.

```
<?php
var_dump(0 == "a"); // 0 == 0 -> true
var_dump("1" == "01"); // 1 == 1 -> true
var_dump("10" == "1e1"); // 10 == 10 -> true
var_dump(100 == "1e2"); // 100 == 100 -> true
switch ("a") {
case 0:
    echo "0";
    break;
case "a": // Эта ветка никогда не будет достигнута, так как "a" уже сопоставленно с 0
    echo "a";
    break;
}
?>
```

5.4. Тернарный оператор

Еще одним условным оператором является тернарный оператор "?:".

```
<?php
```

```
// Пример использования тернарного оператора
$action = (empty($_POST['action'])) ? 'default' : $_POST['action'];
// Приведенный выше код аналогичен следующему блоку с использованием if/else
if (empty($_POST['action'])) {
    $action = 'default';
} else {
    $action = $_POST['action'];
}
?>
```

Выражение (expr1) ? (expr2) : (expr3) интерпретируется как expr2, если expr1 имеет значение TRUE, или как expr3 если expr1 имеет значение FALSE.

5.5. Оператор управления ошибками

PHP поддерживает один оператор управления ошибками: знак (@). В случае, если он предшествует какому-либо выражению в PHP-коде, любые сообщения об ошибках, генерируемые этим выражением, будут проигнорированы.

Если вы установили собственную функцию обработки ошибок с помощью set_error_handler(), то она все равно будет вызвана, однако, если внутри этой функции будет вызвана функция error_reporting(), то она вернет 0, если функция, вызвавшая данную ошибку, была подавлена с помощью @.

В случае, если установлена опция track_errors , все генерируемые сообщения об ошибках будут сохраняться в переменной $php_errormsg. Эта переменная будет перезаписываться при каждой новой ошибке, поэтому в случае необходимости проверяйте ее сразу же.

```
<?php
// Преднамеренная ошибка при работе с файлами
$my_file = @file ('non_existent_file') or
    die ("Ошибка при открытии файла: сообщение об ошибке было таким: '$php_errormsg'"
);

// работает для любых выражений, а не только для функций
$value = @$cache[$key];
// В случае если ключа $key нет, сообщение об ошибке (notice) не будет отображено
?>
```

5.6. Операторы инкремента и декремента

PHP поддерживает префиксные и постфиксные операторы инкремента и декремента в стиле C.

Пример	Название	Действие
++$a	Префиксный инкремент	Увеличивает *$a* на единицу, затем возвращает значение *$a*.
$a++	Постфиксный инкремент	Возвращает значение *$a*, затем увеличивает *$a* на единицу.
--$a	Префиксный декремент	Уменьшает *$a* на единицу, затем возвращает значение *$a*.
$a--	Постфиксный декремент	Возвращает значение *$a*, затем уменьшает *$a* на единицу.

```
<?php
echo "<h3>Постфиксный инкремент</h3>";
$a = 5;
echo "Должно быть 5: " . $a++ . "<br />\n";
echo "Должно быть 6: " . $a . "<br />\n";
echo "<h3>Префиксный инкремент</h3>";
$a = 5;
echo "Должно быть 6: " . ++$a . "<br />\n";
echo "Должно быть 6: " . $a . "<br />\n";
echo "<h3>Постфиксный декремент</h3>";
$a = 5;
echo "Должно быть 5: " . $a-- . "<br />\n";
echo "Должно быть 4: " . $a . "<br />\n";
echo "<h3>Префиксный декремент</h3>";
$a = 5;
echo "Должно быть 4: " . --$a . "<br />\n";
echo "Должно быть 4: " . $a . "<br />\n";
?>
```

5.7. Логические операторы

Пример	Название	Результат
$a and $b	И	**TRUE** если и *$a*, и *$b* **TRUE**.
$a or $b	Или	**TRUE** если или *$a*, или *$b* **TRUE**.
$a xor $b	Исключающее или	**TRUE** если *$a*, или *$b* **TRUE**, но не оба.
! $a	Отрицание	**TRUE** если *$a* не **TRUE**.
$a && $b	И	**TRUE** если и *$a*, и *$b* **TRUE**.
$a || $b	Или	**TRUE** если или *$a*, или *$b* **TRUE**.

```
<?php
// foo() никогда не буде вызвана, так как эти операторы являются шунтирующими (short-
circuit)
$a = (false && foo());
$b = (true  || foo());
$c = (false and foo());
$d = (true  or  foo());

// "||" имеет больший приоритет, чем "or"
// Результат выражения (false || true) присваивается переменной $e
// Действует как: ($e = (false || true))
$e = false || true;

// Константа false присваивается $f, а затем значение true игнорируется
// Действует как: (($f = false) or true)
$f = false or true;

var_dump($e, $f);

// "&&" имеет больший приоритет, чем "and"
// Результат выражения (true && false) присваивается переменной $g
// Действует как: ($g = (true && false))
$g = true && false;

// Константа true присваивается $h, а затем значение false игнорируется
// Действует как: (($h = true) and false)
$h = true and false;

var_dump($g, $h);
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
bool(true)
bool(false)
bool(false)
bool(true)
```

5.8. Строковые операторы

В PHP есть два оператора для работы со строками (string). Первый - оператор конкатенации ('.'), который возвращает строку, представляющую собой соединение левого и правого аргумента. Второй - оператор присваивания с конкатенацией ('.='), который присоединяет правый аргумент к левому.

```
<?php
$a = "Hello ";
$b = $a . "World!"; // $b теперь содержит строку "Hello World!"

$a = "Hello ";
$a .= "World!";     // $a теперь содержит строку "Hello World!"
?>
```

5.9. Операторы, работающие с массивами

Пример	Название	Результат
$a + $b	Объединение	Объединение массива *$a* и массива *$b*.
$a == $b	Равно	**TRUE** в случае, если *$a* и *$b* содержат одни и те же пары

		ключ/значение.
$a === $b	Тождественно равно	**TRUE** в случае, если *$a* и *$b* содержат одни и те же паты ключ/значение в том же самом порядке и того же типа.
$a != $b	Не равно	**TRUE**, если массив *$a* не равен массиву *$b*.
$a <> $b	Не равно	**TRUE**, если массив *$a* не равен массиву *$b*.
$a !== $b	Тождественно не равно	**TRUE**, если массив *$a* не равен тождественно массиву *$b*.

Оператор + возвращает левый массив, к которому был присоединен правый массив. Для ключей, которые существуют в обоих массивах, будут использованы значения из левого массива, а соответствующие им элементы из правого массива будут проигнорированы.

```
<?php
$a = array("a" => "apple", "b" => "banana");
$b = array("a" => "pear", "b" => "strawberry", "c" => "cherry");

$c = $a + $b; // Объединение $a и $b
echo "Union of \$a and \$b: \n";
var_dump($c);

$c = $b + $a; // Объединение $b и $a
echo "Union of \$b and \$a: \n";
var_dump($c);
?>
```

После своего выполнения скрипт напечатает следующее:

```
Union of $a and $b:
array(3) {
  ["a"]=>
  string(5) "apple"
  ["b"]=>
  string(6) "banana"
  ["c"]=>
  string(6) "cherry"
}
Union of $b and $a:
array(3) {
  ["a"]=>
  string(4) "pear"
  ["b"]=>
  string(10) "strawberry"
  ["c"]=>
  string(6) "cherry"
}
```

6. Управляющие конструкции

6.1. if

Конструкция if является одной из наиболее важных во многих языках программирования, в том числе и PHP. Она предоставляет возможность условного выполнения фрагментов кода. Структура if реализована в PHP по аналогии с языком C:

```
if (выражение)
  инструкция
```

выражение вычисляется в булево значение. Если выражение принимает значение TRUE, PHP выполнит инструкцию, а если оно принимает значение FALSE - проигнорирует.

Следующий пример выведет a больше b, если значение переменной *$a* больше, чем *$b*:

```
<?php
if ($a > $b)
  echo "a больше b";
?>
```

Часто необходимо, чтобы условно выполнялось более одной инструкции. Разумеется, для этого нет необходимости обворачивать каждую инструкцию в if. Вместо этого можно объединить несколько инструкций в блок. Например, следующий код выведет a больше b, если значение переменной *$a* больше, чем *$b*, и затем присвоит значение переменной *$a* переменной *$b*:

```
<?php
if ($a > $b) {
  echo "a больше b";
  $b = $a;
}
?>
```

6.2. else

Часто необходимо выполнить одно выражение, если определенное условие верно, и другое выражение, если условие не верно. Именно для этого else и используется. else расширяет оператор if, чтобы выполнить выражение, в случае, если условие в операторе if равно FALSE. К примеру, следующий код выведет a больше чем b, если *$a* больше, чем *$b*, и a НЕ больше, чем b в противном случае:

```
<?php
if ($a > $b) {
  echo "a больше, чем b";
} else {
  echo "a НЕ больше, чем b";
}
?>
```

Выражение else выполняется только, если выражение if эквивалентно FALSE, и если нет других любых выражений elseif, или если они все равны FALSE.

6.3. elseif

Конструкция elseif, как ее имя и говорит есть сочетание if и else. Аналогично else, она расширяет оператор if для выполнения различных выражений в случае, когда условие начального оператора if эквивалентно FALSE. Однако, в отличии от else, выполнение альтернативного выражения произойдет только тогда, когда условие оператора elseif будет являться равным TRUE. К примеру, следующий код может выводить a больше, чем b, a равно b or a меньше, чем b:

```
<?php
if ($a > $b) {
    echo "a больше, чем b";
} elseif ($a == $b) {
    echo "a равен b";
} else {
    echo "a меньше, чем b";
}
?>
```

Может быть несколько elseif в одном if выражении. Первое же выражение elseif (если будет хоть одно) равное TRUE будет выполнено. Выражение elseif выполнится, если предшествующее

выражение if и предшествующие выражения elseif эквивалентны FALSE, а текущий elseif равен TRUE.

6.4. while

Циклы while являются простейшим видом циклов в PHP. Они ведут себя так же, как и их коллеги из языка C. Простейшей формой цикла while является следующее выражение:

```
while (expr)
    statement
```

Смысл выражения while очень прост. Оно указывает PHP выполнять вложенные выражения повторно до тех пор, пока выражение в самом while является TRUE. Значение выражения expr проверяется каждый раз перед началом цикла, поэтому даже если значение выражения изменится в процессе выполнения вложенных выражений в цикле, выполнение не прекратится до конца итерации (каждый раз, когда PHP выполняет выражения в цикле - это одна итерация). В том случае, если выражение while равно FALSE с самого начала, вложенные выражения ни разу не будут выполнены.

Также, как и с оператором if, вы можете группировать несколько выражений внутри одного цикла while, заключая эти выражения между фигурными скобками или используя альтернативный синтаксис:

```
while (expr):
    statement
    ...
endwhile;
```

Следующие примеры идентичны, и оба выведут числа от 1 до 10:

```
<?php
/* пример 1 */
$i = 1;
while ($i <= 10) {
    echo $i++;  /* выводится будет значение переменной
                   $i перед её увеличением (post-increment) */
}
/* пример 2 */
$i = 1;
while ($i <= 10):
    echo $i;
    $i++;
endwhile;
?>
```

6.5. do-while

Цикл do-while очень похож на цикл while, с тем отличием, что истинность выражения проверяется в конце итерации, а не в начале. Главное отличие от обычного цикла while в том, что первая итерация цикла do-while гарантированно выполнится (истинность выражения проверяется в конце итерации), тогда как она может не выполниться в обычном цикле while (истинность выражения которого проверяется в начале выполнения каждой итерации, и если изначально имеет значение FALSE, то выполнение цикла будет прервано сразу).

Есть только один вариант синтаксиса цикла do-while:

```
<?php
$i = 0;
do {
    echo $i;
} while ($i > 0);
?>
```

В примере цикл будет выполнен ровно один раз, так как после первой итерации, когда проверяется истинность выражения, она будет вычислена как FALSE (*$i* не больше 0) и выполнение цикла прекратится.

6.6. for

Цикл for самый сложный цикл в PHP. Он ведет себя так же как его аналог в языке C. Синтаксис цикла for следующий:

```
for (expr1; expr2; expr3)
    statement
```

Первое выражение (*expr1*) всегда вычисляется (выполняется) только один раз в начале цикла.

В начале каждой итерации оценивается выражение *expr2*. Если оно принимает значение TRUE, то цикл продолжается, и вложенные операторы будут выполнены. Если оно принимает значение FALSE, выполнение цикла заканчивается.

В конце каждой итерации выражение *expr3* вычисляется (выполняется).

Каждое из выражений может быть пустым или содержать несколько выражений, разделенных запятыми. В *expr2* все выражения, разделенные запятыми, вычисляются, но результат берется из последнего. Если выражение *expr2* отсутствует, это означает, что цикл будет выполняться бесконечно. (PHP неявно воспринимает это значение как TRUE, также, как в языке C). Часто необходимо прервать цикл, используя условный оператор break вместо использования выражения в цикле for, которое принимает истинное значение.

Рассмотрим следующие примеры. Все из них отображают числа от 1 до 10:

```
<?php
/* пример 1 */
for ($i = 1; $i <= 10; $i++) {
    echo $i;
}
/* пример 2 */
for ($i = 1; ; $i++) {
    if ($i > 10) {
        break;
    }
    echo $i;
}
/* пример 3 */
$i = 1;
for (; ; ) {
    if ($i > 10) {
        break;
    }
    echo $i;
    $i++;
}
/* пример 4 */
for ($i = 1, $j = 0; $i <= 10; $j += $i, print $i, $i++);
?>
```

PHP также поддерживает альтернативный синтаксис с двоеточием для циклов for.

```
for (expr1; expr2; expr3):
    statement
    ...
endfor;
```

Перебор массивов показан ниже

```
<?php
$people = Array(
        Array('name' => 'Kalle', 'salt' => 856412),
        Array('name' => 'Pierre', 'salt' => 215863)
        );
for($i = 0, $size = sizeof($people); $i < $size; ++$i)
{
    $people[$i]['salt'] = rand(000000, 999999);
}
?>
```

6.7. foreach

Конструкция foreach предоставляет простой способ перебора массивов. Foreach работает только с массивами и объектами, и будет генерировать ошибку при попытке использования с

переменными других типов или неинициализированными переменными. Существует два вида синтаксиса:

```
foreach (array_expression as $value)
    statement
foreach (array_expression as $key => $value)
    statement
```

Первый цикл перебирает массив, задаваемый с помощью array_expression. На каждой итерации значение текущего элемента присваивается переменной $value и внутренний указатель массива увеличивается на единицу (таким образом, на следующей итерации цикла работа будет происходить со следующим элементом).

Второй цикл будет дополнительно соотносить ключ текущего элемента с переменной $key на каждой итерации.

```
<?php
/* Пример 1: только значение */
$a = array(1, 2, 3, 17);
foreach ($a as $v) {
    echo "Текущее значение переменной \$a: $v.\n";
}
/* Пример 2: значение (для иллюстрации массив выводится в виде значения с ключем) */
$a = array(1, 2, 3, 17);
$i = 0; /* только для пояснения */
foreach ($a as $v) {
    echo "\$a[$i] => $v.\n";
    $i++;
}
/* Пример 3: ключ и значение */
$a = array(
    "one" => 1,
    "two" => 2,
    "three" => 3,
    "seventeen" => 17
);
foreach ($a as $k => $v) {
    echo "\$a[$k] => $v.\n";
}
/* Пример 4: многомерные массивы */
$a = array();
$a[0][0] = "a";
$a[0][1] = "b";
$a[1][0] = "y";
$a[1][1] = "z";
foreach ($a as $v1) {
    foreach ($v1 as $v2) {
        echo "$v2\n";
    }
}
/* Пример 5: динамические массивы */
foreach (array(1, 2, 3, 4, 5) as $v) {
    echo "$v\n";
}
?>
```

6.8. break

break прерывает выполнение текущей структуры for, foreach, while, do-while или switch. break принимает необязательный числовой аргумент, который сообщает ему выполнение какого количества вложенных структур необходимо прервать.

```
<?php
$arr = array('один', 'два', 'три', 'четыре', 'стоп', 'пять');
while (list(, $val) = each($arr)) {
    if ($val == 'стоп') {
        break;    /* Тут можно было написать 'break 1;'. */
    }
    echo "$val<br />\n";
```

```
}
/* Использование дополнительного аргумента. */
$i = 0;
while (++$i) {
    switch ($i) {
    case 5:
        echo "Итерация 5<br />\n";
        break 1;  /* Выйти только из конструкции switch. */
    case 10:
        echo "Итерация 10; выходим<br />\n";
        break 2;  /* Выходим из конструкции switch и из цикла while. */
    default:
        break;
    }
}
?>
```

6.9. continue

continue используется внутри циклических структур для пропуска оставшейся части текущей итерации цикла и, при соблюдении условий, начала следующей итерации. continue принимает необязательный числовой аргумент, который указывает сколько итераций будет пропущено.

```
<?php
while (list($key, $value) = each($arr)) {
    if (!($key % 2)) { // пропуск нечетных чисел
        continue;
    }
    do_something_odd($value);
}
$i = 0;
while ($i++ < 5) {
    echo "Снаружи<br />\n";
    while (1) {
        echo "В середине<br />\n";
        while (1) {
            echo "Внутри<br />\n";
            continue 3;
        }
        echo "Это никогда не будет выведено.<br />\n";
    }
    echo "Это тоже.<br />\n";
}
?>
```

6.10. switch

Оператор switch подобен серии операторов IF с одинаковым условием. Во многих случаях вам может понадобиться сравнивать одну и ту же переменную (или выражение) с множеством различных значений, и выполнять различные участки кода в зависимости от того, какое значение принимает эта переменная (или выражение). Это именно тот случай, для которого удобен оператор switch.

```
<?php
if ($i == 0) {
    echo "i равно 0";
} elseif ($i == 1) {
    echo "i равно 1";
} elseif ($i == 2) {
    echo "i равно 2";
}

switch ($i) {
    case 0:
        echo "i равно 0";
        break;
    case 1:
        echo "i равно 1";
```

```
        break;
    case 2:
        echo "i равно 2";
        break;
}
?>
```

Важно понять, как оператор switch выполняется, чтобы избежать ошибок. Оператор switch исполняет строчка за строчкой (на самом деле выражение за выражением). В начале никакой код не исполняется. Только в случае нахождения оператора case, значение которого совпадает со значением выражения в операторе switch, PHP начинает исполнять операторы. PHP продолжает исполнять операторы до конца блока switch либо до тех пор, пока не встретит оператор break. Если вы не напишете оператор break в конце секции case, PHP будет продолжать исполнять команды следующей секции case.

Специальный вид конструкции case -- default. Сюда управление попадает тогда, когда не сработал ни один из других операторов case.

```
<?php
switch ($i) {
    case 0:
        echo "i равно 0";
        break;
    case 1:
        echo "i равно 1";
        break;
    case 2:
        echo "i равно 2";
        break;
    default:
       echo "i не равно 0, 1 или 2";
}
?>
```

Выражением в операторе case может быть любое выражение, которое приводится в простой тип, то есть в тип integer, или в тип с плавающей точкой (float), или строку. Массивы или объекты не могут быть здесь использованы до тех пор, пока они не будут разыменованы до простого типа.

Возможен альтернативный синтаксис для управляющей структуры switch.

```
<?php
switch ($i):
    case 0:
        echo "i равно 0";
        break;
    case 1:
        echo "i равно 1";
        break;
    case 2:
        echo "i равно 2";
        break;
    default:
        echo "i не равно to 0, 1 или 2";
endswitch;
?>
```

6.11. return

Если вызвано из функции, выражение return() немедленно прекращает выполнение текущей функции и возвращает свой аргумент как значение данной функции. return() также завершит выполнение выражения eval() или всего файла скрипта.

Если вызывается из глобальной области видимости, выполнение текущего файла скрипта прекращается. Если текущий файл скрипта был подключен с помощью функций include() или require(), тогда управление возвращается к файлу, который вызывал текущий. Более того, если текущий файл скрипта был подключен с помощью include(), тогла значение переданоне return() будет возвращено в качестве значения вызова include(). Если return() вызывается из главного файла скрипта, тогда выполнение скрипта прекращается.

6.12.include

Выражение include() включает и выполняет указанный файл.

Файлы включаются исходя из пути указанного файла, или, если путь не указан, используется путь, указанный в директиве include_path. Если файл не найден в include_path, include() попытается проверить директорию, в которой находится текущий включающий скрипт и текущую рабочую директорию перед тем, как выдать ошибку. Конструкция include() выдаст warning, если не сможет найти файл; поведение отлично от require(), который выдаст фатальную ошибку.

Когда файл включается, его код наследует ту же область видимости переменых, что и строка, на которой произошло включение. Все переменные, доступные на этой строке во включащем файле будут также доступны во включаемом файле. Однако все функции и классы, объявленные во включаемом файле, будут доступны в глобальной области видимости.

Пример # Простой пример include()

```
vars.php
<?php
$color = 'green';
$fruit = 'apple';
?>

test.php
<?php
echo "A $color $fruit"; // A
include 'vars.php';
echo "A $color $fruit"; // A green apple
?>
```

Если включение происходит внутри функции включающего файла, тогда весь код, содержащийся во включаемом файле, будет вести себя так, как будто он был определен внутри этой функции. То есть, он будет в той же области видимости переменных этой функции. Исключением к этому правилу являются магические константы, которые выполняются парсером перед тем, как происходит включение.

Пример # Включение внутри функции

```
<?php
function foo()
{
    global $color;
    include 'vars.php';
    echo "A $color $fruit";
}
/* vars.php в той же области видимости, что и foo(),  *
* поэтому $fruit НЕ доступен снаружи этой области     *
* $color доступен, поскольку мы переменную глобальной */
foo();                    // A green apple
echo "A $color $fruit";   // A green
?>
```

6.13. goto

Оператор goto используется для перехода в другую часть программы. Место, куда необходимо перейти указывается с помощью метки, за которой ставится двоеточие, после оператора goto указывается желаемая метка для перехода. Оператор не является неограниченным "goto". Целевая метка должна находиться в том же файле, в том же контексте. Имеется ввиду, что вы не можете ни перейти за границы функции или метода, ни перейти внутрь одной из них. Вы также не можете перейти внутрь любой циклической структуры или оператора switch. Но вы можете выйти из них, и обычным применением оператора goto является использование его вместо многоуровневых break.

Пример # Использование goto

```
<?php
goto a;
echo 'Foo';
```

```
a:
echo 'Bar';
?>
```

Результат выполнения данного примера:

```
Bar
```

Пример # Использование goto в цикле

```
<?php
for($i=0,$j=50; $i<100; $i++) {
  while($j--) {
    if($j==17) goto end;
  }
}
echo "i = $i";
end:
echo 'j hit 17';
?>
```

Результат выполнения данного примера:

```
j hit 17
```

7. Функции

7.1. Функции, определяемые пользователем

Пример # Псевдокод для демонстрации использования функций

```
<?php
function foo($arg_1, $arg_2, /* ..., */ $arg_n)
{
    echo "Example function.\n";
    return $retval;
}
?>
```

Имена функций следуют тем же правилам, что и другие метки в PHP. Корректное имя функции начинается с буквы или знака подчеркивания, за которым следует любое количество букв, цифр или знаков подчеркивания. В качестве регулярного выражения оно может быть выражено так: [a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*.

Функции не обязаны быть определены до их использования, *исключая* тот случай, когда функции определяются условно, как это показано в двух последующих примерах.

Пример # Функции, зависящие от условий

```
<?php
$makefoo = true;
/* Мы не можем вызвать функцию foo() в этом месте,
   поскольку она еще не определена, но мы можем обратиться к bar() */
bar();
if ($makefoo) {
  function foo()
  {
    echo "Я не существую до тех пор, пока выполнение программы меня не достигнет.\n";
  }
}
/* Теперь мы благополучно можем вызывать foo(),
   поскольку $makefoo была интерпретирована как true */
if ($makefoo) foo();
function bar()
{
  echo "Я существую сразу с начала старта программы.\n";
}
?>
```

Пример # Вложенные функции

```
<?php
function foo()
{
  function bar()
  {
    echo "Я не существую пока не будет вызвана foo().\n";
  }
}
/* Мы пока не можем обратиться к bar(), поскольку она еще не определена. */
foo();
/* Теперь мы можем вызвать функцию bar(), обработка foo() сделала ее доступной. */
bar();
?>
```

Все функции и классы PHP имеют глобальную область видимости - они могут быть вызваны вне функции, даже если были определены внутри и наоборот.

PHP не поддерживает перегрузку функции, также отсутствует возможность переопределить или удалить объявленную ранее функцию.

7.2. Аргументы функции

Функция может принимать информацию в виде списка аргументов, который является списком разделенных запятыми выражений. Аргументы вычисляются слева направо.

PHP поддерживает передачу аргументов по значению (по умолчанию), передачу аргументов по ссылке, и значения по умолчанию. Списки аргументов переменной длины также поддерживаются.

Пример # Передача массива в функцию

```
<?php
function takes_array($input)
{
    echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
?>
```

По умолчанию аргументы в функцию передаются по значению (это означает, что если вы измените значение аргумента внутри функции, то вне ее значение все равно останется прежним). Если вы хотите разрешить функции модифицировать свои аргументы, вы должны передавать их по ссылке.

Если вы хотите, что бы аргумент всегда передавался по ссылке, вы можете указать амперсанд (&) перед именем аргумента в описании функции:

Пример # Передача аргументов по ссылке

```
<?php
function add_some_extra(&$string)
{
    $string .= 'и кое-что еще.';
}
$str = 'Это строка, ';
add_some_extra($str);
echo $str;    // выведет 'Это строка, и кое-что еще.'
?>
```

Функция может определять значения по умолчанию в стиле C++ для скалярных аргументов.

Пример # Использование значений по умолчанию в определении функции

```
<?php
function makecoffee($type = "капуччино")
{
    return "Готовим чашку $type.\n";
}
echo makecoffee();
echo makecoffee(null);
echo makecoffee("эспрессо");
?>
```

Результат выполнения данного примера:

```
Готовим чашку капуччино.
Готовим чашку .
Готовим чашку эспрессо.
```

Обратите внимание, что все аргументы, для которых установлены значения по умолчанию, должны находиться правее аргументов, для которых значения по умолчанию не заданы, в противном случае ваш код может работать не так, как вы этого ожидаете.

Пример # Использование значений по умолчанию

```
<?php
function makeyogurt($flavour, $type = "ацидофил")
{
    return "Готовим чашку из бактерий $type со вкусом $flavour.\n";
}

echo makeyogurt("малины");   // отрабатывает правильно
?>
```

Результат выполнения данного примера:

```
Готовим чашку из бактерий ацидофил со вкусом малины.
```

7.3. Возврат значений

Значения возвращаются при помощи необязательного оператора возврата. Возвращаемые значения могут быть любого типа, в том числе это могут быть массивы и объекты. Возврат приводит к завершению выполнения функции и передаче управления обратно к той строке кода, в которой данная функция была вызвана. Для получения более детальной информации ознакомьтесь с описанием return().

Пример # Использование конструкции return()

```
<?php
function square($num)
{
    return $num * $num;
}
echo square(4);   // выводит '16'.
?>
```

Функция не может возвращать несколько значений, но аналогичного результата можно добиться, возвращая массив.

Пример # Возврат нескольких значений в виде массива

```
<?php
function small_numbers()
{
    return array (0, 1, 2);
}
list ($zero, $one, $two) = small_numbers();
?>
```

Для того, чтобы функция возвращала результат по ссылке, вам необходимо использовать оператор & и при описании функции, и при присвоении переменной возвращаемого значения:

Пример # Возврат результата по ссылке

```
<?php
function &returns_reference()
{
    return $someref;
}
$newref =& returns_reference();
?>
```

7.4. Обращение к функциям через переменные

PHP поддерживает концепцию переменных функций. Это означает, что если к имени переменной присоединены круглые скобки, PHP ищет функцию с тем же именем, что и результат вычисления переменной, и пытается ее выполнить. Эту возможность можно использовать для реализации обратных вызовов, таблиц функций и множества других вещей.

Пример # Работа с функциями посредством переменных

```
<?php
function foo() {
    echo "In foo()<br />\n";
}
function bar($arg = '')
{
    echo "In bar(); argument was '$arg'.<br />\n";
}
// Функция-обертка для echo
function echoit($string)
{
    echo $string;
}
$func = 'foo';
$func();        // Вызывает функцию foo()
$func = 'bar';
$func('test');  // Вызывает функцию bar()
$func = 'echoit';
$func('test');  // Вызывает функцию echoit()
?>
```

Пример # Пример вызова переменного метода со статическим свойством

```
<?php
class Foo
{
    static $variable = 'static property';
    static function Variable()
    {
        echo 'Method Variable called';
    }
}

echo Foo::$variable; // Это выведет 'static property'. Переменная $variable будет раз
решена в нужной области видимости.
$variable = "Variable";
Foo::$variable();  // Это вызовет $foo-
>Variable(), прочитав $variable из этой области видимости.
?>
```

7.5. Анонимные функции

Анонимные функции, также известные как замыкания (closures), позволяют создавать функции, не имеющие определенных имен. Они наиболее полезны в качестве значений callback-параметров, но также могут иметь и множество других применений.

Пример # Пример анонимной функции

```
<?php
echo preg_replace_callback('~-([a-z])~', function ($match) {
    return strtoupper($match[1]);
}, 'hello-world');
// выведет helloWorld
?>
```

Пример # Пример присвоения анонимной функции переменной

```
<?php
$greet = function($name)
{
    printf("Hello %s\r\n", $name);
};
$greet('World');
$greet('PHP');
?>
```

8. Суперглобальные переменные

8.1. $GLOBALS - Ссылки на все переменные глобальной области видимости

Ассоциативный массив (array), содержащий ссылки на все переменные глобальной области видимости скрипта, определенные в данный момент. Имена переменных являются ключами массива.

```
<?php
function test() {
    $foo = "local variable";
    echo '$foo in global scope: ' . $GLOBALS["foo"] . "\n";
    echo '$foo in current scope: ' . $foo . "\n";
}

$foo = "Example content";
test();
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
$foo in global scope: Example content
$foo in current scope: local variable
```

8.2. $_SERVER - Информация о сервере и среде исполнения

Переменная *$_SERVER* - это массив, содержащий информацию, такую как заголовки, пути и местоположения скриптов. Записи в этом массиве создаются веб-сервером. Нет гарантии, что каждый веб-сервер предоставит любую из них; сервер может опустить некоторые из них или предоставить другие, не указанные здесь.

```
<?php
echo $_SERVER['SERVER_NAME'];
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
www.example.com
```

8.3. $_GET - GET-переменные HTTP

Ассоциативный массив параметров, переданных скрипту через URL.

$HTTP_GET_VARS содержит аналогичный набор данных, но не является суперглобальным. (Заметьте, что *$HTTP_GET_VARS* и *$_GET* являются разными переменными и обрабатываются PHP независимо друг от друга)

```
<?php
echo 'Привет ' . htmlspecialchars($_GET["name"]) . '!';
?>
```

Подразумевается, что пользователь ввел в браузере адрес http://example.com/?name=Hannes

Результатом выполнения данного примера будет что-то подобное:

```
Привет Hannes!
```

8.4. $_POST - HTTP POST variables

Ассоциативный массив данных, переданных скрипту через HTTP метод POST.

```
<?php
echo 'Привет ' . htmlspecialchars($_POST["name"]) . '!';
?>
```

Подразумевается, что пользователь отправил через POST name=Hannes

Результатом выполнения данного примера будет что-то подобное:

```
Привет Hannes!
```

8.5. $_FILES - Переменные файлов, загруженных по HTTP

Ассоциативный массив (array) элементов, загруженных в текущий скрипт через метод HTTP POST.

8.6. $_SESSION - Переменные сессии

Ассоциативный массив, содержащий переменные сессии, которые доступны для текущего скрипта. Смотрите документацию по функциям сессии для получения дополнительной информации.

8.7. $_ENV - Переменные окружения

Ассоциативный массив (array) значений, переданных скрипту через переменные окружения.

Эти значения импортируются в глобальное пространство имен PHP из системных переменных окружения, в котором запущен парсер PHP. Большинство значений передаётся из командной оболочки, под которой PHP запущен, и различных системных приложений, полного и точного списка не существует. Пожалуйста, изучите документацию к вашей командной оболочке для получения списка переменных окружения.

```
<?php
echo 'Мое имя пользователя: ' .$_ENV["USER"] . '!';
?>
```

Допустим, скрипт запустил "bjori"

Результатом выполнения данного примера будет что-то подобное:

```
Мое имя пользователя: bjori!
```

8.8. $_COOKIE - HTTP Куки

Ассоциативный массив (array) значений, переданных скрипту через HTTP Куки.

```
<?php
echo 'Привет, ' . htmlspecialchars($_COOKIE["name"]) . '!';
?>
```

Положим, что значение куки с именем "name" было установлено равным "Ханнес".

Результатом выполнения данного примера будет что-то подобное:

```
Привет, Ханнес!
```

II. Работа с СУБД MySQL

1. Поля и их типы в MySQL

База данных с точки зрения MySQL (и некоторых других СУБД) - это обыкновенный каталог, содержащий двоичные файлы определенного формата - таблицы. Таблицы состоят из записей, а записи, в свою очередь, состоят из полей. Поле имеет два атрибута - имя и тип.

Тип поля может быть:

- Целым;
- Вещественным;
- Строковым;
- Бинарным;
- Дата и время;
- Перечисления и множества.

1.1. Целочисленные типы данных

Тип	Диапазон
TINYINT	-128...+127
SMALLINT	-32768...+32767
MEDIUMINT	-8 388 608...+8 388 607
INT	-2 147 483 648...+2 147 483 647
BIGINT	-9 223 372 036 854 775 808...+9 223 372 036 854 775 807

Вещественные типы записываются в виде:

```
ТИП (ДЛИНА, ЗНАКИ) [UNSIGNED]
```

Длина - это колличество знакомест, в которых будет размещено все число при его передаче, а ЗНАКИ - это колличество знаков после десятичной точки, которые будут учитываться. Если указан модификатор UNSIGNED, знак числа учитываться не будет.

1.2. Вещественные числа

Тип	Описание
FLOAT	Небольшая точность
DOUBLE	Двойная точность
REAL	То же, что и DOUBLE
DECIMAL	Дробное число, хранящееся в виде строки
NUMERIC	То же, что и DECIMAL

Любая строка - это массив символов. При поиске с помощью оператора SELECT (мы рассмотрим его далее) не учитывается регистр символов: строки "HELLO" и "Hello" считаются одинаковыми.

Можно настроить MySQL на автоматическое перекодирование символов - в этом случае в базе данных строки будут храниться в одной кодировке, а выводиться - в другой.

В большинстве случаев применяется тип VARCHAR или просто CHAR, позволяющий хранить строки, содержащие до 255 символов. В скобках после типа указывается длина строки:

```
VARCHAR(48);
CHAR(73);
```

Если 255 символов для вашей задачи недостаточно, можно использовать другие типы, например, TEXT.

1.3. Строки

Тип	Описание
TINYTEXT	Максимальная длина 255 символов
TEXT	Максимальная длина 65535 символов (64 Кб)
MEDIUMTEXT	Максимальная длина 16 777 215 символов
LONGTEXT	Максимальная длина 4 294 967 295 символов

1.4. Бинарные типы данных

Тип	Описание
TINYBLOB	Максимум 255 символов
BLOB	Максимум 65535 символов
MEDIUMBLOB	Максимум 16 777 215 символов
LONGBLOB	Максимум 4 294 967 295

Бинарные типы данных также можно использовать для хранения текста, но при поиске будет учитываться регистр символов. К тому же, любой текстовый тип можно преобразовать в бинарный, указав модификатор BINARY:

```
VARCHAR(30) BINARY;
```

1.5. Дата и время

Тип	Описание
DATE	Дата в формате ГГГ-ММ-ДД
TIME	Время в формате ЧЧ:ММ:СС
TIMESTAMP	Дата и время в формате timestamp, выводится в виде ГГГГММДДЧЧММСС
DATETIME	Дата и время в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС

2. Операторы и команды MySQL

2.1. Создание таблиц. Оператор CREATE

Создать таблицу через SQL-запрос позволяет оператор CREATE. Его синтаксис:

```
CREATE TABLE Имя_таблицы
(
Имя_поля1 Тип Модификатор
...
Имя_поляN Тип Модификатор
[первичный ключ]
[внешний ключ]
)
```

Вообще, с помощью операторa CREATE можно создавать и другие объекты, но мы их рассматривать не будем, поскольку их применение весьма ограничено.

В качестве модификаторов можно использовать следующие значения:

- NOT NULL - поле не может содержать неопределенного значения (NULL), то есть поле должно быть явно инициализировано;
- PRIMARY KEY - поле будет первичным ключом (идентификатором записи), по которому можно однозначно идентифицировать запись;
- AUTO_INCREMENT - при вставке новой записи значение этого поля будет автоматически увеличено на единицу, поэтому в таблице не будет двух записей с одинаковым значением этого поля;
- DEFAULT - задает значение, которое будет использовано по умолчанию, если при вставке записи поле не будет инициализировано явно. Значение по умолчанию задается так:

Имя_поля	Тип	DEFAULT	Значение, например
NO	INT	DEFAULT	0
NAME	INT	DEFAULT	'Петров'

Теперь создадим таблицы - "Товар", "Клиенты", "Заказы":

```
CREATE TABLE CLIENTS
(
C_NO     int       NOT NULL,
FIO      char(40)  NOT NULL,
ADDR     char(30)  NOT NULL,
CITY     char(15)  NOT NULL,
PHONE    char(11)  NOT NULL
);
```

Таблица CLIENTS содержит поля C_NO (номер клиента), FIO (Фамилия, Имя, Отчество), ADDR (Адрес), CITY (Город) и PHONE (Телефон). Все эти поля не могут содержать пустого значения (NOT NULL).

```
CREATE TABLE TOOLS
(
T_NO   int          NOT NULL,
DSEC   char(40)     NOT NULL,
PRICE  double(9,2)  NOT NULL,
QTY    double(9,2)  NOT NULL
);
```

Данная таблица будет содержать данные о товарах. Тип double(9,2) означает, что 9 знаков относим под целую чать, и два - под дробную. QTY - это колличество товара на складе.

```
CREATE TABLE ORDERS
(
O_NO     int         NOT NULL,
DATE     date        NOT NULL,
C_NO     int         NOT NULL,
T_NO     int         NOT NULL,
QUANTITY double(9,2) NOT NULL,
AMOUNT   double(9,2) NOT NULL
);
```

Эта таблица содержит сведения о заказах - номер заказа (O_NO), дату заказа (DATE), номер клиента (C_NO), номер товара (T_NO), колличество (QUANTITY) и сумму всего заказа AMOUNT (то есть AMOUNT = T_NO * TOOL_PRICE).

2.2. Добавление данных в таблицу. Оператор INSERT

Для добавления записей используется оператор INSERT:

```
INSERT INTO Имя_таблицы [(Список полей)]
VALUES (Список констант);
```

После выполнения оператора INSERT будет создана новая запись, в качестве значений полей будут использованы соответствующие константы, указанные в списке VALUES.

Теперь добавим данные в наши таблице. Добавить данные можно с помощью оператора INSERT. Рассмотрим пример использование оператора INSERT:

```
INSERT INTO CLIENTS
VALUES (1, 'Иванов И.И.', 'Вокзальная 3', 'Москва', '09599911100');
```

Добавляемые значения должны соответствовать тому порядку, в котором поля перечислены в операторе CREATE. Если вы хотите добавлять информацию в другом порядке, то вы должны указать этот порядок в операторе INSERT, например:

```
INSERT INTO CLIENTS (FIO, ADDRESS, C_NO, PHONE, CITY)
VALUES ('Петров', 'Мира 29', 2, '-', 'Екатеринбург');
```

С помощью INSERT мы можем добавлять данные и в определенные поля, например, C_NO и FIO:

```
INSERT INTO CLIENTS (C_NO, FIO)
VALUES (1, 'Иванов');
```

Однако, в нашем случае сервер MySQL не выполнит такой запрос, поскольку все остальные поля равны NULL (пустое значение), а наша таблица не принимает пустые значения. Аналогично можно добавить данные в другие таблицы.

В качестве примера, добавим данные в таблицу TOOLS:

```
INSERT INTO TOOLS
VALUES (1, 'Клавиатура ABC', 340.98, 5);
```

Обратите внимание, что мы пока не указали первичные ключи таблицы, поэтому нам никто не мешает добавить в таблицу одинаковые записи. Добавить дату в поле DATE можно с помощью функции TO_DATE:

```
INSERT INTO ORDERS
VALUES (1, TO_DATE('01/03/05, 'DD/MM/YY'),1,1,1,340.98);
```

Данная запись означает, что первого марта 2005 года Иванов И.И. (C_NO=1) заказал одну (QUANTITY=1) клавиатуру ABC (T_NO=1).

2.3. Обновление записей. Оператор UPDATE

Синтаксис оператора UPDATE, который используется для обновления записей, выглядит так:

```
UPDATE Имя_таблицы
SET Поле1 = Значение1, ... , ПолеN = ЗначениеN
[WHERE Условие];
```

Если не задано условие WHERE, будет модифицирована вся таблица, а это может повлечь за собой непредсказуемые последствия, поскольку для всех записей будут установлены одинаковые значения полей, поэтому всегда указывайте условие WHERE.

Предположим, нам необходимо обновить запись, если, например, клиент Иванов переехал в другой город и нам нужно отметить это событие в базе данных. Сделаем следующее:

```
UPDATE CLIENTS
SET CITY = 'Псков'
WHERE C_NO = 1;
```

Данный запрос нужно понимать так: найти запись, поле C_NO которой = 1 (это код клиента Иванова), и установить значение CITY равным "Псков".

2.4. Удаление записей. Оператор DELETE

Если нам необходимо удалить всех клиентов, номера которых превышают 5, то мы поступим следующим образом:

```
DELETE FROM CLIENTS
WHERE C_NO > 5;
```

С помощью оператора DELETE можно удалить все записи таблицы, указав условие, которое подойдет для всех записей, например:

```
DELETE FROM CLIENTS;
```

Если вторая часть оператора DELETE-WHERE не указана, занчит, действие оператора распространяется на все записи сразу.

2.5. Выбор записей. Оператор SELECT

Добавление, изменение и удаление записей - это, конечно, очень важные команды, но вы часто будете использовать оператор SELECT, который выбирает данные из таблицы. Синтаксис этого оператора более сложен:

```
SELECT [DISTINCT|ALL] {*| [поле1 AS псевдоним] [,..., полеN AS псевдоним]}
FROM Имя_таблицы1 [,..., Имя_таблицыN]
[WHERE условие]
[GROUP BY список полей] [HAVING условие]
[ORDER BY список полей]
```

Мы полностью не будем рассматривать оператор SELECT, лучше это делать на конкретном примере. Сейчас мы рассмотрим оператор SELECT в общих чертах. Например, для вывода всех записей из таблицы CLIENTS сделайте следующее:

```
SELECT * FROM CLIENTS;
```

В результате вы получите следующий ответ сервера:

C_NO	FIO	ADDR	CITY	PHONE
1	Иванов И.И.	Вокзальная 3	Москва	09599911100
1	Иванов И.И.	Вокзальная 3	Москва	09599911100
2	Петров П.П.	Мира 29	Екатеринбург	3438920437

Обратите внимание на первые две записи - они одинаковые. Теоретически, добавление одинаковых записей возможно - ведь мы не указали первичный ключ таблицы. Если вы хотите исключить одинаковые записи из ответа сервера (но не из таблицы), используйте запрос:

```
SELECT DISTINCT *
FROM CLIENTS;
```

Предположим, вы хотите вывести только фамилию и номер телефона клиента, тогда используйте следующий запрос:

```
SELECT DISTINCT FIO, PHONE
FROM CLIENTS;
```

Если вам нужно вывести все товары, цена на которые превышает 800, то воспользуйтесь таким запросом:

```
SELECT *
FROM TOOLS
WHERE PRICE > 800;
```

Вы можете использовать следующие операторы отношений: <, >, =, <>, <=, >=.

Если в вашей таблице присутствуют несколько однофамильцев, то для вывода информации обо всех из них, используйте модификатор LIKE, например:

```
SELECT *
FROM CLIENTS
WHERE FIO LIKE '%Иванов%';
```

Приведенный запрос можно причитать так: вывести информацию о клиентах, фамилия которых похожа на 'Иванов'.

Если вам необходимо выбрать данные из разных таблиц, то перед именем поля нужно указывать имя таблицы. Вот запрос, который позволяет вывести имена всех клиентов, которые хотя бы один раз покупали товар:

```
SELECT DISTINCT CLIENTS.FIO
FROM CLIENTS, ORDERS
```

```
WHERE CLIENTS.C_NO = ORDERS.C_NO;
```

Оператор SELECT позволяет ипользовать вложенные запросы, однако MySQL их не поддерживает.

2.6. Внутренние функции MIN, MAX, AVG, SUM

При работе с оператором SELECT вам доступны несколько очень полезных внутренних функций MySQL, вычисляющих колличество элементов (COUNT), сумму элементов (SUM), максимальное и минимальное значения (MAX и MIN), а также среднее значение (AVG).

Следующие операторы выведут, соответственно, колличество записей в таблице CLIENTS, самый дорогой товар и сумму цен всех товаров:

```
SELECT COUNT(*)
FROM CLIENTS;

SELECT MAX(PRICE)
FROM TOOLS;

SELECT SUM(PRICE)
FROM TOOLS;
```

2.7. Группировка записей

Оператор SELECT позволяет группировать возвращаемые значения. Например, клиент Иванов (C_NO=1) несколько раз заказывал какой-то товар. Значит, его номер встречается в таблице ORDERS несколько раз. Другой клиент также мог сделать несколько заказов. Мы можем сгруппировать все записи по полю C_NO (номер клиента), а затем вывести сумму заказа каждого клиента.

```
SELECT CLIENTS.FIO, SUM(ORDERS.AMOUNT) AS TOTALSUM
FROM CLIENTS, ORDERS
WHERE CLIENTS.C_NO = ORDERS.C_NO
GROUP BY ORDERS.C_NO;
```

Группировку выполняет оператор GROUP BY, который является частью оператора SELECT. Оператор GROUP BY можно ограничить с помощью HAVING. Этот оператор используется для отбора строк, возвращаемых GROUP BY. HAVING можно считать аналогом WHERE, но только для GROUP BY:

```
HAVING <условие>
```

Например, нас интересуют только клиенты, которые заказали товаров на общую сумму, превышающую 1500:

```
SELECT CLIENTS.FIO, SUM(ORDERS.AMOUNT) AS TOTALSUM
FROM CLIENTS, ORDERS

WHERE CLIENTS.C_NO = ORDERS.C_NO
GROUP BY ORDERS.C_NO
HAVING TOTALSUM > 1500;
```

В этом запросе мы использовали псевдоним столбца TOTALSUM. В некоторых серверах SQL для определения псевдонима не нужно писать служебное слово AS, а некоторые требуют применения знака равенства:

```
SUM(ORDERS.AMOUNT) TOTALSUM или TOTALSUM = SUM(ORDERS.AMOUNT)
```

2.8. Сортировка записей

Пока мы не установили первичный ключ, сортировка таблицы не выполняется. Данные будут отображены в порядке их занесения в таблицу. Для сортировки по полю C_NO результата вывода таблицы CLIENTS ипользуется следующий оператор (сама таблица при этом не сортируется):

```
SELECT *
FROM CLIENTS
ORDER BY C_NO;
```

2.9. Ключи

Предположим, что кто-то добавил в таблицу CLIENTS запись:

```
1 Сидоров Свободы 7 Калининград 0113452103
```

В то же время, до этого номер 1 был закреплен за Ивановым. У нас получилось, что один и тот же номер сопоставлен разным клиентам. Чтобы избежать такой путаницы, необходимо ипользовать первичные ключи:

```
ALTER TABLE CUSTOMER
ADD PRIMARY KEY (C_NO);
```

После этого запроса поле C_NO может содержать только уникальные значения. В качестве первичного ключа нельзя использовать поле, доспускающее значение NULL. Создать певичный ключ можно и проще - при создании таблицы следующим образом:

```
CREATE TABLE CLIENTS
(
C_NO    int      NOT NULL,
FIO     char(50) NOT NULL,
ADDR    char(55) NOT NULL,
CITY    char(20) NOT NULL,
PHONE   char(8)  NOT NULL,
PRIMARY KEY (C_NO);
);
```

Таблица ORDERS содержит сведения о заказах. По полю C_NO этой таблице идентифицируется заказчик. Предположим, что в таблицу ORDERS кто-то ввел значение, которого нет в таблице CLIENTS. Кто заказал товар? Нам нужно не допустить подобной ситуации, поэтому следует использовать подобный запрос:

```
ALTER TABLE ORDERS
ADD FOREIGN KEY(C_NO) REFERENCES CLIENTS;
```

Введенные в таблицу ORDERS номера клиентов C_NO должны существовать в таблице CLIENTS. Аналогично нужно добавить внешний ключ по полю T_NO. Эта возможность называется декларативной целостностью.

Команда ALTER используется не только для добавления ключей. Она предназначена для реорганизации таблицы в целом. Вы хотите добавить еще одно поле? Или установить список допустимых значений для каждого из полей. Все это можно сделать с помощью команды ALTER:

```
ALTER TABLE CLIENTS
ADD ZIP char(7) NULL;
```

Этот оператор добавляет в таблицу CLIENTS новое поле ZIP типа char. Обратите внимание, что вы не можете добавить новое поле со значением NOT NULL в таблицу, в которой уже есть данные. Например, если компания работает только с клиентами Москвы и Екатеринбурга, то целесообразно ввести список допустимых значений для таблицы CLIENTS:

```
ALTER TABLE CLIENTS
ADD CONSTRAINT INVALID_STATE SHECK (CITY IN ('Москва', 'Екатеринбург'));
```

2.10. Использование внешних ключей

Теперь углубимся в изучение SQL. Вы уже знаете, как добавлять первичный ключ, теперь добавим внешний ключ при создании таблицы. Внешние ключи используются для связи одной таблицы с другой. Например, в таблице CLIENTS у нас есть два клиента - Иванов (C_NO=1) и Петров (C_NO=2). Оператор в магазине при оформлении заказа ошибся и указал несуществующий номер, например, C_NO=3. Как мы потом сможем идентифицировать клиента? Для решения такой проблемы и существуют внешние ключи:

```
CREATE TABLE T
(
/* Описание полей таблицы */
FOREING KEY KEY_NAME (LIST)
REFERENCES ANOTHER_TABLE [(LIST2)]
[ON DELETE OPTION]
[ON UPDATE OPTION]
);
```

Здесь:

- KEY_NAME - Имя ключа. Имя не является обязательным, но рекомендуется всегда указывать имя ключа - если вы не укажите имя ключа, вы потом не сможете его удалить;
- LIST - это список полей, входящих во внешний ключ. Список разделяется запятыми;

• ANOTHER_TABLE - это другая таблица, по которой устанавливается не внешний ключ, а необязательный элемент;

• LIST2 - это список полей этой таблицы. Типы полей в списке LIST должны совпадать с типами полей в списке LIST2.

Предположим, что в первой таблице у нас есть поля - NO и NAME - целого и символьного типов соответственно. Во второй таблице у нас есть поля с одинаковыми именами и тапами. Определение внешнего ключа:

```
FOREIGN KEY KEY_NAME (NO, NAME)
REFERENCES ANOTHER_TABLE (NAME, NO)
```

Это определение некорректно, потому что типы полей NO и NAME не совпадают. Нужно использовать такое определение:

```
FOREIGN KEY KEY_NAME (NO, NAME)
REFERENCES ANOTHER_TABLE (NO, NAME)
[ON DELETE <OPTION>]
[ON UPDATE <OPTION>]
```

Если поля имеют одинаковае имена, как в нашем случае, список LIST2 лучше вообще не указывать.

Необязательные параметры ON DELETE <OPTION> и ON UPDATE <OPTION> определяют действие по обновлению информации в базе данных, при удалении информации из таблицы и при ее обновлении. А действия могут быть следующими:

• CASCADE - удаление или обновление значений везде, где оно встречается. Например, у нас есть таблица клиентов и заказов. Иы хотим удалить запись клиента с номером C_NO=1. Из таблицы заказов будут удалены сведения обо всех заказах, сделанных клиентом;

• NOACTION - вы не сможете удалить информацию из таблицы клиентов до тех пор, пока вы не удалите все заказы, сделанные этим клиентом. То есть действие NOACTION запрещает удалять запись из основной таблицы, если она используется в дочерней таблице;

• SETNULL - все значения в дочерней таблице будут заменены на NULL (если значения NULL допускаются);

• С помощью параметра SET_DEFAULT вы можете указать значение по умолчанию. Например, если вы укажите SET_DEFAULT 1, то при удалении клиента с любым номером его заказы бубудт приписываться клиенту с номером 1, который есть в таблице CLIENTS.

2.11. Удаление полей и таблиц. Оператор DROP

Стандартом SQL не предусмотрено удаление столбцов, однако в MySQL мы это можем сделать:

```
ALTER TABLE CLIENTS
DROP ZIP;
```

А удалить таблицу еще проще:

```
DROP ORDERS;
```

2.12. Отключение от СУБД

Используя запрос DISCONNECT можно отключиться от используемой базы данных, а затем, используя запрос CONNECT, подключиться к другой базе данных. В некоторых серверах SQL запрос DISCONNECT не работает, а вместо CONNECT применяется запрос USE.

При использовании PHP нет необходимости использовать данные запросы, поскольку для отключения от сервера MySQL используется функция mysql_close(), а для подключения к серверу MySQL используется функция mysql_connect().

3. Функции PHP для работы с MySQL по категориям

3.1. Функции соединения с сервером MySQL

Основной функцией для соединения с сервером MySQL является mysql_connect(), которая подключает скрипт к серверу баз данных MySQL и выполяет авторизацию пользователя базой данных. Синтаксис у данной функции такой:

```
mysql_connect ([string $hostname] [, string $user] [, sting $password]);
```

Как вы наверно заметили, все параметры данной функции являются необязательными, поскольку значения по умолчанию можно прописать в конфигурационном файле php.ini. Если вы хотите указать другие имя MySQL-хоста, пользователя и пароль, вы всегда можете это сделать. Параметр $hostname может быть указан в виде: хост:порт.

Функция возвращает идентификатор (типа int) соединения, вся дальнейщая работа осуществляется только через этот идентификатор. При следующем вызове функции mysql_connect() с теми же параметрами новое соединение не будет открыто, а функция возвратит идентификатор существующего соединения.

Для закрытия соединения предназначена функция mysql_close(int $connection_id).

Вообще, соединение можно и не закрывать - оно будет закрыто автоматически при завершении работы PHP скрипта. Если вы используете более одного соединения, при вызове mysql_close() нужно указать идентификатор соединения, которое вы хотите закрыть. Вообще не закрывать соединения - плохой стиль, лучше закрывать соединения с MySQL самостоятельно, а не надеясь на автоматизм PHP, хотя это ваше право.

Если вы будете использовать только одно соединение с базой данных MySQL за все время работы сценария, можно не сохранять его идентификатор и не указывать идентификатор при вызове остальных функций.

Функция mysql_connect() устанавливает обыкновенное соединение с MySQL. Однако, PHP поддерживает постоянные соединения - для этого используйте функцию mysql_pconnect(). Аргументы этой функции такие же, как и у mysql_connect().

В чем разница между постоянным соединением и обыкновенным соединением с MySQL? Постоянное соединение не закрывается после завершения работы скрипта, даже если скрипт вызвал функцию mysql_close(). Соединение привязывается к PID потомка веб сервера Apache (от имени которого он и работает) и закрывается лишь тогда, когда удаляется процесс-владелец (например, при завершении работы или перезагрузке веб-сервера Apache).

PHP работает с постоянными соединениями примерно так: при вызове функции mysql_pconnect() PHP проверяет, было ли ранее установлено соединение. Если да, то возвращается его идентификатор, а если нет, то открывается новое соединение и возвращается идентификатор.

Постоянные соединения позволяют значительно снизить нагрузку на сервер, а также повысить скорость работы PHP скриптов, использующих базы данных.

При работе с постоянными соединениями нужно следить, чтобы максимальное число клиентов Apache не преывшало максимального числа клиентов MySQL, то есть параметр MaxClient (в конфигурационном файле Apache - httpd.conf) должен быть меньше или равен параметру max_user_connection (параметр MySQL).

3.2. Функция выбора базы данных

Функция mysql_select_db (string $db [, int $id]) выбирает базу данных, с которой будет работать PHP скрипт. Если открыто не более одного соединения, можно не указывать параметр $id.

```
// Попытка установить соединение с MySQL:
if (!mysql_connect($server, $user, $ password)) {
echo "Ошибка подключения к серверу MySQL";
exit;
}
// Соединились, теперь выбираем базу данных:
mysql_select_db($db);
```

3.3. Функции обработки ошибок

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

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

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

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

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

3.4. Функции выполнения запросов к серверу баз данных

Все запросы к текущей базе данных отправляются функцией mysql_query(). Этой функции нужно передать один параметр - текст запроса. Текст запроса модет содержать пробельные символы и символы новой строки (\n). Текст должен быть составлен по правилам синтаксиса SQL. Пример запроса:

```
$q = mysql_query("SELECT * FROM mytable");
```

Приведенный запрос должен вернуть содержимое таблицы mytable. Результат запроса присваивается переменной $q. Результат - это набор данных, который после выполнения запроса нужно обработать определенным образом.

3.5. Функции обработки результатов запроса

Если запрос, выполненный с помощью функции mysql_query() успешно выполнился, то в результате клиент получит набор записей, который может быть обработан следующими функциями PHP:

- mysql_result() - получить необходимый элемент из набора записей;
- mysql_fetch_array() - занести запись в массив;
- mysql_fetch_row() - занести запись в массив;
- mysql_fetch_assoc() - занести запись в ассоциативный массив;
- mysql_fetch_object() - занести запись в объект.

Также можно определить количество содержащихся записей и полей в результате запроса. Функция mysql_num_rows() позволяет узнать, сколько записей содержит результат запроса:

```
$q = mysql_query("SELECT * FROM mytable");
echo "В таблице mytable ".mysql_num_rows($q)." записей";
```

Запись состоит из полей (колонок). С помощью функции mysql_num_fields() можно узнать, сколько полей содержит каждая запись результата:

```
$q = mysql_query("SELECT * FROM mytable");
echo "В таблице mytable ".mysql_num_fields($q)." полей ";
```

У нас также есть возможность узнать значение каждого поля. Это можно сделать с помощью следующей функции:

```
mysql_result (int $result, int $row, mixed $field);
```

Параметр функции $row задает номер записи, а параметр $field - имя или порядковый номер поля.

Предположим, SQL-запрос вернул следующий набор данных:

```
Email               Name      Last_Name
-------------------------------------------------
ivanov@mail.ru      Ivan      Ivanov
petrov@mail.ru      Petr      Petrov
```

Вывести это в браузер можно следующим образом:

```
$rows = mysql_num_rows($q);
$fields = mysql_num_fields($q);

echo "<pre>";
```

```
  for ($c=0; $c<$rows; $c++) {
     for ($cc=0; $cc<$fields; $cc++) {
     echo mysql_result($q, $c, $cc)."\t";
     echo "\n";
     }
  }
echo "</pre>";
```

Следует отметить, что функция mysql_result() универсальна: зная количество записей и количество полей, можно "обойти" весь результат, но в тоже время, скорость работы данной функции достаточно низка. Поэтому, для обработки больших наборов записей рекомендуется использовать функции mysql_fecth_row(), mysql_fecth_array(), и.т.д.

Функция mysql_fecth_row(int $res) получает сразу всю строку, соответствующую текущей записи результата $res. Каждый следующий вызов функции перемещает указатель запроса на следующую позицию (как при работе с файлами) и получает следующую запись. Если более нет записей, то функция возвращает FALSE. Пример использования данной функции:

```
$q = mysql_query("SELECT * FROM mytable WHERE month=\"$db_m\" AND day=\"$db_d\");
for ($c=0; $c<mysql_num_rows($q); $c++)
{
$f = mysql_fetch_row($q);
echo $f;
}
```

Использовать функцию mysql_fecth_row() не всегда удобно, так как значения всех полей одной записи находятся все в одной строке. Удобнее использовать функцию mysql_fetch_array(), которая возвращает ассоциативный массив, ключами которого будут имена полей.

Функция mysql_fecth_array(int $res [, int $result_type]) возвращает не ассоциативный массив, а массив, заданный необязательным параметром $result_type, который может принимать следующие значения:

- MYSQL_ASSOC - возвращает ассоциативный массив;
- MYSQL_NUM -возвращает массив с числовыми индексами, как в функции mysql_fecth_row();
- MYSQL_BOTH - возвращает массив с двойными индексами, то есть вы можете работать с ним, как с ассоциативным массивом и как со списком (MYSQL_BOTH - это значение по умолчанию для параметра $result_type.

В PHP есть функция, возвращающая ассоциативный массив с одним индексом:

```
mysql_fetch_assoc(int $res);
```

Фактически, данная функция является синонимом для mysql_fetch_array($res, MYSQL_ASSOC);

Пример использования функции mysql_fecth_array():

```
$q = mysql_query("SELECT * FROM mytable WHERE month=\"$db_m\" AND day=\"$db_d\");
for ($c=0; $c<mysql_num_rows($q); $c++)
{
$f = mysql_fetch_array($q);
echo "$f[email] $f[name] $f[month] $f[day] <br>";
}
```

Как видно, использовать функцию mysql_fetch_array() намного удобнее, чем mysql_fetch_row().

3.6. Функции получения информации о результатах SQL-запросов

PHP предоставляет еще несколько полезных функций, которые позволяют узнать информацию о результах SQL-запросов.

- Функция mysql_field_name(int $result, int $offset) возвращает имя поля, находящегося в результате $result с номером $offset (нумерация начинается с 0). Другими словами, функция возвращает имя поля с номером $offset.
- Функция mysql_field_type(int $result, int $offset) возвращает тип поля с номером $offset в результате $result (номер задается относительно результата, а не таблицы);

• Функция mysql_field_flags(int $result, int $offset) возвращает пречисленные через пробел флаги (модификаторы), которые имеются у поля с номером $offset. Перечислим все поддерживаемые MySQL флаги:

Флаг	**Описание**
not_Null	Поле не может содержать неопределенного значения (NULL), то есть поле должно быть явно инициализировано
Primary_Key	Поле будет первичным ключом - идентификатором записи, по которому можно однозначно идентифицировать запись;
auto_increment	При вставке новой записи значение этого поля будет автоматически увеличено на единицу, потому в таблице никогда не будет двух записей с одинаковым значением этого поля;
Unique_Key	Поле должно содержать уникальное значение;
Multiple_Key	Индекс
Blob	Поле может содержать бинарный блок данных
Unsigned	Поле содержит беззнаковые числа
Zerofill	Вместо пробелов используются символы с кодом \0
Binary	Поле содержит двоичные данные
enum	Поле может содержать один элемент из нескольких возможных (элемент перечисления)
timestamp	В поле автоматически заносится текущая дата и время при его модификации

Функция mysql_field_flags() возвращает флаги в виде строки, в которой флаги разделяются пробелами.

4. Функции PHP для работы с MySQL

mysql_affected_rows - Возвращает число затронутых прошлой операцией рядов

int **mysql_affected_rows** ([resource `$link_identifier`])

Возвращает количество рядов, затронутых последним INSERT, UPDATE, REPLACE или DELETE запросом, связанным с дескриптором link_identifier. Возвращает количество измененных записей в случае успеха, и -1 в случае если последний запрос не удался.

`link_identifier` Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня E_WARNING.

При использовании UPDATE, MySQL не обновит колонки, уже содержащие новое значение. Вследствие этого, функция mysql_affected_rows() не всегда возвращает количество рядов, подошедших под условия, только количество рядов, обновлённых запросом.

Запрос REPLACE сначала удаляет запись с указанным первичным ключом, а потом вставляет новую. Данная функция возвращает количество удаленных записей вместе с количеством вставленных.

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Ошибка соединения: ' . mysql_error());
}
mysql_select_db('mydb');
/* здесь функция вернёт корректное число удалённых записей */
mysql_query('DELETE FROM mytable WHERE id < 10');
printf("Удалено записей: %d\n", mysql_affected_rows());
/* если WHERE всегда возвращает false, то функция возвращает 0 */
mysql_query('DELETE FROM mytable WHERE 0');
printf("Удалено записей: %d\n", mysql_affected_rows());
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
Удалено записей: 10
Удалено записей: 0
```

mysql_client_encoding - Возвращает кодировку соединения

string **mysql_client_encoding** ([resource `$link_identifier`])

Возвращает значение переменной MySQL character_set. Возвращает используемую по умолчанию кодировку для данного соединения.

`link_identifier` Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **`E_WARNING`**.

```
<?php
$link    = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$charset = mysql_client_encoding($link);
echo "Текущая кодировка: $charset\n";
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
Текущая кодировка: latin1
```

mysql_close - Закрывает соединение с сервером MySQL

bool **mysql_close** ([resource `$link_identifier`])

mysql_close() закрывает непостоянное соединение с базой данных MySQL, на которое указывает переданный дескриптор. Если параметр link_identifier не указан, закрывается последнее открытое (текущее) соединение. Возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки.

В использовании mysql_close() обычно нет надобности для непостоянных соединений, т.к. они автоматически закрываются в конце скрипта. См. также высвобождение ресурсов.

link_identifier Соединение MySQL. Если не указано, то используется последнее соединение, открытое mysql_connect(). Если соединение не найдено или не установлено, то будет сгенерирована ошибка уровня E_WARNING.

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Ошибка соединения: ' . mysql_error());
}
echo 'Успешно соединились';
mysql_close($link);
?>
```

Результат выполнения данного примера:

```
Успешно соединились
```

mysql_connect - Открывает соединение с сервером MySQL

resource **mysql_connect** ([string `$server` = ini_get("mysql.default_host") [, string `$username` = ini_get("mysql.default_user") [, string `$password` = ini_get("mysql.default_password") [, bool `$new_link` = false [, int `$client_flags` = 0]]]]])

Открывает новое соединение с сервером MySQL или использует уже существующее. Возвращает дескриптор соединения с MySQL в случае успешного выполнения или FALSE в случае возникновения ошибки.

server Сервер MySQL. Может также включать номер порта, например, "hostname:port" или путь к локальному сокету, например, ":/path/to/socket" для локального сервера. Если PHP-директива mysql.default_host не определена (по умолчанию), то значением по умолчанию является 'localhost:3306'. В SQL safe mode этот параметр игнорируется и всегда используется значение 'localhost:3306'.

username Имя пользователя. Значение по умолчанию определяется директивой mysql.default_user. В SQL safe mode этот параметр будет проигнорирован и будет использован пользователь, владеющий процессом сервера.

password Пароль. Значение по умолчанию определяется директивой mysql.default_password. В SQL safe mode этот параметр будет проигнорирован и в качестве пароля будет использована пустая строка.

new_link Если второй вызов функции **mysql_connect()** произошёл с теми же аргументами, то новое соединение не будет установлено. Вместо этого функция вернёт ссылку на уже установленное соединение. Параметр *new_link* может заставить функцию **mysql_connect()** открыть ещё одно соединение, даже если соединение с аналогичными параметрами уже открыто. В SQL safe mode этот параметр игнорируется.

client_flags Параметр *client_flags* должен быть комбинацией из следующих констант: 128 (включает обработку *LOAD DATA LOCAL*), **MYSQL_CLIENT_SSL**, **MYSQL_CLIENT_COMPRESS**, **MYSQL_CLIENT_IGNORE_SPACE** or **MYSQL_CLIENT_INTERACTIVE**. Подробнее читайте в разделе Клиентские константы MySQL. В SQL safe mode этот параметр игнорируется.

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Ошибка соединения: ' . mysql_error());
}
echo 'Успешно соединились';
mysql_close($link);
?>
```

mysql_create_db - Создает базу данных MySQL

bool **mysql_create_db** (string `$database_name` [, resource `$link_identifier`])

Пытается создать базу данных на сервере, с которым ассоциирован переданный дескриптор соединения. Возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки.

database_name Имя создаваемой базы данных.

link_identifier Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **E_WARNING.**

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Ошибка соединения: ' . mysql_error());
}
$sql = 'CREATE DATABASE my_db';
if (mysql_query($sql, $link)) {
    echo "База my_db успешно создана\n";
} else {
    echo 'Ошибка при создании базы данных: ' . mysql_error() . "\n";
}
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
База my_db успешно создана
```

mysql_data_seek - Перемещает внутренний указатель в результате запроса

bool **mysql_data_seek** (resource `$result` , int `$row_number`)

Перемещает внутренний указатель результата запроса, с которым связан переданный дескриптор, к ряду с указанным номером. Следующий вызов к функции получения данных MySQL, такой как mysql_fetch_assoc(), вернёт именно его. Возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки.

Нумерация row_number начинается с 0. row_number должен быть значением в диапазоне от 0 до mysql_num_rows() - 1. Однако, если результат пуст (mysql_num_rows() == 0), то попытка сдвига указателя к нулевому ряду завершится неудачей - будет вызвана ошибка уровня E_WARNING и mysql_data_seek() вернет FALSE.

result Обрабатываемый результат запроса. Этот результат может быть получен с помощью функции mysql_query().

row_number Желаемый номер ряда в полученном дескрипторе результата.

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Ошибка соединения: ' . mysql_error());
}
$db_selected = mysql_select_db('sample_db');
if (!$db_selected) {
    die('Не удалось выбрать базу данных: ' . mysql_error());
}
$query = 'SELECT last_name, first_name FROM friends';
$result = mysql_query($query);
if (!$result) {
    die('Ошибка запроса: ' . mysql_error());
}
/* получение рядов в обратном порядке */
for ($i = mysql_num_rows($result) - 1; $i >= 0; $i--) {
    if (!mysql_data_seek($result, $i)) {
        echo "Не удалось переместиться к ряду $i: " . mysql_error() . "\n";
        continue;
    }    if (!($row = mysql_fetch_assoc($result))) {
```

```
        continue;
    }    echo $row['last_name'] . ' ' . $row['first_name'] . "<br />\n";
}
mysql_free_result($result);
?>
```

mysql_db_name - Возвращает название базы данных из вызова к mysql_list_dbs()

string **mysql_db_name** (resource `$result` , int `$row` [, mixed `$field`])

Возвращает название базы данных из вызова к mysql_list_dbs().Возвращает название базы данных в случае успеха, или FALSE в случае ошибки. В случае возврата FALSE используйте mysql_error() для определения природы ошибок.

result Дескриптор результата, полученный из вызова mysql_list_dbs().
row Индекс в результате.
field Имя поля.

```
<?php
error_reporting(E_ALL);
$link = mysql_connect('dbhost', 'username', 'password');
$db_list = mysql_list_dbs($link);
$i = 0;
$cnt = mysql_num_rows($db_list);
while ($i < $cnt) {
    echo mysql_db_name($db_list, $i) . "\n";
    $i++;
}
?>
```

mysql_db_query - Переключается на указанную базу данных и посылает запрос

resource **mysql_db_query** (string `$database` , string `$query` [, resource `$link_identifier`])

Выбирает базу данных и выполняет запрос к ней. Возвращает ресурс результата запроса к MySQL или FALSE в случае ошибки. Функция также возвращает TRUE/FALSE для INSERT/UPDATE/DELETE запросов для индикации успеха/провала.

database Имя базы данных, на которую произойдет переключение.
query Запрос MySQL. Данные в запросе должны быть корректно проэкранированы.
link_identifier Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **E_WARNING.**

```
<?php
if (!$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')) {
    echo 'Не удалось подключиться к mysql';
    exit;
}
if (!mysql_select_db('mysql_dbname', $link)) {
    echo 'Не удалось выбрать базу данных';
    exit;
}
$sql    = 'SELECT foo FROM bar WHERE id = 42';
$result = mysql_query($sql, $link);
if (!$result) {
    echo "Ошибка DB, запрос не удался\n";
    echo 'MySQL Error: ' . mysql_error();
    exit;
}
while ($row = mysql_fetch_assoc($result)) {
    echo $row['foo'];
}
mysql_free_result($result);
?>
```

mysql_errno - Возвращает численный код ошибки выполнения последней операции с MySQL

int **mysql_errno** ([resource `$link_identifier`])

Возвращает код ошибки последней функции работы с MySQL. Возвращает код ошибки последней функции работы с MySQL, или 0 (ноль), если операция выполнена успешно.

Ошибки работы с MySQL больше не вызывают сообщений в PHP. Вместо этого используйте функцию mysql_errno(), чтобы получить код ошибки. Учтите, что функция возвращает код ошибки только последней выполненной функции (исключая mysql_error() и mysql_errno()). Проверяйте результат работы функции до вызова следующей.

`link_identifier` Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **`E_WARNING`**.

```
<?php
$link = mysql_connect("localhost", "mysql_user", "mysql_password");
if (!mysql_select_db("nonexistentdb", $link)) {
    echo mysql_errno($link) . ": " . mysql_error($link). "\n";
}
mysql_select_db("kossu", $link);
if (!mysql_query("SELECT * FROM nonexistenttable", $link)) {
    echo mysql_errno($link) . ": " . mysql_error($link) . "\n";
}
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
1049: Unknown database 'nonexistentdb'
1146: Table 'kossu.nonexistenttable' doesn't exist
```

mysql_error - Возвращает текст ошибки последней операции с MySQL

string **mysql_error** ([resource `$link_identifier`])

Возвращает текст ошибки выполнения последней функции MySQL. Ошибки работы с MySQL больше не вызывают сообщений в PHP. Вместо этого используйте функцию mysql_error(), для получения сообщения об ошибке. Учтите, что функция возвращает текст ошибки только последней выполненной функции MySQL (исключая mysql_error() и mysql_errno()), поэтому убедитесь, что вы вызываете данную функцию до вызова следующей функции MySQL. Возвращает текст ошибки выполнения последней функции MySQL, или " (пустую строку), если операция выполнена успешно.

`link_identifier` Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **`E_WARNING`**.

```
<?php
$link = mysql_connect("localhost", "mysql_user", "mysql_password");
mysql_select_db("nonexistentdb", $link);
echo mysql_errno($link) . ": " . mysql_error($link). "\n";
mysql_select_db("kossu", $link);
mysql_query("SELECT * FROM nonexistenttable", $link);
echo mysql_errno($link) . ": " . mysql_error($link) . "\n";
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
1049: Unknown database 'nonexistentdb'
1146: Table 'kossu.nonexistenttable' doesn't exist
```

mysql_fetch_array - Обрабатывает ряд результата запроса, возвращая ассоциативный массив, численный массив или оба

array **mysql_fetch_array** (resource `$result` [, int `$result_type` = MYSQL_BOTH])

Возвращает массив, соответствующий обработанному ряду результата запроса и сдвигает внутренний указатель данных вперед.

result Обрабатываемый результат запроса. Этот результат может быть получен с помощью функции mysql_query().

result_type Тип возвращаемого массива. Является константой и может принимать следующие значения: **MYSQL_ASSOC**, **MYSQL_NUM** и **MYSQL_BOTH**.

Возвращает массив строк, соответствующих обработанному ряду результата запроса, или FALSE, если рядов больше нет. Тип возвращаемого массива зависит от значения параметра result_type. При использовании MYSQL_BOTH (по умолчанию), вы получите массив, состоящий как из ассоциативных индексов, так и из численных. MYSQL_ASSOC вернёт только ассоциативные индексы (аналогично функции mysql_fetch_assoc()), а MYSQL_NUM - только численные (аналогично функции mysql_fetch_row()).

Если несколько колонок в результате будут иметь одинаковые названия, то будет возвращена последняя колонка. Чтобы получить доступ к другим колонкам с тем же именем, используйте численные индексы массива или псевдонимы в запросе. В случае псевдонимов используйте именно их - вы не сможете использовать настоящие имена колонок.

Пример # mysql_fetch_array() с MYSQL_NUM

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password") or
    die("Ошибка соединения: " . mysql_error());
mysql_select_db("mydb");
$result = mysql_query("SELECT id, name FROM mytable");
while ($row = mysql_fetch_array($result, MYSQL_NUM)) {
    printf("ID: %s  Имя: %s", $row[0], $row[1]);
}
mysql_free_result($result);
?>
```

Пример # mysql_fetch_array() **с** MYSQL_ASSOC

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password") or
    die("Ошибка соединения: " . mysql_error());
mysql_select_db("mydb");
$result = mysql_query("SELECT id, name FROM mytable");
while ($row = mysql_fetch_array($result, MYSQL_ASSOC)) {
    printf("ID: %s  Имя: %s", $row["id"], $row["name"]);
}
mysql_free_result($result);
?>
```

Пример # mysql_fetch_array() **с** MYSQL_BOTH

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password") or
    die("Ошибка соединения: " . mysql_error());
mysql_select_db("mydb");
$result = mysql_query("SELECT id, name FROM mytable");
while ($row = mysql_fetch_array($result, MYSQL_BOTH)) {
    printf ("ID: %s  Имя: %s", $row[0], $row["name"]);
}
mysql_free_result($result);
?>
```

mysql_fetch_assoc - Возвращает ряд результата запроса в качестве ассоциативного массива

array **mysql_fetch_assoc** (resource `$result`)

Возвращает ассоциативный массив, соответсвующий полученному ряду и сдвигает вперед внутренний указатель результата. Функция mysql_fetch_assoc() аналогична вызову функции

mysql_fetch_array() со вторым необязательным параметром, равным MYSQL_ASSOC. Функция возвращает только ассоциативный массив. Возвращает ассоциативный массив строк, соответствующий полученному ряду, либо FALSE если рядов больше нет.

result Обрабатываемый результат запроса. Этот результат может быть получен с помощью функции mysql_query().

Если два или более столбцов результата имеют одинаковые имена, приоритет будет иметь последний столбец. Для доступа к другому одноименному столбцу (или столбцам), вам необходимо либо обратиться к результату запроса по числовому индексу с помощью mysql_fetch_row() либо добавить псевдонимы к нужным столбцам. Для более подробной информации о псевдонимах смотрите описание примера mysql_fetch_array().

```
<?php
$conn = mysql_connect("localhost", "mysql_user", "mysql_password");
if (!$conn) {
    echo "Unable to connect to DB: " . mysql_error();
    exit;
}
if (!mysql_select_db("mydbname")) {
    echo "Unable to select mydbname: " . mysql_error();
    exit;
}
$sql = "SELECT id as userid, fullname, userstatus
        FROM   sometable
        WHERE  userstatus = 1";
$result = mysql_query($sql);
if (!$result) {
    echo "Could not successfully run query ($sql) from DB: " . mysql_error();
    exit;
}
if (mysql_num_rows($result) == 0) {
    echo "No rows found, nothing to print so am exiting";
    exit;
}
// До тех пор, пока в результате содержатся ряды, помещаем их в ассоциативный массив.
// Замечание: если запрос возвращает только один ряд - нет нужды в цикле.
// Замечание: если вы добавите extract($row); в начало цикла, вы сделаете
//            доступными переменные $userid, $fullname и $userstatus
while ($row = mysql_fetch_assoc($result)) {
    echo $row["userid"];
    echo $row["fullname"];
    echo $row["userstatus"];
}
mysql_free_result($result);
?>
```

mysql_fetch_field - Возвращает информацию о колонке из результата запроса в виде объекта

object **mysql_fetch_field** (resource `$result` [, int `$field_offset` = 0])

Возвращает объект, содержащий информацию о колонке. Эту функцию можно использовать для получения информации о полях в переданном результате запроса. Возвращает object, содержащий информацию о колонке. Объект содержит следующие свойства:

result Обрабатываемый результат запроса. Этот результат может быть получен с помощью функции mysql_query().

field_offset Числовое смещение поля. Если смещение не указано, функция возвращает информацию о первой колонке, которая ещё не была обработана этой функцией. Нумерация *field_offset* начинается с *0*.

- name - название колонки
- table - название таблицы, которой принадлежит колонка
- max_length - максимальная длина колонки
- not_null - 1, если колонка не может быть **NULL**
- primary_key - 1, если колонка является первичным индексом
- unique_key - 1, если колонка является уникальным индексом

- multiple_key - 1, если колонка является неуникальным индексом
- numeric - 1, если колонка численная
- blob - 1, если колонка является BLOB
- type - тип колонки
- unsigned - 1, если колонка не содержит знака (unsigned)
- zerofill - 1, если колонка заполняется нулями (zero-filled)

```
<?php
$conn = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$conn) {
    die('Ошибка при соединении: ' . mysql_error());
}
mysql_select_db('database');
$result = mysql_query('select * from table');
if (!$result) {
    die('Ошибка в запросе: ' . mysql_error());
}
/* получаем данные о колонке */
$i = 0;
while ($i < mysql_num_fields($result)) {
    echo "Информация о колонке $i:<br />\n";
    $meta = mysql_fetch_field($result, $i);
    if (!$meta) {
        echo "Информация недоступна<br />\n";
    }
    echo "<pre>
blob:         $meta->blob
max_length:   $meta->max_length
multiple_key: $meta->multiple_key
name:         $meta->name
not_null:     $meta->not_null
numeric:      $meta->numeric
primary_key:  $meta->primary_key
table:        $meta->table
type:         $meta->type
unique_key:   $meta->unique_key
unsigned:     $meta->unsigned
zerofill:     $meta->zerofill
</pre>";
    $i++;
}
mysql_free_result($result);
?>
```

mysql_fetch_lengths - Возвращает длину каждого поля в результате

array **mysql_fetch_lengths** (resource `$result`)

Возвращает массив длин для каждого поля, содержащегося в последнем ряду результата, полученном из MySQL. Массив (array) длин в случае успеха или FALSE в случае возникновения ошибки.

Возвращает длины каждого поля, содержащегося в последнем ряду, обработанном функциями mysql_fetch_row(), mysql_fetch_assoc(), mysql_fetch_array() и mysql_fetch_object() в массиве, начинающемся с 0.

result Обрабатываемый результат запроса. Этот результат может быть получен с помощью функции mysql_query().

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Ошибка выполнения запроса: ' . mysql_error();
    exit;
}
$row     = mysql_fetch_assoc($result);
$lengths = mysql_fetch_lengths($result);
print_r($row);
print_r($lengths);
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
Array
(
    [id] => 42
    [email] => user@example.com
)
Array
(
    [0] => 2
    [1] => 16
)
```

mysql_fetch_object - Обрабатывает ряд результата запроса и возвращает объект

object **mysql_fetch_object** (resource `$result` [, string `$class_name` [, array `$params`]])

Возвращает объект со свойствами, соответствующими колонкам в обработанном ряду и сдвигает внутренний указатель результата вперед. Возвращает объект (object) со строковыми свойствами, соответствующими полученному ряду, или FALSE, если рядов больше нет.

result	Обрабатываемый результат запроса. Этот результат может быть получен с помощью функции mysql_query().
class_name	Имя класса. Будет создан экземпляр указанного класса, заполнен свойствами и возвращен. Если не указан, возвращается экземпляр **stdClass**.
params	Необязательный массив (array) параметров, передаваемых в конструктор создаваемого экземпляра *class_name*.

```
<?php
mysql_connect("hostname", "user", "password");
mysql_select_db("mydb");
$result = mysql_query("select * from mytable");
while ($row = mysql_fetch_object($result)) {
    echo $row->user_id;
    echo $row->fullname;
}
mysql_free_result($result);
?>
```

mysql_fetch_row - Обрабатывает ряд результата запроса и возвращает массив с числовыми индексами

array **mysql_fetch_row** (resource `$result`)

Возвращает массив с числовыми индексами, содержащий данные обработанного ряда, и сдвигает внутренний указатель результата вперед. Возвращает массив строк с числовыми индексами, содержащий данные обработанного ряда, или FALSE, если рядов не осталось.

result	Обрабатываемый результат запроса. Этот результат может быть получен с помощью функции mysql_query().

Обрабатывает один ряд результата, на который ссылается переданный указатель. Ряд возвращается в виде массива. Каждая колонка располагается в следующей ячейке массива, начиная с нулевого индекса

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Ошибка запроса: ' . mysql_error();
    exit;
}
$row = mysql_fetch_row($result);
echo $row[0]; // 42
echo $row[1]; // email
?>
```

mysql_field_flags - Возвращает флаги, связанные с указанным полем результата запроса

string **mysql_field_flags** (resource `$result` , int `$field_offset`)

Возвращает флаги, связанные с указанным полем. Каждый флаг возвращается как отдельное слово, отделённое от предыдущего пробелом. Полученное значение можно разбить в массив, используя функцию explode().Возвращает строку с флагами, связанными с результатом или FALSE в случае возникновения ошибки.

result Обрабатываемый результат запроса. Этот результат может быть получен с помощью функции mysql_query().

field_offset Числовое смещение поля. *field_offset* начинается с *0*. Если *field_offset* не существует, генерируется ошибка уровня **E_WARNING.**

Возвращаются следующие флаги, если ваша версия MySQL их уже поддерживает: "not_null", "primary_key", "unique_key", "multiple_key", "blob", "unsigned", "zerofill", "binary", "enum", "auto_increment" и "timestamp".

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Ошибка в запросе: ' . mysql_error();
    exit;
}
$flags = mysql_field_flags($result, 0);
echo $flags;
print_r(explode(' ', $flags));
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
not_null primary_key auto_increment
Array
(
    [0] => not_null
    [1] => primary_key
    [2] => auto_increment
)
```

mysql_field_len - Возвращает длину указанного поля

int **mysql_field_len** (resource `$result` , int `$field_offset`)

Возвращает длину указанного поля. Длина указанного поля в случае успеха или FALSE в случае возникновения ошибки.

result Обрабатываемый результат запроса. Этот результат может быть получен с помощью функции mysql_query().

field_offset Числовое смещение поля. *field_offset* начинается с *0*. Если *field_offset* не существует, генерируется ошибка уровня **E_WARNING.**

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Не удалось выполнить запрос: ' . mysql_error();
    exit;
}
// Получит длину поля id так, как указано в структуре базы данных
$length = mysql_field_len($result, 0);
echo $length;
?>
```

mysql_field_name - Возвращает название указанной колонки результата запроса

string **mysql_field_name** (resource `$result` , int `$field_offset`)

Возвращает название колонки с указанным индексом. Название поля по указанному индексу в случае успеха или FALSE в случае возникновения ошибки.

result Обрабатываемый результат запроса. Этот результат может быть получен с помощью функции mysql_query().

field_offset Числовое смещение поля. *field_offset* начинается с *0*. Если *field_offset* не существует, генерируется ошибка уровня **E_WARNING.**

```
<?php
```

```
/* Таблица пользователей состоит из трёх колонок:
 *   user_id
 *   username
 *   password.
 */
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Ошибка соединения с MySQL: ' . mysql_error());
}
$dbname = 'mydb';
$db_selected = mysql_select_db($dbname, $link);
if (!$db_selected) {
    die("Не удалось выбрать базу $dbname: " . mysql_error());
}
$res = mysql_query('select * from users', $link);
echo mysql_field_name($res, 0) . "\n";
echo mysql_field_name($res, 2);
?>
```

Результат выполнения данного примера:

```
user_id
password
```

mysql_field_seek - Устанавливает внутренний указатель результата на переданное смещение поля

bool **mysql_field_seek** (resource `$result` , int `$field_offset`)

Перемещает указатель к полю с указанным смещением. Если следующий вызов функции mysql_fetch_field() не содержит смещения, то будет возвращено смещение, содержащееся в mysql_field_seek(). Возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки.

result	Обрабатываемый результат запроса. Этот результат может быть получен с помощью функции mysql_query().
field_offset	Числовое смещение поля. *field_offset* начинается с *0*. Если *field_offset* не существует, генерируется ошибка уровня **E_WARNING**.

mysql_field_table - Возвращает название таблицы, которой принадлежит указанное поле

string **mysql_field_table** (resource `$result` , int `$field_offset`)

Возвращает название таблицы, которой принадлежит указанное поле. Имя таблицы в случае успеха.

result	Обрабатываемый результат запроса. Этот результат может быть получен с помощью функции mysql_query().
field_offset	Числовое смещение поля. *field_offset* начинается с *0*. Если *field_offset* не существует, генерируется ошибка уровня **E_WARNING**.

```
<?php
$query = "SELECT account.*, country.* FROM account, country WHERE country.name = 'Por
tugal' AND account.country_id = country.id";
// получаем результат из базы данных
$result = mysql_query($query);
// выводит имя таблицы и имя поля
for ($i = 0; $i < mysql_num_fields($result); ++$i) {
    $table = mysql_field_table($result, $i);
    $field = mysql_field_name($result, $i);
    echo  "$table: $field\n";
}
?>
```

mysql_field_type - Возвращает тип указанного поля из результата запроса

string **mysql_field_type** (resource `$result` , int `$field_offset`)

Функция mysql_field_type() аналогична функции mysql_field_name(). Аргументы одинаковы, но вместо имени колонки возвращается её тип. Поля могут быть следующих типов: "int", "real", "string", "blob" и других.

result Обрабатываемый результат запроса. Этот результат может быть получен с помощью функции mysql_query().

field_offset Числовое смещение поля. *field_offset* начинается с *0*. Если *field_offset* не существует, генерируется ошибка уровня **E_WARNING**.

```
<?php
mysql_connect("localhost", "mysql_username", "mysql_password");
mysql_select_db("mysql");
$result = mysql_query("SELECT * FROM func");
$fields = mysql_num_fields($result);
$rows   = mysql_num_rows($result);
$table  = mysql_field_table($result, 0);
echo "Ваша таблица '" . $table . "' содержит " . $fields . " поля и " . $rows .
" запись\n";
echo "В таблице есть следующие поля:\n";
for ($i=0; $i < $fields; $i++) {
    $type  = mysql_field_type($result, $i);
    $name  = mysql_field_name($result, $i);
    $len   = mysql_field_len($result, $i);
    $flags = mysql_field_flags($result, $i);
    echo $type . " " . $name . " " . $len . " " . $flags . "\n";
}
mysql_free_result($result);
mysql_close();
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
Ваша таблица 'func' содержит 4 поля и 1 запись
В таблице есть следующие поля:
string name 64 not_null primary_key binary
int ret 1 not_null
string dl 128 not_null
string type 9 not_null enum
```

mysql_free_result - Освобождает память от результата запроса

bool **mysql_free_result** (resource `$result`)

Высвободит всю память, занимаемую результатом, на который ссылается переданный дескриптор result. Возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки.

Нуждается в вызове только в том случае, если вы всерьёз обеспокоены тем, сколько памяти используют ваши запросы к БД, возвращающие большое количество данных. Вся память, используемая для хранения этих данных автоматически очистится в конце работы скрипта.

result Обрабатываемый результат запроса. Этот результат может быть получен с помощью функции mysql_query().

Если в качестве параметра result передан не ресурс, то будет вызвана ошибка уровня E_WARNING. Стоит также заметить, что mysql_query() возвращает resource только для запросов SELECT, SHOW, EXPLAIN и DESCRIBE.

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Не удалось выполнить запрос: ' . mysql_error();
    exit;
}
/* Используем результат, подразумевая, что после этого он нам больше не нужен */
$row = mysql_fetch_assoc($result);
/* Теперь освобождаем результат и продолжаем дальнейшую работу над нашим скриптом */
mysql_free_result($result);
echo $row['id'];
echo $row['email'];
?>
```

mysql_get_client_info - Возвращает данные о MySQL-клиенте

string **mysql_get_client_info** (void)

Возвращает строку, содержащую версию клиентской библиотеки. Версия клиентской библиотеки MySQL.

```
<?php
printf("Версия клиентской библиотеки MySQL: %s\n", mysql_get_client_info());
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
Версия клиентской библиотеки MySQL: 3.23.39
```

mysql_get_host_info - Возвращает информацию о соединении с MySQL

string **mysql_get_host_info** ([resource `$link_identifier`])

Описывает тип используемого соединения, указанного переданным дескриптором соединения, включая имя хоста. Возвращает строку, описывающую тип используемого соединения, указанного переданным дескриптором соединения или FALSE в случае возникновения ошибки.

`link_identifier` Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **E_WARNING.**

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Ошибка соединения: ' . mysql_error());
}
printf("Тип соединения с MySQL: %s\n", mysql_get_host_info());
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
Тип соединения с MySQL: Localhost via UNIX socket
```

mysql_get_proto_info - Возвращает информацию о протоколе MySQL

int **mysql_get_proto_info** ([resource `$link_identifier`])

Возвращает информацию о протоколе MySQL. Возвращает используемый протокол MySQL в случае успеха или FALSE в случае возникновения ошибки.

`link_identifier` Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **E_WARNING.**

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Ошибка соединения: ' . mysql_error());
}
printf("Версия протокола MySQL: %s\n", mysql_get_proto_info());
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
Версия протокола MySQL: 10
```

mysql_get_server_info - Возвращает информацию о сервере MySQL

string **mysql_get_server_info** ([resource `$link_identifier`])

Возвращает версию сервера MySQL. Возвращает версию сервера MySQL в случае успеха или FALSE в случае возникновения ошибки.

link_identifier Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **E_WARNING.**

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Ошибка соединения: ' . mysql_error());
}
printf("Версия сервера MySQL: %s\n", mysql_get_server_info());
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
Версия сервера MySQL: 4.0.1-alpha
```

mysql_info - Возвращает информацию о последнем запросе

string **mysql_info** ([resource `$link_identifier`])

Возвращает подробную информацию о последнем запросе. Возвращает информацию о запросе в случае успеха, или FALSE в случае ошибки. Смотрите пример ниже для каких запросов возвращается информация и как выглядят возвращаемые значения. Для неперечисленных запросов будет возвращено значение FALSE.

link_identifier Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **E_WARNING.**

Числа расставлены только для примера -- их значения зависят от результата запроса.

```
INSERT INTO ... SELECT ...
String format: Records: 23 Duplicates: 0 Warnings: 0
INSERT INTO ... VALUES (...),(...),(...)...
String format: Records: 37 Duplicates: 0 Warnings: 0
LOAD DATA INFILE ...
String format: Records: 42 Deleted: 0 Skipped: 0 Warnings: 0
ALTER TABLE
String format: Records: 60 Duplicates: 0 Warnings: 0
UPDATE
String format: Rows matched: 65 Changed: 65 Warnings: 0
```

mysql_insert_id - Возвращает идентификатор, сгенерированный при последнем INSERT-запросе

int **mysql_insert_id** ([resource `$link_identifier`])

Возвращает идентификатор, сгенерированный колонкой с AUTO_INCREMENT последним запросом (обычно INSERT).

link_identifier Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **E_WARNING.**

Идентификатор, сгенерированный колонкой с AUTO_INCREMENT последним запросом в случае успеха , 0, если последний запрос не генерирует значение AUTO_INCREMENT value, и FALSE, если соединение MySQL не было установлено.

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Ошибка соединения: ' . mysql_error());
```

```
}
mysql_select_db('mydb');
mysql_query("INSERT INTO mytable (product) values ('kossu')");
printf("Идентификатор последней вставленной записи %d\n", mysql_insert_id());
?>
```

mysql_list_dbs - Возвращает список баз данных, доступных на сервере

resource **mysql_list_dbs** ([resource `$link_identifier`])

Возвращает указатель на результат, содержащий список баз данных, доступных на указанном сервере.

`link_identifier` Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **E_WARNING.**

Возвращает resource результата в случае успеха, или FALSE в случае ошибки. Используйте функцию mysql_tablename(), чтобы получить данные из результата, или любую другую функцию, работающую с результатами запросов, например mysql_fetch_array().

```
<?php
// Без использования mysql_list_dbs()
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$res = mysql_query("SHOW DATABASES");
while ($row = mysql_fetch_assoc($res)) {
    echo $row['Database'] . "\n";
}
// Устарело, начиная с PHP 5.4.0
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$db_list = mysql_list_dbs($link);
while ($row = mysql_fetch_object($db_list)) {
     echo $row->Database . "\n";
}
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
database1
database2
database3
```

mysql_list_processes - Возвращает список процессов MySQL

resource **mysql_list_processes** ([resource `$link_identifier`])

Возвращает список текущих процессов на сервере MySQL. Дескриптор результата (resource) в случае успеха или FALSE в случае возникновения ошибки.

`link_identifier` Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **E_WARNING.**

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$result = mysql_list_processes($link);
while ($row = mysql_fetch_assoc($result)){
    printf("%s %s %s %s %s\n", $row["Id"], $row["Host"], $row["db"],
        $row["Command"], $row["Time"]);
}
mysql_free_result($result);
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
1 localhost test Processlist 0
4 localhost mysql sleep 5
```

mysql_num_fields - Возвращает количество полей результата запроса

int **mysql_num_fields** (resource `$result`)

Возвращает количество полей в результате запроса. Возвращает количество полей в результате запроса (resource) в случае успеха или FALSE в случае возникновения ошибки.

result Обрабатываемый результат запроса. Этот результат может быть получен с помощью функции mysql_query().

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Не удалось выполнить запрос: ' . mysql_error();
    exit;
}
/* возвращает 2, так как id,email === двум полям */
echo mysql_num_fields($result);
?>
```

mysql_num_rows - Возвращает количество рядов результата запроса

int **mysql_num_rows** (resource `$result`)

Возвращает количество рядов результата запроса. Эта команда работает только с запросами SELECT или SHOW, возвращающих актуальный результат запроса. Чтобы получить количество рядов, обработанных функцями INSERT, UPDATE, REPLACE и DELETE, используйте функцию mysql_affected_rows().Количество рядов в результате запроса в случае успеха или FALSE в случае возникновения ошибки.

result Обрабатываемый результат запроса. Этот результат может быть получен с помощью функции mysql_query().

```
<?php
$link = mysql_connect("localhost", "mysql_user", "mysql_password");
mysql_select_db("database", $link);
$result = mysql_query("SELECT * FROM table1", $link);
$num_rows = mysql_num_rows($result);
echo "Получено $num_rows рядов\n";
?>
```

mysql_pconnect - Устанавливает постоянное соединение с сервером MySQL

resource **mysql_pconnect** ([string `$server` = ini_get("mysql.default_host") [, string `$username` = ini_get("mysql.default_user") [, string `$password` = ini_get("mysql.default_password") [, int `$client_flags`]]]])

Устанавливает постоянное соединение с сервером MySQL. Возвращает дескриптор постоянного соединения MySQL в случае успеха, и FALSE в случае ошибки. Работает аналогично mysql_connect() с двумя важными отличиями.

Во-первых, при соединении функция пытается найти уже открытый (постоянный) указатель на тот же сервер с тем же пользователем и паролем. Если он найден, возвращён функцией будет именно он, вместо открытия нового соединения.

Во-вторых, соединение с SQL-сервером не будет закрыто, когда работа скрипта закончится. Вместо этого, оно останется рабочим для будущего использования (mysql_close() также не закрывает постоянные соединения, открытые mysql_pconnect()).

Соединения такого типа называют 'постоянными'.

server Сервер MySQL. Может также включать номер порта, например, "hostname:port" или путь к локальному сокету, например, ":/path/to/socket" для локального хоста. Если директива mysql.default_host не определена (по умолчанию), то по умолчанию используется значение 'localhost:3306'

username Имя пользователя. По умолчанию используется имя пользователя, владеющего серверным процессом.

password Пароль. По умолчанию используется пустая строка.

client_flags Параметр *client_flags* может быть комбинацией следующих констант: 128 (включает обработку *LOAD DATA LOCAL*), **MYSQL_CLIENT_SSL**,

MYSQL_CLIENT_COMPRESS, **MYSQL_CLIENT_IGNORE_SPACE** и **MYSQL_CLIENT_INTERACTIVE**.

mysql_ping - Проверяет соединение с сервером и пересоединяется при необходимости

bool **mysql_ping** ([resource `$link_identifier`])

Проверяет работает ли соединение с сервером. Если оно утеряно, автоматически предпринимается попытка пересоединения. Эта функция может быть использована в скриптах, работающих на протяжении долгого времени. Возвращает TRUE, если соединение в рабочем состоянии и FALSE в противном случае.

link_identifier Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **E_WARNING**.

```
<?php
set_time_limit(0);
$conn = mysql_connect('localhost', 'mysqluser', 'mypass');
$db   = mysql_select_db('mydb');
/* Подразумевается что этот запрос будет выполняться достаточно долго */
$result = mysql_query($sql);
if (!$result) {
    echo 'Запрос #1 не удался, выходим.';
    exit;
}
/* Проверяем что соединение еще работает, если нет, соединяемся заново */
if (!mysql_ping($conn)) {
    echo 'Соединение потеряно, выходим после запроса #1';
    exit;
}
mysql_free_result($result);
/* Соединение еще работает, выполняем еще один запрос */
$result2 = mysql_query($sql2);
?>
```

mysql_query - Посылает запрос MySQL

resource **mysql_query** (string `$query` [, resource `$link_identifier`])

Посылает один запрос (посылка нескольких запросов не поддерживается) активной базе данных сервера, на который ссылается переданный дескриптор link_identifier. Завершится с ошибкой и вернет FALSE, если у пользователя нет доступа к какой-либо из таблиц, фигурирующих в запросе.

query SQL-запрос. Запрос не должен заканчиваться точкой с запятой. Данные в запросе должны быть корректно проэкранированы.

link_identifier Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **E_WARNING**.

Для запросов SELECT, SHOW, DESCRIBE, EXPLAIN и других запросов, возвращающих результат из нескольких рядов, mysql_query() возвращает дескриптор результата запроса (resource), или FALSE в случае ошибки.

Для других типов SQL-запросов, INSERT, UPDATE, DELETE, DROP и других, mysql_query() возвращает TRUE в случае успеха и FALSE в случае ошибки.

Полученный дескриптор результата нужно передать в функцию mysql_fetch_array() или любую другую функцию, работающую с результатами запросов.

Используйте mysql_num_rows() для выяснения количества рядов в результате SELECT-запроса или mysql_affected_rows() для выяснения количества обработанных рядов запросами DELETE, INSERT, REPLACE и UPDATE.

```
<?php
// Эти данные, к примеру, могли быть получены от пользователя
$firstname = 'fred';
$lastname  = 'fox';
// Формируем запрос
// Это лучший способ выполнить SQL-запрос
// Еще примеры можно найти в документации mysql_real_escape_string()
$query = sprintf("SELECT firstname, lastname, address, age FROM friends
    WHERE firstname='%s' AND lastname='%s'",
    mysql_real_escape_string($firstname),
    mysql_real_escape_string($lastname));
// Выполняем запрос
$result = mysql_query($query);
// Проверяем результат
// Это показывает реальный запрос, посланный к MySQL, а также ошибку.
// Удобно при отладке.
if (!$result) {
    $message  = 'Неверный запрос: ' . mysql_error() . "\n";
    $message .= 'Запрос целиком: ' . $query;
    die($message);
}
// Используем результат
// Попытка напечатать $result не выведет информацию, которая в нем хранится
// Необходимо использовать какую-либо mysql-функцию, работающую с результатом запроса
// См. также mysql_result(), mysql_fetch_array(), mysql_fetch_row() и т.п.
while ($row = mysql_fetch_assoc($result)) {
    echo $row['firstname'];
    echo $row['lastname'];
    echo $row['address'];
    echo $row['age'];
}
// Освобождаем ресурсы, ассоциированные с результатом
// Это делается автоматически в конце скрипта
mysql_free_result($result);
?>
```

mysql_real_escape_string - Экранирует специальные символы в строках для использования в выражениях SQL

string **mysql_real_escape_string** (string `$unescaped_string` [, resource `$link_identifier`])

Экранирует специальные символы в unescaped_string, принимая во внимание кодировку соединения, таким образом, что результат можно безопасно использовать в SQL-запросе в функции mysql_query(). Если вставляются бинарные данные, то к ним так же необходимо применять эту функцию. Возвращает строку, в которой экранированы все необходимые символы, или FALSE в случае ошибки.

mysql_real_escape_string() вызывает библиотечную функцмю MySQL mysql_real_escape_string, которая добавляет обратную косую черту к следующим символам: \x00, \n, \r, \, ', " и \x1a.

Эта функция должна всегда (за несколькими исключениями) использоваться для того, чтобы обезопасить данные, вставляемые в запрос перед отправкой его в MySQL.

`unescaped_string`	Экранируемая строка.
`link_identifier`	Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **`E_WARNING`**.

```
<?php
```

```
// Соединение
$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')
    OR die(mysql_error());
// Запрос
$query = sprintf("SELECT * FROM users WHERE user='%s' AND password='%s'",
            mysql_real_escape_string($user),
            mysql_real_escape_string($password));
?>
```

mysql_result - Возвращает данные результата запроса

string **mysql_result** (resource `$result` , int `$row` [, mixed `$field` = 0])

Возвращает содержимое одного поля из набора результата MySQL. Содержимое одного поля из набора результата MySQL в случае успеха, или FALSE в случае ошибки.

Работая с большими результатами запросов, следует использовать одну из функций, обрабатывающих сразу целый ряд результата (указаны ниже). Так как эти функции возвращают значение нескольких ячеек сразу, они НАМНОГО быстрее mysql_result(). Кроме того, учтите, что указание численного смещения работает намного быстрее, чем указание колонки, или колонки с таблицей через точку.

result Обрабатываемый результат запроса. Этот результат может быть получен с помощью функции mysql_query().

row Номер получаемого ряда из результата. Нумерация рядов начинается с *0*.

field Имя или смещение получаемого поля. Может быть как смещением поля, именем поля, так и именем поля вместе с таблицей (таблица.поле). Если для поля был указан псевдоним ('select foo as bar from...'), используйте его вместо имени самого поля. Если не указан, возвращается первое поле.

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Ошибка соединения: ' . mysql_error());
}
if (!mysql_select_db('database_name')) {
    die('Ошибка выбора базы данных: ' . mysql_error());
}
$result = mysql_query('SELECT name FROM work.employee');
if (!$result) {
    die('Ошибка выполнения запроса:' . mysql_error());
}
echo mysql_result($result, 2); // выведет имя третьего сотрудника
mysql_close($link);
?>
```

mysql_select_db - Выбирает базу данных MySQL

bool **mysql_select_db** (string `$database_name` [, resource `$link_identifier`])

Выбирает для работы указанную базу данных на сервере, на который ссылается переданный дескриптор соединения. Каждый последующий вызов функции mysql_query() будет работать с выбранной базой данных. Возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки.

database_name Имя выбираемой базы данных.

link_identifier Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **E_WARNING**.

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Не удалось соединиться : ' . mysql_error());
}
// выбираем foo в качестве текущей базы данных
```

```
$db_selected = mysql_select_db('foo', $link);
if (!$db_selected) {
    die ('Не удалось выбрать базу foo: ' . mysql_error());
}
?>
```

mysql_set_charset - Устанавливает кодировку клиента

bool **mysql_set_charset** (string `$charset` [, resource `$link_identifier`])

Устанавливает кодировку по умолчанию для текущего соединения. Возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки.

charset Корректное название кодировки.

link_identifier Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **E_WARNING.**

mysql_stat - Возвращает текущий статус сервера

string **mysql_stat** ([resource `$link_identifier`])

Возвращает текущий статус сервера. Возвращает строку с данными аптайма, количества потоков, запросов, количеством открытых таблиц и таблиц с сброшенным внутренним кэшем (flush tables), а также количество запросов в секунду. Для получения полного списка других переменных вам необходимо будет использовать SQL-запрос SHOW STATUS. Если передан некорректный link_identifier, то будет возвращен NULL.

link_identifier Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **E_WARNING.**

```
<?php
$link   = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$status = explode('  ', mysql_stat($link));
print_r($status);
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
Array
(
    [0] => Uptime: 5380
    [1] => Threads: 2
    [2] => Questions: 1321299
    [3] => Slow queries: 0
    [4] => Opens: 26
    [5] => Flush tables: 1
    [6] => Open tables: 17
    [7] => Queries per second avg: 245.595
)
```

mysql_thread_id - Возвращает идентификатор текущего потока

int **mysql_thread_id** ([resource `$link_identifier`])

Возвращает идентификатор текущего потока. Если соединение потеряно и вы пересоединились с помощью mysql_ping(), то идентификатор потока изменится. Это означает, что вам не следует получать данный идентификатор и хранить его для дальнейшего использования. Вызывайте функцию тогда, когда он вам нужен. Идентификатор потока в случае успеха или FALSE в случае возникновения ошибки.

link_identifier Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое

соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **E_WARNING.**

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$thread_id = mysql_thread_id($link);
if ($thread_id){
    printf("Идентификатор текущего потока: %d\n", $thread_id);
}
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
Идентификатор текущего потока: 73
```

mysql_unbuffered_query - Посылает запрос MySQL без авто-обработки результата и его буферизации

resource **mysql_unbuffered_query** (string `$query` [, resource `$link_identifier`])

Посылает запрос MySQL query без автоматической обработки и буферизации её результата, в отличие от функции mysql_query(). Это позволяет сохранить достаточно большое количество памяти для SQL-запросов, возвращающих большое количество данных. Кроме того, вы можете начать работу с полученными данными сразу после того, как первый ряд был получен: вам не приходится ждать до конца SQL-запроса.

query — Запускаемый SQL-запрос. Данные в запросе должны быть корректно проэкранированы.

link_identifier — Соединение MySQL. Если идентификатор соединения не был указан, используется последнее соединение, открытое mysql_connect(). Если такое соединение не было найдено, функция попытается создать таковое, как если бы mysql_connect() была вызвана без параметров. Если соединение не было найдено и не смогло быть создано, генерируется ошибка уровня **E_WARNING.**

Для SELECT, SHOW, DESCRIBE и EXPLAIN запросов mysql_unbuffered_query() возвращает resource в случае успеха, или FALSE в случае ошибки.

Для остальных типов SQL-запросов, UPDATE, DELETE, DROP и т.д., mysql_unbuffered_query() возвращает TRUE в случае успеха и FALSE в случае ошибки.

5. Примеры использования PHP-MySQL

Пример 1. Этот простой пример показывает, как соединиться с базой данных, выполнить запрос, распечатать результат и отсоединиться.

```
<?php
// Соединяемся, выбираем базу данных
$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')
    or die('Не удалось соединиться: ' . mysql_error());
echo 'Соединение успешно установлено';
mysql_select_db('my_database') or die('Не удалось выбрать базу данных');

// Выполняем SQL-запрос
$query = 'SELECT * FROM my_table';
$result = mysql_query($query) or die('Запрос не удался: ' . mysql_error());

// Выводим результаты в html
echo "<table>\n";
while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
    echo "\t<tr>\n";
    foreach ($line as $col_value) {
        echo "\t\t<td>$col_value</td>\n";
    }
    echo "\t</tr>\n";
}
echo "</table>\n";

// Освобождаем память от результата
mysql_free_result($result);

// Закрываем соединение
mysql_close($link);
?>
```

Пример 2. Скрипт вывода содержимого таблицы MySQL в виде HTML:

```
<?php
$host = "localhost";
$user = "user";
$password = "secret_password";

// Производим попытку подключения к серверу MySQL:
if (!mysql_connect($host, $user, $password))
{
echo "<h2>MySQL Error!</h2>";
exit;
}

// Выбираем базу данных:
mysql_select_db($db);

// Выводим заголовок таблицы:
echo "<table border=\"1\" width=\"100%\" bgcolor=\"#FFFFE1\">";
echo "<tr><td>Email</td><td>Имя</td><td>Месяц</td>";
echo "<td>Число</td><td>Пол</td></tr>";

// SQL-запрос:
$q = mysql_query ("SELECT * FROM mytable");

// Выводим таблицу:
for ($c=0; $c<mysql_num_rows($q); $c++)
{
echo "<tr>";
$f = mysql_fetch_array($q);
echo "<td>$f[email]</td><td>$f[name]</td><td>$f[month]</td>";
echo "<td>$f[day]</td><td>$[s]</td>";
echo "</tr>";
}
echo "</table>";
?>
```

Список использованных источников

1. Котеров Д. PHP 7 - СПб.: БХВ-Петербург, 2016. - 1088 с.
2. Веллинг Л. Разработка Web-приложений с помощью PHP и MySQL. - 4-е изд. - М.: ООО «И.Д. Вильямс», 2010. - 848 с.
3. Колисниченко Д. PHP и MySQL. Разработка веб-приложений. - М.: БХВ-Петербург, 2015. - 592 с.
4. Маклафлин Б. PHP и MySQL. Исчерпывающее руководство. - СПб.: Питер, 2013. - 512 с.
5. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL и JavaScript. - СПб.: Питер, 2011 - 496 с.
6. Пьюривал С. Основы разработки веб-приложений. - СПб.: Питер, 2015 - 272 с.
7. Янк К. PHP и MySQL. От новичка к профессионалу. - М.: Эксмо, 2013. - 384 с.

Учебное издание

ЛЫСАНОВ ДЕНИС МИХАЙЛОВИЧ

ЯЗЫК РАЗРАБОТКИ СЦЕНАРИЕВ PHP И СУБД MYSQL

Электронный образовательный ресурс по дисциплине

«Web-программирование»

Отпечатано в издательско-полиграфическом центре
Набережночелнинского института
Казанского (Приволжского) федерального университета

Подписано в печать 07.05.2018 г.
Формат 60x84/16. Печать ризографическая.
Бумага офсетная. Гарнитура «Times New Roman».
Усл. печ. л. 4,6. Уч.-изд. л. 3,6.
Тираж 50 экз. Заказ №988

423810, г. Набережные Челны, Новый город, проспект Мира, 68/19
тел./факс (8552) 39-65-99 e-mail: ic-nchi-kpfu@mail.ru