Защита информации в БД

Два наиболее общих подхода к обеспечению безопасности данных в современных СУБД: *избирательный подход* и *обязательный подход.*

Эти два подхода отличаются следующими свойствами:

1. В случае избирательного управления некоторый пользователь обладает различными правами (привилегиями или полномочиями) при работе с данными объектами. Разные пользователи могут обладать разными правами доступа к одному и тому же объекту. Избирательные права характеризуются значительной гибкостью.
2. В случае обязательного управления, наоборот, каждому объекту данных присваивается некоторый классификационный уровень, а каждый пользователь обладает некоторым уровнем допуска. При таком подходе доступом к определенному объекту данных обладают только пользователи с соответствующим уровнем допуска.

3. Для реализации избирательного принципа предусмотрены следующие методы. В базу данных вводится новый тип объектов БД — это пользователи. Каждому пользователю в БД присваивается уникальный идентификатор. Для дополнительной защиты каждый пользователь кроме уникального идентификатора снабжается уникальным паролем, причем если идентификаторы пользователей в системе доступны системному администратору, то пароли пользователей хранятся чаще всего в специальном кодированном виде и известны только самим пользователям.

4. Пользователи могут быть объединены в специальные группы пользователей. Один пользователь может входить в несколько групп. В стандарте вводится понятие группы PUBLIC, для которой должен быть определен минимальный стандартный набор прав. По умолчанию предполагается, что каждый вновь создаваемый пользователь, если специально не указано иное, относится к группе PUBLIC.

5. Привилегии или полномочия пользователей или групп — это набор действий (операций), которые они могут выполнять над объектами БД.

6. В последних версиях ряда коммерческих СУБД появилось понятие роли. Роль — это поименованный набор полномочий. Существует ряд стандартных ролей, которые определены в момент установки сервера баз данных. И имеется возможность создавать новые роли, группируя в них произвольные полномочия. Введение ролей позволяет упростить управление привилегиями пользователей, структурировать этот процесс. Кроме того, введение ролей не связано с конкретными пользователями, поэтому роли могут быть определены и сконфигурированы до того, как определены пользователи системы.

7. Пользователю может быть назначена. одна или несколько ролей.

8. Объектами БД, которые подлежат защите, являются все объекты, хранимые в БД: таблицы, представления, хранимые процедуры и триггеры. Для каждого типа объектов есть свои действия, поэтому для каждого типа объектов могут быть определены разные права доступа.

Проверка полномочий основана на том, что каждому пользователю или процессу информационной системы соответствует набор действий, которые он может выполнять по отношению к определенным объектам.

Проверка подлинности означает достоверное подтверждение того, что пользователь или процесс, пытающийся выполнить санкционированное действие, действительно тот, за кого он себя выдает.

Дополнительные средства защиты БД:

- встроенные средства контроля значений данных в соответствии с типами;
- повышения достоверности вводимых данных;
- обеспечения целостности связей таблиц;
- организации совместного использования объектов БД в сети.

Средства повышения достоверности вводимых значений в СУБД служат для более глубокого контроля, связанного с семантикой обрабатываемых данных (возможность при создании таблицы указывать следующие ограничения на значения).

Более совершенной формой организации контроля достоверности информации в БД является разработка хранимых процедур. Механизм хранимых процедур применяется в БД, размещенных на сервере. Сами хранимые процедуры представляют собой программы, алгоритмы которых предусматривают выполнение некоторых функций над данными.

Решение прикладной задачи, как правило, требует информации из нескольких таблиц. Сами таблицы для удобства обработки и исключения дублирования информации некоторым образом связываются. Функции поддержания логической целостности связанных таблиц берет на себя СУБД. К сожалению, далеко не все СУБД в полной мере реализуют эти функции, в этом случае ответственность за корректность связей возлагается на приложение.

Изменение данных

Данные в таблице можно изменять с помощью функции `mysqli_query()` в сочетании с оператором `UPDATE`. Как и ранее, успешное выполнение оператора `UPDATE` не обязательно означает, что данные были фактически изменены. Для того чтобы узнать количество измененных данных в таблице, вам придется вызвать функцию `mysqli_affected_rows()`. Этой функции нужно передать идентификатор подключения. Этой функцией можно пользоваться в сочетании с любым запросом, в результате которого данные могли быть изменены (`UPDATE`, `INSERT`, `REPLACE` или `DELETE`, но не `SELECT`!).

В листинге 11.4 приведен пример программы, позволяющей администратору базы данных изменять любые данные в поле `domain` таблицы `domains_brNN`.

Листинг 11.4. Использование функции `mysqli_query()` для изменения данных в таблице

```
<html> <head>
<title> Листинг 11-4. Использование функции
mysqli_query()
        для изменения данных в таблице
</title> </head> <body>
<?php
$user = "pGG";  // здесь GG – номер группы
$pass = "pGG"; $db = "sample"; $table = "domains_brNN";
$conn = mysqli_connect("localhost", $user, $pass); if (!$conn) die("Нет соединения с MySQL");
mysqli_select_db($conn, $db)
or die ("Нельзя открыть $db");
if (isset($_POST['domain']) && isset($_POST['id'])) {
    $query = "UPDATE $table
              SET domain='{$_POST['domain']}' WHERE id='{$_POST['id']}' ";
    $result = mysqli_query($conn, $query);
    if (!$result) die
    ("Нельзя обновить: " . mysqli_error($conn)); print"<p>Таблица $table обновлена: "
        . mysqli_affected_rows($conn) . " строк изменено";
}
?>
```

```
<?php
print "<form action='{$_SERVER['PHP_SELF']}'
method='post'>";
?>
<select name="id">
    <?php
    $result = mysqli_query($conn, "SELECT domain, id FROM
$table");
    while ($a_row = mysqli_fetch_array($result)) {
        print "<OPTION VALUE='$a_row[id]'";
        if (isset($id) && $id == $a_row[id])
            print " SELECTED";
        print "> $a_row[domain]\n";
    }
    mysqli_close($conn);
    ?>
</select>
  Введите новое значение:
<input type="text" name="domain">
<p><input type="submit" value="Заменить">
    </form>
</body>
</html>
```

Если аргумент функции mysqli_query() является переменной, значение которой было передано из другой программы с помощью метода POST, и в этой переменной содержится SQL-запрос, в котором есть апострофы, например

$query = "SELECT * FROM domains where id='2' ";

то эти апострофы будут экранированы обратной косой чертой (\). Для того, чтобы можно было использовать значение переменной $query в функции mysqli_query(), необходимо применить функцию stripslashes(), которая убирает все управляющие символы (в том числе и обратную косую черту перед апострофами) из указанной строки.

В листинге 11.5 приведен пример программы, использующей функцию `stripslashes()`.

Листинг 11.5. Использование функции `stripslashes()`

```
<html> <head>
<title>Листинг 11-5. Использование функции stripslashes()
</title> </head> <body>
<?php
$user = "pGG"; $pass = "pGG"; $db = "sample";
$table = "domains_brNN";
    $conn = mysqli_connect("localhost", $user, $pass);
    if (!$conn) die("Нет соединения с MySQL");
    mysqli_select_db($conn, $db) or die ("Нельзя открыть
$db");
    $query = "SELECT * FROM $table WHERE id='2' ";
    settype($query, "string");
    $plain_query = stripslashes($query);
    $result = mysqli_query($conn, $plain_query);
    $a_row = mysqli_fetch_array($result);
    print "<p>Ваше имя: $a_row[myname]";
    mysqli_close($conn);
?>
</body> </html>
```

Перед использованием функции `stripslashes()` желательно явно задать тип «строка» для переменной, которая будет ее аргументом, что делается с помощью функции `settype()`.

Получение информации о базе данных

До сих пор мы рассматривали функции, предназначенные для сохранения и чтения данных. Однако в PHP есть несколько функций для того, чтобы вы могли получить дополнительную информацию о базах данных, доступных в данном подключении.

Список баз данных

Список всех баз данных, доступных для данного подключения, можно получить с помощью специального запроса:

```
$db_list = mysqli_query($conn, "SHOW DATABASES");
```

После этого полученным списком таблиц можно пользоваться аналогично результату любого запроса. В частности, его можно передавать функции `mysqli_fetch_row()`, которая вернет ассоциативный массив с именами баз данных.

Список таблиц в базе данных

Список всех таблиц, доступных для данной базы данных, можно получить с помощью специального запроса:

```
$table_list = mysqli_query($conn, "SHOW TABLES");
```

Если указанная база данных существует и вы имеет соответствующие права, то функция вернет вам идентификатор результата, который можно обработать функцией `mysqli_fetch_row()` или аналогичной ей.

В следующем примере показано, как с помощью функции `mysqli_list_tables()` можно получить список всех таблиц базы данных.

```
$table_list = mysqli_query($conn, "SHOW TABLES"); while ($tab_rows = mysqli_fetch_row($table_list))
      print "$tab_rows[0]<br>\n";
```

Листинг 11.6. Вывод структуры базы данных

```
<html> <head>
<title> Листинг 11-6. Вывод структуры базы данных </title> </head>
<body>
<?php
$user = "pGG"; $pass = "pGG"; // здесь GG - номер группы
$db = "study";
$conn = mysqli_connect("localhost", $user, $pass); if (! $conn ) die("Нет соединения с MySQL");
mysqli_select_db($conn, $db);
$tab_res = mysqli_query($conn, "SHOW TABLES"); print "<dl><dd>\n";
while ($tab_rows = mysqli_fetch_row($tab_res))
{  #1            созд-е массива имен таблиц
    print "<b>$tab_rows[0]</b>\n";
//$tab_rows[0] т.к. работаем только с одной БД
    $query_res = mysqli_query($conn,"SELECT * from $tab_rows[0]");
    $num_fields = mysqli_num_fields($query_res); print "<dl><dd>\n";
    for ($x=0; $x<$num_fields; $x++)
    {  #2
        $properties = mysqli_fetch_field_direct($query_res, $x); print "<i>";
        print $properties->type;
        // тип поля
        print "</i> <i>";
        print $properties->length;
        // max-ая длина поля
        print "</i> <b>";
        print $properties->name;
        // имя поля
        print "</b> <i>";
        print $properties->flags;
        // флаги поля (not null и т.п.) print "</i><br>\n";
    }  #2
    print "</dl>\n";
}  #1
print "</dl>\n";  mysqli_close($conn); ?>  </body> </html>
```

cust

int 4 **cnum** *not_null primary_key*
string 10 **cname** *not_null*
string 10 **city** *not_null*
int 6 **rating** *not_null*
int 5 **snum**

ord

int 4 **onum** *not_null primary_key*
real 7 **amt** *not_null*
date 10 **odate**
int 4 **cnum**
int 4 **snum**

sal

int 4 **snum** *not_null primary_key*
string 10 **sname** *not_null*
string 10 **city** *not_null*
real 7 **comm** *not_null*

Работа с файлами

Включение файлов в документ

В каждый PHP-документ можно включить файл с помощью команды `include()`. После этого текст PHP-программы во включаемом файле будет исполняться точно так же, как если бы он был записан во включающем файле непосредственно.

Листинг 12.1. Использование функции `include()`

```
<html> <head>
<title>Листинг 12-1. Использование функции include()
</title> <body>
<?php
  include ("ls12-2.php");
?> </body> </html>
```

Листинг 12.2. Включаемый файл с PHP-программой

```
<?php
print " А меня вставили (;o)<br>";
print "Да еще и считать заставили... 4 + 4 = ".(4+4);
?>
```

Листинг 12.3. Использование функции `include()`, возвращающей значение

```
<html> <head>
<title> Листинг 12-3. Использование функции include(),
        возвращающей значение </title> </head> <body>
<?php
$addResult = include("ls12-4.php");
print "Вставленный файл возвращает $addResult";
?>
</body> </html>
```

Листинг 12.4. Включаемый файл, возвращающий значение

```
<?php
$retval = ( 4 + 4 );
return $retval;
?>
Этот текст никогда не появится на экране, потому что он
находится после оператора return!
```

Команду `include()` можно использовать внутри условного оператора. В таком случае включаемый файл будет выполняться только при выполнении условия.

Например, в следующем фрагменте команда `include()` не будет выполнена.

```
$test = false;
if ($test)
   {
    include("a_file.txt");  // не будет включен
   }
```

Листинг 12.5. Использование `include()` внутри цикла

```
<html> <head>
<title> Листинг 12-5. Использование include()
        внутри цикла </title> </head> <body>
<?php
for ($x = 1;  $x<=3;  $x++)
    {
     $incfile = "incfile$x".".txt";
     print "Включаем файл $incfile<br>";
     include("$incfile");
     print "\n<p>";
    }
?>
</body> </html>
```

В языке PHP существует команда `require()`, которая действует аналогично команде `include()`. Эту команду тоже можно использовать в циклах, однако возвращать значения она не может.

Исследование файлов

Проверка существования файла

Для того чтобы проверить, существует ли нужный вам файл, применяется функция `file_exists()`. Эта функция принимает строку, содержащую полный или относительный путь к файлу. Если файл найден, то функция возвращает `true`, в противном случае — `false`.

```
if (file_exists("test.txt"))
    print "test.txt найден";
```

Файл или каталог?

Иногда нужно убедиться, что исследуемый объект действительно является

файлом, а не каталогом. Для этого существует функция `is_file()`. Эта функция требует указания пути к файлу и возвращает булево значение:

```
if (is_file("test.txt"))
    print "test.txt - это файл";
```

Иногда нужно убедиться, что исследуемый объект является каталогом. Для этого существует функция `is_dir()`. Эта функция требует указания пути к каталогу и тоже возвращает булево значение:

```
if (is_dir("/tmp"))
    print "/tmp - это каталог";
```

Проверка статуса файла

После того как файл найден и вы убедились в том, что это именно то, что вам нужно, можете узнать, каков статус этого файла, т.е., другими словами, что с ним можно делать — читать его, или писать в него, или выполнять его. Для этого в PHP есть несколько функций.

Функция `is_readable()` сообщает вам о том, можете ли вы читать указанный файл, В системе UNIX бывают случаи, что вы видите файл, но не имеете возможности читать его содержимое. Функция `is_readable()` требует строку, содержащую имя и путь к файлу, и возвращает булево значение:

```
if (is_readable("test.txt"))
    print "test.txt можно читать";
```

Функция `is_writable()` сообщает вам о том, существует ли у вас возможность писать в указанный файл. Функция `is_writable()` требует строку, содержащую имя и путь к файлу, и возвращает булево значение:

```
if (is_writable("test.txt"))
    print "в test.txt можно писать";
```

Функция `is_executable()` сообщает вам о том, можете ли вы исполнять указанный файл. Это определяется на основании расширения имени файла или на основании ваших прав доступа к нему в данной операционной системе.

Функция `is_executable()` требует строку, содержащую имя и путь к

файлу, и возвращает булево значение:

```
if (is_executable("test.txt"))
    print "test.txt можно выполнять";
```

Определение размера файла

Функция `filesize()` определяет размер файла в байтах. Она возвращает значение `false`, если определить размер файла не удается.

```
print "Размер файла
test.txt - "; print
filesize("test.txt");
```

Создание и удаление файлов

Если нужный вам файл еще не существует, то его можно создать с помощью функции `touch()`. Получив строку с именем файла, эта функция создает пустой файл с заданным именем. Если такой файл уже существует, то функция не изменяет его содержания, но изменяет дату модификации.

```
touch("myfile.txt");
```

Существующий файл можно удалить с помощью функции `unlink()`.
Эта функция тоже получает имя файла.

```
unlink("myfile.txt");
```

Информация о дате и времени

С помощью функции `fileatime()` можно узнать, когда к файлу последний раз осуществлялся доступ. Функция требует указания имени файла и возвращает дату последнего доступа к нему. Под доступом к файлу понимается чтение или запись в него. Время возвращается в системе эпохи UNIX, т.е. представляет собой количество секунд, прошедших после 1 января 1970 г. В последующих примерах мы преобразуем это число в понятный для человека формат с помощью функции `date()`. Подробнее о работе с датами и об их преобразовании будет рассказано в 14-й главе «Работа с датами».

```
$atime = fileatime("test.txt");
print "Последний раз доступ к test.txt был ";
print date("D d M Y g:i A",  $atime);
// Вывод: Mon 10 Feb 2003 1:30 PM
```

Дату последней модификации файла можно узнать с помощью функции `filemtime()`. Функция требует указания имени файла и возвращает дату последнего изменения содержимого файла.

```
$mtime = filemtime("test.txt");
print "Последний раз test.txt был модифицирован ";
print date("D d M Y g:i A",  $mtime);
// Вывод: Mon 10 Feb 2003 1:30 PM
```

Так же можно узнать дату последнего изменения файла с помощью функции `filectime()`. В системе UNIX дата изменения устанавливается тогда, когда изменяется содержимое файла, или права доступа к нему, или его владелец. В других системах эта функция возвращает дату создания файла.

```
$ctime = filectime("test.txt");
print "Последний раз test.txt был изменен ";
print date("D d M Y g:i A", $ctime);
// Вывод: Mon 10 Feb 2003 1:30 PM
```

Все функции для создания, удаления или модификации файлов требуют, чтобы были правильно установлены права доступа.

Открытие файла для чтения, записи или добавления

Чтобы с файлом можно было работать, его нужно открыть для чтения, или записи, или же для того и другого. Для этого существует функция `fopen()`. Данной функции передаются два аргумента — строка с именем файла и путем к нему, а также строка, описывающая режим открытия файла. Самые обычные режимы доступа к файлу — это чтение, запись и добавление в конец файла. Соответствующие строки выглядят как `"r"`, `"w"` и `"a"`. Функция `fopen()` возвращает целое число, которое позже используется для доступа к открытому файлу. Это число называется указателем на файл, и его можно присвоить переменной. Для того чтобы открыть файл для чтения, нужно написать такую конструкцию:

```
$fp = fopen("test.txt", "r");
```

Соответственно, для записи следует открывать файл следующим образом:

```
$fp = fopen("test.txt", "w");
```

Для добавления в конец файла функция открытия выглядит так:

```
$fp = fopen("test.txt", "a");
```

Функция `fopen()` возвращает `false`, если открыть файл по какой-то причине не удалось. Поэтому перед тем как начинать работать с файлом, следует проверить значение возвращенного функцией указателя. Сделать это можно таким образом:

```
if ($fp = fopen("test.txt", "w"))
   {// работа с файлом
   }
```

Если открыть файл не удалось, есть возможность прервать выполнение программы:

```
$fp = fopen("test.txt", "w")
 or die ("Не удается открыть файл");
```

Если функция `fopen()` вернет `true`, остальная часть выражения не будет обрабатываться, и функция `die()`, которая просто выводит свой аргумент на экран браузера заканчивает выполнение программы, никогда не будет вызвана. Если `fopen()` вернет `false`, то будет вызвана функция `die()`.

Если все прошло гладко и вы выполнили все необходимые действия с файлом, то по окончании работы его следует закрыть. Для этого существует функция `fclose()`, которой нужно передать указатель на файл, полученный ранее от функции `fopen()`:

```
fclose($fp);
```

Чтение из файла

В языке PHP есть несколько функций для чтения данных из файла.

Чтение строк с помощью `fgets()`

Для чтения строки из файла существует функция `fgets()`, и ей нужно передать указатель на открытый файл. Кроме того, у этой функции есть второй аргумент, указывающий максимальное число символов, которое можно прочесть из файла до того, как встретится конец строки или файла. Функция `fgets()` будет читать данные из файла до тех пор, пока не встретит символ конца строки или конца файла.

```
$line = fgets($fp, 1024); // $fp - это указатель на файл
```

С помощью описанной функции можно читать строки, но вам нужен способ определить достижение конца файла. Для этого существует функция `feof()`. Данная функция возвращает `true` при достижении конца файла и `false` — в противном случае. Этой функции тоже нужно передавать указатель на файл.

```
feof($fp); // $fp - это указатель на файл
```

Листинг 12.6. Открытие файла и чтение из него строк

```
<html> <head>
<title>Листинг 12-6. Открытие файла и чтение из него
       строк </title> </head> <body>
<?php
$filename = "test.txt";
$fp = fopen($filename,"r")
      or die("Нельзя открыть $filename");
while (! feof($fp))
     {
      $line = fgets($fp, 1024);
      print "$line<br>";
     }
?>
</body> </html>
```

Чтение произвольного количества символов с помощью функции `fread()`

Можно читать данные из файла не по строкам, а кусками произвольного размера. Для этого существует функция `fread()`. Эта функция имеет два аргумента — указатель на файл и количество символов, которое нужно прочесть. Возвращает данная функция количество символов, которое ей фактически удалось прочесть. Это число может не совпасть с затребованным при достижении конца файла или в других случаях.

```
$var = fread($fp, 16);
```

В листинге 12.7 приведена измененная программа, читающая данные из файла не по строкам, а порциями по 16 байт.

Листинг 12.7. Чтение файла функцией `fread()`

```
<html> <head>
<title> Листинг 12-7. Чтение файла функцией
        fread()</title> </head> <body>
<?php
$filename = "test.txt";
$fp = fopen($filename,"r")
      or die("Нельзя открыть $filename");
```

```
while (! feof($fp))
    {
     $var = fread($fp,16);
     print "$var<br>";
     }
?>
</body> </html>
```

Функция `fread()` позволяет вам указать, сколько символов нужно прочесть из файла, но она не позволяет выбирать, с какого места начинать чтение. Для этого существует функция `fseek()`. Данная функция имеет два аргумента — указатель на файл и количество символов, на которое нужно отступить от начала файла.

Это число называют смещением.

```
fseek($fp,64);
```

Чтение отдельных символов с помощью `fgetc()`

Функция `fgetc()` работает почти так же, как функция `fread()`, разница заключается в том, что она читает каждый раз по одному символу. Поэтому ей не нужен второй аргумент. Вы просто передаете указатель на файл.

```
$char = fgetc($fp);
```

Запись в файл

Для записи в файл существуют два разных режима. Разница между ними заключается в способе вызова функции `fopen()`. Если вы откроете файл в простом режиме записи, т е. запишите функцию так:

```
fopen("test.txt", "w"),
```

то в существующем файле вся информация будет уничтожена и новые данные записаны в начало файла. Если такого файла не существует, то он будет создан. Если вы откроете файл в режиме добавления, т.е. запишите функцию так:

```
fopen("test.txt", "a"),
```

то все новые данные будут добавлены в конец файла.

Запись в файл с помощью `fwrite()`

У функции `fwrite()` есть два аргумента — указатель на файл и строка. Функция `fwrite()` просто записывает строку в файл.

Листинг 12.8. Запись в файл и добавление в его конец

```
<html> <head>
<title> Листинг 12-8. Запись в файл и
        добавление в его конец </title>
        </head> <body>
<?php
$filename = "test.txt";
 $fp = fopen($filename,"w")
      or die("Нельзя открыть $filename");
print        "Пишем        в
$filename<br>";   fwrite($fp,
"Привет            всем!\n");
fclose($fp);
$fp = fopen($filename,"a")
      or die("Нельзя открыть $filename");
print "Добавляем в конец
$filename<br>"; fwrite($fp,"Еще
дописали :-)\n");  fclose($fp);
?>
</body> </html>
```

Блокировка файла

В PHP есть специальная функция, которая предназначена для предотвращения подобных ситуаций. Функция `flock()` блокирует файл, т.е. не позволяет другим пользователям читать этот файл или писать в него до тех пор, пока процесс, наложивший блокировку, не закончит работу с данным файлом. Эта функция имеет два аргумента — указатель на файл и целое число, указывающее режим блокировки.

В таблице 12.1 приведены режимы блокировки, которые вы можете применить к файлу.

Таблица 12.1 — Режимы блокировки функции `flock()`

Номер режима	Тип блокировки	Описание
1	Частичная	Разрешает другим процессам читать файл, но запрещает запись в него
2	Полная	Запрещает другим процессам как чтение, так и запись в файл
3	Освобождение	Снимает блокировку с файла

Функцию `flock()` нужно вызывать сразу после открытия файла и потом вызывать ее повторно для освобождения.

```
$fp = fopen("test.txt","a");
flock($fp, 2);  // Полная блокировка
// Запись чего-то в файл
flock($fp, 3);  // Освобождение файла
fclose($fp);
```

Блокировка с помощью функции `flock()` не является абсолютной. С ней будут считаться только те программы, которые тоже пользуются этой функцией.

Работа с каталогами

Теперь, когда вы умеете проверять существование файла, можете читать его или писать в него, обратимся к работе с каталогами. В языке PHP есть несколько функций для работы с каталогами, с помощью которых вы можете создавать каталоги, удалять их или читать.

Создание каталога

Для создания каталога предназначена функция `mkdir()`. Она имеет два аргумента — строку, содержащую путь и имя нового каталога, и целое восьмеричное число, описывающее режим создания каталога. Этот аргумент оказывает влияние только в операционной системе UNIX. Восьмеричное число, описывающее режим создания, должно начинаться с нуля, а затем идут три цифры (каждая от 0 до 7), которые представляют собой права доступа (чтения — `r`; записи — `w`; исполнения программ — `x`) для владельца, группы и для всех остальных. Примеры задания прав доступа приведены в таблице 12.2.

Таблица 12.2 — Задание прав доступа к каталогу

	владелец	**группа**	**остальные**
Права доступа	r w x	r w x	r w x
Что разрешается (в виде двоичного числа: 1 – разрешено, 0 – запрещено)	1 1 1	1 1 1	1 1 1
Восьмеричное число	7	7	7
Двоичное число	1 1 1	1 0 1	1 0 0
Восьмеричное число	7	5	4

Функция `mkdir()` возвращает `true` в случае успеха и `false` — в противном случае.

```
mkdir("testdir", 0777); // "разрешено всё всем"
chdir("testdir");       // переход в директорию
testdir
```

Удаление каталога

Для удаления каталога из файловой системы предназначена функция `rmdir()`. Удалить каталог можно только в том случае, если ваша программа имеет на это право и если каталог пуст. У данной функции есть только один аргумент — имя и путь к удаляемому каталогу.

```
rmdir("testdir");
```

Открытие каталога для чтения

Перед тем как читать содержимое каталога, его нужно открыть и получить указатель на него с помощью функции `opendir()`. Эта функция имеет только один аргумент — имя и путь к каталогу. Она возвращает `true`, если каталог был успешно открыт, и `false` — в противном случае. Ошибка при открытии может быть вызвана тем, что каталог не существует или ваша программа не имеет права его читать.

```
$fp = opendir("testdir");
```

Чтение каталога

Точно так же, как вы читали строки из файла с помощью функции `fgets()`, у вас есть возможность читать в каталоге имена файлов и подкаталогов с помощью функции `readdir()`. Эта функция имеет один аргумент — имя каталога — и возвращает строку, содержащую имя найденного объекта, т.е. файла или подкаталога. По достижении конца файла она возвращает `false`. Имейте в виду, что функция `readdir()` возвращает имена объектов, но не пути к ним.

Листинг 12.9. Чтение каталога с помощью функции `readdir()`

```
<html> <head>
<title> Листинг 12-9. Чтение каталога с помощью
        функции readdir()</title> </head> <body>
<?php
$dirname = "testdir";
$dh = opendir($dirname);
while (gettype($file = readdir($dh)) != boolean)
      {
       if (is_dir("$dirname/$file")) print "(D)";
       print "$file<br>";
      }
closedir($dh);
?>
</body> </html>
```

Формы и программы для передачи файлов на сервер

HTML-форма, предназначенная для копирования файлов и имеющая соответствующее поле, должна содержать аргумент `ENCTYPE`:

```
ENCTYPE = "multipart/form-data"
```

Кроме того, PHP требует, чтобы в форме перед полем для копирования файлов находилось скрытое поле. Такое скрытое поле должно называться `max_file_size` и в нем должен быть записан максимальный размер файла (в байтах), который вы разрешаете передавать. Этот размер не должен превышать размер, установленный в поле `upload_max_filesize` в файле `php.ini`, который обычно равен 2 Мбайтам. Теперь можно подготовить само поле для передачи файла. Это обычный элемент `INPUT`, у которого в атрибуте `type` записано `"file"`.

Листинг 12.10. Простая форма для передачи файла

```
<html> <head>
<title> Листинг 12-10. Простая форма для передачи файла
</title> </head> <body>
<?php
print "<form enctype='multipart/form-data'
action='{$_SERVER['PHP_SELF']}' method='post'>";
?>
<input type="hidden" name="MAX_FILE_SIZE" value="51200">
<input type="file" name="fupload"><br>
<input type="submit" value="Передать!">
</form>
</body> </html>
```

PHP сохраняет еще некоторую дополнительную информацию о переданном файле в глобальных переменных. Эту информацию можно найти в массиве с глобальной видимостью $_FILES. Подробнее – в таблице 12.3:

Таблица 12.3 — Элементы массива, описывающие переданный файл

Имя переменной	Описание	Пример
`$_FILES[fupload][name]`	Имя переданного файла	`test.gif`
`$_FILES[fupload][size]`	Размер переданного файла в байтах	`6835`
`$_FILES[fupload][type]`	Тип переданного файла в системе MIME	`image/gif`

Листинг 12.11. Программа обработки переданного файла

```
<html> <head>
    <title> Листинг 12-11. Программа обработки
переданного
        файла </title> </head>

<?php
$file_dir = "/tmp/uploads";
$file_url = "http://www.primer.ru/uploads";
if (isset($_FILES['fupload']))
{ #1
    $fupload = $_FILES['fupload'];
    print "<p>Путь:   {$fupload['tmp_name']}\n";
    print "<p>Имя:    {$fupload['name']}\n";
    print "<p>Размер: {$fupload['size']} байт\n";
    print "<p>Тип:    {$fupload['type']}<p>\n";
    if ($fupload['type'] == "image/gif")
    { #2
        copy($fupload['tmp_name'],
"$file_dir/{$fupload['name']}")
        or die("Нельзя копировать");
        print "<img
src=\"$file_url/{$fupload['name']}\">\n";
    } #2
} #1
?>
<body>
<?php
print "<form enctype='multipart/form-data'
action='{$_SERVER['PHP_SELF']}' method='post'>";
?>
    <p>
        <input type="hidden" name="MAX_FlLE_SIZE"
value="5120000">
    <p>
        <input type="file" name="fupload">
    <p>
       <input type="submit" value="Отправить файл">
</form>
</body> </html>
```