ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - ВАРНА

КАТЕДРА “КОМПЮТЪРНИ НАУКИ И ТЕХНОЛОГИИ”

доц. д-р инж. Огнян Железов

Проектиране на Web приложения

(PHP, MySQL)

Настоящият учебник е разработен в съответствие с учебната програма по дисциплината “Проектиране на Web програмни приложения”, която се изучава в ОКС “Бакалавър” на специалност “Компютърни системи и технологии” в Технически университет – Варна. Желателно е ползващите учебника да имат познания по HTML, тъй като в много случаи се генерира програмно HTML код, който се изпраща към потребителския браузър.

Във второто издание на учебника са разгледани някои нови възможности на PHP 5.5.0.

Авторът се е старал да дава изчерпателно описание на разглежданите функции, обекти, атрибути и методи, така, че учебникът да може да предоставя и необходимите справочни данни на разработчиците на PHP приложения.

Авторът изказва благодарност на рецензента, доц. Недялко Николов за ценните препоръки по оформянето на учебника.

Съдържание

1. Общи сведения за PHP

Определението, дадено за PHP в началната страница на Web сайта http://www.php.net , от който се разпространяват неговите дистрибуции, е:

> PHP е многоплатформен, вграждан в HTML скриптов език с общо предназначение. Основното предназначение на PHP е създаването на Интернет приложения като PHP страници или като самостоятелни CGI скриптове.

1.1. Кратка история на PHP

PHP е създаден през 1994г. от Размус Лердорф (Rasmus Lerdorf). Той използвал създаденият програмен език за да записва адресите на посетителите в неговата лична страница. Първата версия, използвана от други потребители е била достъпна през 1995г. под наименованието "Personal Home Page Tools", откъдето идва и съкращението PHP. Тази първа версия е съдържала много опростен парсер, който разпознавал като валидни инструкции само няколко специални макроса и ограничен набор от инструкции с общо предназначение. Липсват някои от основните за съвременните езици за програмиране възможности, като например конструкцията за организиране на цикли *for*.

При разработването на трета версия PHP е пренаписан изцяло наново с участието на Анди Гутманс и Зеев Сураски. В PHP3 е реализиран нов приложен програмен интерфейс (API), който улеснява използването на допълнителни външни модули. След появяването на трета версия през 1998г. броя на домейните с инсталиран PHP достига един милион.

В PHP4 са направени съществени допълнения и изменения, които увеличават възможностите на PHP:

1) Докато PHP3 постоянно парсва скриптовете при изпълнението им, в PHP4 е въведено компилиране до междинен (т.н. байт-код) и последващо изпълнение от Zend Engine (по името на създателите и Зеев и Анди). Компилирането до междинен код и последващо изпълнение

е технология, която се е оказала плодотворна и се прилага и в други програмни среди като Java на Sun Microsystems, а в последствие и в NET Framework на Microsoft.

2) Във версия PHP4.1.0 се въвеждат суперглобалните променливи като $_GET и $_POST , и от версия PHP4.2.0 се забранява по подразбиране конвертирането на входните променливи в глобални (чрез опцията register_globals), с което се елиминира една от „дупките” в сигурността на интернет приложенията, написани на PHP.
3) PHP5 съдържа нова версия на Zend Engine, която разширява обектните възможности на PHP. Обектните възможности на PHP5 са съизмерими с тези на изцяло обектно ориентирани програмни езици като Java. Добавени са и допълнителни възможности като:

- ❖ Структура try-catch за прихващане на изключения
- ❖ Разширена поддръжка на XML чрез интерфейсите SAX, DOM и XSLT, включени в библиотеката libxml2.
- ❖ Поддръжка на Simple Object Access Protocol (SOAP) за обмен на съобщения и достъп до Web услуги чрез HTTP и други транспортни протоколи.
- ❖ Библиотека MySQLi (MySQL impoved), която предоставя процедурен и обектно ориентиран интерфейс към MySQL 4.1 и по-нови версии
- ❖ Разширение SQLite – вградена SQL библиотека, която предоставя функционалност както при работа със сървър за бази от данни, включително вложени заявки и транзакции.

2. Инсталиране и конфигуриране на среда за изпълнение на PHP скриптове

За да тествате дадените в учебника примери е необходимо да инсталирате и конфигурирате на персоналния си компютър среда за изпълнение на PHP скриптове. Това включва инсталирането на локален Web сървър, PHP и MySQL. Те могат да бъдат инсталирани поотделно или чрез инсталационни пакети, които

инсталират едновременно Web сървър, PHP и MySQL и помощна програма PHPMyAdmin за администриране на MySQL бази от данни. Ще разгледаме по-подробно инсталирането на локален Web сървър, PHP и MySQL като самостоятелни продукти, въпреки, че е по-лесно да се използва инсталационен пакет XAMPP или Wamp Server. В някои случаи се налага да се използва обезателно Internet Information Server на Microsoft (напр. при разработване на ASP.NET приложения на Visual Studio.NET), а инсталационните пакети XAMPP и Wamp Server предлагат инсталиране само на Apache Web сървър.

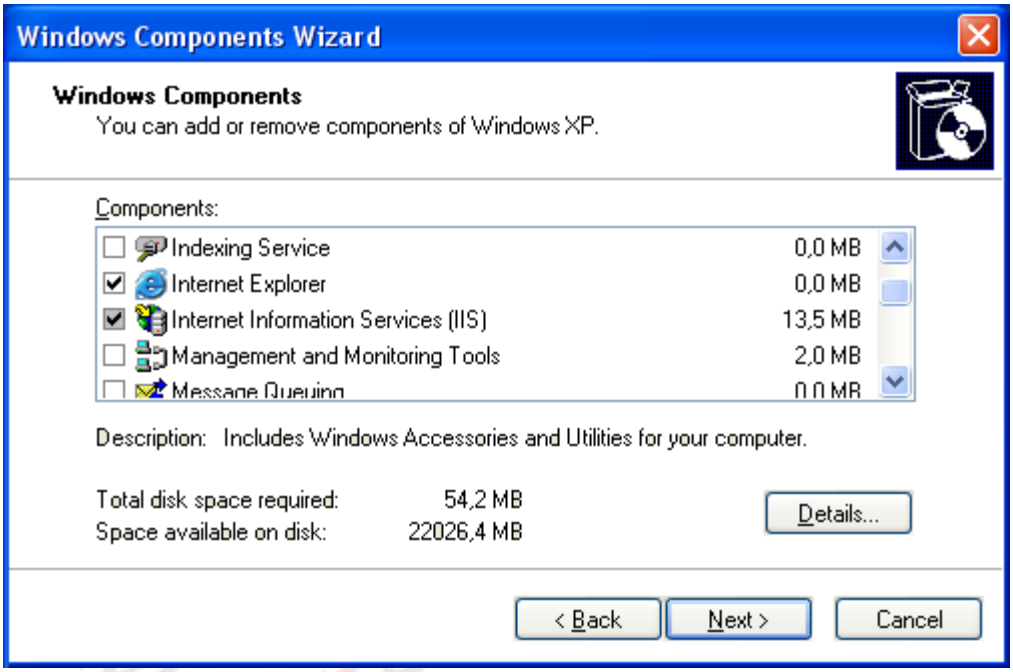
2.1. Инсталиране на Interner Information Server

За Windows 2000/XP

Interner Information Server (IIS) се инсталира като опция от инсталационния диск на Windows 2000/XP. За да го инсталирате:

1) Поставя се в CD-Rom инсталационния диск на Windows
2) От Start Menu на Windows се избира Settings – Control Panel

3) Избера се “Add or Remove Programs”. Извежда се прозорец със списък от инсталираните програми. От лентата вляво се избира “Add/Remove Windows Components”. Извежда се прозорец със списък на опциите за инсталиране

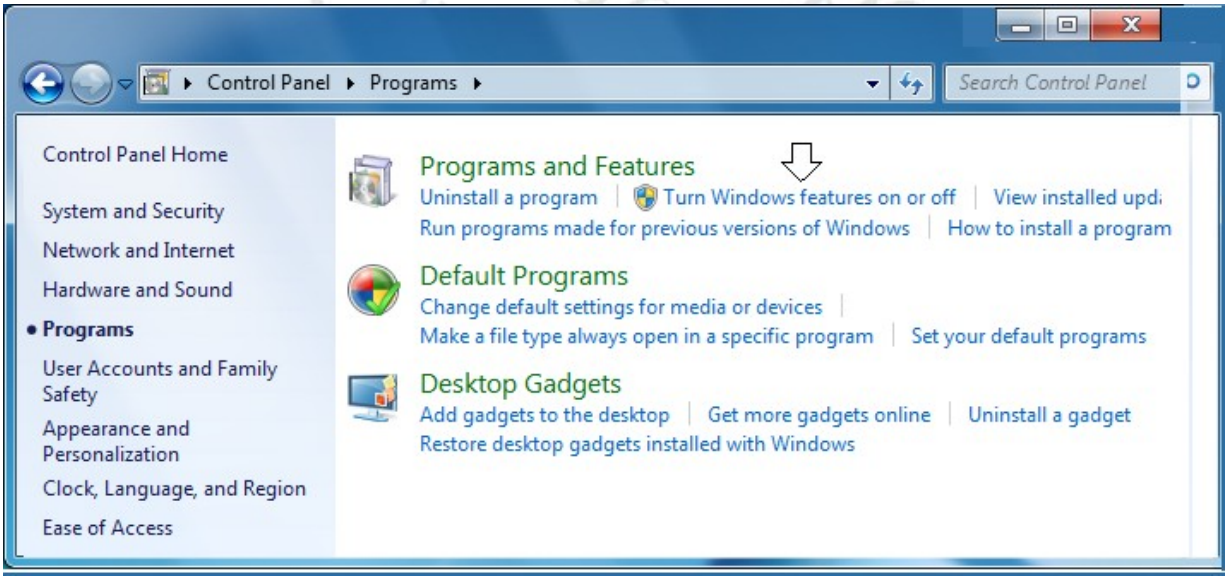


Маркира се опцията Internet Information Services и се изпълняват следващите стъпки от инсталационната програма.

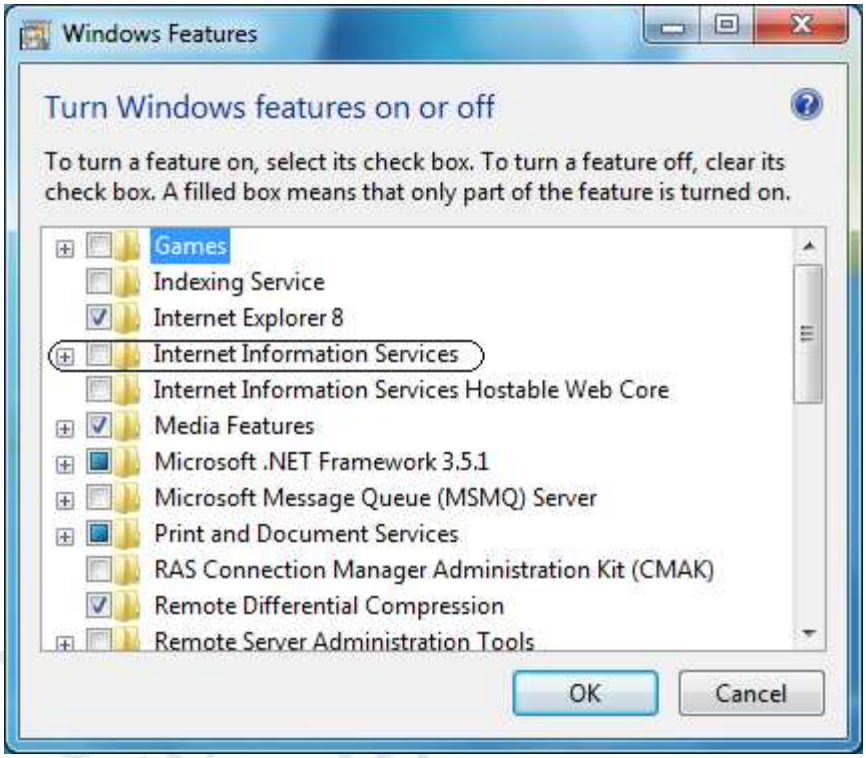
За Windows Windows 7/ Windows 8

Инсталирането е подобно на това за Windows 2000/XP.

1) Поставя се в CD-Rom инсталационния диск на Windows
2) От Start Menu на Windows се избира Settings – Control Panel



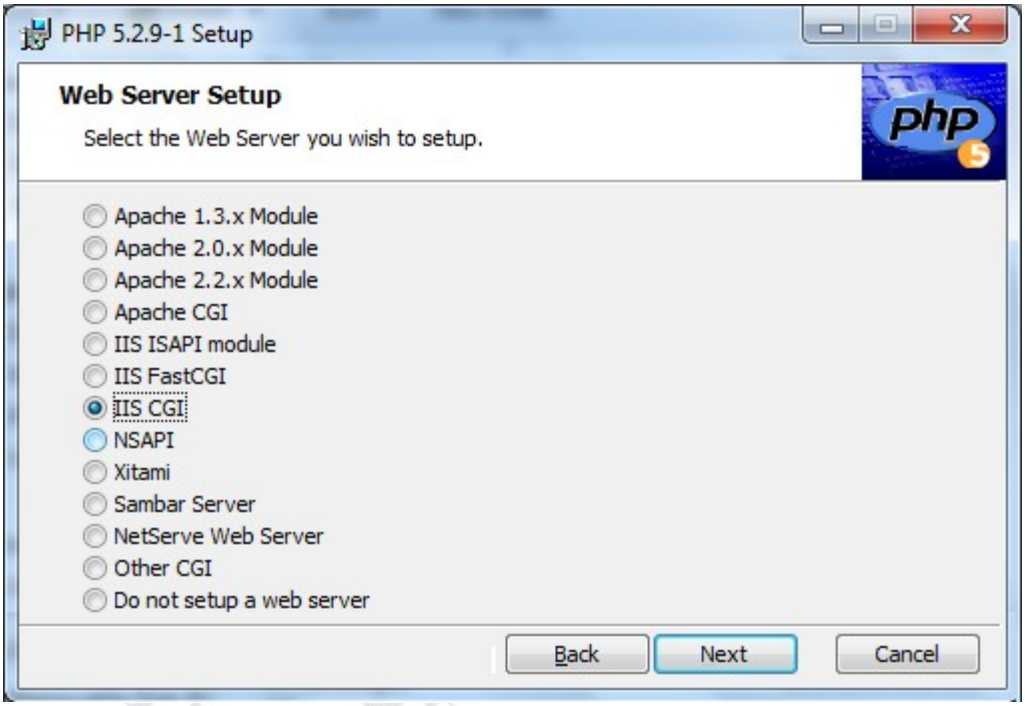
3) От прозореца на Control Panel се избира Programs - Turn Windows features on or off
4) В прозореца “Windows Features” се маркира опцията “Internet Information Services” и се щраква на бутона “OK”



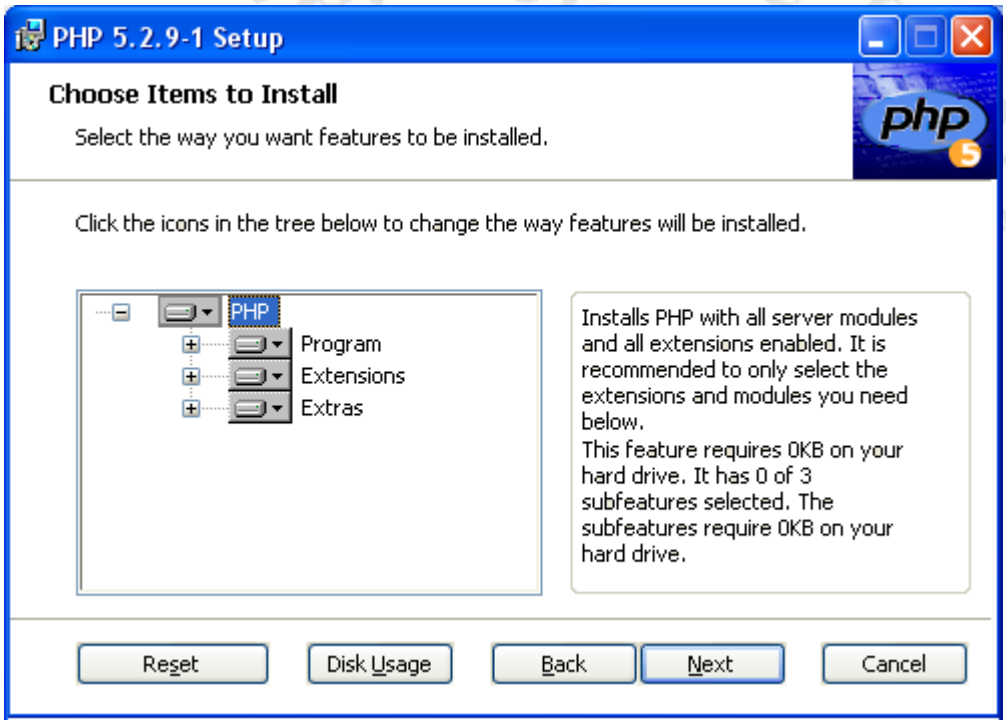
2.2. Инсталиране на PHP в ОС Windows

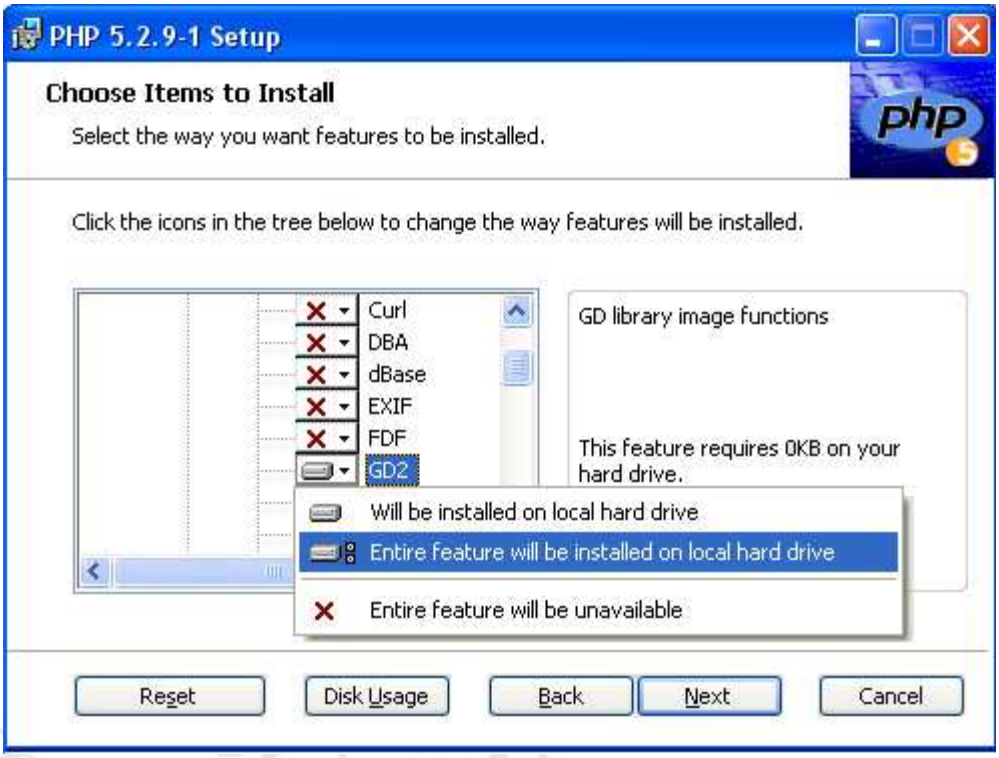
За инсталиране на PHP е необходимо да се изтегли инсталационен продукт от сайта http://www.php.net. За ОС Windows най-лесни за използване са инсталационните програми от групата “Windows Binaries”. Тези инсталационни програми ви дават възможност при инсталиране да изберете кои разширения да се включат в PHP.

При стартиране на инсталационната програма (файлът има разширение .msi), на първите две стъпки от инсталирането се извежда начално съобщение и лицензионно споразумение, което потребителят трябва да потвърди, че приема. На третата стъпка се извежда списък от Web сървъри, които потребителят може да инсталира на компютъра си за да тества Интернет приложения. Ако сте инсталирали Internet information Server, трябва да маркирате „IIS CGI”, както е показано на следващата фигура:



На следващата стъпка от инсталационната програма е необходимо да се изберат разширенията, които ще се добавят към PHP при инсталирането му. В последствие ако разработчикът иска да добави други разширения, той може да стартира отново инсталационната програма и от същия прозорец да избере разширенията, които иска да добави.





При разширяването на групата “Extensions” за разширенията, които искаме да добавим в инсталацията, се щраква върху иконата ✕ на разширението и се избира “Entire feature will be installed on local hard rive”, както е показано на фигурата за разширението GD2. За да се тестват примерите в учебника е необходимо да се инсталират разширенията GD2, MySQL, MySQLi и SQLite.

2.3. Тестване на PHP инсталацията. Модул Hello.php

За да се проверим работоспособността на инсталацията на PHP и да се провери какви разширения са инсталирани, можем да използваме PHP модул със следното съдържание

```
<HTML>
<HEAD>
<TITLE>Hello PHP</TITLE>
</HEAD>
<BODY>
<?php
        echo "Hello PHP";
        phpinfo();
```

```
?>
</BODY>
</HTML>
```

Пример 1 Модул Hello.php

Модулът съдържа код на HTML документ, в секцията BODY на който е включен PHP блок, който съдържа само две инструкции. Първата инструкция включва конструкцията echo, с която се извежда в прозореца на браузъра низа "Hello PHP". С втората инструкция се извиква функцията phpinfo(), която извежда информация за версията и разширенията на инсталирания PHP.

За да се тества модулът, той може да се запише в главната директория на инсталирания локален Web сървър. За IIS това по подразбиране е директорията \inetpub\wwwroot на системното дисково устройство. За да се изпълни модулът, той трябва да се заяви от локалния Web сървър с адреса http://localhost/Hello.php.

2.4. Конфигуриране на PHP – конфигурационен файл php.ini

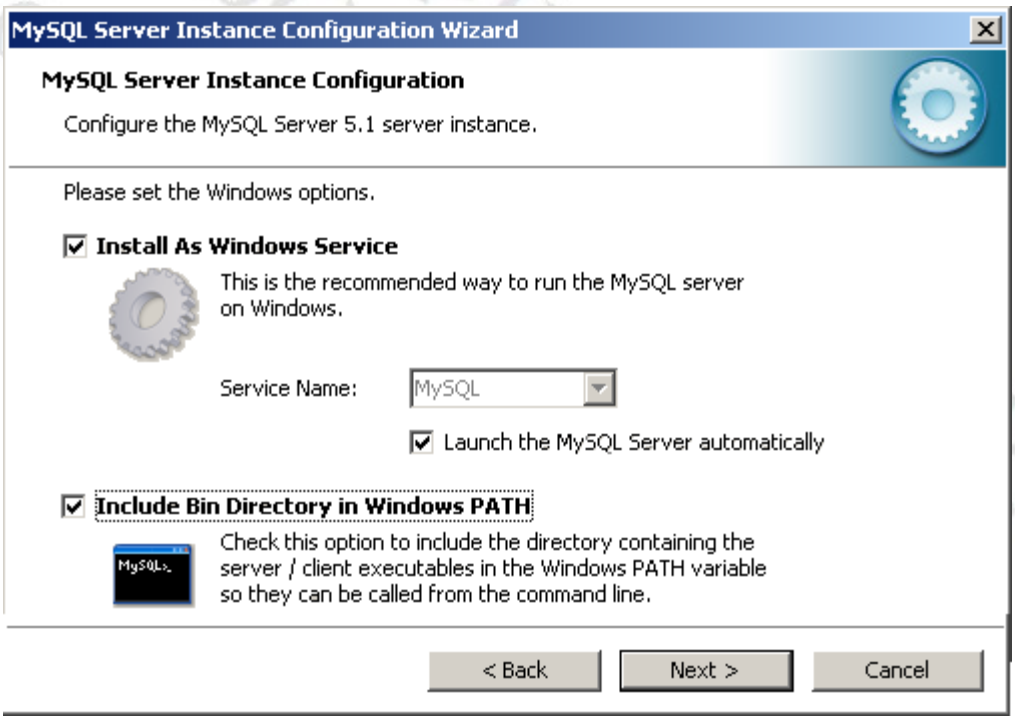
Конфигурационният файл php.ini съдържа набор от параметри с различно значение, някои от които може да се наложи да бъдат променени. По правило той се записва в инсталационната директория на PHP. Спецификацията на файла се извежда като параметър “Loaded Configuration File” от функцията phpinfo(). Функцията извежда и стойностите на конфигурационните параметри. Можете да видите тази информация например като заявите с модула Hello.php, от предходния пример.

Един от параметрите, които е добре да промените, за да улесните тестването на PHP модули на компютъра си е параметъра display_errors. По подразбиране, за последните версии на PHP5 той има стойност Off, с което се забранява извеждането на информация грешките, генерирани при компилирането и изпълнението на PHP модулите. Препоръчва се информацията за грешките да се записва в .log файл. Ако обаче компютърът се използва за изучаване на PHP и тестване на PHP приложения, много по добре е информацията за грешките да се извежда

непосредствено в прозореца на браузъра. За целта във файла php.ini трябва да се зададе стойност „On” на този параметър.

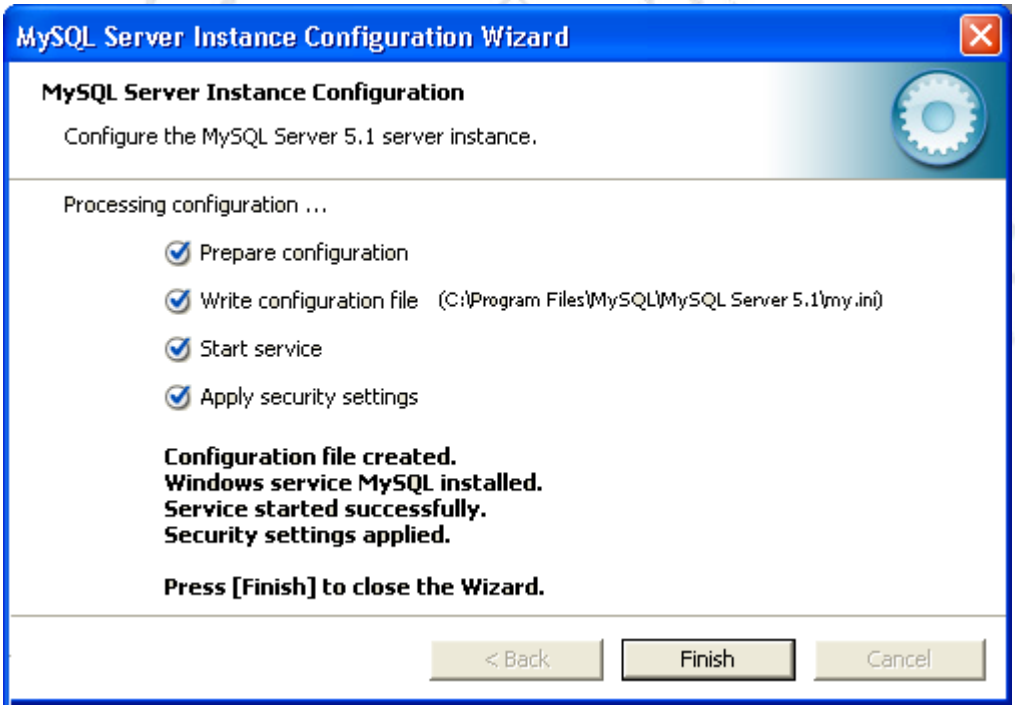
2.5. Инсталиране на MySQL сървър в ОС Windows

За инсталиране на сървъра за бази от данни MySQL е необходимо да се изтегли инсталационен продукт MySQL Community Server от сайта http://dev.mysql.com/downloads/. След инсталирането (най-просто е да се инсталира с опцията „Typical”) се стартира помощната програма за конфигуриране „Configuration Wizard”. Ако на третата стъпка се избере „Standard Configuration”, на четвърта стъпка трябва да се избере опцията „Install As Windows Service”. Опцията „Include Bin Directory in Windows Path” улеснява стартирането на клиентската програма mysql в прозореца „Command Prompt” и също е полезно да се включи.



На следващата стъпка от програмата за конфигуриране трябва да се въведе парола на потребителя root, който се създава при инсталирането. Предвидени са опции за създаване на анонимен акаунт (без парола) и за разрешаване на достъп от отдалечени (инсталирани на друг компютър в мрежата) клиенти.

От съображения за сигурност не се препоръчва създаване на анонимен акаунт и затова тази опция (която беше по подразбиране разрешена до версия 4 на MySQL), във версия 5 по подразбиране е изключена.



При успешно конфигуриране информацията за него се записва в конфигурационен файл my.ini и MySQL сървър се стартира като Windows услуга.

2.6. Инсталационни пакети XAMPP и WampServer

Инсталационните пакети XAMPP и WampServer инсталират:
Apache Web сървър
PHP
MySQL
PHPMyAdmin

Инсталирането им чрез пакетите предоставя на разработчика конфигурирана среда за тестване на PHP приложения. Всички продукти се разпространяват като Open Source, което означава, че може да се изтегли и текста на програмите. Можете да ги изтеглите от адресите:
http://xampp.en.softonic.com/ за пакет XAMPP
http://www.wampserver.com/en/download.php за пакет WampServer
http://www.phpmyadmin.net/home_page/downloads.php за самостоятелно инсталиране на phpMyAdmin

3. Основни елементи на PHP

Създаването на програми на кой да е език за програмиране изисква познаване и правилно използване на елементите на езика – символи, ключови думи, константи, променливи, функции, обекти, коментари. Изучаването на тези програмни елементи, тяхното дефиниране и използване, правилата (синтаксиса) за записване на командите и структурата на програмите са начален етап за изучаване на всеки език за програмиране. Ще разгледаме последователно тези елементи и изисквания.

3.1. Константи

Всяка програма съдържа постоянна информация, която не се променя по време на нейното изпълнение. Такъв тип информация се представя посредством константи.

D1.Константата е величина, която не може да променя стойността си по време на изпълнение на програмата.и се определя от елементите {тип, стойност}.

В PHP константите се включват в самите команди или се дефинират, с което им се присвоява име. Ето един пример за константа, включена в команда:

```
$x= 3.1416;
```

С тази команда в променливата $x се записва числото 3.1416, което, както сте се досетили, е константата π, зададена с точност до четвъртия знак след десетичната точка. Независимо от това колко пъти ще се изпълни тази команда в програмата, резултатът винаги ще бъде един и същ, тъй като вдясно от оператора за присвояване "=" стои константата 3.1416.

Използване на подобни константи многократно в текста на програмите се счита за лош стил на програмиране. По-добре е на константите, които имат някакво специално значение за програмата да се присвоят на имена в началото на програмата, и след това те да се използват с дефинираните си имена, а не непосредствено със стойностите си.

В програмите на PHP се използват три типа константи – числови (целочислени и дробни), символни и логически.

- Целочислени (тип integer) – могат да бъдат в десетична, осмична и шестнадесетична бройна система:
  - Десетични – започват с цифра, различна от 0, например 15
  - Осмични – започват с 0, следвани от цифри от 0 до7 за разрядите на числото, например 017. Преобразувана в десетична, тази константа има стойност $1*8^1+7*8^0=15_{10}$.
  - Шестнадесетични – започват с 0x, следвани от цифри за разрядите на числото, като цифрите със стойност от 10 до 15 се задават с първите букви от латинската азбука от A до F. Например константата 0xAC* е шестнадесетична. Преобразувана в десетична, тази константа има стойност $10*16^1+12*16^0=172_{10}$.

* Могат да се използват и малките букви a-f.

- ❖ Дробни (тип float) – могат да бъдат само десетични. Прието е като разделител между цялата и дробната част да се използва десетична точка вместо десетична запетая, например 12.54. Запетаята в PHP се използва като разделител например при изброяване на елементи на списъци.
- ❖ Символни (тип string) – PHP дава възможност за използване на символни константи, които се записват като последователност от символи, ограничени отляво и отдясно с двойни кавички или апострофи, напр. “Hello “ или ‘World’
- ❖ Логически (тип boolean) – логическите константи “истина” и “неистина” се записват с наименованията си на английски съответно като **TRUE** и **FALSE**.

3.1.1. **Дефиниране на константи**

Дефинирането на константа е всъщност създаване на константа, на която се присвоява име и стойност от определен тип. В PHP това става чрез вградената в езика функция define. Тя има следната декларация

bool **define** (string name, mixed value [, bool case_insensitive])
където:
name е името на създаваната константа
value е стойността на създаваната константа
case_insensitive незадължителен параметър, който показва дали при изписване на името на константата ще се прави разлика между малки и главни букви (при подразбиращата се стойност FALSE) или не се прави такава разлика (при стойност TRUE).

```
define(“PI”,3.14);
```

Пример 2 Дефиниране на константа с име PI и стойност 3.14

В дадения по-горе пример се дефинира константа PI със стойност 3.14 . Тъй като не е зададена стойност на параметъра case_insensitive се подразбира, че при използване константата трябва да се записва както е дефинирана – с главни букви.

Важно е да се запомни, че именуваните константи в PHP са глобални, независимо къде (в рамките на PHP приложението) са дефинирани.

3.1.2. **Проверка за съществуване на именувана константа**

PHP предоставя функцията defined, чрез която може да се провери дали е дефинирана константа със зададено име.
Декларация:

bool **defined** (string name)

Функцията връща логическа стойност TRUE ако константата с име name е дефинирана, и FALSE ако не е дефинирана.

```
if(defined("PI")) echo PI;
```

Пример 3 Проверка за съществуване на константа чрез функция defined

В дадения по-горе пример се проверява дали константата PI е дефинирана, и ако е дефинирана се извежда нейната стойност чрез конструкцията echo.

В PHP са включени голям брой именувани константи. По-голямата част от тях обаче се създават в разширенията на PHP и са достъпни само ако даденото разширение е активно – динамично включено или компилирано съвместно с PHP. Функцията get_defined_constants() връща масив с всички достъпни именувани константи в PHP модула. По-долу е даден код на PHP модул, който извежда в прозореца на браузъра имената и стойностите на всички достъпни именувани константи.

```
<HTML>
<BODY>
<OL>
<?php
  print_r(get_defined_constants());
?>
</OL></BODY></HTML>
```

Пример 4 Извеждане на всички достъпни именувани константи

В примера е използвана функцията print_r, която е разгледана в т. “Итерация на масиви”.

3.1.3. **Получаване на стойност на именувана константа – функция constant**

По правило достъп до именуваните константи се извършва чрез името на константата, както е показано в пример 3. PHP обаче предоставя освен това и функцията constant, която връща стойността на константа по зададено име като параметър на функцията. Описание:

mixed constant (string name)

Функцията constant дава възможност за динамично генериране на името на константата, както е демонстрирано в дадения по-долу пример:

```
for($n=0;$n<10;$n++)
{ //Ако константата A$n съществува, се извежда стойността ù
      if(defined(“A$n”)) echo constant(“A$n”);
}
```

Пример 5 Получаване на стойност на константи чрез функция constant

В примера е организиран цикъл с конструкция for, в който стойността на променливата $n се променя от 0 до 9. При всяко изпълнение на цикъла чрез слепване на низа “A” със стойността на $n се генерира име на константа от A0 до A9, получава се стойността на константата чрез функцията constant и се извежда с конструкция echo.

3.1.4. **“Магически” константи**

PHP предоставя пет предварително дефинирани константи, стойността на които не е постоянна, а зависи от мястото в кода, в което се използват. Това са:

__LINE__ съдържа текущия номер на реда от PHP модула
__FILE__ съдържа пълната файлова спецификация PHP модула
__FUNCTION__ съдържа името на функцията, която се изпълнява или празен низ извън кода на функция.
__CLASS__ съдържа името на класа на обекта, чийто код се изпълнява, или празен низ извън кода на обект.
__METHOD__ съдържа името на метода, който се изпълнява или празен низ извън кода на метод

Тези константи например могат да се използват при извеждане на съобщение за грешка от код на програмиста, за да се определи по-точно инструкцията, при изпълнение на която е генерирана грешката.

3.2. Променливи

Освен константи всеки език за програмиране включва променливи, които представят информацията, която се променя по време на изпълнение на програмата.

D2 Променливата е величина, която може да променя стойността си по време на изпълнение на програмата и се определя от тройката {име, тип, стойност}.

Стойността, съхранявана в променливата се използва чрез нейното име. За всяка променлива при изпълнение на програмата се заделя място в оперативната памет на компютъра. При избор на име на променлива трябва да се спазват следните правила:

! ● Преди имената на променливите в PHP задължително се поставя символа $ (изключение от това правило е само достъпът до променливи-полета на обекти чрез оператора "->"). Имената на променливите трябва да започват с буква или символ за подчертаване "_" и освен тях могат да съдържат и цифри.

Имената на променливите трябва да са уникални (различни) за всички променливи, декларирани в даден контекст (извън функции, във функции или в класове). PHP (както повечето съвременни езици за програмиране) прави разлика между малките и големи букви в имената на променливите. Например за PHP. $x и $X са две различни променливи, докато например за VB Script (скриптов език, създаден от Microsoft и поддържан от Internet Explorer) x и X са една и съща променлива.

В PHP (както и в други скриптови езици)променливите не се декларират явно, а се създават при първото им използване.

3.2.1. **Инициализиране на променливи**

Инициализирането на променливите в не е задължително, но не се счита за добра практика да се използват подразбиращите се стойности, които те получават при създаването си. Стойността на неинициализирана променлива зависи от контекста, в който тя се използват, например:

при използване като логически тип връща FALSE
при използване като числов тип (integer или float) връща 0
при използване като символен тип (string) връща празен низ
при използване като масив (тип array) връща празен масив

Инициализиране със стойност

По подразбиране променливите се инициализират със стойност в съответствие със следния синтаксис:

$variable=expression;

Стойността на израза се изчислява и посредством оператора “=”се присвоява на променливата $variable.

```
$x=5;
$a=”Hello”;
```

Пример 6 Инициализиране на променливи по стойност

Инициализиране с адрес

От версия 4 на PHP е добавена възможност за инициализиране на променливи с адрес. Това инициализиране използва следния синтаксис

$variable2=&$variable1;

При това инициализиране чрез оператора "&" променливата $variable2 получава адреса в паметта (а не просто стойността) на променливата $variable1. Като резултат от това стойността на двете променливи е записана на едно и също място в оперативната памет, т.е. в действителност това е една променлива с две различни имена.

```
$x=5;
$y=&$x;
$y=$y+1;
echo $x;
```

Пример 7 Инициализиране на променлива с адрес

В примера на променливата $y се присвоява адреса на променливата $x, след което се увеличава с единица стойността на $y и се извежда чрез конструкция echo стойността на $x. Тъй като $x и $y са всъщност една и съща променлива с две различни имена, стойността на $x също се оказва увеличава с единица.

3.2.2. **Индиректно обръщение към променливи**

Индиректното обръщение към променливи дава възможност за използване на съдържанието на променлива като име на променлива. Такава възможност липсва в повечето от съвременните езици за програмиране. Индиректното обръщение използва следния синтаксис:

$$variable

При това съдържанието на променливата трябва да бъде символен низ, който да отговаря на изискванията за име на променлива. Да разгледаме един пример.

```
$var=”p1”;
$$var=12;
echo $p1;
```

Пример 8 Индиректно обръщение към променлива

В примера променливата $var получава като стойност символния низ “p1”. В инструкцията $$var=12; съдържанието на $var се използва като име на променлива, в резултат на което се създава променливата $p1 и в нея се записва 12. Стойността на променливата $p1 се извежда посредством конструкцията echo.

Индиректното обръщение дава възможност за създаване на променливи, чиито имена се генерират програмно, което разширява много възможностите на PHP. Ще разгледаме подобен пример, когато разглеждаме получаването в PHP модул на данни от HTML формуляр.

3.2.3. **Управление на променливи**

PHP предоставя набор от функции, свързани с управлението на променливите. Ще разгледаме най-често използваните от тях.

Проверка за съществуване на променлива – функция isset

Функцията isset връща логическа стойност TRUE ако всички променливи в списъка съществуват и имат стойност, различна от NULL. Ако поне една от проверяваните променливи не съществува или има стойност NULL функцията връща FALSE. Тя има следното описание:

bool isset(mixed var [, mixed var [, ...]])

Проверка за “празна” променлива – функция empty

Функцията empty връща логическа стойност TRUE ако проверяваната променлива има стойност false, празен низ, array(), NULL, "0", 0, или не съществува. До версия PHP 5.5.0 функцията приема като аргумент само променлива, а от тази версия аргументът може да бъде и израз. Функцията има следното описание:

bool empty (mixed var)

Унищожаване на променлива – функция unset

Функцията unset унищожава зададена променлива
Описание:

void unset (mixed var [, mixed var [, mixed ...]])

От версия 4 на PHP функцията не връща стойност. При опит да се получи стойност от unset се генерира грешка при компилиране (parse error).

```
unset($p1);              //унищожава променлива
unset($ar1[“a1”]);       //унищожава елемент от масив
```

Пример 9 Унищожаване на променливи с функция unset

Ако глобална променлива се унищожава във функция се унищожава само нейното локално копие, а извън функцията тя продължава да съществува. За да се унищожи глобалната променлива достъпът до нея трябва да се осъществи посредством масива $GLOBALS

```
$x=5; //Създаване и инициализиране на променливата $x
f1();   //Обръщение към функцията
echo $x; //Извеждане на стойността на $x
function f1()
{
        unset($GLOBALS['x']); //унищожаване на променливата $x
}
```

Пример 10 Унищожаване на променлива с глобална видимост

В примера променливата $x се унищожава вътре във функцията, но тъй като достъпът до нея се извършва чрез масива $GLOBALS тя се унищожава и извън функцията. Като резултат при извеждането на стойността на $x чрез инструкцията echo $x; се извежда съобщение за грешка “PHP Notice: Undefined variable: x”.

Проверка за тип на променлива – функции is_type

PHP предоставя набор от функции, чрез които може да се провери дали в дадена променлива е записан определен тип данни. Възможността за такава проверка показва, че действително се съхранява не само стойността, но и типа на данните в променливите. Функциите се представят със следното описание:

bool is_type (mixed var)

където *type* в името is_type е типа на данните, за който се проверява съдържанието на зададената променлива var. Такива са функциите is_bool, is_int, is_float, is_string, is_array, is_object, и т.н. Функциите връщат логическа стойност TRUE ако в променливата е записан съответния тип данни, и стойност FALSE ако е записан друг тип данни. Ако променливата не съществува, се извежда съобщение за грешка, както в предния пример.

3.2.4. **Област на действие (видимост) на променливите**

В PHP променливите са видими в контекста, в който са създадени. Променливите, декларирани в главния скрипт не са непосредствено видими във функциите. Това е още един случай, в който PHP се отличава от повечето езици за програмиране. Все пак е дадена възможност за достъп от кода на функциите до променливи, декларирани в главния скрипт, посредством масива $GLOBALS. Като ключ за достъп се използва името на променливата (без символа $ в началото). Обръщение във

функция към променлива, създадена извън функцията е показано в предходния пример. Ако заменим функцията unset с конструкцията echo , вместо да се унищожи променливата, ще бъде изведена нейната стойност.

Друга възможност за достъп до променливи, декларирани извън кода на дадена функция ни дава ключовата дума global. Ако във функцията една променлива се обяви като глобална чрез ключовата дума global, то чрез нея получаваме достъп до променлива, създадена извън функцията. Да разгледаме един пример

```
$x=5; //Създаване и инициализиране на променливата $x
f1();   //Обръщение към функцията
function f1()
{
      global $x;  //Обявяване на $x за глобална променлива
      echo $x;    //Извеждане на $x
}
```

Пример 11 Обявяване на променлива за глобална с ключовата дума global

3.3. Елементарен изход – конструкции echo и print

PHP предоставя две конструкции за изпращане на символен низ към прозореца на потребителския браузър, които на практика не

3.3.1. Конструкция echo

Описание:

void echo (string arg1 [, string arg2..])

Конструкцията echo извежда стойността на всички аргументи. Стойностите на аргументите се изчисляват (ако са изрази) и се преобразуват в символен низ, ако са от друг тип. Скобите не са задължителни при един аргумент и обезателно трябва да липсват при повече от един аргумент. При наличие на скоби при повече от един аргумент се извежда грешка при компилиране “parse error”.

Конструкцията echo има синтаксис от вида <?=exp ?>, където exp е променлива или израз на PHP. Този блок за извеждане може да се вмъква в HTML код за да се изведе стойността на израза exp, преобразувана в символен низ, както е показано в дадения по-долу пример

```
<ul>
<li>sin(1)=<?= sin(1) ?></li>
<li>cos(1)=<?= cos (1) ?></li>
<ul>
```

Пример 12 Използване на блок за извеждане <?=exp ?> в код на HTML списък

Примерът съдържа код на HTML списък, който извежда стойностите на функциите sin(1) и cos(1) чрез блок за извеждане.

Блоковете за извеждане могат да се използват само тогава, когато на опцията short_open_tag в конфигурационния файл php.ini е зададена стойност true. Ако опцията не е включена, за извеждане на стойността на променливата трябва да се използва процедурен блок <?php echo exp; ?>, с обикновена конструкция echo.

3.3.2. **Конструкция print**

Описание:

int print (string arg)

Конструкцията print извежда стойността на аргумента arg. Стойността на аргумента се изчислява (ако е израз) и се преобразува в символен низ, ако е от друг тип. Скобите не са задължителни.

Конструкции echo и print на практика не се различават при извеждане на стойността на един израз. Все пак трябва да се отбележи, че конструкцията echo може да извежда стойностите на няколко израза, докато print – само на един. Друга съществена разлика е, че print връща стойност 1 и следователно може да участва в изрази, а echo – не. Напр. $x = print “Hi”; записва в $x 1.

3.4. Основни типове данни

В PHP се използват четири примитивни типове данни: boolean, integer, float и string, два съставни типа: array и object, и два специални типа: resource и NULL. Тук ще разгледаме примитивните типове и типът NULL. Съставните типове притежават допълнителни свойства и свързани с тях програмни елементи и се разглеждат самостоятелно.

3.4.1. Тип boolean

Представя логически тип данни, който може да приема стойности TRUE и FALSE.

В много случаи при проверка на условия се прави преобразуване на други типове данни в тип boolean. За програмиста е необходимо да знае в какви случаи данните се преобразуват в boolean стойност TRUE, и в какви случаи – във FALSE. Правилата за преобразуване са както следва

В стойност FALSE се преобразуват:	Всички останали стойности от тези типове данни се преобразуват в логическа стойност TRUE
integer 0 float 0.0 string “” string “0” масив без елементи	

3.4.2. Тип integer

Представя цели числа със знак. PHP (за разлика от C) не поддържа целочислен тип без знак. Броя разряди, с които се представят стойностите, зависи от операционната система, но по правило е 32 бита (4 байта).

Ако при изчисления се получи стойност, която излиза от диапазона на типа integer, тя автоматично се преобразува в тип float.

3.4.3. Тип float

Представя реални числа. Данните от тип float се записват в 64 бита (8 байта). Трябва да се има предвид, че заради физическото представяне на реалните числа в двоична бройна система, някои десетични дроби, напр. 0.1 и 0.7 не могат да се преобразуват в техния двоичен еквивалент без загуба на точност. Например резултатът от изпълнението на функцията floor((0.1+0.7)*10) ще бъде 7 а не 8, тъй като вътрешното представяне на резултата от изчисляването на израза в скобите е 7.999, а функцията връща неговата цяла част, т.е. 7.

3.4.4. **Тип string**

Низовете в PHP са последователности от символи, които винаги скрито се терминират (т.е. ограничават се в края на низа) с null. При операциите с низовете обаче се използва информацията за дължината на низа, която се съхранява в променливите от тип string при тяхното машинно представяне, а не терминиращият символ в края.

В PHP символните константи могат да се записват в три форма́та:

а) Ограничени с двойни кавички
б) Ограничени с апострофи
в) Във формат “here-doc”

Низове, ограничени с двойни кавички

Този формат на символните константи предоставя някои възможности, които другите формати не притежават:

а) Дава възможност за включване на символни последователности:

Има някои символи, които не могат да се въведат в символните константи, ограничени с двойни кавички, непосредствено от клавиатурата или няма да се приемат във вида, в който са въведени. Те могат да се въведат със символни последователности, започващи с “\” , например:

\n – преминаване на нов ред (LF или код $0x0A_{16}$)
\r – връщане в началото на реда (CR или код $0x0D_{16}$)
\t – табулация (HT или код $0x09_{16}$)

\\ – символ наклонена черта
\" – символ двойни кавички
\' – символ апостроф
\x[0-9A-Fa-f]{1,2} – двуцифрен код на символ в шестнадесетична бройна система. Така по принцип може да се въведе всеки символ, но на практика има смисъл да се въвеждат само символи, които ги няма на клавиатурата, и за които няма специални кодови последователности.

б) В низовете, ограничени с двойни кавички, могат да се включват имена на променливи, които, при компилирането на PHP модула се заместват със стойността на променливите, преобразувани в символен низ.
Пример:

```
$user_ime=”Peter”;
echo “Your name is $user_ime”;
```

Пример 13 Включване на променлива в низ, ограничен с двойни кавички

Като резултатът ще бъде изведен низа “Your name is Peter”. Заместването на променливи в редица случаи спестява на програмиста многократни слепвания на променливи със символни константи, например при динамично генериране на SQL команди за операции с бази от данни.

Низове, ограничени с апострофи

Низовете, ограничени с апострофи, не поддържат специалните символни последователности, започващи с “\” и заместването на променливите с техните стойности. В този формат низа се извежда така, както е записан, т.е. ако в него се съдържа например $x ще се изведе $x, а не стойността на променлива с това име. Поддържат се само два специални символа:

\' – апостроф
\\ – обратна наклонена черта “\”

Низове във формат here-doc

Този формат позволява вграждането в скрипт на текст, който може да съдържа произволен брой кавички, апострофи, преминаване на нов ред и други "специални" за низовете символи без необходимост от специално кодиране. Форматът изисква в първия ред да се запише <<< и низ (маркер), който не се среща никъде в текста, а на последния ред да се запише само низа-маркер. Да разгледаме един пример:

```
$t=<<<STRING_HERE
Надпис № 57 на канасюбиги Омуртаг от 822г.
Кансюбиги Омуртаг е от Бога владетел на земята, в която се е родил. Живеейки в лагера в Плиска, той изгради малък аул на Тича. И постави в аула четири колони и върху колоните два лъва. Нека Бог удостои поставения от Бога владетел да тъпче добре с краката си императора, докато тече Тича и докато...като владее над многото българи и като подчинява враговете си да преживее в радост и веселие сто години!
STRING_HERE;
echo $t;
```

Пример 14 Използване на символна константа във формат here-doc

3.5. Операции, оператори и операнди.

Всяка компютърна програма в съответствие с предназначението си извършва зададени операции над стойностите на променливи и константи. Основните елементи на една операция са операторите и операндите.

D3 Операторите в PHP са символи за обозначаване на операции. Константите и променливите*, които участват при изпълнението на операциите се наричат операнди.

* Като променливи величини в операциите могат да се използват и стойности на елементи на масиви, функции, атрибути и методи на обекти.

Приоритетът на операторите определя реда на изпълнение на съответните им операции в изразите. Например в израза 2+3*5 първо ще се изпълни операцията умножение, а след това операцията събиране, защото умножението има по-висок приоритет.

Асоциативността на операторите определя реда на групирането им с операндите и следователно и реда на изчисляване. В зависимост от това операторите могат да бъдат ляво асоциативни, дясно асоциативни или неасоциативни. Нека например е даден изразът a Θ b Θ c. Ако операторът Θ е ляво асоциативен, този израз ще се интерпретира като (a Θ b) Θ c и ще се изчислява отляво надясно. Ако операторът е дясно асоциативен, изразът ще се интерпретира като a Θ (b Θ c) и ще се изчислява отдясно наляво. Ако операторът е неасоциативен, изразът може да съдържа синтактична грешка или да има специално значение.

3.5.1. **Таблица на операторите**

Дадената по-долу таблица съдържа операторите на PHP, като подреждането е в намаляващ ред според техния приоритет. Както се вижда от таблицата, най-висок приоритет има оператора new, а най-нисък – операторът “,”

Асоциативност	**Оператори**
неасоциативен	new
лява	[
неасоциативен	++ --
неасоциативен	~ - (int) (float) (string) (array) (object) @
неасоциативен	instanceof
дясна	!
лява	* / %
лява	+ - .
лява	<< >>
неасоциативен	< <= > >=
неасоциативен	== != === !==
лява	&

лява	^
лява	|
лява	&&
лява	||
лява	? :
дясна	= += -= *= /= .= %= &= |= ^= <<= >>=
лява	and
лява	xor
лява	or
лява	,

Таблица 1 Оператори на PHP

Според броя на операндите, които участват при изпълнението на операцията, операциите (и съответно операторите) на PHP са:

- едноместни (унарни) – с един операнд
- двуместни (бинарни) – с два операнда
- триместни (тернарни) –

Според предназначението си операциите на PHP могат да бъдат класифицирани в следните основните групи:

3.5.2. **Аритметични операции**

Основните аритметични операции са събиране, изваждане, умножение, делене и модул. Те се задават с операторите:

+	за събиране
-	за изваждане
*	за умножение
/	за делене
%	за модул

Аритметичните операции са двуместни (бинарни) операции. Допълнително пояснение изисква само операцията модул (%). Това е операция, с помощта на която можем да определим остатъка от целочислено делене. Тя се изпълнява над целочислени

операнди. Стойността, която се получава при изпълнение на операцията делене по модул е остатъкът от деленето на две цели числа, например:

5 % 3 равно на 2	5 делено на 3 е равно на 1 и **остатък 2**
0 % 3 равно на 0	Резултатът от деленето е 0, което е цяло без остатък – **остатък 0**
-10 % 3 равно на -1	частното е отрицателно число (-3) и за да получим делимото (-10) трябва да добавим **остатък -1**

Пример 15 Аритметична операция модул

PHP предоставя три едноместни (унарни) аритметичните операции:

++	инкрементиране (увеличаване с 1)
--	декрементиране (намаляване с 1)
-	смяна на знака (умножение с -1)

Тези операции може да се считат като по-кратък запис на бинарни аритметични операции, например операцията x++ е еквивалентна на x=x+1. Интересно е да се отбележи, че съвременните микропроцесори притежават машинни команди за изпълнение на унарните операции инкрементиране, декрементиране и смяна на знака.

3.5.3. **Логически операции**

Логическите операции са операции, които се извършват над логически операнди и връщат логически резултат TRUE (логическа истина) или FALSE (логическа неистина). Ако някой от операндите съдържа числова стойност, различна от 0, при изпълнение на логическа операция стойността му се приема за TRUE, а ако съдържа стойност 0, се приема за FALSE. Такова неявно преобразуване на числови стойности в логически се

извършва от всички С-подобни езици за програмиране PHP предоставя следните логически операции:

&&	логическо умножение (бинарна)
||	логическо събиране (бинарна)
!	логическо отрицание (унарна)

Логическите операции могат да се разглеждат като изчисляване на стойности на логически функции. От Булевата алгебра е известно, че логическите функции И, ИЛИ и НЕ образуват функционално пълна база, което означава, че произволно сложна логическа функция може да се представи с логически израз, в който се използват само тези функции. Логическата функция $Z = $X && $Y (логическо умножение) се представя посредством таблицата 3a. Вижда се, че тя дава като резултат TRUE (лог. истина) само ако и двата операнда имат стойност TRUE. . Логическата функция $Z = $X || $Y (лог. събиране) се представя посредством таблицата 3b. Тя дава като резултат TRUE (лог. истина) ако поне един двата операнда има стойност TRUE.

X	Y	Z
false	false	false
false	true	false
true	false	false
true	true	true

3a) логическо умножение (И)

X	Y	Z
false	false	false
false	true	true
true	false	true
true	true	true

3b) логическо събиране (ИЛИ)

X	Z
false	true
true	false

3c) логическо отрицание (НЕ)

Логически операции – таблици на истинност.

Логическата функция $Z=!$X (логическо отрицание), представена посредством таблицата 3c, връща стойност, обратна на стойността на операнда.

3.5.4. **Операции за сравнение**

Операциите за сравнение са операции, които се извършват сравняване на съдържанието на два операнда и връщат логически резултат. По принцип сравнение може да се извършва между стойности от един и същ тип. Операторите за сравнение на PHP дават възможност да се извършва сравнение между две числови стойности или между два символни низа. PHP предоставя следните оператори за сравнение:

Оператор	Значение: връща логическа истина при:
= =	равно на
! =	различно от
>	по-голямо от
<	по-малко от
>=	по-голямо или равно на
<=	по-малко или равно на
===	еквивалентно
!==	нееквивалентно

Например операцията X != 12 ще върне стойност false ако стойността на променливата X е равна на 12 и true във всички други случаи. По-подробно е необходимо да се поясни сравняването на символните низове. Два символни низа могат да бъдат равни (т.е. еквивалентни) само ако имат еднаква дължина и еднакви символи. При сравняването на два символни низа с някой от операторите “>” , “>” , “>=” или “<=” се извършва последователно сравняване на кодовете на символите от двата низа и сравнението завършва тогава, когато се открие, че различие. Резултата от последното сравнение на кодове на символи се приема за резултат от операцията сравнение на низовете. Например ако се сравняват низовете “Hello” и “Harry” различие ще се установи при сравняване на вторите по ред символи в низовете – “e” от “Hello” и “a” от “Harry”. Кода на буквата “e” (латиница) е 101, а кода на “a” е 97, и поради това при сравняването на двата низа ще се получи че първият низ има по-

голяма стойност от втория. Следователно в дадения по-долу пример променливата Z ще получи стойност true.

```
$X = ”Hello”, $Y = ”Harry”, $Z = ($X>$Y);
```

Пример 16 Сравняване на символни низове

3.5.5. **Побитови операции**

Побитовите (bitwise) операции са бинарни операции, които се извършват над съответните битове на един или два целочислени операнда и връщат като резултат целочислена стойност. В PHP стойностите на числовите променливи се записват в четири байта (32 двоични разряда). При побитовите операции стойността на всеки от 32-та двоични разряда на резултата се получава като логическа функция от стойностите на същите по номер битове на двата операнда. PHP предоставя следните побитови операции:

&	побитово умножение AND (бинарна)
|	побитово събиране OR (бинарна)
^	побитово изключващо събиране XOR (бинарна)
~	побитово инвертиране NOT (унарна)

Таблиците на истинност на логическите функции, чрез които се изчисляват битовете на резултата са аналогични на таблиците на истинност за логическите операции:

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

4a) побитово умножение (AND)

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

4b) побитово събиране (OR)

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

4c) изключва-що събиране (XOR)

X	Z
0	1
1	0

4d) побитово отрицание (NOT)

Побитови операции – таблици на истинност.

Примери:

```
$X = 7, $Y = 12 ;
$Z1 = $X & $Y;
$Z2 = $X | $Y;
$Z3 = $X ^ $Y;
$Z4 = ~$X;
```

За да получим нагледна представа за резултата от побитовите операции на двата операнда $X и $Y трябва да ги представим като 32 разрядни двоични числа и да извършим операциите със съответните битове за да получим битовете на резултата. В случая можем да се ограничим до разглеждането на първите осем разряда (младшия байт) тъй като останалите разряди съдържат само нули По-долу е даден пример на побитовото умножение на операндите $X и $Y дава като резултат 32 разрядно число, което съдържа 1 само в разряд b2 и поради това стойността му е 4.

разряд и тегло	b7 128	b6 64	b5 32	b4 16	b3 8	b2 4	b1 2	b0 1
X	0	0	0	0	0	1	1	1
Y	0	0	0	0	1	1	0	0
Z	0	0	0	0	0	1	0	0

Пример 17 Побитово умножение Z1 = X & Y. Резултат Z=4

разряд и тегло	b7 128	b6 64	b5 32	b4 16	b3 8	b2 4	b1 2	b0 1
X	0	0	0	0	0	1	1	1
Y	0	0	0	0	1	1	0	0
Z	0	0	0	0	1	1	1	1

Пример 18 Побитово събиране Z2 = X | Y. Резултат Z=15

разряд и	b7	b6	b5	b4	b3	b2	b1	b0
тегло	128	64	32	16	8	4	2	1
X	0	0	0	0	0	1	1	1
Y	0	0	0	0	1	1	0	0
Z	0	0	0	0	1	0	1	1

Пример 19 Побитово изключващо събиране $Z3 = $X ^ $Y.
Резултат $Z3=11

разряд и	b7	b6	b5	b4	b3	b2	b1	b0
тегло	128	64	32	16	8	4	2	1
X	0	0	0	0	0	1	1	1
Z	1	1	1	1	1	0	0	0

Пример 20 Побитово инвертиране $Z = ~$X.
Резултат $Z4=($2^{32}$-1)-7

Резултатът от побитовото инвертиране на $X е 32-разрядно двоично число, което има единици във всички разряди с изключение на разрядите b0, b1 и b2, и поради това стойността му (като цяло положително число) е равна на сумата от теглата на всички разряди (2^{32}-1) намалена със сумата от теглата на разрядите b0, b1 и b2*.

3.5.6. **Измествания**

Измествания са особен вид побитови операции. В тях участват два целочислени операнда, като при изпълнение на операцията съдържанието на левия операнд (operand1) се измества наляво или надясно на брой разряди (битове), равен на стойността на десния операнд (operand2). Съдържанието на operand1 участва в съответната операция като цяло число, представено с 32 двоични разряда (4 байта), а operand2 – като цяло положително число. PHP предоставя два оператора за изместване:

* Детайлното разглеждане на побитовите операции излиза извън рамките на настоящия курс.

<< изместване наляво
>> изместване надясно

Измествания в PHP имат някои ограничения, които е важно да се имат предвид. Не трябва да се прави изместване надясно на операнди с размер по-голям от 32 бита в 32 битови микропроцесорни системи. Също така не трябва да се прави изместване наляво ако резултатът е число, изискващо повече от 32 бита машинно представяне.

Интересно е да се отбележи, че аритметичните устройства на микропроцесорите включват схеми за извършване на операциите измествания и поради това притежават машинни команди за извършване на тези операции. Измествания са били въведени като команди на езика за програмиране С защото създателите му са искали програмите, написани на С да се доближават по бързодействие до програмите, написани на машинния език на процесора.

Изместване наляво <<

При операцията изместване наляво `operand1 << operand2` се получава числова стойност, равна на съдържанието на operand1, изместено наляво на брой разряди, равен на operand2. При това изместване най-старшите битове от съдържанието на operand1 се «избутват» извън 32-та бита на резултата, а в най-младшите битове се «вмъкват» нули. Резултатът от тази операция е еквивалентен на умножение на operand1 по 2 на степен operand2. Резултатът обаче ще бъде такъв само ако в «избутаните» старши битове на operand1 са се съдържали само нули.
Пример: нека operand1=3. Тогава съдържанието му като 32-разрядно двоично число ще има вида:

operand1=00000000 00000000 00000000 00000011

При изпълнение на операцията `operand1 << 2` резултатът, представен като 32 разрядно двоично число ще има вида:

00000000 00000000 00000000 00001100

, което, представено като десетично число е $12=3*2^2$. – съдържанието на operand1, умножено по 2^2.

Изместване надясно >>

При операцията аритметично изместване надясно operand1 >> operand2 се получава числова стойност, равна на съдържанието на operand1, изместено надясно на брой разряди, равен на operand2. При това изместване най-младшите битове от съдържанието на operand1 се «избутват» извън 32-та бита на резултата, а в най-старшите битове се «вмъкват» нули, но съдържанието на най-старшия бит b_{31} се копира (запазва стойността си). Резултатът от тази операция е еквивалентен на делене на operand1 по 2 на степен operand2 със запазване на знака на operand1. По-точно, получава се цялата част от деленето, защото «избутаните» младши разряди на operand1 се губят.
Пример: нека operand1=-12. Тогава съдържанието му като 32-разрядно двоично число ще има вида*:

operand1=11111111 11111111 11111111 11110100

При изпълнение на операцията operand1 >> 2 резултатът, представен като 32 разрядно двоично число ще има вида:

10111111 11111111 11111111 11111101 ,

което, представено като десетично число е $-3 = -12/2^2$. – съдържанието на operand1, разделено на 2^2.

В някои случаи обаче при изместване на отрицателни числа може да не се получи точно остатъка от целочислено делене с 2^n.

---

* Числото е представено в т.н. допълнителен код. Детайлното разглеждане на машинното представяне на числата излиза извън рамките на настоящия курс.

Например операцията “-11>>1” би трябвало да даде резултат “-5”, но в действителност дава резултат “-6”. Така, че за препоръчване е операцията “изместване надясно” да се използва само за положителни числа.

3.5.7. **Операции със символни низове**

Освен операциите за сравнение на символни низове, разгледани в т 3.5.4 PHP предоставя още два оператор за работа със символни низове – операторът конкатенация “.” и операторът за инкрементиране ++.

Конкатенацията е бинарна операция, която “слепва” съдържанието на два символни низа и връща като резултат получения символен низ.

```
$X=”Hello “; $Y=”World”;
$Z=$X . $Y;
```

Пример 21 Слепване на символни низове

Резултатът от операцията конкатенация $X . $Y в дадения пример е символния низ “Hello World”, получен от слепването на двата символни низа, съдържащи се в операндите $X и $Y.

Инкрементиране на символни низове

В PHP низовете се инкрементират съгласно конвенцията на езика Perl. Ако последният символ е буква или цифра, той се инкрементира до следващата буква или цифра.
Пример:
$a=”ab”; ++$a; има за резултат “ac”
$b=”12”; ++$b; има за резултат “13”

Ако последната буква е 'Z' или 'z' или ако последната цифра е '9', символът се инкрементира както следва:

'Z' – на 'A'
'z' – на 'a'
9 – на 0
, след което предходния символ се инкрементира по същите правила. Да разгледаме отново пример с низове с букви и цифри:
$a=”az”; ++$a; има за резултат “ba”
$b=”19”; ++$b; има за резултат “20”

В тая връзка може да се постави въпроса: а какво е отношението на създателите на PHP към делото на славянските първоучители Кирил и Методий, и по-точно: може ли по същите правила да инкрементираме низове, съдържащи символи на кирилица? Отговорът е отрицателен. Символите остават непроменени, тъй като кодовете им не попадат в интервала от кодове на символите от латинската азбука.

Възможностите за операции със символни низове в PHP обаче съвсем не се изчерпват с операциите за сравнение и конкатенация. Както ще бъде разгледано по-нататък, PHP предоставя на програмиста богат набор от функции, чрез които се извършват операции със символни низове.

3.5.8. **Условен оператор “?”**

Условният оператор “?” е единственият оператор на PHP, който извършва операция над три операнда. Той има следния синтаксис:

operand1 ? operand2 : operand3

където:
operand1 е логически операнд (променлива или константа) или израз, който връща логически резултат
operand2, operand3– операнди или изрази, които връщат стойност.

Стойността, която се получава при изпълнение на операцията зависи от стойността на operand1:

ако operand1 има стойност true	операцията връща стойността на operand2
ако operand1 има стойност false	операцията връща стойността на

operand3

Условният оператор “?” не е просто съкратен запис на конструкцията if-else, защото, за разлика от конструкциите, разгледани по-нататък, операторите връщат стойност. Например като използваме условния оператор “?” можем да присвоим стойността, която се получава от него на някаква променлива или да я използване като фактически параметър при извикване на функция*.

```
$s=$x ? “x има стойност true” : “x има стойност false”;
```

Пример 22 Използване на условния оператор “?”

В примера в ако променливата $x има стойност логическа истина (true) , в променливата $s се записва низа “x има стойност true”, в противен случай в $s се записва низа “x има стойност false”.

3.5.9. **Оператор за присвояване**

Операторът за присвояване “=” служи за присвояване на стойности на дадена променлива. Операцията присвояване е бинарна и има следния синтаксис:

operand1 = operand2

където operand1 може да бъде само име на променлива, а operand2– константа, променлива, функция или израз, който връща стойност. Стойността на operand2 се присвоява на operand1. Например при изпълнение на операцията `$y=$x+2` интерпретаторът на PHP ще изчисли аритметичната сума от съдържанието на променливата $x и константата 2 и ще запише (или, както е прието да се казва, ще присвои) резултата на променливата $y.

Операцията присвояване връща стойността, която се присвоява на operand1. Поради това например изразът* `1+($x=5)` не съдържа синтактична грешка. Първа ще се изпълни операцията в скобите,

* Функциите се разглеждат в т. 4.5

която ще върне стойност 5, след което ще се извърши операцията събиране и ще се получи резултат 6. Записът `1+$x=5` обаче съдържа синтактична грешка, тъй като може да се присвоява стойност само на променлива, а в случая се прави опит да се присвои стойност на израза 1+x.

3.5.10. **Съкратен запис на операции**

Практиката в програмирането е показала [6], че най-често използваните операции са операциите за присвояване и аритметичните операции. Някои комбинации от тези операции са толкова често срещани, че в езика C и C-подобните езици (по синтаксис) като PHP са предвидени специални форми, които опростяват използването им. Това са конструкции от вида:

```
променлива1 = променлива1 оператор втори_операнд
```

За подобни конструкции е предвиден съкратен запис от вида:

```
променлива1 оператор = втори_операнд
```

В PHP се използват следните съкратени записи на операции.

Съкратена операция	**Значение**
$x += $y	$x = $x + $y
$x -= $y	$x = $x - $y
$x *= $y	$x = $x * $y
$x /= $y	$x = $x / $y
$x %= $y	$x = $x % $y
$x <<= $y	$x = $x << $y
$x >>= $y	$x = $x >> $y
$x &= $y	$x = $x & $y

---

* Изразите като програмни конструкции се разглеждат в следващата глава

Съкратена операция	Значение
$x ^= $y	$x = $x ^ $y
$x |= $y	$x = $x | $y

Таблица 2 Съкратен запис на операции в PHP

3.5.11. **Оператори за преобразуване на типове**

Явното преобразуване на типове в PHP има същия синтаксис, както в езика C: името на желания тип се поставя в скоби преди променливата (или израза), чийто тип трябва да бъде преобразуван Могат да се използват следните унарни оператори:

- (int), (integer) - преобразува в тип integer (целочислен)
- (float), (double), (real) - преобразува в тип float (реален, съдържащ стойност във формат с плаваща запетая)
- (bool), (boolean) - преобразува в тип boolean (логически)
- (string) - преобразува в тип string (символен низ)
- (binary) - преобразува в тип binary (двоичен низ) (PHP 6)
- (array) - преобразува в тип array (масив)
- (object) - преобразува в тип object (обект)

Да разгледаме един пример:

```
$x=5;
$y=(bool)$x;
echo “y=$y”;
```

Пример 23 Използване на оператор за преобразуване на тип

В примера стойността на променливата $x от тип int се преобразува в тип bolean. Тъй като стойността на $x е различна от 0, $y получава стойност TRUE, а конструкцията echo ще изведе низа y=1.

Автоматично преобразуване на типове

При извършване на операции PHP автоматично преобразува операндите в съответствие с типа на операцията. Например ако

оператора е “.” (слепване) операндите автоматично се преобразуват в символни низове.

```
$x=2.3;
$y=5.4;
echo $x.$y;
```

Пример 24 Автоматично преобразуване на типа на операндите при операция слепване

В примера променливите $x и $y съдържат стойности от тип float. При операция слепване обаче те се преобразуват в символни низове и затова конструкцията echo ще изведе низа “2.35.4”. Ако се опитаме обаче да слепим непосредствено константите от тип float със следната инструкция `echo 2.3.5.4;` ще се генерира грешка при компилирането, тъй като PHP не може да определи кой от символите “.” трябва да се счита за десетичен разделител, и кой – за операнд.

Ако не сме сигурни какъв резултат ще получим при автоматичното преобразуване на типове, е по-добре да използваме явно преобразуване с дадените по-горе оператори.

3.5.12. **Оператор за приглушаване – скриване на съобщение за грешка**

Операторът за приглушаване @ се поставя преди изрази и забранява извеждането на съобщение за грешка при изчисляване на израза. Операторът @ може да се включи и преди променливи, извикване на функции и директиви include, тъй като те са частен случай на изрази и връщат стойност. Да разгледаме един пример:

```
$fspec='my_file.txt';
$my_file = @file($fspec) or
   die ("Файлът $fspec не може да бъде прочетен");
```

Пример 25 Използване на оператора за приглушаване

В примера с функцията file се прави опит за прочитане на съдържанието на текстов файл. Ако при това се генерира грешка се извежда съобщение “Файлът my_file.txt не може да бъде прочетен”

и се прекратява изпълнението на модула чрез функцията die. Ако преди функцията нямаше оператор за приглушаване щеше да се изведе по-дълго и неясно съобщение, което се генерира от самата функция file.

Операторът @ не може да се включва преди програмни конструкции, като декларации на функции, класове, конструкции if-else и цикли.

Операторът @ може да потиска съобщенията дори за грешки, които предизвикват прекратяване на изпълнението на PHP модула. В такъв случай изпълнението на PHP модула ще бъде прекратено без извеждането на каквото и да е съобщение за оператора, което в повечето случай не е добра практика.

3.6. Изрази

Изразите са едни от основните елементи на езиците за програмиране. Изразът е формула за изчисляване на стойност, която включва константи и променливи, свързани със символи за операции и кръгли скоби.

При изчисляване на изразите компилаторът отчита приоритета на операциите. (вижте т.3.5.1. Таблица на операторите). По правило операциите са ляво асоциативни, което означава, че стойностите на изразите, в които те участват, се изчисляват отляво надясно. Дясно асоциативен е само операторът за присвояване “=” и съкратените записи “+=”, “-=” и т.н. Неасоциативни са например операторите за преобразуване на тип (int), (float) и т.н. , операторът new и операторите за сравнение.

4. Конструкции за управление

Всяка една програма, в съответствие с алгоритъма си, съдържа последователност от инструкции и конструкции за управление. Конструкциите за управление реализират условните разклонения в алгоритъма на програмата. От това следва, че всички разклонени програми съдържат конструкции за управление. PHP предоставя на

програмиста две конструкции, които осъществяват условно разклонение - if-else и switch и четири конструкции за организиране на цикли – for, while, do-while и foreach.

4.1. Условна конструкция if-else

Тази конструкция се използва тогава, когато трябва да се изпълни една инструкция или блок от инструкции само в случай, че се изпълнява зададено условие. Клаузата else дава възможност да се изпълни друга инструкция или блок от инструкции в случай, че не се изпълнява зададеното условие. Конструкцията има следния синтаксис:

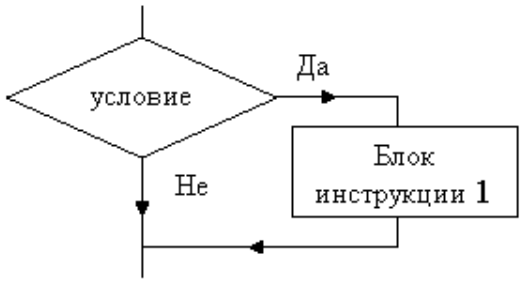
```
if (условие)
{
      Блок инструкции 1
}
```

Фиг. 1 Конструкция if

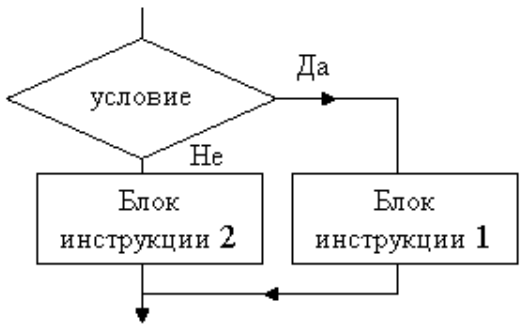
Ако се добави клауза else конструкцията добива вида:

```
if (условие)
{
      Блок инструкции 1
}
else
{
      Блок инструкции 2
}
```

Фиг. 2 Конструкция if - else



Фиг. 2 Блокова схема на конструкция if



Фиг. 3 Блокова схема на конструкция if-else

където:

- условие е израз, който връща логически стойност (true или false) или числова стойност, която се преобразува в логическа (стойност 0 се преобразува във false, а стойност, различна от 0 – във true).
- блок инструкции 1 – Блок от инструкции, който се изпълнява, когато изразът условие има стойност логическа истина (true).
- блок инструкции 2 – Блок от инструкции, който се изпълнява, когато изразът условие има стойност логическа неистина (false).

Ако трябва да се изпълни само една инструкция, тя може да не се включва във фигурни скоби, например конструкцията

```
if($x>5){ $y=1; }
else { $y=2; }
```

Пример 26 Конструкция if-else**.**

може да се замени с

```
if($x>5)  y=1;
else  $y=2;
```

Важно е да се запомни, че клаузата else трябва да следва непосредствено след блока инструкции след if, в противен случай клаузата ще представлява синтактична грешка в скрипта. Така например конструкцията

```
if($x>5)  $y=1; $x=5;
else  $y=2;
```

съдържа синтактична грешка, защото клаузата else не следва непосредствено след инструкцията `$y=1;` а между тях е включена още една инструкция - `$x=5;`.

Пример: Да се състави скрипт, който да изчислява стойността на функцията

$$y(x) = \begin{cases} x+5 \ \ за \ \ x<5 \\ x-1 \ за \ x>=5 \end{cases}$$

Както се вижда от зададената дефиниция, y(x) е по части непрекъсната функция. За да се определи по каква формула трябва да се изчислява нейната стойност, е необходимо да се направи проверка дали стойността на x е по-малка от 5 или е по-голяма или равна на 5. Тези условия са взаимно-изключващи се, следователно е достатъчна само една проверка, която в PHP ще се извърши посредством оператор за сравнение. За изчисляването на стойността на y може да се използва следната условна конструкция if-else:

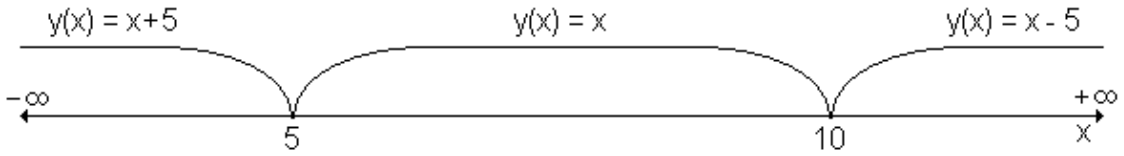
```
if($x<5) $y = $x + 5;
else    $y = $x – 1;
```

Блоковете след if и else могат да съдържат всички допустими за PHP инструкции, включително и други конструкции if-else. Посредством вложени конструкции if-else може да се реализира произволно сложна разклонена програма. Да разгледаме още един

пример за изчисляване на стойност на функция, при който да е необходимо да се проверяват две условия:
Да се състави скрипт, който да изчислява стойността на функцията

$$y(x) = \begin{cases} x + 5 \text{ за } x < 5 \\ x \text{ за } 5 \le x < 10 \\ x - 5 \text{ за } x \ge 10 \end{cases}$$

От зададеното условие се вижда, че за да се определи по каква формула трябва да се изчислява стойността на y(x) е необходимо да се провери в кой от зададените три интервала на числовата ос попада стойността на променливата x.



Фиг. 3 Разположение на интервалите от пример 27 върху числовата ос.

Тъй като трите интервала I1 (-∞ , 5) , I2 [5 , 10) и I3 [10 , +∞) покриват цялата числова ос -∞ , +∞ , т.е. всички възможни стойности на x, и не се припокриват взаимно, достатъчни са две проверки за да се установи в кой интервал попада стойността на x и така да се определи по коя формула следва да се изчислява стойността на y(x). Скриптът, който извършва това изчисление може да се реализира посредством две вложени структури if – else.

```
if($x<5) {
        $y=$x+5;
}else {//Вложена конструкция if-else
        if($x<10) $y=$x;
        else $y=$x-5;
}
```

Пример 27 Пример за вложена конструкция if-else

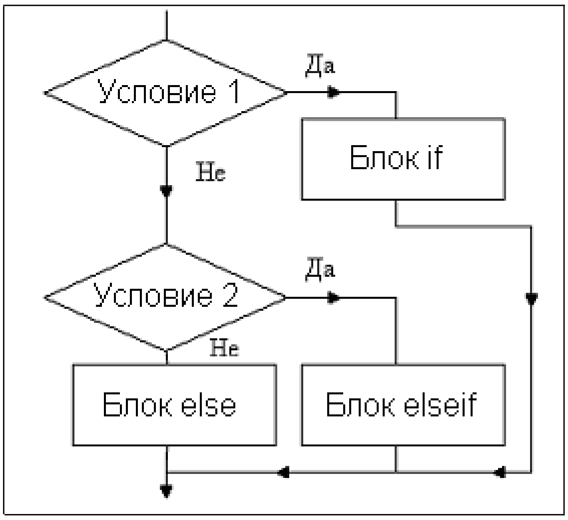
4.2. Клаузата elseif

Клаузата elseif, както се вижда от наименованието ù, е комбинация от if и else. Чрез нея в конструкцията if може да се включи проверка на допълнително условие и изпълнение на блок

инструкции в случай, че при изчисляване на условието на if е получена стойност FALSE. За разлика от клаузата else обаче, блокът на elseif се изпълнява само ако при изчисляване на условието на elseif се получи стойност TRUE.

```
if (условие)
{
        Блок инструкции if
}
elseif(условие2)
{
        Блок инструкции elseif
}
else { Блок инструкции else
}
```

Синтаксис на конструкция if-else с клауза elseif



Фиг. 4 Блокова схема на конструкция if-elseif-else

Освен клаузи elseif, конструкцията if може да съдържа в края и клауза else, както е показано в следващия пример.

С използването на клаузата elseif можем да извършим изчислението от предната задача със следния код:

```
if($x<5)
{
        $y=$x+5;
}
elseif($x<10)
{
        $y=$x;
{
else{
        $y=$x-5;
}
```

или, ако си спестим блоковете (което е възможно, тъй като в тях има само по една инструкция) получаваме:

```
if($x<5) $y=$x+5;
elseif($x<10) $y=$x;
else $y=$x-5;
```

Пример 28 Пример за използване на if с клауза elseif

PHP предоставя и алтернативен синтаксис за конструкцията if-elseif-else, в който вместо с фигурна скоба блоковете започват със символа двоеточие, а цялата конструкция завършва с endif; . Тъй като този синтаксис няма никакви предимства с дадения по-горе (използван във всички “С-подобни” програмни езици) няма да бъде разглеждан по-подробно.

4.3. Конструкция switch

Конструкцията switch дава възможност за многократно разклонение чрез сравнение на стойността на зададен израз с набор от константи. Тя има следния синтаксис:

```
switch(expression)
{
    case label 1:
        инструкции 1
        break;
    case label 2:
        инструкции 2
        break;
    ……….
    [ default:
      блок инструкции
      default ]
}
```

При изпълнение на конструкцията първо се изчислява стойността на израза *expression* (в частност вместо израз може да бъде зададена само една променлива), след което стойността се сравнява последователно с константите label1, label2 … и ако се открие съвпадение на стойността на израза с някой етикет (константа) се изпълняват инструкциите, които следват след етикета.

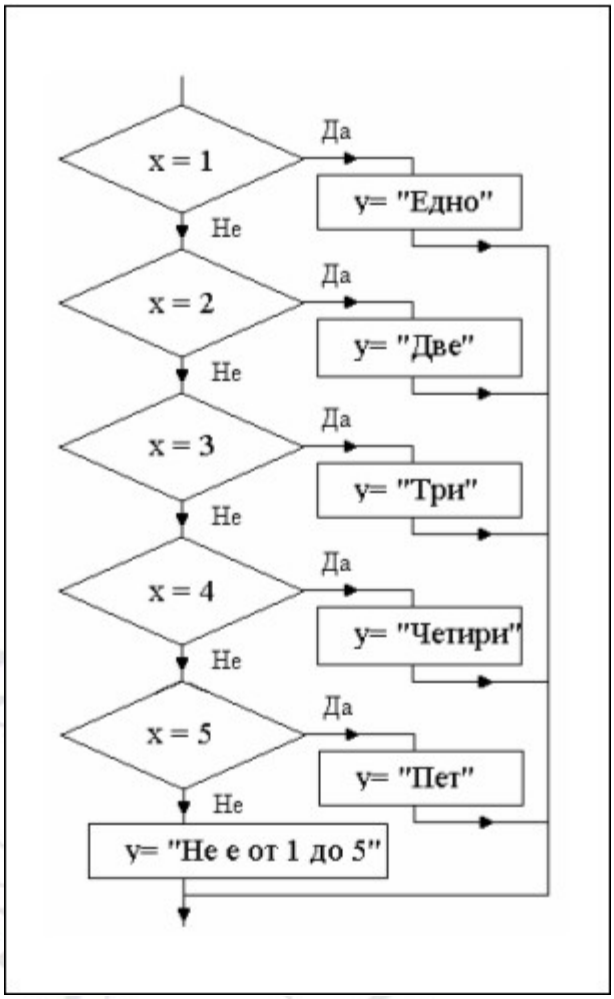
Ако не се открие съвпадение с никой от етикетите, се изпълняват инструкциите, които следват след незадължителната клауза default (ако е включена).Да разгледаме един пример:

Да се състави скрипт, който да сравнява съдържанието на променлива $x с числата от 1 до 5 и при съвпадение да присвоява на променлива y низ, съответстващ на стойността на променливата x. Ако стойността на $x е извън интервала [1 ÷ 5] на $y трябва да се присвои низ “Не е от 1 до 5”.

Реализацията на скрипта посредством структура switch има вида:

```
switch($x)
{
        case 1:
                $y=”Едно”;
                break;
        case 2:
                $y=”Две”;
                break;
        case 3:
                $y=”Три”;
                break;
        case 4:
                $y=”Четири”;
                break;
        case 5:
                $y=”Пет”;
                break;
        default:
                $y=” Не е от 1 до 5”;
}
```

Пример 29 Използване на конструкция switch



Фиг. 5 Сравняване на съдържанието на променлива с набор от константи

В примера конструкцията switch извършва последователно сравнение на стойността на променливата $x с константите от 1 до 5. При откриване на съвпадение се изпълняват инструкциите който следват след съответната клауза case, като с оператора break се излиза от блока на конструкцията switch. Ако не се открие съвпадение на стойността на $x с константите от 1 до 5 се изпълнява инструкцията след клаузата default.

В PHP, за разлика от C, конструкцията switch може да сравнява не само числови константи, но и символни низове.

По правило всяка последователност от инструкции, която следва след някоя клауза case, завършва с break, защото чрез този безусловен преход се излиза от конструкцията switch и изпълнението на скрипта продължава със следващата след switch инструкция. Включването на break след всеки case обаче съвсем не е задължително. Ако break липсва, PHP ще продължи с изпълнението на следващите инструкции до достигане на края на блока на switch или до достигане на безусловен преход break. Да разгледаме един такъв пример:

```
switch($x)
{
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
                $y=” x е от 1 до 5”;
                break;
        default:
                $y=” Не е от 1 до 5”;
}
```

Пример 30 Използване на структура switch с клаузи case с общ безусловен преход break.

В този пример стойността на $x се сравнява последователно с константите 1, 2, 3, 4 и 5. Ако се получи съвпадение на $x с една от тези стойности, се изпълнява инструкцията

y=” x е от 1 до 5”;

5. Конструкции за организиране на цикли

В програмирането цикъл се нарича многократно (повтарящо се) изпълнение на един и същ блок инструкции, като при всяко ново изпълнение се обработват нови данни.

Циклични алгоритми се използват в много случаи, например при операции с масиви от данни, при итерационни изчисления (при които резултатите от изпълнението на блока инструкции се използват като входни данни при следващото му изпълнение) и др. Поради това всички програмни езици предоставят конструкции за организиране на цикли.

PHP предоставя за организиране на цикли конструкциите for, while do-while и foreach. Конструкцията foreach извършва итерация на масиви и атрибути на обекти и поради това ще бъде разгледана в последствие. Тук ще разгледаме първите три конструкции (for, while do-while), който по синтаксис и алгоритъм за изпълнение са същите, както в C и “C-подобните” програмни езици.

5.1. Конструкция за организиране на цикли for

Конструкцията for е най-често използваната конструкция за организиране на цикли. Тя, за разлика от конструкциите while и do-while поема по-голяма част от организацията на цикъла, и поради това дава по-компактен сорс код на цикъла.

Синтаксис:

```
for(инициализация ; условие ; инкрементиране) инструкция
```

или

```
for(инициализация ; условие ; отброяване)
{
        блок инструкции
}
```

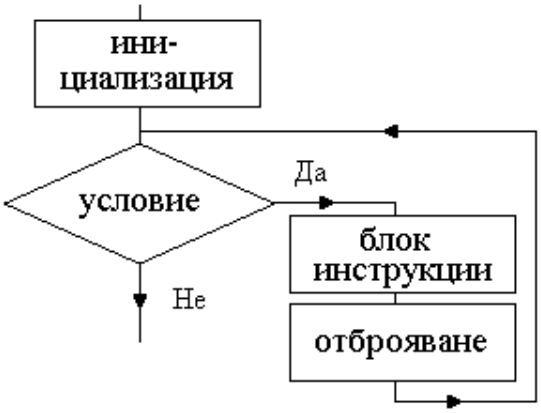
инициализация е инструкция или списък от инструкции, които се изпълняват еднократно като начално установяване на цикъла*.
условие е израз, който връща логическа стойност или числова стойност, която се преобразува в логическа. Много често изразът съдържа една или няколко операции за сравнение и логически операции.
отброяване по правило е инструкция, която променя стойността на индекса (брояча) на цикъла.

Блокът инструкции се повтаря до тогава, докато изразът, зададен като **условие** връща резултат логическа истина. Условието се проверява преди изпълнението на блока инструкции, следователно конструкцията for реализира цикъл с префиксна проверка. Ако в цикъла трябва да се изпълнява само една инструкция не е необходимо тя да се включва в блок {…}. Блоковата схема, която представя алгоритъма, съответстващ на конструкцията for е дадена на фиг. 9:
Да разгледаме един пример за реализиране на цикъл посредством конструкция for.

Да се състави програма, която да изчислява сумата от първите 100 натурални (т.е. цели положителни) числа.

Съгласно условието, трябва да се изчисли сумата S = 1 + 2 + 3 ...+ 100. Ако се използва математическия символ за сума "Σ" стойността на S, която трябва да се изчисли, може да се представи във вида: $S = \sum_{n=1}^{100} n$

* Вижте общата блокова схема на цикъл, дадена в т. 4.



Фиг. 6 Блокова схема на конструкция for

За да се реализира многократното сумиране като цикъл, съгласно даденото по-горе определение, то трябва да се представи като многократно изпълнение на един и същ блок инструкции. Използва се натрупвано събиране, като към променливата S при всяко изпълнение на блока на цикъла се добавя поредното цяло положително число.

```
$S=0; $n=1;  //инициализация
        $S = $S + $n;  $n = $n + 1;
        $S = $S + $n;  $n = $n + 1;   } N двойки
            .....                     } инструкции
        $S = $S + $n;  $n = $n + 1;
```

След задаването на начални стойности на променливите $S и $n по този алгоритъм се повтаря 100 пъти двойката инструкции `$S=$S+$n;  $n=$n+1;` След 100 изпълнения на двойката инструкции, в променливата $S се получава крайния резултат – сумата на числата от 1 до 100. Програмата, реализирана с конструкция for има вида:

```
$S=0; //Инициализация
for($n=1;$n<=100;$n++)
{
      $S=$S+$n;
}
echo “S=$S”;
```

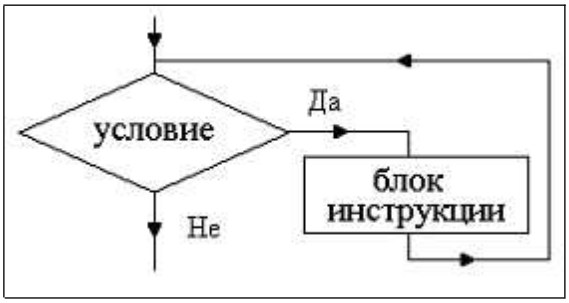
Пример 31 Изчисляване на сумата на първите 100 натурални числа с конструкция for

5.2. Конструкция while

Конструкцията while, за разлика от конструкция for включва само проверка на условието за продължаване на цикъла. Тя има следния синтаксис:

```
while(условие)
{
        блок инструкции
}
```

като (както при for) при една инструкция в тялото на цикъла не е необходимо тя да се включва в блок. Както се вижда от блоковата



Фиг. 7 Блокова схема на конструкция while

схема, проверката на условието за продължаване на цикъла е (както при for) преди изпълнението на блока инструкции (префиксна проверка). Изчисляването на сумата S=1+2+ … 100, реализирано с конструкция while има вида:

```
$S=0; $n=1; //Инициализация
while($n<=100)
{
        $S=$S+$n;
        $n++;        //Отброяване
}
echo “S=$S”;
```

Пример 32 Изчисляване на сумата на първите 100 натурални числа с конструкция while

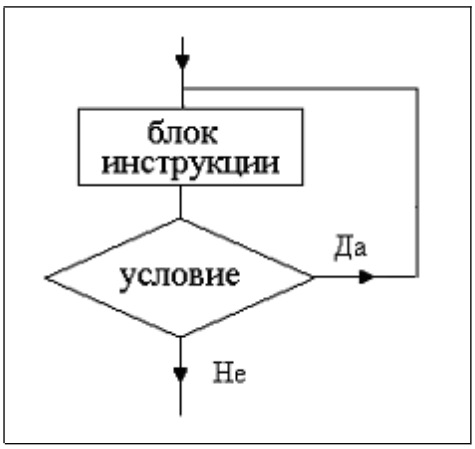
5.3. Конструкция do-while

Конструкцията do-while, както конструкция while включва само проверка на условието за продължаване на цикъла. Тя има следния синтаксис:

```
do
{
```

```
        блок инструкции
}
while(условие);
```

като (както при for и while) при една инструкция в тялото на цикъла не е необходимо тя да се включва в блок. Както се вижда от блоковата схема, разликата между do-while от една страна, и while и for от друга е, че при do-while проверката на условието за продължаване на цикъла е след изпълнението на блока инструкции (постфиксна проверка). Предния пример, реализи-ран с конструкция do-while има вида:



Фиг. 8 Блокова схема на конструкция do-while

```
$S=0; $n=1; //Инициализация
do
{
        $S=$S+$n;
        $n++;        //Отброяване
} while($n<=100);
echo “S=$S”;
```

Пример 33 Изчисляване на сумата на първите 100 натурални числа с конструкция do-while

При сравнение на синтаксисите на конструкциите може да се забележи и друга разлика: единствено конструкцията do-while завършва със символа за край на инструкция “;”.

5.4. Безусловен преход break

Безусловният преход break дава възможност за прекратяване на изпълнението на цикъл, структура switch или блок инструкции
Синтаксис:

```
        break  [N]
```

където label е незадължителен параметър, който показва изпълнението на колко на брой вложени конструкции ще се прекрати с break.

Да разгледаме един пример:

```
for($m=0;$m<10;$m++)
{
	for($n=0;$n<10;$n++)
	{
		if($n==3)break; //Излизане от конструкциите for
	}
}
echo "m=$m  n=$n";
```

Пример 34 Използване на оператор break за излизане от цикли for

В примера конструкциите for организират цикли, който трябва да се изпълнят по 10 пъти (за стойности на $m и $n от 0 до 9), но когато n достигне стойност $n=3 се изпълнява break и се прекратява изпълнението едновременно и на двата цикъла. Така, че на практика вътрешният цикъл се изпълнява 4 пъти (за стойности на $n от 0 до 3), а външният – само веднъж и се извежда низа “m=0 n=3”. Ако break беше без параметъра 2, тогава щеше да се прекратява изпълнението само на вътрешния цикъл, а външния щеше да се изпълни 10 пъти, както е зададено с условието. Тогава щеше да се изведе низа “m=10 n=3”.

6. Масиви

Масивите в PHP са от асоциативен тип, което означава, че се съхраняват двойки ключ-стойност (а не просто индексирани стойности, както е в повечето езици за програмиране). Масив в PHP може да бъде създаден чрез функцията array със следния синтаксис:

```
array([ключ=>]стойност, [ключ=>]стойност, .....)
```

Ключовете са незадължителни (това означават квадратните скоби в сиснтаксиса). Когато липсват, автоматично се присвоява на

ключа целочислена стойност, с едно по-голяма от последния използван целочислен ключ.

Ключът на елемент от масив може да бъде от тип integer или от тип string. Стойността на елемент от масив може да бъде от всеки тип, включително и масив, т.е може да имаме масив, някои от елементите на който са масиви. В дадения по-долу пример се създават масива с функция array, като първия елемент на масива е вложен масив.

$a1=array(“p1”=>array(3,7), “p2”=>”Hi”);

PHP предоставя набор от функции за операции с масиви. Тук ще бъдат разгледани някои от тях, класифицирани според предназначението си.

6.1. Запълване на последователни елементи от масив със зададена стойност – функция array_fill

Функцията array_fill дава възможност да създадем масив, в който елементите имат последователни целочислени индекси и една и съща зададена стойност. Функцията има следния синтаксис:

array array_fill (int start_index, int num, mixed value)

където:

start_index е индекса (целочислен ключ) на първия елемент от масива

num е броя елементи на създавания масив

value е стойността, която получават всички елементи от масива

В даденият по-долу пример се създава масив, който съдържа 100 елемента с начален индекс 2, като всички елементи съдържат низа „Hello”.

```
$a=array_fill(2,100,”Hello”);
print($a);
```

Пример 35 Създаване на масив, запълнен със зададена стойност чрез функция **array_fill**

6.2. Операции с ключовете на масив

PHP предоставя две функции за операции с ключовете на масиви – функция за проверка за съществуване на елемент по зададен ключ и функция за извличане на всички ключове

6.2.1. Проверка за съществуване на елемент по зададен ключ - функция **array_key_exists()**

Описание:
bool array_key_exists(mixed key, array search)
където:
key е ключа, за който се прави проверка за съществуване
search е масива, в който се търси елемент със зададения ключ

Функцията връща логическа стойност TRUE ако елемент с ключ *key* се съдържа в масива search, и FALSE – ако не съществува.

6.2.2. Извличане на ключовете от масив – функция array_keys

Описание:
array array_keys (array input [, mixed search_value [, bool strict]])
където:
input е масивът, от който се извличат ключовете
search_value (незадължителен) – ако е зададен, се извличат ключовете само на елементите, които имат тази стойност.
strict (незадължителен) – определя оператора за сравнение на стойностите на елементите със search_value. При стойност TRUE сравнението е за еквивалентност (оператор “===”), а при FALSE (подразбиращо се) сравнението е за равенство (оператор “==”)

Функцията връща масив с целочислени индекси (начален индекс 0), който съдържа извлечените ключове. Да разгледаме един пример:

```
$array = array("Peter", "John", "Peter", "Peter", "Harry");
print_r(array_keys($array));
```

```
echo "<HR>";
print_r(array_keys($array, "Peter"));
```

Пример 36 Извличане на ключове на елементи от масив

В примера първото обръщение към функцията array_keys извлича всички ключове от масива в масив с последователни целочислени индекси. Съдържанието на извлечения масив се извежда от функцията print_r.

При второто обръщение към функцията array_keys извлича само ключовете на елементите, които съдържат низа "Peter"

6.3. Извличане на стойностите на елементите на масив– функция array_values

Описание:

array array_values(array input)

където input е зададеният масив, от който се извличат стойностите на елементите. Функцията (както и array_keys) връща масив с целочислени индекси (начален индекс 0), който съдържа извлечените стойности.

Функциите array_keys и array_values могат да се използват например когато искаме на извършим итерация (т.е. последователна обработка*) на елементите от масив, както е дадено в следващия пример:

```
$array = array(“Author”=>"Peter", “Publisher”=>"John");
$names= array_values($array); //Извличане на стойностите
$t="<OL>";
for($n=0;$n<2;$n++){
      $t.="<LI>$ names[$n]</LI>"; //Добавяне на ред към списъка
}
$t.="</OL>";
echo $t; //Извеждане на списъка
```

* Терминът “итерация” има и друго значение – итерационен цикличен алгоритъм е алгоритъмът, при който резултатите от текущото изпълнение на блока на цикъла са входни данни за следващото изпълнение.

Пример 37 Итерация на масив с конструкция for

В примера се извличат стойностите на елементите на масива $array и съдържанието на получения масив $names се извежда като номериран списък чрез итерация с конструкция for

6.4. Получаване на броя на елементите на масив– функция count

int count (mixed var [, int mode])
където:
var е масив или обект, чийто брой елементи връща функцията
mode е параметър за рекурсивност. При стойност COUNT_ RECURSIVE, ако масивът е многомерен, се преброяват всички негови елементи. При подразбиращата стойност 0 се преброяват елементите само по първи ключ.

6.5. Сортиране на елементите на масив– функция sort

Функцията сортира елементите на зададения масив в нарастващ ред. Начинът на сравнение се определя от параметъра sort_flags
Описание:
bool sort(array &array [, int sort_flags])
където:
array е масивът, чийто елементи се сортират,
sort_flags е параметър, който определя начина на сортиране. Допустими стойности на параметъра са константите:

- SORT_REGULAR – сравнява стойностите на елементите в съответствие с техния тип.
- SORT_NUMERIC – сравнява стойностите на елементите като числа
- SORT_STRING - сравнява стойностите на елементите като символни низове
- SORT_LOCALE_STRING - сравнява стойностите на елементите в съответствие с локалните настройки „current locale”. Този тип сравнение е добавен в PHP 4.4.0 и 5.0.2.

Локалните настройки могат да се променят с функцията setlocale() (или с i18n_loc_set_default() за PHP 6).

Да разгледаме един пример:

```
$a=array(5, 12, 3, 2);
sort($a, SORT_STRING);
print_r($a);
```

Пример 38 Сортиране на масив с функция sort със сравнение на елементите от тип SORT_STRING

В дадения по-горе пример се създава масив с числени стойности. При сортиране обаче се използва тип сравнение SORT_STRING, т.е. стойностите се сравняват като символни низове. Резултатът е Array ([0] => 12 [1] => 2 [2] => 3 [3] => 5). Това подреждане (което не е правилно ако се сортират числа) се получава, тъй като при сравнение на символни низове се сравняват кодовете на първите символи и ако са еднакви се продължава със сравнението на следващите. Тъй като кода на символа „1” е най малък, в сортирания масив стойността „12” се оказва първа, въпреки че числото 12 не е най-малкото, а на-голямото в масива.

В следващия пример елементите на масива се сортират като числа и затова получаваме подреждане в нарастващ ред на числовите стойности Array ([0] => 2 [1] => 3 [2] => 5 [3] => 12).

```
$b=array(5,12,3,2);
sort($b,SORT_NUMERIC);
print_r($b);
```

Пример 39 Сортиране на масив с функция sort със сравнение на елементите от тип SORT_NUMERIC

Недостатък на функцията sort е това, че тя може да прави сортиране само в нарастващ ред. PHP предоставя по-мощната функция, с която могат да се сортират няколко масива, като за всеки масив се задават два параметъра – за тип на сравнение и за посока на сортиране. Тя има следното описание:

bool array_multisort(array ar1 [, mixed arg [, mixed ... [, array ...]]])
където:

първият параметър е масив, след който следват един или два незадължителни параметъра, които определят типа на сравнение и посоката на сортиране. След първия масив в същия ред могат да се включат и параметри за други масиви.

Параметърът за сортиране (който липсва във функцията sort) може да има някоя от следните стойности:
SORT_ASC – Сортиране в нарастващ ред (подразбиращо се)
SORT_DESC - Сортиране в намаляващ ред

Да разгледаме един пример:

```
$a=array(5, 12, 3, 2); $b=array(5,12,3,2);
array_multisort( $a, SORT_STRING, SORT_ASC,
                 $b,SORT_NUMERIC, SORT_DESC );
```

Още по-големи възможности дава функцията за сортиране bool usort (array &$array , callback $cmp_function). Тя използва като втори параметър потребителска функция, с която се сравняват съседните елементи при сортиране. Функцията за сравнение трябва да връща числов резултат >0 , ==0 или <0 съответно ако първият от параметрите е по-голям, равен или по-малък от втория. Така с подходяща функция за сравнение $cmp_function може да се сортират елементите на масива по различни критерии. Пример:

```
function cmp($a, $b){return $a-$b;} $c=array(3,1,2); usort ($c,"cmp");
```

6.6. Итерация на елементите на масив– конструкция foreach

Конструкцията foreach е въведена в PHP 4, като, по подобие на Perl, Visual Biasic, и други програмни езици, е предназначена за итерация на масиви. При опит за итерация на друг тип данни се генерира грешка "Warning: Invalid argument supplied for foreach()". Конструкцията има съкратен синтаксис:

```
foreach (array_expression as $value)
{ блок инструкции }
```

и разширен синтаксис:

```
foreach (array_expression as $key => $value)
{ блок инструкции }
```

който включва допълнителния параметър $key и двойката символи “=>”.

Преди изпълнението на цикъла указателят на елементите на масива се установява на първия елемент. При всяко изпълнение на блока инструкции на цикъла променливата $value получава стойността на поредния елемент от масива, а променливата $key (ако се използва разширения синтаксис) получава ключа на поредния елемент.

Да разгледаме един пример:

```
$a=array("k1"=>"Test","k2"=>"construction","k3"=>"foreach");
$t="<OL>";
foreach($a as $key=>$val)
{
      $t = $t."<LI>$key=$val</LI>\n";
}
$t = $t."</OL>";
echo $t;
```

Пример 40 Итерация на масив с конструкция foreach

В примера се извършва итерация на масива $a като при всяко изпълнение на тялото на цикъла към променливата $t се добавя низ с HTML код на елемент от номериран списък, който съдържа ключа и стойността на поредния елемент от масива. След изпълнението на цикъла и добавяне на етикета за край на списъка, кодът на номерирания списък в променливата $t се извежда в прозореца на браузъра с конструкция echo.

Важно е да се запомни, че по подразбиране конструкцията foreach работи не с масива, а с негово копие, и поради това изменението на стойността на елемент от масива в тялото на цикъла не се запазва след изпълнението на foreach.

В PHP5 е добавена възможност за променяне на стойностите на масива при итерация. За целта е необходимо преди променливата $value да се постави символа &. Това ще има като резултат предаването на съдържанието на елемента от масива по адрес, при което променянето на стойността се запазва след приключването на изпълнението на цикъла. Можем да се убедим в това като изпълним PHP модул, съдържащ следния код:

```
$a=array(1,2,3,4);
foreach($a as &$value){
    $value=$value*2;
}
print_r($a);
```

Пример 41 Променяне на стойностите на елементите на масив при итерация с конструкция foreach

6.7. Итерация на елементите на масив– функции each, list и reset

Преди включването на конструкцията foreach в PHP, за итерация (т.е. последователна обработка) на елементите на масив най-често е използвана двойката функции each и list.

Функцията each връща текущата двойка ключ-стойност в 4-елементен масив с ключове 0, 1, “key” и “value”.

Описание:

array each (array &array)

където array е масивът, от който се извлича двойка ключ-стойност

Функцията each премества вътрешния указател на масива на следващия елемент. Ако преди изпълнението на each вътрешния указател е бил позициониран след последния елемент, each връща стойност FALSE. Да разгледаме един пример.

```
$a=array(‘a’=>’one’,’b’=>’two’);
$b=each($a);
print_r($b);
```

Пример 42 Извличане на двойка ключ-стойност от масив с функция each

Резултатът от изпълнението на функцията each от примера е масив със следното съдържание:

```
Array
(
    [1] => one
    [value] => one
    [0] => a
```

[key] => a
)

Както се вижда от примера, 4-елементния масив, който връща each, съдържа два елемента за извлечения ключ и два за извлечената стойност.

Вторият елемент от двойката функции за итерация на масиви, list, е всъщност конструкция, която извлича последователни стойности от масив с целочислени индекси с начален индекс 0.

Описание:

void list (mixed varname, mixed ...)

където varname е име на променлива, в която се извлича стойност на елемент от входния масив.

След изпълнение на конструкцията list указателят на масива сочи пак първия елемент, така, че при ново изпълнение на list четенето на стойностите на елементите ще започне отново от елемента с индекс 0. Да разгледаме един пример.

```
$a=array(“apple”,”banana”,”orange”);
list($b1,$b2)= $a;
print “b1=$b1<br>b2=$b2<br>”;
```

Пример 43 Извличане на стойности от масив с целочислени индекси чрез конструкция list

В примера конструкцията list извлича и присвоява на променливите $b1 и $b2 стойностите на първите два елемента от масива $a. Техните стойности “apple” и ”banana” се извеждат с конструкцията print.

Да разгледаме един пример за итерация на масив с двойката each – list:

```
$a=array("k1"=>"Test","k2"=>"iteration","k3"=>"each-list");
$t="<OL>";
reset($a); //Установяване на указателя на масива на първия елемент
while(list($key,$val) = each($a))
{
      $t = $t."<LI>$key=$val</LI>\n";
}
$t = $t."</OL>";
echo $t;
```

Пример 44 Итерация на масив с each - list

Примерът дава същия резултат, както приведения по-горе пример за итерация на масив с конструкцията foreach. Преди всяко изпълнение на тялото на цикъла се изпълнява функцията each($a), която извлича ключа и стойността на поредния елемент от масива в 4-елементен масив, два от елементите на който имат индекси 0 и 1. При изпълнение на конструкцията list($key,$val) стойностите на тези елементи (които съдържат всъщност ключа и стойността на поредния елемент от масива) се присвояват на променливите $key и $val. В тялото на цикъла към символния низ в $t се прибавя код на елемент от HTML списък, който съдържа стойностите на $key и $val.

Инициализация на указателя на масив – функция reset

Обърнете внимание на извикването на функцията reset($a) преди изпълнение на цикъла while. Функцията reset($a) инициализира указателя на масива, като го установява на първия елемент. Така след нейното извикване можем да бъдем сигурни, че ще се изведе съдържанието на всички елементи от масива. Трябва да се има предвид, че под "първи елемент" в случая се разбира първия създаден (и неунищожен с unset) елемент в масива.

7. Функции

Опитът в програмирането показва, че много често в една и съща програма се налага няколко пъти да се извършват едни и същи изчисления или да се решават едни и същи задачи. В такива случаи е по-рационално такива части от алгоритъма да се отделят като самостоятелни модули с възможност за многократно използване на включения в тях код. В PHP, както и във всички съвременни езици за програмиране, са включени структурни единици, наречени функции, които предоставят на програмиста възможност да използва многократно в програмата си кода, включен в тях.

7.1. Деклариране на функции.

Ако програмистът иска да създаде своя функция, той трябва задължително да я декларира. Без деклариране могат да се използват само вградените в PHP функции и функциите, които са методи на създадени обекти и се извикват чрез тях. В PHP функциите се декларират посредством ключовата дума function, следвана от списък от формални параметри в скоби и тяло (код) на функцията, включено във фигурни скоби.
Синтаксис:

```
function f_name ($arg1, $arg2, …, $argN)
{
      //statements
      .....
      .....
      return expression
}
```

където:

- f_name - е името на функцията
- $arg1, $arg2, …, $argN - е списък от параметри на функцията
- statements - са инструкции, включени в тялото на функцията
- return – е незадължителен оператор, чрез който функцията връща стойността на израза expression, след което се прекратява изпълнението на функцията

Тялото на една функция може да включва няколко оператора return, В PHP3 е било необходимо функциите да бъдат декларирани преди тяхното извикване. От PHP4 това изискване не важи, тъй като е въведено компилиране до междинен (т.н. байт-код) и последващо изпълнение от Zend Engine.

7.2. Условно деклариране

В PHP е включена възможност за създаване на функция при изпълнение на определено условие. За целта декларацията на

функцията се включва в блока на конструкция if, както е показано в дадения по-долу пример:

```
if($x > 0)
{
	function get_log_X($x)
	{
		return log($x);
	}
}
```

Пример 45 Условно създаване на функция

В примера ако $x > 0 се създава функцията get_log_X, която връща натурален логаритъм от стойността на $x.

В PHP ако една функция е декларирана в друга функция, то за да бъде достъпна вътрешната функция, външната функция трябва да бъде изпълнена поне веднъж.

7.3. Предаване на параметри на функцията и връщане на резултати.

По правило при изпълнението на функция извикващата програма предава на функцията набор от параметри. Параметрите, които се предават на функцията при нейното изпълнение е прието да се наричат фактически параметри.

Предаване на параметри по стойност

По подразбиране в PHP извикващата програма предава параметри на функциите по *стойност*. Стойностите на фактическите параметри при извикване на функцията се копират във формалните параметри*, с които функцията е декларирана, и се използват като входни данни при нейното изпълнение.

* В действителност стойностите на параметрите се копират във временни променливи, които се унищожават след изпълнение на функцията, но можем да считаме, че те са асоциирани с формалните параметри.

При предаването по стойност, променливите, които участват при изпълнението на операциите във функцията са временни променливи и затова променянето на техните стойности не се предава на фактическите параметри след завършването на изпълнението на функцията.

Например ако декларираме функция addition(a1,a2,s), и я извикаме с фактически параметри x, y и z, изменението на стойността на третия параметър s вътре във функцията няма да промени стойността на фактическия параметър z, така че след изпълнението на функцията addition променливите x, y и z ще имат същите стойности, както преди нейното изпълнение. От това следва, че функцията echo $z; ще изведе стойност 0.

```
function addition($a1,$a2,$s)
{
      $s= $a1+$a2;
}
$x=5; $y=7; $z=0;
addition($x, $y, $z);
echo $z;    //Извеждане на стойността на z
```

Пример 46 Предаване на параметри по стойност.

Предаване на параметри по адрес

В PHP предаването на параметър по адрес се указва, като преди символа $ на формалния параметър в декларацията на функцията се поставя символа &.

При предаване на параметри по адрес физическият адрес, на който се записват измененията в съдържанието на параметъра вътре във функцията съвпада с физическия адрес на фактическия параметър извън функцията. Поради това измененията на стойността на такъв параметър се запазват след изпълнението на функцията.

В дадения по-горе пример, ако в декларацията на функцията укажем че параметъра $s се предава по адрес, тогава изчислената му стойност във функцията ще се запази извън нея.

```
function addition($a1,$a2,&$s)
{
        $s= $a1+$a2;
}
$x=5; $y=7; $z=0;
addition($x, $y, $z);
echo $z;     //Извеждане на стойността на z
```

Пример 47 Предаване на параметър по адрес.

От това следва, че функцията echo $z; ще изведе изчислената във функцията стойност 12.

Задаване на подразбираща се стойност на параметри

PHP дава възможност да се зададат подразбиращи се стойности на параметри, като синтаксисът е същия, както в езика C: В списъка от параметри на функцията параметрите, на който се задават подразбиращи се стойности трябва да бъдат най-вдясно. Даденият по-долу пример съдържа декларация на функцията addition, в която на параметрите $a1 и $a2 са зададени подразбиращи се стойности 1. Ако функцията се извика само с един параметър, той ще получи стойност 2 – сумата от подразбиращите се стойности на $a1 и $a2.

```
function addition(&$s, $a1=1, $a2=1)
{
        $s= $a1+$a2;
}
```

Пример 48 Задаване на подразбиращи се стойности на параметри

7.4. Достъп до фактически параметри на функция чрез func_get_args.

PHP предоставя функцията func_get_args(), която, изпълнена в тялото на функция, връща масив с нулево базирани целочислени индекси, който съдържа стойностите на фактическите параметри, с които е извършено обръщението към функцията

Важно е да се запомни, че в PHP не е задължително функциите да се извикват със същия брой параметри, с който са декларирани. Можем да декларираме функция изобщо без параметри, а да я извикаме с набор от фактически параметри, достъп до които в кода на функцията можем да извършим чрез масива, който връща функцията func_get_args(). Да разгледаме един такъв пример:

```
$name="Peter";
function get_name( )
{
        $arg=func_get_args( );
        if(count($arg))
        {
        return $arg[0];
        }
}
echo "User name is ".get_name($name); //Извежда стойността на
                                                    //$name
```

Пример 49 Достъп до параметри на функцията посредством func_get_args( )

В примера функцията е декларирана без параметри, но се извиква с един фактически параметър – променливата $name. В кода на функцията достъп до параметъра се осъществява посредством масива, който връща функцията func_get_args( )

7.5. Променливи, декларирани във функциите.

В PHP променливи могат да се създават както в кода на функциите, така и извън тях. Променливите, създавани в скрипта извън декларациите на функциите са глобални – те са достъпни и запазват стойността си (или както се казва още, са видими) в целия код на скрипта, но не непосредствено (както в другите езици за

програмиране) а чрез масива $GLOBALS. За разлика от тях променливите, създавани във функциите, са локални – те са достъпни само във функцията, в която са декларирани. Тези променливи се създават при извикване на функцията и се унищожават след завършване на нейното изпълнение. Да разгледаме един пример:

```
function addition($a1, $a2)
{
	$s=$a1+$a2;  //Създаване на локална променлива $s
	return $s;
}
	$x=5; $y=7;
	$z=addition($x, $y);
	echo "z = $z <BR>";  // Извеждане на стойността на $z
	echo "s = $s <BR>";  // Извеждане на стойността на $s
```

Пример 50 Деклариране на локални променливи във функция

В примера във функцията addition се декларира променливата s и в същата инструкция се присвоява на $s сумата от стойностите на параметрите на функцията – $a1 и $a2. Операторът return прекратява изпълнението на функцията и връща чрез името на функцията стойността на s. В скрипта извън функцията се присвоява на глобалната променлива z резултата от изпълнението на функцията. Стойностите на $z и $s се извеждат посредством функции alert. Тъй като обаче променливата $s е локална – декларирана вътре във функцията addition – обръщението към нея извън функцията ще генерира грешка и ще се изведе съобщение “PHP Notice: Undefined variable: s ” ($s е недефинирана). Ако превърнем инструкцията *echo "s = $s
";* в коментар като поставим пред нея два символа наклонена черта “//”, останалата част от скрипта ще се изпълни и ще се изведе стойността на $z.

7.6. Статични променливи във функциите.

Подобно на C, PHP дава възможност за създаване на локални за функцията променливи като статични. Статичните променливи са

достъпни само вътре във функцията, но запазват стойността си след изпълнението на функцията. Те могат да бъдат инициализирани вътре във функцията, което става при нейното първото изпълнение. Да разгледаме един пример, в който се отброяват извикванията на функция посредством статична променлива.

```
function fn()
{
        static $counter=0; //Инициализиране при първо извикване
        $counter++;        //Отброяване при извикване
        echo "<BR>Брой извиквания=$counter";
}
```

Пример 51 Използване на статична променлива във функция

7.7. Връщане на резултат от функция

Функциите в PHP връщат резултат чрез оператора return. Той има следния синтаксис:

```
return expression;
```

където expression може да бъде всеки валиден израз на PHP, а в частност и просто име на променлива.

Извикващата програма получава резултат от изпълнението на функцията чрез нейното име. Например ако функцията addition трябва да връща резултат, то в нейната декларация трябва да се включи оператор return както в дадения по-долу пример:

```
function addition($a1, $a2)
{
        return $a1+$a2;
}
$x=5; $y=7;
echo addition(x,y);      //Извеждане на стойността на addition
```

Пример 52 Връщане на резултат от функция.

В примера извикването на функцията addition е включено като параметър на конструкцията echo. Като резултат върнатата посредством името на функцията addition стойност ще се изведе в диалоговата кутия.

Една функция по принцип може да съдържа повече от един оператор return. След изпълнението на оператор return изпълнението на функцията завършва и изпълнението на извикващата програма продължава със следващата инструкция.

7.8. Връщане на резултат по адрес

В тоя случай стойността, която връща функцията се предава по адрес, За връщане на резултат по адрес е необходимо:

a) В декларацията на функцията преди името на функцията да се постави символа &.
b) При извикване на функцията преди името и също трябва да се постави &.

Да разгледаме един пример:

```
$name="Peter";
function &get_name()
{
	return $GLOBALS["name"];
}
$new_name=&get_name(); //Присвояване на върнатата стойност по адрес
$new_name="John";
echo "User name is ".$name; //Извежда стойността на $name
```

Пример 53 Връщане на резултат от функция по адрес.

В примера функцията get_name() връща стойността на променлива, създадена извън функцията. Върнатата по адрес стойност се присвоява на променливата $new_name. Така двете променливи $name и $new_name се оказват с един и същ адрес. При променяне на стойността на $name стойността на $new_name също се оказва променена.

Вградени функции.

PHP предоставя на програмиста няколко голям брой вградени функции с различно предназначение, които могат да се използват навсякъде без да е необходимо да се декларират. Информация за тях можете да намерите в PHP Maual, който се инсталира заедно с PHP ако се избере съответната опция от групата Extras на Windows инсталационния файл.

7.9. Проверка за съществуването на функция – function_exists

PHP предоставя функцията function_exists, чрез която може да се провери, дали дадена функция е достъпна в PHP кода.
Описание:

bool function_exists (string function_name)

където function_name е името на функцията, за която се прави проверка.

Функцията function_exists връща логическа стойност TRUE ако проверяваната функция е достъпна, и FALSE в противен случай

7.10. Масив с имена на достъпни функции

Функцията get_defined_functions() връща масив с целочислени индекси, елементите на който съдържат имената на достъпните функции. Можем да изведем съдържанието му с функцията print_r

```
echo “<pre>”;
print_r(get_defined_functions());
echo “</pre>”;
```

Пример 54 Извеждане имената на всички достъпни функции.

8. Операции със символни низове

Символните низове са един от основните типове данни, с които работят интернет приложенията. Тъй като PHP е създаден като процедурно ориентиран език, той не включва клас String с методи за операции с низове, а предоставя набор от функции за основните символни операции: търсене, заместване, разделяне на части, и др. Ще разгледаме последователно функциите на PHP за тези операции.

8.1. Функции за търсене на подниз

PHP предоставя два вида функции за търсене на подниз: функции, които връщат индекс на подниза и функции, които връщат самия подниз.

Функции за търсене, които връщат индекс на подниз

Основната функция е strpos. Тя има следното описание
int strpos (string main_string, mixed search_string [, int offset])
където:
main_string е зададеният символен низ
search_string е търсеният подниз
offset – (незадължителен) е началният индекс, от който започва търсенето

Функцията връща индекс на открития подниз или FALSE ако низа не е открит.

PHP предоставя няколко подобни функции със същите параметри и значение:
stripos – при сравнението на search_string с main_string не прави разлика между малки и главни букви. (case-insensitive).
strrpos – търси подниза от дясно наляво (сравнението на символите обаче се извършва в нормалния ред)
strripos – търси подниза от дясно наляво като не прави разлика между малки и главни букви (case-insensitive right to left)

Да разгледаме един пример:

```
$s=”Hello World”;
if(strpos($s,”Hello”)!==FALSE)
{
```

```
    echo “Открит е подниза ‘Hello’ на позиция “.strpos($s,”Hello”);
}
```

Пример 55 Търсене а подниз с функция strpos

В примера се търси подниза ”Hello” в низа $s. Тъй като подниза е на индекс 0, е важно резултата, който връща strpos, да се сравнява за не-еквивалентност а не за неравенство с FALSE. При сравнение за неравенство 0 се приема за FALSE (в съответствие с правилата за конвертиране в лог. стойност) и поради това ще се получи грешен резултат че подниза не е открит.

Функции за търсене, които връщат подниз

Основната функция е strstr. Тя има следното описание
string strstr (string main_string, string search_string)
където:
main_string е зададеният символен низ
search_string е търсеният подниз

Функцията търси първото появяване на search_string в main_string и връща останалата част от откритата позиция до края на main_string или FALSE ако подниза не е открит.

PHP предоставя още една подобна функция със същите параметри и значение:
stristr – при сравнението на search_string с main_string не прави разлика между малки и главни букви. (case-insensitive).

Да разгледаме един пример:

```
$s=”Hello World”;
if(strstr($s,” Wo”)!==FALSE)
{
    echo “Открит е подниза ‘Wo’;
}
```

Пример 56 Търсене а подниз с функция strstr

В примера се търси подниза ”Wo” в низа $s. Функцията strstr връща останалата част от низа – “World” от позицията на откритият подниз ”Wo” до края на низа.

8.2. Получаване на дължината на низ – функция strlen

Функцията връща дължината на низа като целочислена стойност.
Описание: int strlen (string test_string)
където test_string е зададения низ

8.3. Извличане на подниз от зададена позиция – функция substr

Функцията връща подниз от зададен начален индекс и със зададена дължина (или до края на низа).
Описание: string substr (string string, int start [, int length])
където:
string е зададения низ
start е началният индекс, от който се извлича подниза
length – (незадължителен) е дължината на извличания подниз

Ако параметърът length не е зададен, се извлича подниз от индекс start до края на низа.

8.4. Разделяне на низ

PHP предоставя функции за разделяне на низ на части (поднизове) по два критерия: по зададена дължина на поднизовете и по зададен разделител, Ще разгледаме функциите, с които могат да се извършат такива разделяния:

Разделяне на низ по зададена дължина на подниз

Разделяне по зададена дължина на подниз може да се извърши с функцията str_split
Описание:

array str_split (string main_string [, int split_length])

където:
main_string е зададения низ
split_length – (незадължителен) е дължината на поднизовете

Функцията връща масив, в елементите на който са записани получените при разделянето поднизове. Ако split_length липсва във всеки елемент има само по един символ (както ако split_length=1)
Да разгледаме един пример:

```
$str=”Hello PHP”;
$a=str_split($str,2); //разделяне на поднизове с дължина 2
echo “<pre>”.print_r($a,TRUE).”</pre>”;
```

Пример 57 Разделяне на низ на поднизове с функция str_split

Низът се разделя на поднизове с дължина 2 символа.
Полученият масив се извежда чрез конструкция echo, като функция print_r с втори параметър TRUE връща (вместо да извежда) низ със съдържанието на масива. В прозореца на браузъра се извежда:

```
Array
(
    [0] => He
    [1] => ll
    [2] => o
    [3] => PH
    [4] => P
)
```

Разделяне на низ по зададен разделител

За разделяне на низ на части по един зададен разделител PHP предоставя функцията explode. Тя има следното описание:

array explode (string separator, string string [, int limit])

където:
separator е зададеният разделител
string е низа, който се разделя
limit – (незадължителен) задава макс. брой разделяния

Да разгледаме един пример:

```
$str=”Hello PHP”;
$a=explode(” “,$str); //разделяне с разделител – празен интервал
echo “<pre>”.print_r($a,TRUE).”</pre>”;
```

Пример 58 Разделяне на низ по зададен разделител с функция explode

Низът се разделя по разделител “ “ (празен интервал). Полученият масив $a съдържа два елемента с низовете “Hello” и “PHP”.

Разделяне на низ с разделител - регулярен израз

Функцията explode дава възможност за разделяне с един разделител. В някои случаи е необходимо да се направи разделяне по няколко възможни разделителя. За тази цел може да се използва функцията preg_split, в която разделителят се задава като регулярен израз. Тя има следното описание:
array preg_split (string pattern, string subject [, int limit [, int flags]])
където:
pattern е низ, съдържащ регулярен израз – шаблон за разделител
subject е зададеният символен низ, който ще се разделя
limit – (незадължителен) е максималният брой разделяния, които може да се направят
flags – (незадължителен) може да съдържа сума от параметри със следното значение:

- PREG_SPLIT_NO_EMPTY – указва да не се връщат празни низове
- PREG_SPLIT_OFFSET_CAPTURE – указва от всяко разделяне да се получава вложен масив, който освен стойността на подниза да съдържа и индекса (отместването) на позицията му в зададения низ
- PREG_SPLIT_DELIM_CAPTURE - указва низовете, разпознати като разделители също да се включват като елементи в получения масив.

Както се вижда от описанието на функцията preg_split, тя е доста по-сложна и ни дава много по-големи възможности от explode. Да разгледаме един пример:

```
$str="Hello, World of  PHP";
$a=preg_split("/[ ,\,]/",$str, -1, PREG_SPLIT_NO_EMPTY);
echo "<pre>".print_r($a,TRUE)."</pre>";
```

Пример 59 Разделяне на низ по списък от разделители с функция preg_split

В примера регулярният израз съдържа клас с два символа – празен интервал и запетая, за които се търси съвпадение. Зададеният низ съдържа и двата символа и те се използват като разделители при разделянето.

```
Array
(
    [0] => Hello
    [1] => World
    [2] => of
    [3] => PHP
)
```

Получените празни поднизове не се записват в масива, тъй като е зададен параметъра PREG_SPLIT_NO_EMPTY. Получава се следния резултат:

9. Функции за операции с файлове

Файловата система (ФС) е структура, състояща се от директории и файлове (обекти на ФС), която организира съхраняването и достъпа до данните върху дисковите запомнящи устройства на компютъра. На практика цялото програмно обезпечение и всички локални данни, които се използват от компютъра се съхраняват във файловете на ФС. Поради това всеки език за програмиране и всяка среда за разработване на програмни продукти (с малки изключения*) предоставят средства за операции с обекти на ФС.

PHP предоставя набор от функции за операции с файлове. Едно от големите предимства на PHP е, че с един и същ набор от функции се извършват файлови операции както с файлове от локалната ФС, така и с файлове с мрежов достъп. Ще разгледаме функциите за основните файлови операции: отваряне на файл, четене, запис и затваряне на файл.

9.1. Отваряне на файл или URL – функция fopen

* Например в Java аплетите, от съображения за сигурност, по подразбиране е забранен достъпът до локалната файлова система.

Функцията fopen свързва ресурс, зададен с filename, с поток от данни.
Описание:
resource fopen (string filename, string mode [, bool use_include_path [, resource zcontext]])
където:
filename е спецификация на локален файл или адрес на мрежов ресурс
Ако низа filename започва с protocol://, където protocol е някой от поддържаните протоколи за обмен на данни,, то filename се счита за адрес на ресурс (URL) и при успешно изпълнение, fopen ще върне манипулатор за този ресурс. Ако зададеният протокол не се поддържа, се извежда съобщение за грешка.Ако низа filename започва с “ftp://” (т.е. ако протокола е File Transfer Protocol) се създава ftp връзка към зададения сървър, изпраща се заявка и при получаване на отговор от сървъра се връща ресурс за файлова операция. Ако сървърът не поддържа пасивен режим “Passive Mode”, функцията fopen връща стойност за неуспешна операция (FALSE). При създаване на FTP връзка файлът може да бъде отворен за четене или за запис, но не и за двете операции едновременно.

Ако низа filename започва с “php://stdin” , с “php://stdout” или с “php://stderr” , се създава поток съответно за системен вход, системен изход или за извеждане на системното изходно устройство на съобщение на грешка. Ако спецификацията на файла започва с нещо друго, напр. с “/”, “/” или “C:” или с име на файл, се отваря файл от локалната файлова система и функцията fopen връща манипулатор за този файл

Параметърът mode задава режима на отваряне на файла. Той приема следните стойности:

‘r’	- отваряне на файла или ресурса само за четене
‘r+’	- отваряне за четене и запис
‘w’	- отваряне само за запис
‘w+’	- отваряне за четене и запис. Указателя на файла за запис се установява в началото, което означава, че се надписва старото му съдържание.
‘a’	- отваряне само за запис с добавяне към старото съдържание. Указателят за запис се установява в края на файла. Ако файла

	не съществува, се прави опит за създаването му.
‘a+’	- отваряне за четене и запис с добавяне към старото съдържание. Ако файла не съществува, се прави опит за създаването му.
‘x’	- отваряне само за запис. Указателя на файла за запис се установява в началото на файла. Ако файлът съществува, се генерира грешка и fopen връща стойност FALSE. Ако файла не съществува, се прави опит за създаването му.
‘x+’	- отваряне за четене и запис. Указателя на файла за запис се установява в началото на файла. Ако файлът съществува, се генерира грешка и fopen връща стойност FALSE. Ако файла не съществува, се прави опит за създаването му.

Освен дадените по-горе символи атрибутът mode може да съдържа и символа “b” (от binary). Той е от значение само за операционни системи, които разграничават файловите операции с текстови файлове от тези с двоични файлове. В ОС Linux този параметър се игнорира. За ОС Windows е желателно символът “b” да се включва в параметъра mode при операции с двоични файлове.

Параметърът use_include_path е от значение тогава, когато е зададен относителен адрес на файла. Ако use_include_path има стойност '1' или TRUE, освен в текущата директория, зададеният файл се търси и в директорията, зададена като стойност на параметъра include_path в конфигурационния файл boot.ini.

Параметърът zcontext задава набор от параметри, които модифицират поведението на потока, създаван от fopen. Ресурсът zcontext се може да бъде създаден с функцията stream_context_ create().

Примери:

$fp=fopen(“http://www.php.net/”, “r”); //отваряне на файл на интернет документ

$fp=fopen(“/home/proba.txt/”, “r”); . //отваряне на локален файл

9.2. Функции за четене от файл

Функциите за четене от файл, които предоставя PHP, поддържат файлов указател, като четат последователно съдържанието на файла от позицията на указателя и го преместват след прочитане на определен брой байтове или символи.

Четене на ред от файл – функция fgets

Функцията прочита текстов ред или length-1 байта от текущата позиция на файловия указател
Описание:
string fgets (resource handle [, int length])
където:
handle е файлов манипулатор, получен от функцията fopen
length е броя байтове, който трябва да бъде прочетен

Четенето завършва в следните случаи:

a) когато се прочетат length-1 байтове
b) когато се достигне до символ за край на ред (този символ се включва в прочетените данни
c) когато се достигне до края на файла

За успешно изпълнение на операцията е необходимо handle да съдържа валиден файлов манипулатор, получен с някоя от функциите fopen, popen или fsockopen. При успешна операция функцията връща прочетеното в символен низ, а при грешка връща стойност FALSE.
Да разгледаме един пример:

```
echo “<pre>”;
$fp=fopen(“proba.txt” ,”r”) or die(“<br>can’t open file for reading”);
while(!feof($fp))
{
        $buffer=fgets($fp,4096);
        echo $buffer;
}
fclose($fp); //затваряне на файла
echo “</pre>”;
```

Пример 60 Прочитане на текстов файл по редове

В примера се прочита и извежда последователно по редове съдържанието на файла proba.txt, който трябва да е записан в същата директория, в която е PHP модула. Обърнете внимание на използването на функцията feof($fp) за проверка нали е достигнат края на файла и функцията fclose($fp) за затваряне на файла. Етикетът за преформатиране <pre> е необходим за да се отработва символа за преминаване на нов ред при извеждане на текста във Web браузъра.

Ако при отваряне на файла функцията fopen() върне стойност FALSE, се изпълнява функцията die(), която извежда съобщението “can’t open file for reading” и прекратява изпълнението на PHP модула.

Четене на зададен брой байтове от файл – функция fread

Функцията прочита length байта от текущата позиция на файловия указател.
Описание:
string fread(resource handle int length)
където:
handle е файлов манипулатор, получен от функцията fopen
length е броя байтове, който трябва да бъде прочетен

Четенето завършва в следните случаи:

a) когато се прочетат length байтове
b) когато се достигне до края на файла
c) (за четене от мрежов ресурс) докато има достъпни символи
d) (за дефиниран от потребителя поток) когато се прочетат 8192 байта

Както се вижда от описанието, разликата между fgets и fread е в това, че fread не прекратява четенето, когато прочете байт, който съдържа символа за край на ред. Може да се каже, че fgets е по-подходяща за четене от текстови файлове, а fread е по-подходяща за двоични файлове.

Четене на символ от файл – функция fgetc

Функцията прочита един символ от текущата позиция на файловия указател
Описание:
string fgetc(resource handle)
където:
handle е файлов манипулатор, получен от функцията fopen

След прочитане на символа файловият указател се мести на следващия символ, така че с последователни извиквания на fgetc може да се чете съдържанието на файла символ по символ.

Прочитане на цялото съдържание на файл – функция file_get_contents

Функцията отваря файла, прочита цялото му съдържание в символен низ, като започва четенето от отместване offset, и затваря файла. Максималния брой байтове, които да бъдат прочетени, може да се ограничи с параметъра maxlen.
Описание:
string file_get_contents (string filename [, bool use_include_path [, resource context [, int offset [, int maxlen]]]])
където:
filename е спецификация (или URL) на файла
use_include_path – указва при стойност TRUE да се претърсва за относителен адрес и директорията, зададена като стойност на параметъра include_path в конфигурационния файл php.ini
context – има същото значение, както във функцията fopen
offset – отместване на първия прочетен байт от началото на файла
maxlen – максимален брой байтове за четене

Да разгледаме пример за прочитане и извеждане на съдържанието на текстов файл, реализиран с функцията file_get_contents:

```
echo “<pre>”;
$buffer=file_get_contents(“http://www.abv.bg”);
echo htmlspecialchars($buffer);
echo “</pre>”;
```

Пример 61 Прочитане на HTML документ с функцията file_get_contents

В примера се прочита и извежда съдържанието на началната страница на сайта на abv.bg. Обърнете внимание на прекодирането на прочетения в $buffer символен низ с функцията htmlspecialchars. С нея се прекодират специалните за HTML символи като “<” и “>”. Благодарение на това те се извеждат, а не се отработват като маркери за HTML етикети. В прозореца на браузъра ще се изведе сорс кода на HTML страницата, докато, ако липсваше прекодиране с htmlspecialchars(), щеше да се изведе съдържанието на HTML страницата. Етикетът за преформатиране “<pre> се изпраща, както и в предния пример, за да се отработва символа за преминаване на нов ред.

9.3. Функции за запис във файл

Функциите за запис във файл, които предоставя PHP, поддържат файлов указател за запис, като записват последователно във файла (или в общия случай, във файлов изходен поток) от позицията на указателя и го преместват след запис на определен брой байтове или символи.

При изпълнение в ОС (напр. Windows), които правят разлика между двоични и текстови файлове, е желателно при отваряне на файла за запис в стойността на параметъра mode да се добавя символа “b”.

!Важно е да се има предвид, че за да се извърши операция запис, е необходимо Интернет приложението да има права за запис в съответната директория и файл, а ако файла липсва, и права за създаването му. В противен случай се генерира грешка “Permission denied ”.

Запис на символен низ във файл – функция fwrite

Функцията записва съдържанието на зададен символен низ във файл. Ако е зададен броя байтове за запис, записът ще завърши

когато length брой байтове бъдат записани или когато бъде достигнат края на низа (ако е с по-малка дължина). Ако параметърът length не е зададен, се записва цялото съдържание на зададения низ.
Описание:
int fwrite (resource handle, string string [, int length])
където:
handle е файлов манипулатор, получен от функцията fopen
length – (незадължителен) е броя байтове за запис

Да разгледаме един пример:

```
$fp=fopen(“proba.txt” ,”wb”) or die(“<br>can’t open file for writing”);
$s=”First line \nSecond Line\n”;
fwrite($fp, $s); //запис във файла
fclose($fp); //затваряне на файла
```

Пример 62 Запис на символен низ във файл с функцията fwrite

В примера файлът proba.txt се отваря за запис и с функцията fwrite() се записва съдържанието на низа $s, след което файлът се затваря с fclose($fp). Ако при отваряне на файла функцията fopen() върне стойност FALSE, се изпълнява функцията die(), която извежда съобщението “can’t open file for writing” и прекратява изпълнението на PHP модула.

Отваряне, запис на символен низ във файл и затваряне – функция file_put_contents

Описание:
int file_put_contents(string filename, mixed data [, int flags [, resource context]])
където:
filename е спецификация (или URL) на файла
data – Данни за запис. Могат да бъдат: символен низ, масив или ресурс от тип stream (поток).
flags – Може да приема сума от следните константи:

FILE_USE_INCLUDE_PATH	Търси името на файла *filename* в include директорията, зададена като стойност на параметъра include_path от файла php.ini.

FILE_APPEND	Ако файла *filename* вече съществува, добавя записа към съдържанието на файла, вместо да го надписва.
LOCK_EX	Извършва привилегировано заключване на файла по време на записа..
FILE_TEXT	Данните се записват в текстов режим. Ако unicode семантиката е достъпна, подразбиращото се кодиране е UTF-8. Друго кодиране може да се зададе ако се създаде потребителски контекст или ако се използва функцията **stream_default_encoding()** за да се промени текущия контекст. Този флаг достъпен от PHP6 и не може да се използва съвместно с **FILE_BINARY**.
FILE_BINARY	Данните се записват като двоични данни. Този флаг достъпен от PHP6 и не може да се използва съвместно с **FILE_TEXT**.

context – има същото значение, както във функцията fopen

Да разгледаме пример за запис на съдържанието на символен низ във файл

```
$s=”First line \nSecond Line\n”;
file_put_contents($fp, $s); //запис във файла
```

Пример 63 Запис на символен низ във файл с функцията file_put_contents

В примера функцията file_put_contents отваря файла proba.txt за запис, записва в него низа $s и затваря файла. Примерът включва само две инструкции, тъй като и трите операции с файла се извършват от file_put_contents.

9.4. Затваряне на файл – функция fclose

Функцията затваря файла (файловия поток), асоцииран с файловия манипулатор.
Описание:
bool fclose (resource handle)
където handle е файловия манипулатор

Желателно е при завършване на операциите с файла той да се затвори, тъй като така се освобождават ресурсите, които са заделени за операциите с него.

9.5. Други функции за операции с файлове

Поради ограничения обем на курса тук ще бъде дадено кратко описание само на ограничен набор от функции за файлови операции.

- Проверка за съществуване на файл или директория:
  bool file_exists (string filename) – връща лог. стойност TRUE ако файла съществува.
- Получаване на размера на файл с брой байтове:
  int filesize (string filename) – връща дължината на файла
- Проверка дали файла е разрешен за четене:
  bool is_readable (string filename) – връща TRUE ако файла е разрешен за четене
- Проверка дали файла е разрешен за запис:
  bool is_writeable (string filename) – връща TRUE ако файла е разрешен за запис
- Проверка дали файла е изпълнима програма:
  bool is_executable (string filename) – връща TRUE ако файла е изпълнима програма
- Изтриване на файл:
  bool unlink (string filename [, resource context]) – връща TRUE ако файла е изтрит успешно
- Изтриване на директория:

bool rmdir (string dirname [, resource context]) – връща TRUE ако директорията е изтрита успешно

- Създаване на директория:

bool mkdir (string pathname [, int mode [, bool recursive [, resource context]]]) – връща TRUE ако директорията е създадена успешно

където:

mode – задава права за достъп до директорията във формат на ОС Linux (стойността по подразбиране е 0777)

10. Получаване на данни от Web браузър. Масиви $_GET, $_POST и $_REQUEST

Web браузърите използват два метода за изпращане на данни към интернет приложение:

Метод GET: Изпращаните данни се добавят към адреса (URL) след символа „?” като двойки име=стойност разделени със символа „&”, зададен с атрибута action. Даденият по-долу пример съдържа код на HTML формуляр за изпращане на данни по метод GET:

```
<form method= "get" action="http://index.php">
<input name="fname" value="John"><input name="age" value="22">
<input type="submit" value="Send">
</form>
```

Пример 64 Изпращане на данни от HTML формуляр по метода GET

Формулярът съдържа две текстови кутии с имена (name) съответно fname и age, чието съдържание при изпращане по метода GET ще се добави към адреса, зададен в атрибута action както следва:

http://index.php?fname=John&age=22

При използване на метода GET данните могат непосредствено да се добавят към адреса на хипервръзка, както е в следващият пример:

```
<a href="http://index.php?fname=John&age=22 ">Go to start page</a>
```

Пример 65 Изпращане на данни от хипервръзка по метода GET

В примера данните са добавени Методът GET има ограничения за обема на изпращаните данни данни. Друг недостатък на метода е, че изпращаните данни (включително и паролите) се виждат като добавени към URL в адресната лента на браузъра.

За получаването на данни, изпратени по метода GET PHP предоставя глобалния масив $_GET. Получените данни се записват в масива като двойки *име-стойност*. Например изпратените чрез горният формуляр данни могат да се получат в PHP модул със следният код:

```
<?php
$name = $_GET["fname"];
$age = $_GET["age"];
echo "Hello $name , your age is $age";
?>
```

Пример 66 Получаване на данни, изпратени по метода GET

Метод POST: Този метод не притежава недостатъците на метода GET, поради което се предпочита. Данните се изпращат към сървъра в стандартния изходен поток и поради това няма ограничения за обема им и не се виждат в адресната лента на браузъра. Даденият по-долу пример съдържа код на HTML формуляр за изпращане на данни по метод POST:

```
<form method= "post" action="http://index.php">
<input name="fname" value="John"><input name="age" value="22">
<input type="submit" value="Send">
</form>
```

Пример 67 Изпращане на данни от HTML формуляр по метода POST

За получаването на данни, изпратени по метода POST PHP предоставя глобалния масив $_POST. Получените данни се записват в масива като двойки *име-стойност*. Например изпратените чрез горният формуляр данни могат да се получат в PHP модул със следният код:

```
<?php
$name = $_POST["fname"];
$age = $_POST["age"];
echo "Hello $name , your age is $age";
?>
```

Пример 68 Получаване на данни, изпратени по метода POST

В HTML кода на PHP модула също може да се съдържа формуляр, чрез който да се изпращат данни към друг или към същия PHP модул, както е показано в следващият пример:

```
<HTML><BODY>
<form method= "post">
Name:<input name="fname" value="John"><br/>
Age: <input name="age" value="22">"><br/>
<input type="submit" value="Send">"><br/>
</form>
<?php
 if(isset($_POST["fname"] && isset($_POST["age"]))
 {
        $name = $_POST["fname"];
        $age = $_POST["age"];
        echo "Hello $name , your age is $age";
 {
?>
<BODY><HTML>
```

Пример 69 Код на PHP модул, който си изпраща данни чрез формуляр

В примера PHP модула изпраща чрез формуляра данни на себе си, тъй като във формуляра липсва атрибута *action*. При първото зареждане на PHP модула обаче той не получава данни и следователно масивът $_POST е празен. Поради това (за да не се генерира грешка при опит за четене от празен масив) в PHP кода е включена конструкция if, чрез която се проверява дали са получени данни в масива $_POST. След като се зареди PHP модула в браузъра ако се щракне на бутона “Send” данните ще се изпратят към сървъра и при следващото зареждане на PHP модула (по-точно,

на HTML кода, който той изпраща) ще се изведе текста "Hello John , your age is 22".

10.1. Масив $_REQUEST

Глобалният масив $_REQUEST съдържа данни, изпратени и по метод GET и по метод POST. Използването му вместо масивите $_GET и $_POST в известен смисъл улеснява разработчика, тъй като в процеса на разработка могат да се изпращат данни по метода GET за да се виждат като добавени към адреса на PHP модула, а в последствие да се изпращат по метода POST без да се налага да се променя PHP кода.

11. Прехвърляне на файлове от потребителския компютър (file upload)

HTML 4.0 и по-новите версии дават възможност чрез формуляр да се изпрати към сървърната програма съдържанието на файл от компютъра на потребителя. Това може да се види например в сайтове за администриране на (безплатен) сайт на потребител. Потребителят може да записва файлове в своя сайт като ги изпраща чрез елемент FILE от формуляр от предоставената му страница за поддържане на сайта (File manager).

Важно е да се запомни, че за да може да се изпраща съдържанието на файл чрез HTML формуляр, формулярът трябва да има зададена стойност multipart/form-data на атрибута enctype, и метод за изпращане “POST”, както е показано в следващия пример:

```
<form method="post" enctype="multipart/form-data">
<input type="file" name="files[]"><br>
<input type="file" name="files[]"><br>
<input type="submit" value="upload">
</FORM>
```

Пример 70 HTML формуляр за изпращане съдържанието на файлове

В примера HTML формуляра съдържа два елемента от тип FILE с едно и също име, завършващо с двойка отваряща и затварящи квадратни скоби. Това съвсем не е задължително, но улеснява операциите с файловете в PHP модула Формулярът няма атрибут ACTION, което означава, че той ще изпрати данните (в случая съдържанието на файловете) на същия модул, в който се съдържа.

За получаването на съдържанието на файлове, изпратено от HTML формуляр, PHP предоставя глобалния масив $_FILES. Масивът $_FILES съдържа подмасиви с имената на елементите от тип FILE във формуляра, които съдържат следните елементи като вложени масиви, в които се записват параметри на прехвърлените файлове:

name	съдържа имената на прехвърляните файлове
type	съдържа типа на данните във файловете. Подразбира се тип “text/plain”
tmp_name	съдържа спецификацията на файловете във временната директория на сървъра, в която се записват
error	съдържа 0 за файловете, които са прехвърлени успешно и 1 за тези, които не са прехвърлени.
size	Съдържа размера на прехвърлените файлове в байтове

Както се вижда от параметъра tmp_name, прехвърлените файлове се записват във временна директория на сървъра. По правило разработчикът иска да ги запише в директорията на PHP приложението, или в друга работна директория, в която приложението има права за операции с файлове.

В случай, че елементите от тип FILE в HTML формуляра са с едно и също име, завършващо с двойка отваряща и затварящи квадратни скоби, (както е в дадения пример) данните за тях в масива $_FILES се оказват в един подмасив, с което се улеснява тяхната обработка.

Прехвърляне на файловете от временната в зададена работна директория

За прехвърляне файловете от временната в зададена работна директория PHP предоставя функцията move_uploaded_file. Тя има следното описание:

bool move_uploaded_file (string filename, string destination)

където:

filename	е спецификацията на файла във временната директория.Тя може да бъде получена от съответния (по номер на файла) елемент от подмасива tmp_name на масива $_FILES.
destination	е спецификацията на преместения файл

Функцията проверява, дали файла, зададен с filename е валиден прехвърлен файл (от PHP по HTTP протокол). Ако файла е валиден прехвърлен файл, .той се премества в destination и функцията връща стойност TRUE.

При невалиден файл не се извършва преместване и функцията връща стойност FALSE. Стойност FALSE се получава и ако файла не може да бъде преместен по някаква причина. Тези проверки преди прехвърлянето на файла са важни за да се избегне възможността за неоторизиран достъп до прехвърлен файл от други потребители.

Да разгледаме кода на PHP модул, който прехвърля файловете получени от дадения по-горе HTML формуляр в зададена директория:

```
echo "<PRE>".print_r($_FILES,TRUE)."</PRE>";
if(isset($_FILES))
if(isset($_FILES["files"]))
foreach($_FILES["files"]["error"] as $key=>$error){
        if($error==UPLOAD_ERR_OK){
                $tmp_name=$_FILES['files']['tmp_name'][$key];
                $name=$_FILES["files"]["name"][$key];
                move_uploaded_file($tmp_name,"C:\\tmp\\$name");
        }
}
```

Пример 71 Преместване на файлове, получени от HTML формуляр

В примера, тъй като елементите от тип FILE от формуляра имат едно и също име files[] , завършващо с отваряща и затваряща

квадратна скоба, данните за прехвърляните файлове се оказват в един и същ вложен масив files в масива $_FILES. Чрез итерация на вложения масив files успешно прехвърлените файлове се преместват със същите имена в директорията C:\tmp.

Ако към формуляра се добавят още елементи от тип FILE със същото име, без никакви промени в PHP кода успешно прехвърлените файлове ще бъдат записани в същата директория.

12. Контролни структури за включване на код

Възможността за включване на код на един PHP модул в друг е изключително важна за оптимизацията на кода на PHP приложенията. Тя позволява структурирането на програмата от функционални блокове, които могат да се използват в различни по предназначение приложения.

12.1. Конструкции include и require

Конструкциите include и require позволяват включването в PHP модул на файл с PHP код. При изпълнението на такава конструкция PHP чете файла, компилира го до междинен код и го изпълнява като част от кода на текущия PHP модул. За разлика от C/C++ директивата include, конструкциите include и require имат поведение, подобно на функциите, тъй като могат да връщат резултат. Единствената разлика между конструкциите include и require е в това, че ако файлът за вмъкване не бъде открит конструкцията include генерира само предупреждение (warning) и изпълнението на PHP кода продължава, докато require генерира “Fatal error” и изпълнението на PHP модула се прекратява. Конструкциите имат следния синтаксис:

```
include file_name;
require file_name;
```

където file_name е името (или спецификацията) на PHP модула, който ще се вмъква. Допуска се file_name да бъде в скоби, т.е.

include (file_name);
require (file_name);
също е синтактически правилно.

При използването на относителен адрес на вмъквания файл той първо се търси в зададената в спецификацията поддиректория на текущата работна директория, а след това – в директорията, в която се съдържа текущо изпълнявания PHP модул. Разлика между текущата работна директория и директорията на текущия модул може да се получи например когато вмъкваният модул също съдържа конструкция include. Ако текущата работна директория е / my_site/ и вмъкнем файл със спецификация “include/a.php” , а този файл съдържа конструкция require "b.php” . тогава файла b.php първо ще бъде потърсен в /my_site/ , а след това в /my_site/include .

Ако спецификацията на вмъквания файл започва с “./” или с “../” , тогава той се търси само в зададената в спецификацията поддиректория на текущата работна директория.

Кодът от включеният с някоя от конструкциите файл се изпълнява със същата област на видимост както извикващият код. Например ако конструкцията include a.php е включена в тялото на функция, променливите, които се създават в нейния код ще са локални променливи за функцията. Включените във вмъквания код декларации на функции и класове обаче ще имат глобална видимост.

Ако в конфигурационния файл php.ini на параметъра allow_url_include е зададена стойност “on”, то е разрешено вмъкването на PHP модули, външни за PHP приложението от зададен интернет адрес (URL).

Съществуват две разновидности на include/require , които в повечето случаи са по-полезни. Конструкциите
include_once (file_name);
require_once (file_name);
имат същото действие както разглежданите include/require с тази разлика, че ако даден файл е вече веднъж вмъкнат в даден PHP модул, той не се вмъква повторно. Например ако дадено PHP приложение в един модул са събрани декларации на често използвани функции, той може да бъде вмъкнат с include once във всеки от останалите модули, дори някой от тях също да се вмъква в

друг модул. Вмъкването с include ще доведе до грешка, тъй като PHP не допуска повторно деклариране на функция.

13. Операции с бисквидки

Бисквидките са въведени от Netscape, но се поддържат от всички съвременни браузъри. Те представляват текстова информация, която се изпраща от интернет приложение към потребителския браузър в отговор на заявка, изпратена от браузъра към Web сървъра. Заглавната част, с която се изпраща бисквидка към Web браузъра има следния формат:

Set-Cookie: ИМЕ=СТОЙНОСТ [;expires = ДАТА] [; path=ПЪТ] [; domain=ИМЕ на Домейн] [;secure]

Значението на полетата в заглавната част е следното:

- поле ИМЕ=СТОЙНОСТ – това поле е задължително и задава името и стойността на бисквидката, Останалите полета са незадължителни, но трябва да се запишат в зададения ред. В имената и стойностите не се допускат символи “,” , “;” и празни интервали.
- поле expires = ДАТА – определя моментът (датата и времето), след който изтича валидността на бисквидката. Ако полето липсва, браузърът унищожава бисквидката след изтичане на сесията (т.е. периода на активност) на текущия потребител на интернет приложението. Датата има следния формат:

  Weekday, DD-Mon-YY HH:MM:SS GMT

  където съответно Weekday е името на деня от седмицата, DD – деня от месеца, Mon – името на месеца (съкратено до 3 символа), YY – годината (съкратена до 2 цифри), HH – часа, MM – минутите, SS – секундите на GMT – спрямо нулевия меридиан (т.н. Гринуичко време). Пример:

  Monday 20-Sep-10 12:00:00 GMT
- поле path=ПЪТ – асоциира бисквидката с пътя до името на заявения модул или документ. Например

http://abv.bg/cgi-bin/search.php ще асоциира бисквидката с пътя /cgi-bin/

- поле domain=ИМЕ на Домейн – асоциира бисквидката с име на домейн и поддомейна (в URL поддомейна е непосредствено преди името на домейна. Например ако URL е www.yahoo.com асоциирането ще бъде с yahoo.com (домейн com, поддомейн yahoo).
- поле secure – ако е включено в бисквидката, тя може да се изпраща само по криптиран канал за комуникация (HTTPS протокол)

Когато браузърът заявява документ (или ресурс) с URL адрес от Web сървъра, той предварително претърсва записаните бисквидки за да провери дали полетата path и domain на някои от тях не съответстват на адреса на заявката. Ако има такива бисквидки, браузърът ни изпраща към Web сървъра в заглавната част на HTTP заявката в следния формат:

Cookie ИМЕ1=СТОЙНОСТ1; ИМЕ2=СТОЙНОСТ2; и т.н.

Стойността на бисквидката автоматично се URL кодира при изпращане и съответно URL декодира при получаване. Програмните езици за създаване на интернет приложения предоставят на програмиста средства за:

- запис на бисквидки на потребителския компютър
- получаване на имената и стойностите на изпратените от потребителския браузър бисквидки.

Ще разгледаме функциите, които предоставя PHP за операции с бисквидки.

13.1. Изпращане на бисквидка към потребителския браузър – функция setcookie

Функцията създава бисквидка и я изпраща в заглавната част на HTTP отговора, който Web сървъра изпраща към потребителския браузър. Функцията има следното описание:

bool setcookie (string name [, string value [, int expire [, string path [, string domain [, bool secure [, bool httponly]]]]]])

където:

name	име на бисквидката
value	стойност на бисквидката
expire	Задава моментът в секунди, след който изтича валидността на бисквидката. За да се зададе като времеинтервал, интервалът на валидност (в секунди) се добавя към текущото време, получено от функция time(). Например time( ) + 10 означава, че валидността на бисквидката ще изтече след 10 секунди.
path	път в URL адреса, с който се асоциира бисквидката. Ако се зададе само “/”, това означава, че бисквидката ще се асоциира с целия домейн.
domain	домейн, с който се асоциира бисквидката. Ако низа започва с “.”, бисквидката се асоциира със всички поддомейни на зададения домейн
secure	при стойност TRUE указва бисквидката да се изпраща само по HTTPS протокол
httponly	при стойност TRUE разрешава изпращане на бисквидката само по HTTP протокол (т.е. забранява изпращането и по метода GET – с добавяне към URL

Аналогична на setcookie е функцията setrawcookie, но с тази разлика, че тя не извършва URL кодиране на стойността на бисквидката.

13.2. Изпращане на заглавна част към потребителския браузър – функция header

PHP предоставя функция, с която към Web браузъра може да се изпрати произволна заглавна част. Тя има следното описание:

void header (string text [, bool replace [, int http_response_code]])

където:

text е текста на заглавната част

replace – при стойност TRUE указва да се замести съществуваща заглавна част със същото име със зададената.

http_response_code – задава HTTP код на отговор

За да се изпрати към Web браузъра бисквидка с тази функция в text трябва да се запише текста на бисквидката в зададения в по-

горе формат на заглавна част Set-Cookie. Функцията може да се използва и за съвсем други цели, например за зареждане в прозореца на браузъра на друг документ с инструкцията:

```
header('Location: http://www.yahoo.com/');
```

Пример 72 Зареждане в прозореца на браузъра на HTML документ чрез функция header

13.3. Четене на достъпните бисквидки чрез масива $_COOKIE

Изпратените от потребителския компютър бисквидки са достъпни в PHP модулите на интернет приложението посредством масива $_COOKIE. Можем най-лесно да изведем имената и стойностите на всички бисквидки като изведем съдържанието на масива посредством функция print_r, например с инструкцията.

echo “<pre>”.print_r($_COOKIE, TRUE).”</pre>”;

Да разгледаме един пример за създаване на бисквидка с функция setcookie:

```
echo "<pre>".print_r($_COOKIE, TRUE)."</pre>";
setcookie("Author","John", time( )+10);
$_COOKIE["Color"]="Yellow";
```

Пример 73 Създаване на бисквидка с функция setcookie и извеждане на достъпните бисквидки чрез масива $_COOKIE

В примера се извежда съдържанието на масива $_COOKIE чрез функцията print_r и се използва функцията setcookie за създаване на бисквидка с време на съществуване 10 секунди. Това се получава като към текущотото време се, получавано от функцията time( ) се добавя 10 (секунди). При първото заявяване на модула масивът $_COOKIE е празен, но при повторно заявяване (напр. с бутона Refresh на браузъра) в него се появява бисквидката, създадена с функцията setcookie. Елементът с ключ “Color” и стойност “Yellow”, записан в масива, не се появява след Refresh,

тъй като записът на елемент в $_COOKIE не изпраща бисквидка към потребителския браузър.

Ако се отвори страница от друго интернет приложение, след 10 секунди валидността на създадената със setcookie бисквидка “Author” изтича и при повторно заявяване на модула с примера се оказва унищожена и отново се извежда празен масив $_COOKIE.

14. PHP сесии

Потребителската сесия на интернет приложение е период на активност на потребител, притежаващ уникален IP адрес, през който той извършва операции в рамките на приложението. Броят на сесиите на даден Web сайт се използва за измерване на трафика, който той създава. Администраторът на сайта определя интервала от време (session timeout), след който, ако няма обмен на данни между потребителя и Web сървъра в рамките на сайта, сесията се прекратява.

Ако Web сайтът съдържа само HTML документи, те по правило не обменят данни помежду си и всяка HTML страница визуализира собствени данни. В интернет приложенията отделните модули по правило обменят данни, като някои от тези данни са актуални само в рамките на сесията. По тази причина съвременните програмни езици за създаване на интернет приложения предоставят на програмиста средства за съхраняване на данни, които са привързани към текущата сесия и са достъпни докато трае сесията.

PHP предоставя за работа със сесии:

1) Уникален идентификатор на сесия.
2) Функции за стартиране и прекратяване на сесия
3) Глобален масив $_SESSION за съхраняване на данни от сесията
4) Функции за операции със сесийните променливи

14.1. Идентификатор на сесия

PHP създава идентификатора на сесията чрез MD5 хеширане на клиентския IP адрес, текущия час и някаква случайна комбинация, представена като символно предстявяне на шестнатдесетично число. Този идентификатор се изпраща към потребителския браузър, който, при заявяване на друг модул от интернет приложението го изпраща обратно към Web сървъра. Идентификаторът на сесията може да се предава като бисквидка или да се добавя към URL адресите при навигация по страниците на интернет приложението.

От съображения за сигурност е по-добре идентификаторът на сесията (session_id) да се предава като бисквидка. За тази цел посредством функцията ini_set трябва да се активира следната настройка

ini_set(“session.use_cookies”,1) ,

или да се промени непосредствено във файла php.ini. Ако сме сигурни, че на потребителския компютър е разрешено записването на бисквидки, можем да активираме настройката

ini_set(“session.use_only_cookies”,1) ,

която означава, че подадените в URL идентификатори на сесията ще бъдат отхвърлени от PHP. По-висока сигурност може да се получи при използване на криптирана връзка (SSL).

Програмистът може да зададе сам идентификатор на сесията посредством функцията session_id( ). Тя има следното описание:

string session_id ([string id])

където незадължителният параметър id задава идентификатор на сесията.

За да се приложи създаденият идентификатор, функцията session_id трябва да се извика преди стартирането на сесията. Ако се извика без параметър, функцията служи само за получаване на текущия идентификатор на сесията.

14.2. Функции за стартиране и прекратяване на сесия

Функцията session_start() стартира PHP сесия или рестартира текущата сесия ако вече е била стартирана. При стартиране на сесията нейният идентификатор (session_id) се изпраща с HTTP отговора от Web сървъра към потребителския браузър. За да може да се изпрати идентификатора на сесията като бисквидка, функцията session_start() трабва да се изпълни преди да е изпратено каквото и да е към потребителя, т.е. трябва да се изпълни преди тага <HTML>.

Ако опцията session_auto_start във файла php.ini е включена, сесията се стартира автоматично при заявяване на PHP модул от приложението.

Функцията session_destroy() унищожава всички данни, асоциирани с текущата сесия и прекратява сесията. Това е полезно да се прави за да се избегне възможността за достъп до лични данни на потребителя преди изтичане на контролното време за автоматично прекратяване на сесията (session timeout).

Функцията session_unset() унищожава само сесийните променливи, но не прекратява сесията. За унищожаване на отделна сесийна променлива програмистът може да използва функцията unset. Например инструкцията unset($_SESSION ['counter']); ще унищожи променливата counter.

Да разгледаме един пример:

```
session_start();
echo "Session id=".session_id( )."<br>";
session_destroy();
echo "Session id=".session_id( )."<br>";
```

Пример 74 Стартиране и прекратяване на сесия

В примера се стартира сесия и се извежда генерирания от PHP идентификатор на сесията. Сесията се прекратява с извикване на функцията session_destroy(), след което се прави отново опит да се изведе идентификатора на сесията с функцията session_id(), но след прекратяването на сесията тя връща празен низ.

14.3. Запис на данни за сесията. Масив $_SESSION

За запис на сесийни променливи, достъпни за всички модули от приложението в рамките на сесията, PHP предоставя на програмиста глобалния масив $_SESSION. Записът на сесийна променлива е просто запис на елемент в този масив. Например инструкцията

$_SESSION[“favourite_color”]=”green”;

записва в масива сесийната променлива “favourite_color” със стойност ”green”.

Важно е да се запомни, че данните за сесията се съхраняват на сървъра, а не на потребителския компютър, и поради това записът им не може да бъде забранен чрез настройка на потребителския браузър, както това може да се направи с бисквидките.

Проверка дали дадена сесийна променлива съществува (т.е. дали е записана в масива $_SESSION) може да се направи с функцията isset , както това се прави за съществуване на елемент от масив. Да разгледаме един пример, с който се отброяват посещенията на страници в рамките на дадена сесия на PHP приложение.

```
if(isset($_SESSION[“visits”]))
{
      $_SESSION[“visits”]++;
}else{
      $_SESSION[“visits”]=1;
}
echo “<br>Брой отваряния на модули ”. $_SESSION[“visits”];
```

Пример 75 Брояч на отварянията на страници в сесия чрез сесийна променлива.

Този код трябва да се включи във всеки PHP модул от интернет приложението. При първото му изпълнение в рамките на сесията ще се създаде сесийната променлива visits и ще и се присвои стойност 1. Всяко следващо извикване на модул от приложението в сесията ще се отброява с увеличаване на стойността на visits с 1.

14.4. Съхраняване на данните от сесията. Функция session_save_path

Функцията session_save_path дава възможност за променяне на директорията, в която се съхранява файла със сесийните променливи за текущата сесия. Тя има следното описание:

string session_save_path([string path])

където path е спецификацията на директорията, в която ще се запише файла със сесийните данни. Очевидно това е директория, в която PHP трябва да има права за запис.

За да има резултат от изпълнението на функцията session_save_path, тя трябва да се изпълни преди стартирането на сесията. Данните за сесията се съхраняват във файл с име, получено от слепването на представката “sess_” и идентификатора на сесията.

Ако се изпълни без параметъра path, функцията само връща пътя до директорията, в която се записва файла със сесийните данни. Можем да изведем пътя до тази директория с инструкцията

```
echo "Директорията за сесийни променливи е ".session_save_path();
```

14.5. Задаване на параметри на сесийните бисквидки. Функция session_set_cookie_params

Функцията задава параметри на сесийната бисквидка, които са дефинирани в конфигурационния файл php.ini. Действието на тази функция е само по време на изпълнението на текущия PHP модул, и поради това, за да са валидни параметрите за цялото приложение, тя трябва да се извиква в началото на всеки модул преди извикването на функцията session_start(). Функцията има следното описание:

void session_set_cookie_params (int lifetime [, string path [, string domain [, bool secure [, bool httponly]]]])

където:

lifetime	е времето за живот на бисквидката в секунди. Подразбиращата се стойност е 0, което означава “докато

	се затвори браузъра.
path	е пътят, с който се асоциира бисквидката. Подразбиращата се стойност е “/” , т.е директорията на PHP приложението
domain	е домейнът, с който се асоциира бисквидката. Подразбиращата се стойност е none, което означава, че за домейн се приема host адреса на Web сървъра, който генерира бисквидката.
secure	Показва дали бисквидката трябва да се изпраща по криптирана връзка. Подразбиращата се стойност е “off”.
httponly	Показва, дали бисквидката ще се предава само по HTTP протокола. Подразбиращата се стойност е FALSE.

Текущите стойности на тези параметри могат да бъдат получени от функцията session_get_cookie_params( ). Тя връща масив, в който са записани стойностите на дадените по-горе параметри с ключове имената на параметрите.

15. Обектно ориентирано програмиране в PHP

PHP е създаден като процедурно ориентиран език, в които основните програмни единици, са функциите. Обектно ориентирани възможности са добавени в PHP още в трета версия, но те са били доста ограничени, Поради стремежа за запазване на съвместимостта между версиите, в PHP4 обектните възможности на езика не са разширени съществено.

В PHP5 обектния модел е изцяло преработен, добавени са нови възможности и е променено поведението на обектите. В настоящия курс ще бъде разглеждан обектния модел на PHP5, като само в някои случаи ще бъде правено сравнение с модела на PHP4.

16. Класове и обекти

Класът е въведен в програмните езици като разширение на типът *структура*. Докато структурата първоначално е можела да съдържа само различни типове данни, класът може да съдържа и функции (методи на класа). От своя страна структурата е разширение на класическия тип масив, който се дефинира като съвкупност от променливи от един и същ тип с общо име и последователни целочислени индекси.

Класът е обобщен тип данни , който включва:

a) Променливи, които се наричат полета (fields), свойства или атрибути (properties) на класа.
b) Функции, които се наричат методи ка класа. Методите изпълняват определени операции с променливите на класа.
c) Манипулатори на събития – специален тип променливи, на които се присвоява адреса на функции, дефинирани от програмиста извън кода на класа. Обектите, създадени чрез класа извикват за изпълнение функцията, присвоена на манипулатор при възникване на определено за манипулатора събитие. Поради това такива функции се наричат събитийни процедури.

Класът е шаблон за създаване на обекти, така, както типът string е шаблон за създаване на символни низове. Декларацията на клас описва какви атрибути и методи ще има обекта, създаден от класа и каква е тяхната област на видимост.

16.1. Декларация на клас

Декларацията на клас започва с ключовата дума class, следва име на класа и блок, в който са декларирани атрибутите и методите на класа с техните модификатори за достъп. Да разгледаме един пример:

```
class Person
{
	private $name; //Атрибут на класа
	function getName()  //Метод на класа
	{
		return $this->name;
	}
```

```
}
```

Пример 76 Декларация на клас с един атрибут и метод

Важно е да се запомни,че в декларацията на PHP клас не може да има два метода с едно и също име, докато в други обектно ориентирани езици (като C++, Java, C#) е допустимо в един клас да се декларират няколко метода (включително и няколко конструктори), които имат едно и също име, но различен набор от параметри.

В PHP методите трябва да бъдат извиквани със същия набор от параметри, с които са декларирани. Допуска се обаче на някои от параметрите на метода да бъдат зададени подразбиращи се стойности, и тогава при обръщение към метода на тези параметри могат да не се задават стойности.

16.2. Създаване на обекти – оператор new и конструктор на класа

Обект от даден клас (инстанция на класа) се създава посредством оператора new , следван от името на класа и кръгли скоби, в които може е зададен списък от параметри). При изпълнение на оператора new се записва в оперативната памет нов обект, с негови собствени атрибути и методи, създадени по декларациите, включени в тялото на класа. Обектът се създава от функцията- конструктор на класа, ако такава е дефинирана. Ако в тялото на класа не е дефинирана функция-конструктор, то PHP генерира и извиква т.н. празен конструктор (без параметри). В случай че класът е наследник на друг клас, при извикването на генерирания празен конструктор PHP извиква конструктора на родителския клас.

Операторът new връща референция (указател*) към създадения обект. Тази референция по правило се присвоява на променлива, която по-нататък ще наричаме обектна променлива. Посредством

* Трябва да се прави разлика между референциите в PHP и указателите, използвани в езика C. Езика C предоставя операции за указателите, докато референциите в PHP дават само възможност за достъп до атрибутите и методите на обекта.

обектната променлива се осъществява достъп до атрибутите и методите на класа.

16.3. Деклариране на конструктор на класа

Конструкторът на класа в PHP5 е метод на класа, който има име __construct и следното описание (по синтаксис на C):

```
void __construct( [mixed arg1, mixed arg2, …])
{
      //class members
}
```

където:

void е ключова дума на езика C, с която се указва, че функцията не връща стойност

[mixed arg1, mixed arg2, …] – (незадължителен) е списък от параметри на функцията - конструктор

class members – членове на класа (атрибути и методи), които се декларират в тялото на класа.

В PHP4 вместо с името __construct конструкторът на класа се е декларирал с име, съвпадащо с името на класа. Тази възможност е запазена в PHP5 за да се осигури обратна съвместимост, но за нови разработки се препоръчва използването на новото унифицирано наименование на конструктора.

Както е посочено по-горе в PHP клас не може да има два метода с едно и също име. Това се отнася и за конструкторите, които са специализирани методи – един PHP клас може да съдържа само една функция-конструктор.

Да разгледаме един пример:

```
class Person
{
      $name; //Атрибут на класа
      function __construct($name)  //Конструктор на класа
      {
            $this->name=$name; //Запис на стойност в атрибута name
      }
}
$John = new Person(“John”); //Създаване на обект от клас Person
```

```
echo $John->name; //Извеждане на стойността на атрибута name
```

Пример 77 Декларация на клас с конструктор и създаване на обект от класа

В примера в декларацията на класа Person е включен конструктор, който има параметър $name, стойността на който се записва в атрибута $name на класа . При създаване на обект от класа, се извиква конструктора с параметър “Peter”, а със следващата инструкция се извежда стойността на атрибута $name на обекта $Peter.

16.4. Деструктор на класа

Деструкторът е метод на класа, който се извиква автоматично (т.е. без да е необходимо програмистът да пише код за това) при унищожаване на обект от класа. Виртуалната машина на PHP се грижи за това, след изпълнението на даден PHP модул да бъдат унищожени всички създадени при изпълнението му ресурси.

Функцията - деструктор на класа в PHP има име __destruct и следното описание (по синтаксис на C):

```
void function __destruct()
{
      //statements
}
```

където:

void е ключова дума на езика C, с която се указва, че функцията не връща стойност

statements са инструкциите в тялото на функцията

Трябва да се има предвид, че точният момент на извикване на деструктора не съвпада с момента на унищожаване на последната обектна променлива на обекта, което показва, че виртуалната машина на PHP използва т.н. “събирач на боклука” (garbage collector), извикван асинхронно, както това се прави например в Java.

16.5. Достъп до атрибути и методи на обект чрез променливата $this

При изпълнението на метод в контекста (т.е. в кода) на обект, автоматично се генерира променлива $this, която съдържа референция (указател) към обекта. Може да се каже, че в кода на класа променливата $this изпълнява ролята на обектна променлива. Посредством нея и оператора “->”се осъществява достъп до атрибутите и методите на класа.

В дадения по-горе пример, в декларацията на класа Person е включен конструктор с параметър $name, чрез който се инициализира атрибутът на класа $name. Това става с инструкцията

```
$this->name=$name; //Запис на стойност в атрибута name
```


Важно е да се запомни, че в PHP след оператора “->” преди името на променлива не се поставя символа “$”.

16.6. Модификатори за достъп до членовете на класа

Една от базовите идеи на обектно ориентираното програмиране (ООП) е капсулирането на данните и защитата на достъпа до атрибутите и методите на обектите. В PHP3 и PHP4 при деклариране на атрибути се използва ключовата дума “var”, като всички атрибути са публични, т.е достъпни както в кода на класа, така и извън него. Методите на класовете в PHP3 и PHP4 се декларират без модификатор за достъп и също са винаги публични.

В PHP5 са въведени модификатори за достъп, аналогични на използваните в C++ и Java:

- public – указва, че членът на класа е достъпен както в кода на класа, така и извън него.
- protected – указва, че членът на класа е достъпен в кода на класа, в родителските класове и класовете-наследници (ако има такива).
- private – указва, че членът на класа е достъпен само в кода на класа.
- final – указва, че членът на класа не може да се наследява

- var – поддържа се за съвместимост със старите версии, еквивалентен на public

Да разгледаме една модификация на предния пример, в която атрибутът $name е деклариран с модификатор protected.

```
class Person
{
	protected $name; //Атрибут на класа
	function __construct($name)  //Конструктор на класа
	{
		$this->name=$name; //Запис на стойност в атрибута name
	}
}
$John = new Person("John"); //Създаване на обект от клас Person
echo $John->name; //Извеждане на стойността на атрибута name
```

Пример 78 Декларация на клас с атрибут с модификатор protected

В тоя случай при опит за достъп до атрибута с модификатор protected извън кода на класа виртуалната машина на PHP генерира грешка "Cannot access protected property Person::$name ... "

16.7. Статични атрибути и методи на класа. Оператор ::

Статичните атрибути и методи в обектния модел на PHP (както и в C++) са атрибути и методи на самия клас. Те се декларират с ключовата дума static, преди която може да се постави модификатор за достъп. Както и за не-статичните членове на класа, ако модификатор за достъп липсва, това е еквивалентно на модификатор public.

Статичните атрибути и методи се създават по време на компилирането на декларацията на класа, а не по време на създаването на обект от класа. Също така, достъпът до тях извън кода на класа се осъществява непосредствено чрез името на класа, а не чрез обектна променлива.

В кода на класа и в кода на родителски класове и класове-наследници, достъпът до статични атрибути и методи на класа се осъществява с ключовата дума self , следвана от оператора за принадлежност "::" (scope resolution operator).

Да разгледаме един пример

```
class Person
{
	public static $meaning=”Personal info”;
	// Статичен метод за извеждане на стойността на $meaning
	public static function show_meaning()
	{
		echo “Person meaning = ”.self:: $meaning.”<br>”;
	}
} // край на декларацията на класа
Person:: show_meaning(); //Извикване на статичния метод на класа
```

Пример 79 Достъп до статични атрибути и методи с оператора “::”

16.8. Деклариране на константи в класовете

PHP дава възможност за деклариране на константи в класовете с ключовата дума const по следния синтаксис:

const constant_name = constant_value;

Стойността, която се присвоява на декларираната константа може да бъде само константа. Не се допуска присвояване на константата на стойност на променлива или израз.

Достъпът до константите на класовете става чрез името на класа и оператора за принадлежност “::” , следван от името на константата, т.е. по същия синтаксис, както достъпа до статичните членове на класа. В кода на класа достъп до константа на класа се извършва посредством ключовата дума self – синоним на текущия клас. Да разгледаме един пример:

```
class MyClass
{
	const CLASS_NAME=__CLASS__;
	function show_class_name()
	{
		echo self:: CLASS_NAME.”<br>”;
	}
} //край на класа
```

```
MyClass:: show_class_name(); //Извикване на статичния метод
echo “Class Name=”. MyClass:: CLASS_NAME
```

Пример 80 Достъп до константи на класа с оператора “::”

В декларацията на класа MyClass е декларирана константа CLASS_NAME, която получава стойността на “магическата” константа __CLASS__ , която съдържа името на текущия клас. Достъпът до константата CLASS_NAME в кода на класа се осъществява с ключовата дума self, а извън кода на класа – чрез името на класа, и в двата случая следвани от оператора за принадлежност “::” и името на константата.

16.9. Клониране на обекти

В PHP5 операторът new, с който се създава обект чрез конструктора на класа, връща указател към обекта, който се присвоява на променлива. Чрез тази обектна променлива може да се осъществява достъп до членовете на класа (атрибути и методи). Поради това ако стойността на обектната променлива се присвои на друга променлива, и двете променливи ще сочат към един и същ обект.

Да разгледаме един пример:

```
class Person
{
      public $name=”Peter”
}
$a=new Person();
$b=$a;
$b -> name=”John”;
echo s->name;   //извежда се John
```

Пример 81 Присвояване на стойност на обектна променлива

Тъй като обектната променлива $b сочи към същия обект, към който сочи $a, то при променянето на атрибута $name чрез $b чрез $a се прочита променената стойност.

Ако примерът се изпълни от PHP4, ще се изведе “Peter”, защото в PHP4 присвояването `$b = $a` създава копие на обекта, което се

присвоява на $b. Поради това променянето на атрибута на обекта, който се представя от $b не променя атибута на обекта, който се представя от $a.

В PHP5 инструкцията `$b = $a` присвоява на $b адреса на обекта, който се съдържа в $a и следователно след изпълнението на тази инструкция двете променливи сочат към един и същ обект.

Така, че, въпреки че по-нататък ще наричаме променливата в PHP5, чрез която са достъпни атрибутите и методите на обект "обектна променлива", тя всъщност представлява указател към обекта. За разлика от C++ обаче, с този указател могат да се извършват само две операции – присвояване чрез оператор "=" и достъп до член на класа чрез оператор "->". При определени условия (ще бъде разгледано по-нататък) тази обектна променлива може да се преобразува в символен низ с оператора (string).

В PHP5 клонирането на обект, т.е. създаването на пълно копие на обект, с което се копират и стойностите на атрибутите на обекта, се извършва с оператора clone по следния синтаксис:

$cloning_of_object = **clone** $object;

В някои случаи е необходимо след клонирането да променим автоматично (т.е. без допълнителни инструкции) стойностите на някои атрибути на обекта-клонинг. За тази цел PHP предоставя специализирания метод __clone(). Ако той се включи в декларацията на класа, методът се извиква автоматично за изпълнение в клонирания обект след неговото създаване. Да разгледаме един пример:

```
class Person
{
        public $name="Peter";
        public $clone_number=0;
        function __clone( )
        {
                $this -> clone_number++; //Отброяване на клонирането
        }
} //end of class
$a = new Person( );
echo "a clone_number =".$a-> clone_number; //номер на оригинала
```

```
$b = clone $a;  //Клониране на $a
echo “<br>b clone_number =”.$b-> clone_number; //номер на
                                                    //клонинга
echo “<br>b clone name =”.$b-> name;                //име на клонинга
```

Пример 82 Клониране на обект

В примера при създаване на обект се инициализират атрибутите $name и $clone_number. При създаване на клонирания обект $b се извиква автоматично метода __clone(), който увеличава с единица $clone_number. Затова при извеждането на атрибутите на клонирания обект $b атрибутът $name се оказва със същата стойност, но $clone_number е със стойност 1.

16.10. Наследяване

Наследяването, заедно с капсулирането и *полиморфизма, е един от трите фундаментални механизми на обектно ориентираното програмиране. Наследяването дава възможност на общите за дадена група класове атрибути и методи да се декларират в един родителски клас и да се наследят от всички класове, които се нуждаят от същата функционалност. Това не само намалява обема на кода на програмите, но и създава ясна логическа връзка между класовете. Така например за програмиста на Java е достатъчно да знае, че базовият клас Object притежава метода toString(), за да бъде сигурен, че всеки от дефинираните в езика класове притежава този метод.

В PHP наследяването се дефинира с ключовата дума extends. Да разгледаме един пример:

```
class Person
{
      protected $egn;
      public $name;
      public function getID()
      {
            return $this-> egn;
```

* В програмирането полиморфизъм (многообразие) най-общо е способността на типа A да се появява и да се използва като тип B

```
        }
        public function getName( )
        {
                return $this->name;
        }
} //край на класа Person
class Student extends Person  //класът Student наследява Person
{
        protected $fnom;
        function __construct($fnom)  //Конструктор на класа Student
        {
                $this->fnom=$fnom; // Инициализация на атрибута $fnom
        }
        public function getID( )
        {
                return $this-> fnom;
        }
} // край на класа Student
$John = new Student("062034"); //Създаване на обект от клас Student
$John->name="John"; //Достъп до атрибут на родителския клас
echo "<br>".$John->getName( ); // Извикване на метод на род. клас
echo "<br>".$John->getID( ); // Извикване на предефиниран метод
```

Пример 83 Наследяване на клас в PHP

В примера се декларира клас Person и клас Student – наследник на класа Person. Създава се обект $John от клас Student. Чрез обект от класа Student се осъществява достъп до атрибута $name и метода getName( ) на родителския клас Person, тъй като те не са предефинирани в класа Student. При извикване на метода getID( ) обаче се извиква метода на класа Student, тъй като той предефинира метода на класа Person.

В PHP не се разрешава множествено наследяване. Един клас може да наследява само един родителски клас.

Атрибутите и методите на родителския клас, декларирани с модификатори public и protected са достъпни в класовете-наследници. В класа- наследник тези атрибути и методи могат да бъдат надписвани, т.е. да бъдат декларирани отново със същите

имена. В дадения пример в класа Student, наследник на Person, е предефиниран метода getID( ), така, че ако обект от класа Student извика този метод, ще се изпълни метода, деклариран в класа Student. Ако обаче чрез обект от класа Student извършим операция с атрибута $name, това ще бъде атрибута $name на родителския клас, тъй като в класа Student той не е предефиниран.

Важно е да се отбележи също така, че достъп до атрибути и методи на класа-наследник или на родителския клас извън кода на класа е възможен само ако те са декларирани с модификатор public.

Синонимът parent

Запазената дума parent в PHP е синоним на родителския клас в кода на класа-наследник. В кода на методите на класа-наследник може посредством parent и оператора за принадлежност "::" да се извикват методи на родителския клас. Това, очевидно има смисъл да се прави само за тези методи, които са предефинирани в класа-наследник. Ако метод на родителския клас не е предефиниран, той може да бъде извикан в кода на класа-наследник само с името си. Да разгледаме един пример:

```
class Person
{
	public $name;
	public function __construct($name)
	{
		$this-> name=$name;
	}
	public function getName( )
	{
		return $this->name;
	}
} //край на класа Person
class Student extends Person  //класът Student наследява Person
{
	protected $fnom;
	function __construct($name. $fnom)
	{
```

```
            $this->fnom=$fnom;
            parent:: __construct($name); //
        }
        public function getFnom( )
        {
            return $this-> fnom;
        }
} // край на класа Student
$John = new Student("John ","062034"); //Създаване на обект
                                        // от клас Student
echo "<br>Name=".$John->getName( );  // Извикване на метод на
                                        // на родителския клас
echo "<br>Fnom=".$John-> getFnom( ); // Извикване на метод
                                        // на класа-наследник
```

Пример 84 Достъп до метод на родителския клас чрез синонима parent

В примера в класа-наследник Student е дефиниран конструктор, с който се инициализират атрибутите на класа и неговия родителски клас. Атрибутът $fnom се инициализира непосредствено, докато за инициализирането на атрибута $name се извиква конструктора на родителския клас. Тъй като обаче в класа-наследник също има метод с име __construct, обръщението към конструктора на родителския клас става чрез синонима parent и оператора “::” .

Синонимът parent може да се използва също така за достъп до статични променливи и методи на родителския клас. При достъп до променливи преди името на променливата се поставя символа $ (както и при използване на синонима на текущия клас self).

16.11. Final методи и класове

В PHP5 е въведена ключовата дума final, която, включена в декларацията на метод, забранява предефинирането на метода в класове-наследници. При опит за предефиниране се генерира грешка от тип “Fatal error”. Ако целият клас бъде деклариран като final, той не може да бъде наследяван.

Да разгледаме един пример:

```
class Person
{
	protected $name;
	final public function getName()
	{
		return $this->name;
	}
}
class Student extends Person
{
	public function getName()
	{
		return parent:: getName();
	}
}
```

Пример 85 Деклариране на Final метод

В примера методът getName( ) на класа Person е деклариран с ключовата дума Final. Тъй като в класа-наследник Student се прави опит той да бъде предефиниран, компилаторът на PHP генерира грешка “PHP Fatal error: Cannot override final method Person:: getName() “

16.12. Преобразуване на обект в символен низ – метод __toString( )

По подразбиране обектите нямат символно представяне. Ако например създадем обект от класа Person, деклариран в предния пример и се опитаме да го преобразуваме в символен низ със следния код

```
$Peter = new Person();
$t = (string) $Peter;
```

се генерира грешка “PHP Catchable fatal error: Object of class Person could not be converted to string”.

PHP дава възможност за получаване на символно представяне на клас посредством един специален метод – методът __toString( ). Ако този метод се включи в декларацията на класа, то той се извиква автоматично при опит да се преобразува обекта в символен низ. Ако класът съдържа метода __toString( ) то при преобразуване на обект от класа в символен низ се получава това, което методът връща като резултат. Да разгледаме един пример.

```
class Person
{
        public $name;
        public function __construct($name)
        {
                $this-> name=$name;
        }
        function __toString( )
        {
                return “Class is “.__CLASS__;
        }
} //край на класа Person
$Peter = new Person(“Peter”);
echo $Peter;
```

Пример 86 Преобразуване на обект в символен низ

В примера при изпълнение на инструкцията `echo $Peter;` се прави преобразуване на $Peter в символен низ. В резултат на това се извиква автоматично метода __toString( ) , и низа “Class is Person“, който той връща, се получава като резултат от преобразуването на обекта $Peter в символен низ и се извежда от конструкцията echo.

16.13. Извеждане съдържанието на обект – функция print_r

Функция print_r , която се използва за извеждане в четим (readable) вид на съдържанието на масиви, може да се използва и за извеждане на съдържанието на обекти. Например, ако декларираме клас Person и създадем обект от този клас,

```
class Person
{
	public $name;
	private $egn;
	public function __construct($name, $egn)
	{
		$this-> name=$name;
		$this-> egn =$ egn;
	}
} //край на класа Person
$Peter = new Person(“Peter”,”6212191105”);
echo “<pre>”. print_r($Peter, TRUE). “</pre>”;
```

Пример 87 Извеждане на съдържанието на атрибутите на клас с функция print_r

Когато на функцията print_r се зададе като втори параметър логическата константа TRUE, както е в примера, тя връща символен низ, вместо да го изпраща непосредствено към потребителския браузър. При изпълнението на примера в прозореца на браузъра ще се изведе

```
Person Object
(
   [name] => Peter
   [egn:private] => 6212191105
)
```

Обърнете внимание на това, че се извежда съдържанието на всички атрибути на класа, в това число и на атрибута egn с модификатор private, който е видим само в кода на класа. Това прави функцията print_r много удобна за извеждане на текущото състояние на обект.

16.14. Преобразуване на променлива в обект

Преобразуването на променлива в обект става с оператора за преобразуване на тип (object), както е показано в следния пример:

```
$s=“Hello”;
```

```
$obj = (object) $s;
echo “<pre>”. print_r($obj, TRUE). “</pre>”;
```

Пример 88 Преобразуване на променлива в обект

Когато се преобразува променлива в обект се създава обект от вградения в PHP клас stdClass. В горния пример, когато преобразуваме променлива, която съдържа символния низ “Hello” в клас ще получим

```
stdClass Object
(
   [scalar] => Hello
)
```

Както се вижда от резултата, полученият при преобразуването обект от клас stdClass има един атрибут с име scalar и със стойност – стойността на преобразуваната променлива.

При преобразуване на масив в обект, елементите на масива се преобразуват в атрибути на обекта, като ключовете на елементите стават имена на атрибутите, а стойностите им – съответно стойности на атрибутите. За да може да се извършва непосредствено обръщение към атрибутите на получения масив е необходимо ключовете на изходния масив да са символни низове, тъй като синтаксисът на PHP не допуска имената на променливи (в т.ч. атрибути на обект) да са числа. Да разгледаме един пример:

```
$a=array(“a”=>”apple”, “b”=>”banana”, “orange”);
$obj = (object) $a;
echo “<pre>”. print_r($obj, TRUE). “</pre>”;
```

Пример 89 Преобразуване на масив в обект

Резултата от преобразуването е:

```
stdClass Object
(
   [a] => apple
   [b] => banana
   [0] => orange
)
```

Очевидно към атрибутите $a и $b на получения обект можем да осъществим достъп съответно с `$obj->a` и `$obj->b` , но опитът да

направим същото за атрибута $0 (получен от ключа 0 на третия елемент от масива) ще означава синтактична грешка. Достъп до този атрибут все пак може да се осъществи, но само при итерация на атрибутите на обекта с конструкция foreach.

16.15. Итерация на атрибутите на обект

Конструкцията foreach дава възможност освен итерация на елементите на масив да се извършва итерация на достъпните (в дадения контекст) атрибути на обект. Да разгледаме един пример

```
class Person
{
	public $name;
	private $egn;
	public function __construct($name, $egn)
	{
		$this-> name=$name;
		$this-> egn =$egn;
	}
	function showProperties( )
	{
		$t="<OL>";
		foreach($this as $key=>$val){ $t.="<LI>$key =
$val</LI>"; }
		echo $t."</OL>";
	}
} //създаване на обект от клас Person
$Peter = new Person("Peter", "0620301105");
$Peter -> showProperties( );
$t="<OL>";
foreach($Peter as $key=>$val){ $t.="<LI>$key = $val</LI>"; }
echo $t."</OL>";
```

Пример 90 Итерация на атрибутите на обект с конструкция foreach

В примера съдържанието на всички атрибути на обекта се извежда в номериран HTML списък чрез метода showProperties( ) , а чрез итерация на обектната променлива $Peter се извежда

съдържанието само на достъпният извън кода на класа атрибут $name.

Конструкцията foreach дава възможност за променяне съдържанието на атрибутите на обекта, така, както при итерация на масиви може да се променя съдържанието на елементите на масив. За това е необходимо в конструкцията foreach преди името на променливата, в която се получава стойността на атрибута да се постави символа “&”. Тогава се осъществи достъп до атрибутите на обект (а не до техни копия, като е по подразбиране).

16.16. Автоматично вмъкване на файлове с декларации на класове – функция __autoload

PHP предоставя на разработчиците, които използват обектно-ориентирано програмиране, функция __autoload, която може да вмъкне автоматично файл с декларацията на клас, ако тя отсъства в PHP модула, в който трябва да се създаде обект от този клас.

Функцията __autoload се извиква автоматично (т.е. без да е нужно специално обръщение към нея) когато при компилиране на кода на PHP модула не бъде открита декларация на клас, от който трябва да се създаде обект. При извикване на __autoload тя получава като параметър името на класа, чиято декларация не е открита. От името на класа в кода на функцията се получава спецификацията на файла, който съдържа декларацията на класа и този файл се вмъква в PHP модула с конструкция require_once.

Могат да се посочат следните предимства от използването на функцията __autoload:

1) По името на класа е лесно да се открие името на файла, който съдържа неговата декларация.
2) В PHP модула се включват само декларациите на класовете, които са необходими.

Да разгледаме един пример:

```
function __autoload($class_name) {
   require_once “classes/$class_name.php”;
}
```

```
$obj  = new MyClass1();
```

Пример 91 Използване на функция __autoload за автоматично зареждане на декларация на клас

В примера в кода на функцията __autoload от името на класа се получава параметър на конструкцията за вмъкване на файл. Файловете с декларации на класове трябва да са записани в поддиректория classes на директорията, в която е PHP модула, да имат име, съвпадащо с името на класа и разширение .php. Така за класа MyClass1 се търси за вмъкване файл с относителна спецификация “classes MyClass1.php”.

16.17. Достъп до недекларирани атрибути и методи на класове – методи __get(), __set() и __call()

Методите __get(),__set() и __call() в документацията на PHP могат да бъдат открити под името “Магически методи”. С тяхното използване може да се заобиколи едно от съществените ограничения на обектния модел на PHP – това, че в PHP клас не може да се декларират два метода с едно и също име, но с различен набор от параметри (както това е възможно в C++ , Java и други обектно ориентирани езици).

Методите __get(), и __set() дават възможност за четене и запис на стойности на атрибути на клас, които не са достъпни в даден контекст или изобщо не са декларирани в класа. Те имат следния синтаксис:

```
mixed __get ( string $name )
{
  //statements
}
void __set ( string $name , mixed $value )
{
  //statements
}
```

където:

$name е параметър, в който се получава името на недекларирания атрибут, до който се осъществява достъп

$value е параметър,в който се получава стойността за запис в недекларирания атрибут

Да разгледаме един пример:

```
class MyClass
{
	protected $x=array(“a”=>”apple”, “b”=>”banana”);
	//Прочитане на недекларирaн атрибут
	function __get($name)
	{
		return $this->x[$name]; //четене от елемент на масива $x
	}
	//Запис в недекларирaн атрибут
	function __set($name, $value)
	{
		$this->x[$name] = $value; //запис в елемент на масива $x
	}
}
$obj = new MyClass( );
$obj->p=”Pineaple”;
echo “<br>”.$obj->a;
echo “<br>”.$obj->b;
echo “<br>”.$obj->p;
```

Пример 92 Достъп до недекларирани атрибути и методи

В примера се създава обект от класа MyClass, в който са включени методите __get( ), и __set( ). При опит за запис в недекларирaн атрибут $p се извиква автоматично метода __set( ), който записва стойността в масива $x, като използва името на недекларирания атрибут като ключ на елемента от масива. Така с инструкцията `$obj->p=”Pineaple”;`, чрез кода в метода __set( ) всъщност се добавя елемент в масива $x.

При четене на стойността на недекларираните атрибути $a, $b и $p се извиква метода __get( ) , който прочита и връща стойността на елемент от масива $x, като използва името на недекларирания атрибут като ключ на елемента от масива.

Извикване на недекларирaн метод

За извикване на недекларирaн метод PHP предоставя “магическия” метод __call( ). Той има следното описание:

mixed __call (string $name, array $arguments)

където:

$name е параметър, в който се получава името на недекларирания метод

$arguments е масив, в който се съдържат параметрите, с които е извикан недекларираният метод

Да разгледаме един пример:

```
class MyClass
{
        private function __call($name, $parameters)
        {
           echo “Method $name called<br>”;
           echo "parameters<pre>".print_r($parameters, TRUE)."</pre>";
        }
}
$obj = new MyClass( );
$obj->MyMethod(“Hello”,7); //извикванена недекларирaн метод
```

Пример 93 Извикване на недекларирaн метод

В примера при извикване на недекларирания метод MyMethod на обект от класа MyClass PHP извиква “магическия” метод call, който получава като първи параметър $name името на извикания метод, а като втори параметър – масива $parameters, съдържащ параметрите на извикания метод. Методът __call( ) извежда стойностите на тези два параметъра, и като резултат в прозореца на браузъра се извежда:

```
Method MyMethod called
parameters
Array
(
    [0] => Hello
    [1] => 7
)
```

Така, че ако включим “магическия” метод __call() в декларацията на PHP клас, при извикване на метод на обект от класа никога няма да получим грешка “Call to undefined method”. Какво ще получим като резултат от извикването на недекларирaн метод зависи само от кода, включен в метода __call().

17. Абстрактни методи и класове. Интерфейси

Когато се организира йерархия от класове, които трябва да притежават един и същ интерфейс (т.е. набор от достъпни атрибути и методи), е необходимо да се използва механизъм, който да задължава всеки от класовете да поддържа определени методи. Обектния модел на PHP (както и на повечето обектно ориентирани програмни езици) предоставя две такива възможности: наследяване на абстрактни класове и реализиране на интерфейси.

17.1. Абстрактни методи и класове

Абстрактният метод е метод, който се декларира само с име и набор от формални параметри без тяло на функцията.

Абстрактният клас е клас, който съдържа декларация на поне един абстрактен метод. Декларацията на такъв клас трябва да започва с ключовата дума abstract.

Важно е да се запомни, че от абстрактен клас не могат да се създават обекти. Ползата от абстрактният клас е само тази, че той задължава всички негови класове-наследници, които не са абстрактни, да реализират неговите абстрактни методи.

Да разгледаме един пример:

```
abstract class Shape
{
      abstract function surface();
}
class Rectangle extends Shape
{
      public $a;
      public $b;
```

```
        function __construct($a,$b)
        {
                $this->a=$a; $this->b=$b;
        }
        function surface()
        {
                return ($this->a)* ($this->b);
        }
}
class Circle extends Shape
{
        public $r;
        function __construct($r)
        {
                $this->r=$r;
        }
        function surface()
        {
                return pi( ) *($this->r)*($this->r);
        }
}
$R1 = new Rectangle(3,4);
echo "<br>Sr=". $R1-> surface( ); //Извежда лицето на правоъгълника
$C1=new Circle(5);
echo "<br>Sc=". $C1-> surface( ); // Извежда лицето на кръга
```

Пример 94 Използване на абстрактен клас

В примера в абстрактния клас Shape е дефиниран абстрактния метод surface( ). В класовете-наследници Rectangle и Circle методът surface е реализиран и връща лицето на съответната фигура.

17.2. Интерфейси

Наименованието interface произхожда от латинските думи inter – между и face – лице. Интерфейсът в най-общия случай определя алгорътъма, апаратните и програмни средства за взаимодействие между класове или компоненти.

В програмирането, интерфейсът, с който един обект взаимодейства с останалата част от програмата, се състои от атрибутите и методите на обекта, които са достъпни извън него. Когато програмистът използва обекти от даден клас, е необходимо той да знае, дали обектите притежават определени атрибути и методи. Основната програмна конструкция, която използва ООП за да задължи даден клас да притежава определен набор от методи се нарича интерфейс или интерфейсен клас. Интерфейсът в ООП (и в частност в PHP) е абстрактен клас, който може да съдържа само константи и абстрактни методи. В PHP той се декларира с ключовата дума interface. Ако в предния пример ако вместо абстрактния клас Shape декларираме интерфейс Shape, декларацията ще има вида:

```
interface Shape
{
	public function surface();
}
```

Пример 95 Деклариране на интерфейс

Ако един клас наследява интерфейсен клас и реализира абстрактните методи, които са декларирани в него, се казва, че класа реализира интерфейса. В декларацията на класа това се указва с ключовата дума implements, след която следва списък от имената на интерфейсите, които класът реализира. В PHP (както и в Java, C# и други съвременни езици за програмиране) един клас може да наследява само един родителски клас, но може да реализира няколко интерфейса. Да разгледаме пример за декларация на клас Rectangle, който реализира интерфейса Shape.

```
class Rectangle implements Shape
{
	public $a;
	public $b;
	function __construct($a,$b)
	{
		$this->a=$a; $this->b=$b;
	}
	function surface()
```

```
    {
        return ($this->a)* ($this->b);
    }
}
```

Пример 96 Декларация на клас, който реализира интерфейса Shape

Наследяване на интерфейси

Един интерфейсен клас може да наследява други интерфейсни класове. При това има две различия между наследяването на класове и наследяването на интерфейси:

3) Интерфейсът може да наследява повече от един интерфейсен клас (докато класът може да наследява само един клас). В декларацията на интерфейсния клас това се указва с ключовата дума extends, следвана от списък с имената на интерфейсните класове, които се наследяват.
4) При наследяване не се допуска дублирането на абстрактни методи в родителските интерфейсни класове.
5) В интерфейса-наследник не се разрешава повторно дефиниране на абстрактен метод, който вече е дефиниран в някой от родителските интерфейси.

18. Обработка на изключения. Класът Exception

При изпълнението на кода на PHP могат да възникнат грешки по различни причини. За програмата генерирането на грешка е събитие, което е прието да се нарича изключение (exception). Някои от изключенията са предвидени от разработчиците на PHP и за тях се извеждат съобщения и (в зависимост от вида на грешката) се прекратява или продължава изпълнението на кода на PHP модула. Пример за изключение, за което PHP извежда съобщение е опитът за отваряне на несъществуващ файл с функция fopen(). В тоя случай функцията извежда съобщението "*failed to open stream: No such file or directory*" и връща стойност FALSE. Генерираното

изключение обаче е от тип Warning (предупреждение) и изпълнението на кода на PHP модула продължава.

PHP5 дава възможност на програмиста да "прихване" генерираното изключение, да прекрати нормалното изпълнение на кода на програмата и да изпълни специално подготвен код за обработка на изключението. Моделът на PHP5 за обработка на изключения е аналогичен на този, използван в повечето от съвременните езици за програмиране – структурирана обработка посредством структурата try-catch. Тя има следния синтаксис:

```
try {
 //Код, който се проверява за изключения

} catch (Exception $e) {
      // Код, който се изпълнява при генерирано изключене
}
} finally {
      // Код, който се изпълнява винаги (независимо
      // дали е генерирано изключение)
}
```

Ако в блока try се генерира изключение, се прекратява изпълнението на кода и се изпълнява кода в блока catch. След всеки try блок трябва да има поне един catch блок, но за прихващане на различни типове изключения след try могат да се включат няколко различни catch блока. Във всеки catch блок се прихваща изключение от типа, който е зададен като параметър на catch. Ако обработваме всички изключения само с един блок catch, му се предава параметър от базовия клас за изключения Exception (както е показано в дадената по-горе синтаксис на структурата try-catch).

!В PHP5 блока finally се поддържа от версия 5.5.0.

Ако се обработват няколко типа изключения, първо трябва да се обработят специализираните изключения и последно – всички останали чрез базовия клас Exception, както е показано в примера за оператора throw.

18.1. Класът Exception

Класът Exception е вграден в PHP5. Той има следната декларация:

```
class Exception
{
   protected $message = 'Unknown exception';   // exception message
   protected $code = 0;                        // user defined exception code
   protected $file;                            // source filename of exception
   protected $line;                            // source line of exception

   function __construct($message = null, $code = 0);

   final function getMessage();                // message of exception
   final function getCode();                   // code of exception
   final function getFile();                   // source filename
   final function getLine();                   // source line
   final function getTrace();                  // an array of the backtrace()
   final function getTraceAsString();          // formated string of trace

   /* Overrideable */
   function __toString();                      // formated string for display
}
```

Декларация на вградения клас Exception

Както се вижда от декларацията, конструкторът на класа има два параметъра - $message и $code, като и двата параметъра имат подразбиращи се стойности (и следователно обект от клас Exception може да се създаде и чрез конструктор без зададени параметри. Видимите извън класа get методи връщат информация за генерираното изключение – съобщение, код, номер на реда, име на файла, информация от трасиране на стековата памет.

Ако някакъв клас наследява класа Exception, той може да предефинира само конструктора на класа и метода __toString(). Предефинирането на този метод дава възможност да се изведе допълнителна информация за изключението чрез код, включен в предефинирания метод.

Препоръчва се ако се предефинира конструктора на класа, да се извика в него конструктора на родителския клас Exception.

18.2. Генериране на собствено изключение – оператор throw

Операторът throw предоставя на програмиста възможност да генерира собствено изключение тогава, когато при определено условие иска да прекрати нормалното изпълнение на кода на PHP модула. Операторът throw има следното описание:

```
throw new Exception([ string exception message, [int exception number]]);
```

Операторът throw трябва да получи като операнд обект от клас Exception, който се създава с оператора new и конструктора на класа Exception (или негов наследник). По правило на конструктора на класа Exception се задава като параметър символен низ с текст на съобщението за грешка, но може да се зададе и втори параметър – номер на грешката. Да разгледаме един пример:

```
// Деклариране на клас, наследник на Exception
class MyException extends Exception
{
   // Предефиниране на конструктора със задължителен първи
   // параметър
   public function __construct($message, $code = 0) {
      // извикване на конструктора на родителския клас
      parent::__construct($message, $code);
   }

   // Извеждане на съобщение на грешката
   public function __toString() {
      return __CLASS__ . ": [{$this->code}]: {$this->message}\n";
   }
}
// Код, в който се генерира и прихваща изключението
```

```
try
{
        // код, който с проверява за изключение
        // .....................................................................
        // Генериране на собствено изключение
        throw new MyException(“Thrown generated Exception”);
}
catch( MyException $e ){ echo “First Caught is ”.$e; }
catch( Exception $e ){ echo “Last Caught is ”.$e; }
```

Пример 97 Генериране и обработване на изключение в PHP5

В примера се декларира клас – наследник на класа Ecxeption, в който се предефинират конструктора на класа и метода __toString(). В блока try на структурата try-catch е включен оператор throw за генериране на изключение, който получава като параметър обект от клас MyException. Структурата съдържа два блока catch, като с първия се прихваща изключение от клас MyException, а с втория – останалите изключения. В случая ще се прихване изключение MyException и ще се изведе съобщение

19. Операции с бази от данни

Увод

В много случаи при създаване на Интернет приложения се налага продължително съхраняване на информация с възможност за нейното извличане и актуализиране от потребителите на сайта. Въпреки че по принцип е възможно да се съхранява такава информация и в текстови файлове, за предпочитане е да се работи с бази от данни, за които програмните среди предлагат добре развити средства за проектиране.
Интернет приложението, което извършва операции с база от данни по правило се реализира като система клиент-сървър на две нива, като клиента е PHP приложението , а като сървър се използува сървър за база от данни, който е по правило е инсталиран на компютъра, изпълняващ функциите на WEB сървър.

20. Модел на релационните бази от данни

Най-разпространеният модел за организация на база от данни е релационния модел. За пръв път той е формулиран от д-р E. F. Codd през 1970г. Д-р Код формулира известните дванадесет правила за този релационен модел, (който всъщност за тринадесет, тъй като номерирането им започва от 0 и завършва на 12). Сърцевината на релационният модел е вложена най-вече в първите три правила:

0) Релационните системи за управление на бази от данни (СУБД) трябва да могат да управляват напълно БД чрез техните релационни възможности.
1) Правило за информацията: Цялата информация за дадена релационна БД, включително имената на таблиците и колоните, се представя явно като стойности в таблиците.
2) Гарантиран достъп: Всяка стойност в РБД задължително трябва да бъде достъпна чрез комбинация от име на таблицата, стойност на първичен ключ и име на колона.

При релационния модел данните са разделени в множества, които имат структурата на таблица. Релационната таблица се състои от отделни елементи с данни, наречени полета. Едно множество от полета образува запис в таблицата.

21. SQL

SQL (Structured Query Language) – език за структурирани заявки, е разработен в края на 70-те години от IBM за техния продукт DB2. SQL се е утвърдил като стандартен език за команди към СУБД, който се поддържа на практика от всички сървъри за бази от данни. Посредством SQL програмистът на бази от данни може да извършва:

- Извличане на информация от БД.
- Актуализиране на съдържанието на БД
- Променяне на структурата на БД
- Променяне на системните настройки за сигурността
- Създаване на потребители и задаване на права на потребителите

Ще разгледаме основните SQL команди, тъй като те са необходими като параметри на функциите, които PHP предоставя за операции с БД. Обемът на курса не дава възможност за разглеждане на всички възможности на езика, затова са разгледани SQL командите са основните операции с БД а в следващите теми са дадени примери за използването им.

21.1. Оператори, изрази и условия на SQL

SQL не е език за програмиране с общо предназначение, както например C, Java или PHP. Въпреки това той притежава основните елементи на езиците за програмиране: константи, променливи, изрази, конструкции за управление, пълен набор от оператори за основните операции , както и набор от вградени функции с различно предназначение. Настоящия курс е ориентиран към разработването на PHP приложения, което не дава възможност за

пълно разглеждане на възможностите на SQL. В тази глава се разглеждат основните елементи на SQL във връзка с използването на SQL команди при операции с бази от данни в PHP приложенията.

Оператори за сравнение

Операторите за сравнение са най-често използваните оператори в SQL, тъй като те участват при изчисляването на условия, които се включват в повечето SQL команди. Операциите за сравнение могат да се изпълняват за различен тип операнди, но винаги връщат логически резултат TRUE или FALSE. Левият операнд на операторите за сравнение по правило е единична стойност, получена в общия случай от изчисляването на израз. Десният операнд освен единична стойност може да бъде списък, интервал или шаблон. Дадената по-долу таблица съдържа операторите за сравнение на SQL.

=	Равно
!= или <>	Различно
>	По голямо
>=	По голямо или равно
<	По малко
<=	По малко или равно
in	Равно на елемент от списъка
not in	Различно от всички елементи от списъка
between	Между две зададени стойности
not between	Извън интервала от две зададени стойности
begins with	Започва със зададена стойност
contains	Съдържа зададена стойност
not contains	Не съдържа зададена стойност
is null	Незададена стойност
is not null	Зададена стойност
like	Като зададена стойност, Символът. “%” в шаблона дава съвпадение при коя да е последователност от символи. Символът “_”дава съвпадение за един

	произволен символ на дадената позиция в низа.
not like	Не е като зададен шаблон. Значението на символите “%” и “_” е същото като за оператора like.

Таблица 3 Оператори за сравнение на SQL

Аритметични оператори

SQL предоставя оператори за основните аритметични операции.

Operator	**Description**
+	Сумиране
-	Изваждане
*	Умножение
/	Делене
%	Модул (остатък от целочислено делене)
-	(при един операнд) Променяне на знака на стойността

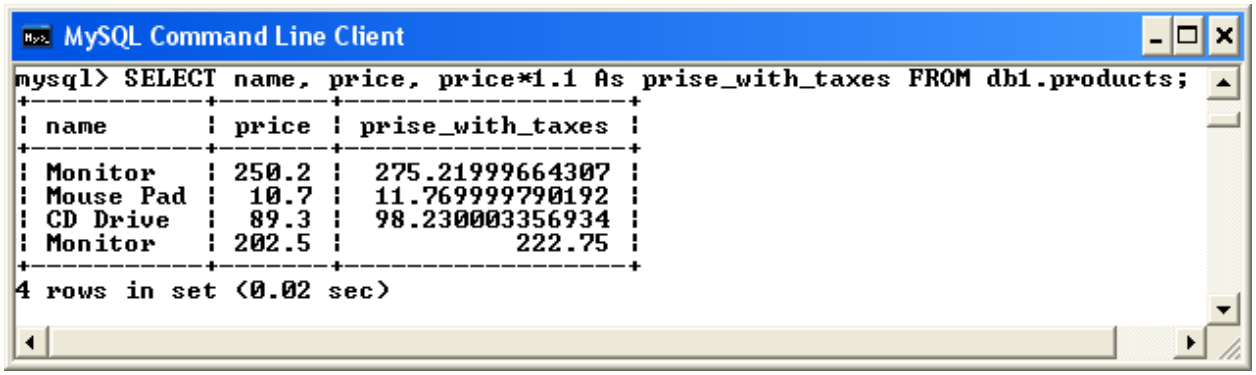
Аритметичните операции могат се използват в изрази, с които с извършват изчисления с данните, съдържащи се в числови полета на извлечени записи. Например ако в таблица prices се съдържа цена на стоки без начислена такса за обслужване (или за данъчно облагане), чрез аритметична операция със стойността на полето може да се изведе и цената с добавената стойност, както е показано в дадената по-долу SQL команда SELECT

```
SELECT name, price, price*1.1 As price_with_taxes FROM
db1.products
```

Пример 98 SQL команда SELECT, която извежда стойности, изчислени с аритметична операция

Резултатът от изпълнението на командата е показан в прозореца на клиентската програма mysql, с която е изпълнена командата. Обърнете внимание на клаузата As след изчислената

стойност. Тя дава възможност да се изведе име на колоната, зададено от програмиста. Ако клаузата As и низа след нея липсваха, щеше да се изведе като име на колоната формулата, по която е извършено изчислението, което не винаги е желателно.



Логически оператори

SQL притежава оператори за основните логически операции: логическо умножение AND, логическо събиране OR и логическо отрицание NOT. Те участват в изрази, с които се изчисляват по сложни условия, както е демонстрирано в следващите примери:

```
price >5 AND price <10
```

Този израз ще върне логическа стойност TRUE ако стойността на полето price е едновременно по-голяма от 5 и по-малка от 10.

```
price <=5 OR price >=10
```

Този израз ще върне логическа стойност TRUE ако стойността на полето price е или по-малка или равна на 5 или по-голяма или равна на 10. Очевидно вторият израз ще върне стойност, точно обратна стойността, получена от първия израз. Това означава, че втория израз може да бъде заместен от инвертирания с оператор NOT резултат от първия израз.

```
NOT (price >5 AND price <10)
```

Изрази в SQL

Изразите в SQL (както и във всеки език за програмиране) са формули за изчисляване на стойност, написани по синтаксиса на езика. Изразите могат да съдържат константи, променливи, имена на полета, вградени функции, оператори и кръгли скоби. Изразите (за разлика от конструкциите в езиците за програмиране) винаги връщат стойност от определен (от последната операция) тип. Например дадените по-горе изрази връщат логическа стойност, докато изразът (price +2)*1.05 връща числова стойност, тъй като последната операция, която се изпълнява, е умножение с числова константа.

Условия в SQL. Клауза WHERE

Условията в SQL са изрази, които връщат логическа стойност. Най-често те се използват с клаузата WHERE, с която се указва изпълнението на зададената SQL команда само за тези записи, за които условието след WHERE връща логическа истина. Примери за използването на клауза WHERE с SQL команди SELECT, INSERT и DELETE са дадени по-долу. Освен с клауза WHERE условия се изчисляват и в SQL конструкции като IF-ELSE и конструкции за организиране на цикли, които се използват в съхранените процедури (SQL Stored Procedures).

21.2. Извличане на записи – SQL команда SELECT

SQL командата *Select е най-често изпълняваната SQL команда, тъй като интернет приложенията, независимо дали изпълняват други операции с БД, на практика винаги визуализират извадки от съхраняваната в БД информация. Командата Select има следния синтаксис:

SELECT [predicate] {* | списък от полета }
FROM списък от таблици
[WHERE условие]

---

* SQL не прави разлика между малки и главни букви в синтаксиса на командите

[GROUP BY списък от полета за групиране]
[HAVING условие за групиране]
[ORDER BY списък от колони за сортиране]
[ASC | DESC]

където:

predicate	(незадължителен) – определя кои записи ще се извеждат. Допустими стойности са: ALL – извеждат се всички записи DISTINGT- извежда се само един от повтарящите се в извадката записи DISTINGT ROW- извежда се само един от повтарящите се в цялата таблица записи TOP n [PERSENT] – (не се използва от MySQL) извежда се зададен брой (или процент) n от първите записи
* | списък от полета	Извеждат се всички (*) или зададен списък от полета. Когато се извеждат полета от няколко таблици, преди името на полето може да се запише и името на таблицата, следвана от “.”
FROM	След клаузата се записва списък от имена на таблици, от които се извличат записи.
WHERE условие	След клаузата се записва условие. Ще бъдат извлечени само записите, за които условието има стойност логическа истина - TRUE
GROUP BY списък от полета	След клаузата се записва списък от полета, по които се групират стойностите на полетата от записите
HAVING условие за групиране	След клаузата се записва условие, по което се извличат записите. Използва се само с клауза GROUP BY
ORDER BY	Задава списък от колоните, по които се извършва сортиране на извлечените записи.
ASC | DESC	Определя реда на сортиране чрез клаузата ORDER BY. ASC задава нарастващ ред, DESC – намаляващ.

Limit [start,] num_row	(за MySQL) Ограничава броя на извлечените записи. където start е номера на първия запис (ако липсва, се подразбира 0), а num_row – брой записи, които да се извлекат

Да разгледаме един пример за SQL команда Select.

```
SELECT ime, fnom, result FROM students WHERE result > 5
```

Пример 99 Извличане на записи с SQL команда SELECT

Командата извлича от таблица students записи със съдържанието на полетата ime, fnom и result , за които стойността в полето result е по-голяма от 5.

21.3. Вградени SQL функции

SQL предоставя голям набор от вградени функции за различни видове операции. Най-общо тези функции могат да се разделят на две групи:

- Скаларни функции – това са функции, които извършват операции със стойностите от всеки един запис и връщат отделен резултат за всеки запис. Например функцията datediff(d1,d2) връща разликата d1-d2 в брой дни между дати (от тип date или datetime).
- Агрегатни (обобщаващи) функции - това са функции, които извършват операции със стойностите на едно или няколко полета от всички извлечени записи и връщат един резултат за всички обработвани записи.

SQL предоставя скаларни функции за операции с различни типове данни – за операции с дата и време, за аритметични и символни операции. Трябва да се има предвид, функциите не могат да се изпълняват непосредствено, така, както се изпълняват SQL командите като командата SELECT например. Програмистът, който иска да използва SQL функция има две възможности:

1) Да включи функцията (самостоятелно или в израз) в списъка от полета на SQL командата SELECT. В тоя случай стойността, която връща функцията е достъпна чрез съответното поле в записите, получени от командата SELECT.

2) Да включи функцията в SQL израз в команда или съхранена процедура.

Да разгледаме един пример: Нека е дадена таблицата products от база от данни db1 на MySQL със следното съдържание.

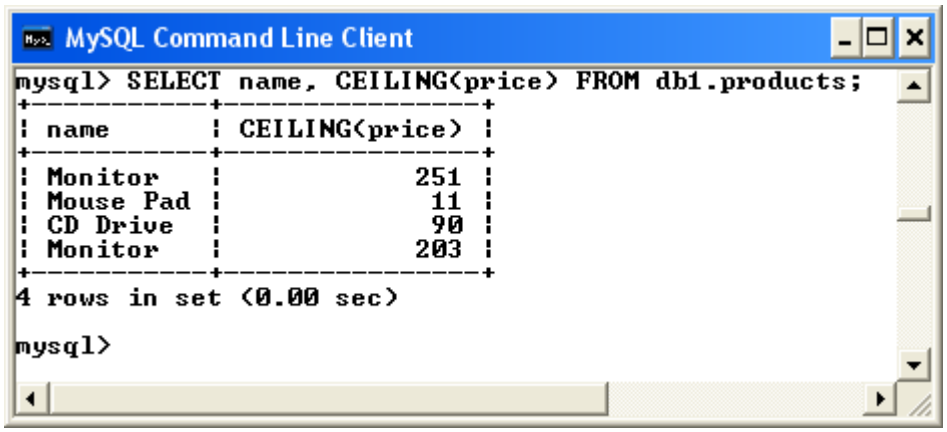
id	name	price
1	Monitor	250.2
2	Mouse Pad	10.7
3	CD Drive	89.3
4	Monitor	202.5

Ако изпълним SQL командата

```
SELECT name, CEILING(price) FROM db1.products
```

Пример 100 Използване на скаларна функция с SQL команда SELECT

функцията CEILING връща най-малкото цяло число, което е равно или по-голямо от стойността на аргумента. На фигурата е показан резултата от изпълнението на командата



Трябва да се има предвид, че наборите от вградени функции не са едни и същи за различните реализации на SQL. Например в ORACLE функцията, която е аналогична на функцията CEILING на MySQL има име CEIL.

21.4. Агрегатни (обобщаващи) функции

Агрегатните функции изчисляват обобщаваща информация за дадена извадка от записи. SQL предоставя пет агрегатни функции

COUNT - изчислява броя на извлечените записи
SUM - намира сумата от стойности на зададено поле в записите
MAX - намира най-голямата стойност на зададено поле
MIN - намира най-малката стойност на зададено поле
AVG - намира средната стойност на зададено поле

За да се получи резултата от изпълнението на някоя от тези функции, тя трябва да се включи в списъка от полета на SQL команда SELECT, като за функциите SUM, MAX, MIN и AVG трябва да се зададе като аргумент името на поле (или израз, в който участват имена на полета), по чиято стойност за извлечените записи се изчислява стойността на функцията. За функцията COUNT може да се зададе като аргумент символът "*" или име на поле – резултатът (брой на извлечените записи) очевидно е един и същ. Да разгледаме два примера за използването на агрегатни функции. Ще използваме таблицата products от MySQL БД db1 от предния пример.

Ако изпълним SQL командата

```
SELECT COUNT(*) FROM db1.products WHERE price > 50
```

Пример 101 Използване на SQL функцията COUNT

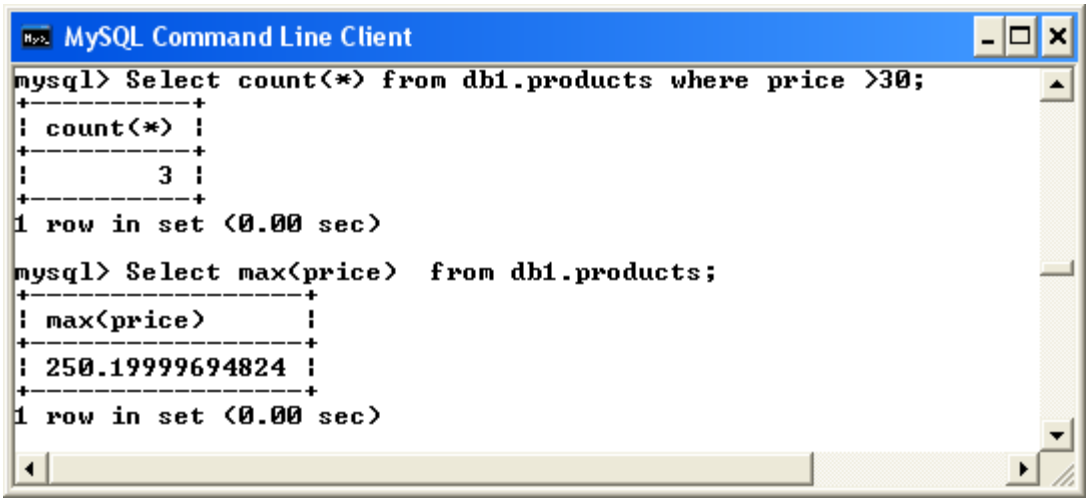
ще получим като резултат един запис с едно поле, в което ще се съдържа 3 – броя на извлечените записи чрез командата SELECT, отброени от функцията COUNT. Това са записите, за които се изпълнява условието, зададено с клаузата WHERE..

Ако изпълним SQL командата

```
SELECT MAX(price) FROM db1.products
```

Пример 102 Използване на SQL функцията MAX

ще получим като резултат един запис с едно поле, в което ще се съдържа 250 - максималната стойност, съдържаща се в колоната price, получена чрез функцията MAX.



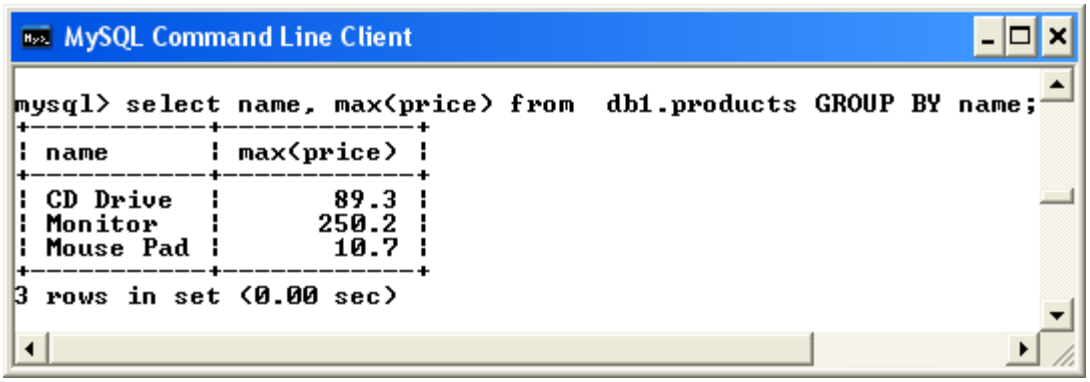
Фигура 1 Резултати от изпълнението SQL командите от примери 94 и 95 чрез клиентската програма mysql

Важно е да се запомни, че агрегатните функции могат да се използват заедно с полета в списъка от полета на SQL командата SELECT само ако се извършва групиране на записите чрез клаузата GROUP BY. Например за таблицата products, за да се получат най-големите цени по групи продукти може да се изпълни SQL командата

```
SELECT NAME, MAX(price) FROM  db1.products GROUP BY name;
```

Пример 103 Използване на SQL функцията MAX с клауза за групиране GROUP BY

В случая, тъй като в полето name само стойността “Monitor” се съдържа два пъти, за нея функцията MAX ще върне по-голямата от двете стойности на полето prise. За останалите два продукта “Mouse Pad” и “CD Drive”, тъй като са единствени, групиране на практика не се извършва и функцията MAX връща единствената стойност.



Фигура 2 Резултат от изпълнението SQL командата от пример 96 чрез клиентската програма mysql

21.5. Добавяне на записи – SQL команда INSERT

SQL командата Insert добавя запис към таблица от базата от данни. Тя има две версии:

SQL команда Insert...Values

SQL командата Insert...Values добавя един запис в края на таблицата. Тя има следния синтаксис

INSERT INTO име_на_таблица [(списък_от_полета)]
VALUES (списък от стойности)

където:

име_на_таблица	е името на таблицата, към която се добавя записа. Преди името на таблицата може да се добави и името на БД и разделител “.”
списък_от_полета	(незадължителен) е списъкът от полетата, за които се задават стойности в добавяния запис
списък_от_стойности	е списъкът от стойности на полетата

Трябва да се спазват следните три правила:

- Типът на стойностите в списъка от стойности трябва да съответства на типа на полетата в списъка от полета. На

практика синтаксисът на SQL прави разлика между символни и числови стойности. Символните стойности трябва да са ограничени с апострофи, а числовите – не.

- Размерът на данните не трябва да надвишава максималния размер за стълба. Това важи особено за символните низове. Затова при създаването на таблици е желателно максималния размер на колоните от символен тип да се задава по-голям от максималната разрешена дължина на записваните низове.
- Позицията на данните в списъка VALUES трябва да съответства на реда на полетата в списъка след името на таблицата. Ако списъка с полета отсъства, задължително трябва да се зададат стойности за всички полета в добавения запис. Това на практика означава, че съкратения синтаксис (без списък от полета) не може да се използва, когато таблицата съдържа полета, на които сървърът за БД сам генерира стойности (напр. от тип AUTO_INCREMENT), тъй като за тях стойности (различни от NULL) в списъка VALUES не трябва да се задават.

Да разгледаме един пример:

```
INSERT INTO students ( ime, fnom, result ) VALUES (‘Иван Иванов’, ‘072023’,4.5)
```

Пример 104 Добавяне на запис с SQL команда INSERT … VALUES

Командата добавя запис в таблица students като задава стойности на полетата ime, fnom и result.

SQL команда Insert...Select

Версията Insert-Select на SQL командата INSERT се използва тогава, когато е необходимо с една команда да се добавят в таблицата от БД много записи извлечени от други таблици от БД. Тя има следния синтаксис:

```
INSERT INTO име_на_таблица
(списък_от_полета)
SELECT списък_от_полета
FROM списък_от_таблици
```

[WHERE условие]

където:

име_на_таблица	е името на таблицата, към която се добавя записа. Преди името на таблицата може да се добави и името на БД и разделител “.”
списък_от_полета	е списъкът от полетата, за които се задават стойности в добавяния запис
списък_от_таблици	е списъкът от таблици, от които се извличат стойностите на полетата
WHERE условие	(незадължително) задава условие, по което се извличат записите.

Както се вижда от синтаксиса, в таблицата се добавят толкова записи, колкото се извличат от една или няколко таблица от БД с вложената SQL команда SELECT. За типа и подреждането на полетата в списъка на INSERT и вложената команда SELECT важат същите изисквания, както за версията Insert … Values. Трябва да се има предвид едно допълнително ограничение:

- Не се допуска от една и съща таблица да се извличат и да се добавят записи.

Да разгледаме един пример:

```
INSERT INTO tmp (students.ime, students.fnom, cources.course_name)
SELECT ime, fnom, course_name FROM students, cources WHERE
students.fnom= cources.fnom
```

Пример 105 Добавяне на записи в таблица с SQL команда INSERT … SELECT

Командата добавя в таблица tmp записи, които извлича от таблица students и таблица cources, като извлечените записи съдържат полета и от двете таблици, извлечени по условие да съвпадат факултетните номера от записите на двете таблици. Това свързване на двете таблици students и cources в SQL се нарича вътрешно свързване (inner join). Смисълът му е: свързва се всеки запис от първата таблица със всеки от втората, но се извличат само тези, който отговарят на за да деното условие.

21.6. Актуализиране на полета в записи – SQL команда UPDATE

SQL командата UPDATE дава възможност за променяне на съдържанието на полета в записи в таблица от БД при изпълнение зададено условие. Тя има следния синтаксис:

UPDATE име_на_таблица SET
име_на_поле = израз [,име_на_поле = израз] …
WHERE условие

където:

име_на_таблица	е името на таблицата, в която се актуализират записи. Преди името на таблицата може да се добави и името на БД и разделител “.”
име_на_поле	е име на поле от записите в таблицата, чиято стойност ще се променя
израз	е формула по синтаксиса на SQL, чиято стойност се изчислява и присвоява на полето.
WHERE условие	(незадължително) задава условие, при изпълнение на което се актуализират полетата в записите.

Изразите в SQL могат да съдържат оператори, константи, имена на полета и вградени в SQL функции. Важно е да се отбележи, че (въпреки твърденията че SQL е стандартизиран) има разлики в някои оператори за между SQL версиите, които се поддържат от различните сървъри за бази от данни.

Да разгледаме един пример: Таблица prices съдържа цени на стоки, предлагани в търговски обект. Преди Новогодишните празници собственикът иска да увеличи цените на стоките, които са под 5 лева, с 10%, (например за да обяви след това “промоция” от 5%). Това може да се направи с изпълнението на следната SQL команда:

```
UPDATE prices SET price=price*1.1 WHERE price<5
```

Пример 106 Актуализиране на записи в таблица с SQL команда UPDATE

Този пример показва колко мощна е SQL командата UPDATE. В примера тя извършва операция, която, ако се изпълнява с команда, която работи само с един запис, би изисквала организиране на цикъл през всички записи в таблицата и много по-голямо време за изпълнение.

21.7. Изтриване на записи от таблица – SQL команда DELETE

SQL командата DELETE дава възможност за изтриване на записи от таблица от БД при изпълнение зададено условие. Тя има следния синтаксис:

DELETE FROM име_на_таблица WHERE условие

където:

име_на_таблица	е името на таблицата, от която се изтриват записи. Преди името на таблицата може да се добави и името на БД и разделител “.”
WHERE условие	(незадължително) задава условие, при изпълнение на което се изтриват записите.

SQL командата DELETE изтрива само цели записи, а, не части от тях. Трябва да се има предвид, че при изтриването на записи от таблица, чието съдържание е логически свързано със съдържанието на други таблици, може да се наруши свързаността на данните. За да се избегне това, е необходимо да се създават връзки между таблиците, които да задължават сървъра за базата от данни да осигурява каскадно изтриване и каскадно актуализиране на свързаните записи. Създаването на връзки между таблиците е разгледано по нататък във връзка с операциите с MySQL бази от данни.

Да разгледаме един пример:

```
DELETE FROM students WHERE fnom=”072023”
```

Пример 107 Изтриване на записи в таблица с SQL команда DELETE

В случая, тъй като факултетния номер има уникална (т.е. неповтаряща се стойност) за всеки студент, ще бъде изтрит само

един запис. Ако, например, в БД има таблица courses, в която са записани избираемите дисциплини, които изучава всеки студент, то след изтриване на запис от таблица students за да се запази свързаността на БД трябва да се изтрият и свързаните с него записи от таблица courses.

22. Разширение SQLite

SQLite е разширение, което добавя към PHP5 вградена машина за бази от данни. SQLite е малка C-библиотека, разработена от [1]D. Richard Hipp през 2000г. SQLite разширението на PHP има следните предимства:

1) Самостоятелност: SQLite не използва модела клиент-сървър. Той се вгражда в приложението и има нужда само от достъп и права за четене и запис за файловете на базата от данни.
2) Не е необходимо администриране – за създаване на базата от данни е необходимо само да се изпълни функция от библиотеката.
3) Високо бързодействие: като нова реализация SQLite не е натоварена с проблеми за съвместимост. За повечето заявки SQLite има бързодействие не по-малко, а в някои случаи по-високо от MySQL.
4) Наличие едновременно на процедурен и обектно-ориентиран интерфейс.

Недостатъци:

1) Липсва сървърен процес: това води до проблеми когато е необходимо да се работи с няколко бази от данни едновременно. По правило сървърите за бази от данни работят в отделен процес и поради това могат да изпълняват асинхронни заявки. При липса на отделен процес, както е при SQLite, заявките се изпълняват винаги синхронно, което означава, че PHP модулът при операция с БД винаги изчаква да завърши операцията в БД и тогава продължава изпълнението на PHP кода.

---

[1] Повече информация за библиотеката и автора можете да намерите на http://www.sqlite.org/

2) Липсва вградена възможност за обработка на двоични данни. При запис на двоични данни те трябва да се кодират като символен низ, и съответно при извличането им с команда SELECT – да се разкодират. Тези ограничения са валидни за SQLite 2, докато от версия 3.0 е добавен типа данни BLOB (Binary Large Object), който позволява запис на големи по обем двоични данни.
3) Транзакциите заключват всички таблици. Повечето сървъри за бази от данни могат да заключват отделни таблици или дори отделни записи. Заключването на цялата база от данни при операции за запис има като резултат много изчаквания и съответно ниска скорост при многопотребителски достъп

Най-подходяща област на приложение на SQLite са интернет приложения, в които основно се изпълняват операции за четене на данни, а много по-рядко – добавяне или актуализиране, за което да се прави заключване на БД. В такива случаи SQLite операциите се изпълняват с висока скорост. Правата за достъп до БД се администрират лесно, тъй като те същност са права за достъп до един файл – файла за SQLite базата от данни.

22.1. Инсталиране на разширение SQLite в ОС Windows

SQLite е включен като опция в инсталационната програма на PHP5 и за инсталирането му е достатъчно да се маркира тази опция. Като резултат във файла php.ini трябва да се появят редовете:

```
extension=php_pdo.dll
extension=php_sqlite.dll
```

Първото разширение включва интерфейс за достъп до бази от данни PHP Data Objects (PDO). Всеки драйвер за база от данни, който реализира PDO интерфейса, предоставя стандартизиран набор от функции за операции с базата от данни, който представлява ниво на абстракция над специализираните функции за конкретния драйвер.

Разширението php_sqlite.dll включва в PHP библиотеката от SQLite функции.

22.2. Задаване на директория за временни файлове

Ако в ОС Windows SQLitece инсталира от потребител, който няма администраторски права, тогава променливата от обкръжението TMP на командния интерпретатор не се инициализира, и SQLite създава временни файлове в Windows директорията, което не е желателно. В тоя случай е полезно да се използва функцията putenv за да се зададе стойност на променливата TMP, например

```
putenv(“TMP=C:/temp”);
```

Пример 108 Задаване на директория за съхраняване на временни файлове на PHP

За зададената директория (в случая “C:/temp”) Web сървърът трябва да има права за създаване и запис във файлове. Инициализацията на променлива на обкръжението посредством функция putenv е валидна само в рамките на текущата заявка към Web сървъра. Можем да изведем текущата стойност на променливата на обкръжението с функцията getenv, например

```
echo “<BR>”.getenv(“TMP”);
```

Интерфейси на PHP за операции SQLite бази от данни

Разширението SQLite предоставя два интерфейса за операции с SQLite бази от данни – процедурно ориентиран и обектно ориентиран. Те предоставят еднакви възможности и поради това (за по-голяма яснота на описанието) ще разгледаме само процедурно ориентирания интерфейс. За сравнение на двата интерфейса ще разгледаме само създаването на обект SQLite от обектно ориентирания интерфейс.

22.3. Създаване на база от данни на SQLite

Създаването на база от данни на SQLite е на практика създаване на специално форматиран файл. За създаването на БД се изпълнява функцията sqlite_open за отваряне на БД – ако базата от данни не съществува, тя се създава. Функцията има следното описание:

resource sqlite_open (string filename [, int mode [, string &error_message]])

където:

filename	е спецификацията на файла за БД
mode	е режим на отваряне на файла. Понастоящем стойността му се игнорира. Предвижда се да се използва в случай че се отваря файла на БД в режим “само за четене”. Подразбиращата се стойност е 0666, което означава права за четене и запис за всички потребители
error_message	Този параметър трябва да бъде променлива. Ако при изпълнението на функцията възникне грешка, в променливата се записва съобщението за грешка.

При успешно изпълнение функцията връща ресурс – манипулатор за базата от данни, който се използва от функциите, които извършват операции с базата от данни. При възникване на грешка функцията връща стойност FALSE Да разгледаме един пример:

```
if ($db = sqlite_open(“mysqlitedb”, 0666, $sqliteerror)) {
      // Операции с SQLite базата от данни
} else {
   die($sqliteerror); //Извеждане на съобщение за грешка
}
```

Пример 109 Отваряне на SQLite база от данни

В примера файла за SQLite базата от данни има име “mysqlitedb” и е в същата директория, в която е PHP приложението. При успешно отваряне на SQLite базата от данни се изпълняват операции с нея, в противен случай се извежда съобщение за грешка и се прекратява изпълнението на PHP модула с функция die.

22.4. Типове данни в SQLite

SQLite версия 1 поддържа само два типа данни:

a) INTEGER
b) Подразбиращ се тип, подобен на типа VARCHAR (текстов тип за низове с променлива дължина) но без ограничението му от 255 символа.

Във версия 2 и 3 са въведени типовете

CHAR(n) За текстови данни с фиксирана дължина на полето
VARCHAR(n) За текстови данни с променлива дължина на полето
REAL За числови данни с плаваща запетая
TEXT За текстови данни
BLOB За двоични данни с голям обем
CLOB За текстови данни с голям обем

22.5. Създаване на таблица в SQLite база от данни

SQL командата за създаване на таблица в SQLite база от данни има следния синтаксис:

```
CREATE [ TEMP | TEMPORARY ] TABLE [ IF NOT EXISTS ]
[име_на_БД.] име_на_таблица
(
    дефиниция_на_колона [ ограничения ]
    [, дефиниция_на_колона [ ограничения ]
    ------------------------------
)
```

където:

дефиниция_на_колона	име_на_колона [тип_данни]
ограничения	задават допълнителни параметри и изисквания към стойностите в колоната: Not Null – за задължително задаване на стойност PRIMARY KEY – поле за първичен ключ UNIQUE – поле с уникална стойност CHECK (exp) – задава израз за проверка за валидност на данните

	DEFAULT value – задава начална стойност ако полето в записа не е инициализирано.

Да разгледаме един пример:

```
CREATE TABLE Address (
  id     INTEGER PRIMARY KEY,
  ime    VARCHAR(20),
  email  VARCHAR(20)
);
```

Пример 110 SQL команда за създаване на таблица в SQLite база от данни

В примера SQL командата създава таблица, която има поле

В SQLite базите от данни типът INTEGER PRIMARY KEY има специално значение. Полето от този тип се приема за поле на което SQLite автоматично генерира уникална стойност при създаване на записа. По правило в сървърите за бази от данни, (напр. ORACLE, MySQL) за да се получи такова поле, му се задава параметър (constraint) AUTO_INCREMENT.

Клаузата IF NOT EXISTS не се поддържа от SQLite 2. За да се избегне извеждането на предупреждение че таблицата вече съществува, преди функцията, с която се изпълнява SQL командата CREATE TABLE може да постави оператора за “приглушаване” @.

22.6. Изпълнение на заявки над SQLite база от данни

Според резултата от изпълнението си заявките към SQLite базата от данни биват два вида:

- Заявки, които връщат извадка от записи (динамична таблица). Такива заявки се изпълняват чрез SQL командата SELECT
- Заявки, които не връщат извадка. Такива заявки се изпълняват чрез SQL команди като INSERT, UPDATE и DELETE.

Ще разгледаме функциите, които предоставя разширението SQLite за тези два типа операции:

22.7. Изпълнение на заявки, които връщат извадка от записи – функция sqlite_query.

Основната функция за изпълнение на заявки, които връщат извадка от записи (динамична таблица) е функцията sqlite_query. Тя има следното описание:

resource sqlite_query(resource dbhandle, string query [, int result_type [, string &error_msg]])

където:

dbhandle	е манипулатор за БД, получен от функцията sqlite_open.
query	е SQL команда, която се изпълнява над базата от данни
result_type	Определя индексирането на масива, в който се получават извлечените записи. Приема като стойност някоя от следните константи SQLITE_ASSOC – масивът има като ключове имената на полетата от извлечените записи SQLITE_NUM – масивът има като ключове последователни целочислени индекси (от 0). SQLITE_BOTH– масивът има и двата типа ключове за всяка стойност – ключ със стойност от името на полето и ключ с целочислен индекс.
error_msg	Този параметър трябва да бъде променлива. Ако при изпълнението на функцията възникне грешка, в променливата се записва съобщението за грешка.

Функцията sqlite_query връща ресурс - манипулатор към динамична таблица, която съдържа извлечените записи. Манипулаторът се използва като параметър на функции като sqlite_fetch_array() и sqlite_seek(), чрез които се осъществява достъп до извлечените записи.

22.8. Последователно четене на записи от SQLite БД – функция sqlite_unbuffered_query.

В случай че ни е нужно еднопосочно последователно четене на записи, например ако искаме само да покажем съдържанието на таблица от базата от данни или извлечени по определен критерий

записи от нея, тогава е по-добре да използваме функцията sqlite_unbuffered_query. Тя има по-високо бързодействие и използва много по-малко оперативна памет на Web сървъра, което е съществено предимство при голям брой едновременно постъпващи заявки за четене от БД. Функцията има следното описание:

resource sqlite_unbuffered_query (resource dbhandle, string query [, int result_type [, string &error_msg]])

Както се вижда от описанието, функцията sqlite_unbuffered_query има същите параметри със същото значение, както функцията sqlite_query. Разликата е във вида на получавания ресурс. Докато sqlite_query връща динамична таблица, sqlite_unbuffered_query връща ресурс от записи, които могат да се четат само последователно. Такъв тип извадка се нарича извадка с еднопосочен курсор.

22.9. Извличане на данни от прочетен запис в масив – функция sqlite_fetch_array

Функцията sqlite_fetch_array извлича запис от извадка, получена от функцията sqlite_query или sqlite_unbuffered_query. Тя има следното описание:

array sqlite_fetch_array (resource result [, int result_type [, bool decode_binary]])

където:

result	е ресурс (извадка от записи) получен от функцията sqlite_query или sqlite_unbuffered_query
result_type	Определя индексирането на масива, в който се получават извлечените записи. Приема като стойност някоя от следните константи SQLITE_ASSOC – масивът има като ключове имената на полетата от извлечените записи SQLITE_NUM – масивът има като ключове последователни целочислени индекси (от 0). SQLITE_BOTH– масивът има и двата типа ключове за всяка стойност – ключ със стойност от името на

полето и ключ с целочислен индекс.

decode_binary При стойност TRUE (подразбираща се) PHP декодира двоичните данни, които са били кодирани с функцията sqlite_escape_string(). Този параметър по правило не се задава.

Функцията връща масив с прочетените данни или FALSE ако няма повече записи за четене.

Функцията sqlite_escape_string(string item) кодира “специалните” символи в символен низ, така, че да могат да се запишат като символни данни в SQLite база от данни. Трябва обаче да се има предвид, че ако поле от запис съдържа прекодирани данни, те не могат непосредствено да се сравняват с оператори за сравнение или оператор LIKE в клауза WHERE на SQL командите.

Да разгледаме един пример, който извежда съдържанието на SQLite таблица в прозореца на браузъра.

```
if ($db = sqlite_open('I:\\mysqlitedb', 0666, $sqliteerror)) {
        //Прочитане на всички записи от таблицата
        @$result = sqlite_unbuffered_query($db, "select * from
        Address",SQLITE_ASSOC,$select_error);
        if($result!==FALSE){
                echo "<TABLE BORDER width='100'>";
                //Четене и извеждане на запис
                while ($a = sqlite_fetch_array($result, SQLITE_ASSOC)) {
                        echo "<TR>\n";
                        foreach($a as $value){
                                echo "<TD>$value</TD>\n";
                        }
                        echo "</TR>";
                } //end of while
                echo "</TABLE>";
        }else{
                echo "<BR> Таблицата не е открита в БД ";
        }
        sqlite_close($db); //Затваряне на БД
} else {
   die($sqliteerror); //БД не може да бъде отворена
}
```

Пример 111 Извеждане на съдържанието на SQLite таблица

В примера записите от таблиците се прочитат с функцията sqlite_unbuffered_query, а четенето на поредния запис в масив става с функцията sqlite_fetch_array. Цикълът while завършва когато няма повече записи в извадката и sqlite_fetch_array връща стойност FALSE. Итерацията през полетата на текущо извлечения запис се извършва с конструкция foreach. Съдържанието на прочетените записи се извежда в HTML таблица.

22.10. Получаване на брой извлечени записи и брой полета в запис

Функцията sqlite_num_rows връща броя прочетени записи след изпълнение на SQL команда SELECT с функция sqlite_query. Тя има следното описание:
int sqlite_num_rows (resource result)
където dbhandle е манипулатора на БД.

Важно е да се запомни, че функцията връща брой записи само при извличане на буферирана извадка от записи с функция sqlite_query. При изпълнение с параметър – ресурс, получен от функция sqlite_unbuffered_query функцията sqlite_num_rows извежда съобщение “Row count is not available for unbuffered queries”.

Функцията sqlite_num_fields връща броя на полетата в прочетените записи след изпълнение на SQL команда SELECT с функция sqlite_query или sqlite_unbuffered_query. Тя има следното описание:
int sqlite_num_fields (resource result)
където dbhandle е манипулатора на БД.

Получената стойност може да използваме например ако искаме да извършим итерация на полетата в запис от получената извадка с конструкция for.

22.11. Изпълнение на SQL команди, които не връщат извадка от записи – функция sqlite_exec

За изпълнение на SQL команди, които не връщат извадка от записи, SQLite предоставя функцията sqlite_exec. Тя има следното описание:

bool sqlite_exec(resource dbhandle, string query [, string &error_msg])

където параметрите имат същото значение, както за функцията sqlite_query, но липсва параметърът result_type.

dbhandle	е манипулатор за БД, получен от функцията sqlite_open.
query	е SQL команда, която се изпълнява над базата от данни
error_msg	Този параметър трябва да бъде променлива. Ако при изпълнението на функцията възникне грешка, в променливата се записва съобщението за грешка.

Функцията връща логическа стойност TRUE ако SQL командата е изпълнена успешно, и FALSE в противен случай. Да разгледаме един пример:

```
if ($db = sqlite_open('mysqlitedb', 0666, $sqliteerror)) {
  @$query = sqlite_exec($db,"Create Table Address(id INTEGER
  PRIMARY KEY, ime varchar(10), email varchar(10))"
  ,$create_error); //Създаване на таблица
  $query = sqlite_exec($db,"Insert Into Address(ime, email) Values
  ('Peter','p@abv.bg')",$insert_error);  //Добавяне на запис
  sqlite_close($db);
} else {
   die($sqliteerror); //БД не може да бъде отворена
}
```

Пример 112 Използване на функцията sqlite_exec за създаване на таблица в SQLite БД и добавяне на запис към нея

В примера с функцията sqlite_exec се изпълняват SQL команда CREATE TABLE , с която се създава таблица в базата от данни 'mysqlitedb' и SQL команда INSERT … VALUES , посредством която се добавя запис към таблицата.

22.12. Изпълнение на операции в транзакция в SQLite

Транзакциите при операции над БД са средство за групово изпълнение на команди, при което или се запазва резултата от

изпълнението на всички команди в транзакцията, или се анулират всички направени в транзакцията изменения.

SQLite поддържа основните SQL команди за транзакции:

- BEGIN за стартиране на транзакция
- ROLLBACK – за прекратяване на транзакция с отказване на измененията
- COMMIT - за прекратяване на транзакция с потвърждаване на измененията.

Транзакциите са от изключително значение за поддържането на целостта на данните. Особено важно е това, че при инцидентно прекратяване на операциите с БД по време на транзакция, сървърът за базата от данни анулира всички изменения като изпълнява автоматично ROLLBACK. Да разгледаме един пример:

```
if ($db = sqlite_open('mysqlitedb', 0666, $sqliteerror)) {
  @$query = sqlite_exec($db,"Create Table Address(id INTEGER
  PRIMARY KEY, ime varchar(10), email varchar(10))"
  ,$create_error); //Създаване на таблица
  $query = sqlite_exec($db,"BEGIN”);
  $query = sqlite_exec($db,"Insert Into Address(ime, email) Values
  ('Peter','p@abv.bg')",$insert_error);  //Добавяне на запис
  $query = sqlite_exec($db,"ROLLBACK”);
  sqlite_close($db);
} else {
   die($sqliteerror); //БД не може да бъде отворена
}
```

Пример 113 Изпълнение на команда в SQLite БД в транзакция

Примерът а аналогичен на предния пример, с тази разлика, че изпълнението на командата, която добавя запис към таблицата Address е включено в транзакция, която завършва с анулиране на измененията с ROLLBACK. Като резултат, към таблицата запис не се добавя.

22.13. Затваряне на SQLite база от данни – функция sqlite_close

Функцията затваря SQLite базата от данни. Тя има следното описание:
void sqlite_close (resource dbhandle)
където dbhandle е манипулатора на БД.

Ако БД е отворена с функцията sqlite_popen, след затварянето ù с sqlite_close тя ще бъде премахната от списъка на отворените БД (persistent list).

22.14. Разширение SQLite3

От версия 5.3.0 PHP предоставя разширението SQLite3 с обектно ориентиран интерфейс за операции с SQLite3 бази от данни. Базовият клас SQLite3 на разширението предоставя основните методи за операции с SQLite база от данни. Конструкторът на класа има следното описание:

SQLite3::__construct (string $filename [, int $flags [, string $encryption_key]])

където:

filename — е спецификацията на файла за БД

flags — (незадължителен) – определя режима на отваряне. Подразбиращата се стойност е SQLITE3_OPEN_READWRITE | SQLITE3_OPEN_CREATE.
Стойността е лог. сума от следните флагове:

- SQLITE3_OPEN_READONLY: Отваря базата от данни само за четене.
- SQLITE3_OPEN_READWRITE: Отваря базата от данни за четене и запис.
- SQLITE3_OPEN_CREATE: Създава базата от данни ако не съществува.

encryption_key — (незадължителен) – задава ключ за криптиране на данните

Пример:

```
$db = new SQLite3('mysqlitedb.db');
```

Пример 114 Създаване на обект от клас SQLite3 за база от данни

Класът SQLite3 предоставя методи за изпълняване на команди над базата от данни, които по предназначението си са аналогични на методите, предоставяни от процедурно ориентираният интерфейс на разширението SQLite.

23. MySQL

MySQL е многопотребителски сървър за бази от данни. Той е създаден от софтуерната фирма MySQL AB, основана в Швеция от Дейвид Аксмарк, Ален Ларсон и Майкъл Видениус (David Axmark, Allan Larson, Micael Widenius MySQL се разпространява като Open Source / Free software.

MySQL функционира в мрежова среда и използва архитектура от типа клиент-сървър.

През януари 2008г. беше публикувана информация, че Sun Microsystems е закупила всички акции на MySQL AB, така, че понастоящем MySQL AB е подразделение на Sun Microsystems.

23.1. Сървърна програма

MySQL Server или mysqld е сървърната програма на MySQL. Тя управлява достъпа до базите от данни с мрежов достъп, на локалните дискови устройства и в оперативната памет. MySQL е многонишкова и може да едновременно поддържа голям брой връзки с клиентски програми.

В MySQL сървър 5.0 са включени някои допълнителни функции:

- поддържане на символични връзки
- поддържане на допълнителни типове таблици InnoDB и BDB
- поддържане на именувани канали за връзка (named pipes).

Размер на таблиците

MySQL версия 3.22 поддържа таблици с размер до 4 GB. Таблиците от тип MyISAM могат да имат размер до 2^{63} байта (осем милиона терабайта). Трябва обаче да се има предвид, че някои операционни системи имат собствени ограничения на максималния размер на файл. Като резултат размера на MySQL таблиците на практика се ограничава от операционната система, а не от MySQL сървър.

23.2. Клиентски програми

Това са програмите, които се използват за осъществяване на връзка със сървъра с цел манипулиране на информацията в базите от данни, които се управляват от него. Фирмата MySQL AB предлага няколко клиентски програми:

1) MySQL Query Browser, MySQL Administrator и MySQL Workbench са клиентски програми, които предоставят графични интерфейси за операции с MySQL бази от данни.
2) mysql е програма с текстов команден интерфейс, подобен на MSDOS. В ОС Windows тя се стартира в прозореца на Command Prompt.

Дистрибуция на MySQL Server и клиентски програми могат да се изтеглят от сайта на фирмата http://dev.mysql.com/ downloads/ .

23.3. Клиентска програма mysql

Както беше посочено по-горе, клиентската програма mysql се стартира в команден режим. Ако при инсталиране на MySQL се маркира опцията за добавяне на пътя до поддиректорията bin на MySQL, в която е инсталирана клиентската програма mysql, тя може да се стартира в прозореца на Command Prompt само въвеждане на mysql и натискане на бутона Enter.

В повечето случаи обаче mysql се стартира с параметри , които се въвеждат в командния ред след името на програмата. Ще разгледаме основните параметри, с които може да се стартира MySQL.

Формат на параметрите в командния ред

Клиентската програма mysql има два форма́та на параметрите в командния ред:

- Дълъг формат: всека параметър се задава с дума, предшествана от двойно тире, след което следва символа = и стойност. Пример: --host=localhost

- Кратък формат: всяка опция се задава с буква, предшествана от единично тире, след което следва празен интервал и стойност. Пример: -h 127.0.0.1

По-долу ще бъдат разгледани създаването на връзка с MySQL сървър и изпълнението на някои команди с клиентската програма mysql.

23.3.1. **Създаване и прекратяване на връзка със сървъра**

За създаване на връзка със сървъра е необходимо да се зададат следните параметри:

--host=име_на_хост или -h име_на_хост
Задава името или IP адреса на компютъра, на който е инсталиран MySQL Server. Когато MySQL е инсталиран на същия компютър, на параметъра трябва да се зададе стойност localhost или 127.0.0.1

--port=номер на порт или -P номер на порт
Задава номера на порта, към който се извършва свързването с компютъра, на който е инсталиран MySQL Server. Този параметър се прилага само за TCP/IP връзки. Подразбиращият се номер на порт е 3306.

--protocol=протокол
Задава протокола, по който се осъществява връзката. Възможни стойности са:
tcp – за TCP/IP протокол (подразбиращ се)
socket – връзка чрез Unix сокет (само за Unix/Linux ОС)
pipe – връзка по именуван канал към локален сървър (само за Windows ОС)
memory – връзка към локален сървър чрез споделена памет (само за Windows ОС)

--shared-memory-base-name=име
Задава име на споделен блок памет – само за Windows ОС при използване на протокол от тип “memory”.

В ОС Unix/Linux името на host компютъра localhost има специално значение за MySQL. То означава, че клиентската програма трябва да се свърже към сървъра посредством socket файл

на Unix. В тоя случай се елиминира съдържанието на параметъра port ако е зададен.

Ето как може да изглежда в ОС Windows командният ред за стартиране на mysql със зададени параметри за host и port:

```
C:\ > mysql --protocol=tcp --host=127.0.0.1
```

където " > " е системното съобщение (prompt) на командния интерпретатор, след което следва името на клиентската програма mysql и параметрите protocol и host, с които тя се стартира. Ако MySQL приеме подразбиращото се име и празна парола, ще се създаде връзка и ще се изведе съобщение от вида:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8 to server version: 5.0.21-community-nt
```

Същия резултат обаче можем да получим ако стартираме MySQL без параметри, тъй като стойностите tcp за протокол и 127.0.0.1 (или localhost) за host са подразбиращите се стойности за връзка с локално инсталиран MySQL Server.

Връзката на клиентската програма mysql с MySQL Server може да се прекрати с командата quit или exit.

23.3.2. **Идентификация на потребител**

Идентификация на потребител на MySQL се осъществява посредством параметрите --user=потребител и –password=парола. За потребител може да се използва и краткия формат –u потребител. Тези параметри определят потребителското име и паролата на акаунта, който ще се използва за достъп до сървъра. Сървърът ще осъществи връзка само ако в таблицата user на БД mysql на MySQL сървър има регистриран потребител с такова име и парола.

За да определи кой акаунт ще бъде приложен, сървърът използва стойността на параметъра user в комбинация с host - името на компютъра, с който се осъществява връзката. Това означава, че е възможно да има различни акаунти с едно и също потребителско име, но с различни стойности на host.

За Windows подразбиращото се име на потребител е ODBC, и поради това, ако стартирате клиентската програма mysql без

параметри, MySQL ще създаде връзка за потребител “ODBC@localhost”.
--password = парола - този параметър служи за задаване на парола. Не съществува подразбираща се парола. Ако се пропусне този параметър, в MySQL трябва да е регистриран акаунт, който да позволява свързване без задаване на парола. За Windows при инсталиране с подразбиращи се опции потребителят ODBC, който се създава като подразбиращ се, дава възможност за свързване без задаване на парола към локално инсталирания MySQL сървър.

При инсталиране на MySQL се инсталиращата програма предлага създаването на потребител root с парола, въведена от оператора. Не се препоръчва създаването на акаунти без парола, тъй като това дава възможност за неоторизиран достъп. Ето как може да изглежда в ОС Windows командният ред за стартиране на mysql със зададени параметри user и password:

```
C:\ > mysql --user=root --password=admin
```

или с използване на краткия формат

```
C:\ > mysql –u root –p admin
```

В MySQL информацията за акаунтите се съхранява в таблицата user на базата от данни mysql, която се създава по подразбиране. Можем да изведем съдържанието на тази таблица като изпълним чрез mysql командата

```
select host, user, password from mysql.user;
```

Резултатът ще има вида:

```
+-----------+-----------+-------------------------------------------+
| host      | user      | password                                  |
+-----------+-----------+-------------------------------------------+
| localhost | root      | *4ACFE3202A5FF5CF467898FC58AAB1D615029441 |
| localhost | ognzhe_db | *BEF2AF5DC9CA1751C3D1DA2E29B12621A90F5E9C |
+-----------+-----------+-------------------------------------------+
```

Вижда се, че данните за паролата не са в четим вид, тъй като се криптират при съхраняване в таблицата.

В дадената по-долу таблица са дадени някои от командите на клиентската програма mysql.

?	(\?) Синоним на командата `help'.
Connect	(\r) Рестартиране на връзката към сървъра. Незадължителни параметри са db и host.
Delimiter	(\d) Разделител между команди: Края на текстов ред също се приема разделител.
Ego	(\G) Send command to mysql server, display result vertically.
Exit	(\q) Излизане от mysql (Аналогична на quit).
Go	(\g) Изпраща команда към MySQL сървър.
Help	(\h) Извеждане на списъка на командите (help).
Notee	(\t) Прекратява записа на изхода на mysql във файл.
Prompt	(\R) Променя системното съобщение (prompt) на mysql.
Quit	(\q) Излизане от mysql.
Rehash	(\#) Изчиства хеш паметта на mysql за изпълнение на команди.
Source	(\.) Изпълнява SQL скриптов файл. Спецификацията на файла се задава като параметър.
show databases	Извежда списък с достъпните бази от данни
show tables	Извежда списък с достъпните таблици за подразбиращата се база от данни. Ако не е зададена подразбираща се БД с команда use, се извежда съобщение за грешка "No database selected"
status	(\s) Извежда информация за сървъра.
tee	(\T) Задава изходен файл [to_outfile]. След изпълнение на командата изхода от mysql се записва във файла (но се извежда и на дисплея)..
use	(\u) Задава подразбираща се база от данни. Името на БД се задава като параметър.
charset	(\C) Задава друго множество от символи

Таблица 4 Команди на клиентската програма mysql

Клиентската програма mysql не е така удобна за използване, както програмите, които предоставят графичен потребителски интерфейс, като MySQL Query Browser, MySQL Administrator и MySQL Workbench. Тя обаче ни дава възможност да изпълним автоматично (например при включване на съответните команди в

пакетен (bach) файл зададен блок от команди на mysql и SQL команди, което може да бъде полезно за администрирането на MySQL сървър.

23.4. Администриране на потребители

Администрирането на потребители на MySQL включва създаване на потребители и задаване на права на потребители. Създаването на потребител става с SQL командата CREATE USER. Тя има следния синтаксис:

```
CREATE USER user [IDENTIFIED BY [PASSWORD] 'password']
    [, user [IDENTIFIED BY [PASSWORD] 'password']] ...
```

където:

user е името на създавания акаунт
password (незадължителен) е паролата на създавания акаунт

Командата CREATE USER създава нов MySQL акаунт. Тя може да бъде изпълнена от потребител, който има глобална привилегия CREATE USER или привилегия INSERT за базата от данни mysql. Командата CREATE USER създава нов запис в таблицата mysql.user, в който няма зададени привилегии. Ако акаунтът вече съществува, се генерира грешка. Синтаксисът за име на акаунт е 'user_name'@'host_name'. Пример:

```
CREATE USER 'peter'@'localhost' IDENTIFIED BY 'peter_pan';
```

Важно е да се запомни, че командата CREATE USER само създава потребител, но не задава привилегии на потребителя.

Задаване на привилегии на потребител

Задаването на привилегии на потребител става с SQL командата GRANT. Освен, че задава привилегии командата създава потребителя, ако той не съществува, така, че командата GRANT замества напълно и командата CREATE USER. Тя има следния синтаксис.

```
GRANT
```

```
priv_type [(column_list)]
  [, priv_type [(column_list)]] ...
ON [object_type] priv_level
TO user [IDENTIFIED BY [PASSWORD] 'password']
    [, user [IDENTIFIED BY [PASSWORD] 'password']] ...
[REQUIRE {NONE | ssl_option [[AND] ssl_option] ...}]
[WITH with_option ...]
```

където:

priv_type	е типът на задаваните привилегии. Може да се задават отделно привилегии за всяка операция като списък , напр. INSERT, DELETE, UPDATE. Привилегията ALL задава привилегии за всички операции, с изключение на привилегията WITH GRANT OPTION (задаването на привилегии), която трябва да се зададе отделно.
column_list	(незадължителен) списък от полета, за който се задават привилегиите
object_type	(незадължителен) Задава типа на обекта от БД, за който се задават привилегии. Възможни стойности са: TABLE, FUNCTION, PROCEDURE
priv_level	Задава ниво на привилегията. Привилегиите могат да бъдат: Глобални – (*) за всички БД на дадения сървър. С команда REVOKE ALL ON *.* се премахват само глобалните привилегии. На ниво БД – (db_name.*)Тези привилегии се записват в таблиците mysql.db и mysql.host. Премахват се с REVOKE ALL ON db_name.* На ниво таблица (db_name.tbl_name). Премахват се с REVOKE ALL ON db_name. tbl_name. На ниво полета от таблица. Списъка на полетата се задава в column_list, а таблицата – като за привилегия на ниво таблица. Тези привилегии се записват в таблицата mysql.column_priv
TO user	Задава акаунта, на който се задават привилегиите. user се задава във формат 'user_name'@'host_name'
IDENTIFIED	(незадължителна) Ако присъства, тя присвоява

BY	парола на новосъздаден акаунт или променя паролата на съществуващ. Ако акаунтът съществува, и в командата GRANT не е зададена клауза IDENTIFIED BY, паролата на акаунта остава непроменена.
REQUIRE	Задава допълнителни изисквания, които могат да бъдат логическа сума от стойностите NONE – не изисква криптирани връзки към акаунта SSL – разрешава само криптирани връзки към акаунта X509 – изисква клиента да притежава сертификат CIPHER 'cipher' ISSUER 'issuer' – изисква клиента да притежава сертификат, издаден от 'issuer' SUBJECT 'subject'
with_option	Задава допълнителни опции, които могат да бъдат логическа сума от стойностите GRANT OPTION – право за задаване на права на потребител MAX_QUERIES_PER_HOUR count – максимален брой (count) заявки, които се обслужват за потребителя за един час. MAX_QUERIES_PER_HOUR count – максимален брой (count) заявки за променяне, които се обслужват за потребителя за един час. MAX_CONNECTIONS_PER_HOUR count – максимален брой (count) свързвания, които може да осъществява потребителя за един час.

Да разгледаме няколко примера:

```
GRANT ALL ON *.* TO 'teacher'@ 'localhost';
```

Пример 115 Задаване на всички привилегии за потребител 'teacher'@ 'localhost' чрез команда GRANT

```
GRANT SELECT, INSERT, DELETE, UPDATE ON db1.* TO student@localhost IDENTIFIED BY ‘abc123’
```

Пример 116 Задаване на привилегии и парола за потребител на база от данни db1 чрез команда GRANT

Получаване на информация за привилегиите на акаунт

Информация за привилегиите на акаунт на MySQL може да се получи чрез SQL командата SHOW GRANTS. За акаунта на текущия потребител тя може да се изпълни без параметри

```
SHOW GRANTS;
```

или

```
SHOW GRANTS FOR CURRENT_USER();
```

За друг регистриран акаунт командата има вида:

```
SHOW GRANTS FOR 'user_name'@'host_name'
```

Ако акаунтът в е защитен с парола, SQL командата SHOW GRANTS извежда след списъка с привилегии низа IDENTIFIED BY, следван от паролата на акаунта.

Отнемане на привилегии

Отнемането на привилегии на акаунт става с SQL командата REVOKE. Тя има следния синтаксис:

```
REVOKE
   priv_type [(column_list)]
     [, priv_type [(column_list)]] ...
   ON [] priv_level
   FROM user [, user] ...
```

където:

priv_type	е списъкът на привилегиите, които се отнемат
column_list	(незадължителен) е списък от полетата от таблица, за които се отнемат привилегиите
object_type	е типа на обекта от БД, за който се отнемат привилегии (TABLE, FUNCTION или PROCEDURE)
priv_level	задава нивото на привилегиите, които се отнемат. Има същите допустими стойности както за

	командата GRANT
user	е името на акаунта, за който се отнемат привилегии във формат ‘user_name@host_name’

Пример:

```
REVOKE DELETE, UPDATE ON tb1.* FROM ‘student@localhost’
```

Пример 117 Отнемане на привилегии от акаунт ‘student@localhost’ за операции DELETE и UPDATE над таблица tb1

23.5. Типове данни в MySQL

MySQL дава възможност за съхранение на два основни типа данни – числови и символни. Те обаче имат по няколко варианта, които е полезно да се познават, за да може таблиците на БД да се дефинират по-прецизно в съответствие с информацията, която ще съдържат.

Числови типове данни

Числовите типове данни имат следните характеристики:

1. Множеството (диапазона) от стойности, които се представят от типа.
2. Броят битове (байтове) за съхранение.
3. Брой позиции при извеждане на стойността.
4. Точността и порядъкът за числовите стойности, съхранявани във формат с плаваща запетая.

Целочислени типове данни

Това са типовете:

Име на типа	Представяне	Диапазон от стойности
TINYINT	1байт	-128 до 128 или от 0 до 255
SMALLINT	2 байта	-2^{15} до $+2^{15}$ или 0 до $2^{16}-1$
MEDIUMINT	3 байта	
INT	4 байта	
BIGINT	8 байта	

За целочислените типове могат да се декларират брой позиции за показване в кръгли скоби след името на типа, например:

country INT(4)

Това означава, че стойността ще се изведе на не по-малко от зададения брой позиции, като незначещите нули в началото ще се изведат като празни интервали. Ако са нужни повече от зададения брой позиции за значещи разряди, те ще се изведат, т.е. значещи разряди няма да бъдат загубени.

Типът BIT

Този тип може да се определи като целочислен тип, който дава възможност да се определят броя двоични разряди, които се използват за съхраняване на стойности . Формат: BIT(n) , където n е броя битове за съхраняване на стойността. Типът BIT определено дава възможност за най-икономично (от гледна точка на разхода на памет) съхраняване на стойности. Можем например да декларираме колона с данни от тип BIT, с дължина само един бит, например:
flag BIT(1).

Типове данни с плаваща запетая

Това са типовете:

FLOAT 4 байта
DOUBLE 8 байта

Данните от тип FLOAT и DOUBLE се съхраняват като мантиса и степенен показател. Типовете дават възможност да се задават броя разряди на мантисата и степенния показател като списък след името на типа, например:
radius FLOAT(10,3)

Когато не са зададени, се използва максималната точност според броя байтове за представяне.

Типове десетични данни с фиксирана запетая

При съхраняване на данни за финансови операции е по-полезно те да се съхраняват в оригиналния си десетичен формат с фиксирана запетая, тъй като така се избягват евентуална загуба на точност при преобразуване на числата от шестнадесетичен в десетичен формат и обратно. Такава възможност предоставя типът DECIMAL. Той има формат: DECIMAL(общ_брой_разряди, брой_разряди_след_десетичната_точка).

Низови типове данни

MySQL предоставя няколко низови типове данни, който се различават по следните характеристики:

Тип на елементите в низа – двоични данни или символи
Дължина на полето при съхраняване на низа – фиксирана или променлива, в зависимост от дължината на низа
Основните низови типове са:

CHAR	символен	с фиксирана дължина
VARCHAR	символен	с променлива дължина
TEXT	символен	с променлива дължина
BINARY	двоичен	с фиксирана дължина
VARBINARY	двоичен	с променлива дължина
ENUM	символен	с фиксиран набор от стойности
SET		с фиксиран набор от стойности, Всяка стойност се представя с 1 в един от битовете на дума с дължина от един до 8 байта (в зависимост от броя на декларираните възможни стойности).

При декларирането на поле от тип ENUM или SET след името на типа в скоби се записва списък от възможните стойности.

```
country ENUM(‘BG’,’FR’,’UK’)
```

Пример 118 Декларация на поле от тип ENUM

В примера се декларира поле от тип ENUM, в което може да се записват като стойности само низовете ‘BG’,’FR’ и ’UK’ и NULL, тъй като полето не е декларирано с ограничение NOT NULL.
Да разгледаме един пример за използване на поле от тип ENIM

```
CREATE TABLE test.countries (
id MEDIUMINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
code ENUM('BG','UK','FR'),
name CHAR(30) NOT NULL
);
INSERT INTO test.countries (code,name) Values ('BG','Bulgaria');
INSERT INTO test.countries (code,name) Values ('RU','Russia');
```

Пример 119 Използване на типът данни ENUM

Примерът съдържа блок от SQL команди, които могат да се въведат и изпълнят последователно с клиентската програма mysql. С първата SQL команда CREATE TABLE се създава таблица

countries. Полето code е от тип ENUM и приема стойности от зададения в дефиницията му списък. Първата SQL команда INSERT добавя запис към таблицата, в който в полето code се записва стойност 'BG'. С втората SQL команда INSERT се прави опит да се запише в полето code стойност, която не е декларирана в списъка на типа ENUM. MySQL отказва да изпълни тази команда и генерира грешка “Data truncated for column 'code' at row 1”.

23.6. Създаване на таблици в MySQL бази от данни

Таблиците в MySQL база от данни се създават с SQL командата CREATE TABLE. Командата има следния синтаксис:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS]
[db_name.]tbl_name
(
дефиниция_на_колона
[,дефиниция_на_колона]
……
)
[ENGINE=машина_за_съхранение]
```

където:

db_name	(незадължително)
tbl_name	е име на таблицата
дефиниция_на_колона	задава името, типа и параметрите и колоната
машина_за_съхранение	Задава типа на таблицата (което определя и машината за съхранение на данните). MySQL:поддържа следните типове таблици: MyISAM – индексирани таблици, не поддържат транзакции, заключване на ниво таблица MERGE – колекция от MyISAM таблици, използвани като една таблица. MEMORY – временни таблици в оперативната памет BDB – поддържат транзакции и заключване на ниво страница

	InnoDB– поддържат транзакции и заключване на ниво запис

Синтаксис на дефиниция_на_колона
име тип[(размер)] [NOT NULL | NULL] [DEFAULT default_value]
[AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
[COMMENT 'string']
машина_за_съхранение – определя типа на таблицата
Да разгледаме един пример:

```
CREATE TABLE IF NOT EXISTS db1.tb1
 (
id INTEGER PRIMARY KEY,
 ime VARCHAR(20),
 email VARCHAR(20)
) ENGINE=InnoDB ";
```

Пример 120 SQL команда за създаване на таблица

Командата CREATE TABLE от примера създава в база от данни db1 таблица с име .tb1 с три полета: id, ime и email съответно от тип INTEGER и VARCHAR. Типът на таблицата е InnoDB, което означава, че операциите с тази таблица могат да се изпълняват в транзакция.

Значение на параметрите на поле в MySQL таблица

Параметрите на полето в MySQL таблица дефинират допълнителни свойства и органичения на стойностите, които могат да се записват в полето:

NULL | NOT NULL	Разрешава/забранява в полето да се съдържа недефинирана стойност NULL
AUTO_INCREMENT	При добавяне на запис MySQL записва автоматично в полето стойност, с 1 по-голяма от последната, генерирана за това поле.
UNIQUE [KEY]	Указва, че полето трябва да има уникална стойност. При опит за дублиране на стойности се генерира грешка. Допуска се

	обаче стойност NULL
PRIMARY [KEY]	Указва, че полето трябва да има уникална стойност, която еднозначно идентифицира записа. При опит за дублиране на стойности се генерира грешка. Не се допуска стойност NULL
COMMENT	Задава пояснителен текст (коментар) за полето

23.7. Индексиране на MySQL таблица

Индексирането на таблиците е особено важно за ускоряване на търсенето (което неявно се прави при повечето операции). То създава логическо подреждане на данните, което може да се различава от физическото подреждане на записите в таблицата.

Търсенето на записи в таблици в БД се осъществява по два метода:

a) Последователен достъп (Sequential Access Method) – MySQL преминава последователно през всички записи докато открие тези, който отговарят на зададеното условие.
b) Непосредствен достъп (Direct Access Method) – MySQL поддържа дървовидна структура, в която записите са подредени логически по стойността на индекса. Търсенето на записи, чиито индекси отговарят на зададено условие, е много по-бързо, защото става по дървовидната структура, а не с последователно четене на всички записи.

Важно е да се запомни, че при задаване на условие с индексирано поле, например в клауза WHERE, за да се използва индексирането, полето не трябва да бъде включено в израз или функция, които се изчисляват за всеки запис. Например за командата “SELECT * from clients where age >30” индексирането по age ще се използва, докато за командата “SELECT * from clients where age-30>0” индексиране няма да се използва, независимо от това дали таблицата е индексирана по това поле.

Клъстерен (clustered) индекс: това е индекс по който физически се подреждат записите в таблицата. От това следва че:

1) Търсенето по клъстерен индекс е най-бързо
2) Клъстерният индекс трябва да бъде с уникални стойности. По правило това е първичният ключ на таблицата.
3) Желателно е стойностите в колоната-клъстерен индекс да се променят възможно най-рядко, тъй като това води до преизчисляване на всички индекси.

Проверка дали SQL заявка използва индексиране – SQL команда Explain

SQL командата EXPLAIN, използвана с команда SELECT връща информация (във вид на таблица) за това как се изпълнява заявката. В полето key се съдържа името на индекса, ако е използван такъв, в противен случай се връща празен низ. Например командата "EXPLAIN SELECT * from clients where age >30" връща

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	clients	range	IDX_age	IDX_age	5		1	Using where

а командата EXPLAIN SELECT * from clients WHERE age-30 > 0

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	clients	ALL					2	Using where

Вижда се, че във втория случай полетата possible_keys (възможни индекси) и key (използван индекс) са празни. Това показва, че при изпълнение на втората команда индексирането не се използва, така че първата команда ще се изпълнява по-бързо.

SQL предоставя три възможности за създаване на индекси в таблица от БД.

1) Чрез добавяне на дефиниция на индекс с някоя от клаузите PRIMARY KEY, UNIQUE, INDEX, или FULLTEXT в командата CREATE TABLE, с която се създава таблицата, например:

```
CREATE TABLE students (
      name VARCHAR(50),
      f_nom VARCHAR(20),
```

```
    INDEX(f_nom)
)
```

Пример 121 Дефиниране на индекс при създаване на таблица

2) При модифициране на таблица с SQL командата ALTER TABLE,

```
ALTER TABLE students ADD INDEX f_nom (f_nom)
```

Пример 122 Дефиниране на индекс при модифициране на таблица с команда ALTER TABLE

3) Посредством SQL командата CREATE INDEX, Командата има следния синтаксис:

```
CREATE [UNIQUE] INDEX [index_name] ON tablename
(index_columns);
```

където:

index_name	е името на създавания индекс
tablename	е името на таблицата, в която се създава индекса
index_columns	е списък от имената на колоните, от които се създава индекса

Следващия пример демонстрира използването на командата CREATE INDEX:

```
CREATE UNIQUE INDEX f_nom ON students (f_nom);
```

Пример 123 Дефиниране на индекс посредством командата CREATE INDEX

23.8. Създаване на релации между таблици от тип InnoDB

Типът таблици InnoDB поддържа използването на външни ключове. Това позволява декларирането на релации между колони от таблици от този тип като машината за съхраняване на данни InnoDB поддържа интегритета на БД чрез забраняване на операции, нарушаващи релациите и чрез каскадно изпълнение на операции DELETE и UPDATE.

При дефиниране на връзката между две таблици, таблицата, която съдържа външен ключ се нарича реферираща (свързана) таблица, а таблицата, която съдържа първичен ключ (или уникален индекс) чрез който се свързва с реферираща таблица, се нарича реферирана (главна) таблица.

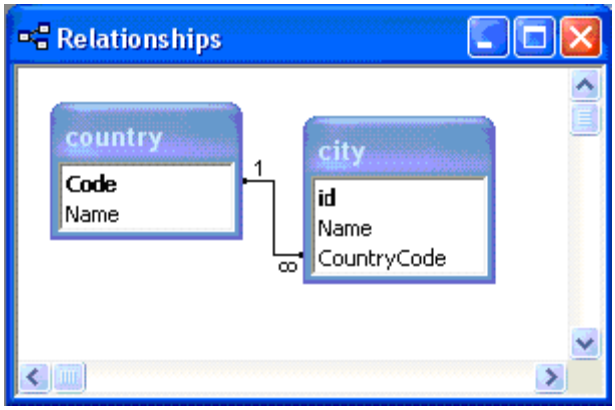
При релация по външен ключ свързаните полета от реферящата и реферираната таблица трябва да са от един и същ тип, като и двете таблици трябва да имат индекси. Ако реферящата таблица няма дефиниран индекс, InnoDB го създава автоматично. Ограничението (Constraint) FOREIGN KEY за дефиниране на външен ключ има следния синтаксис[3]:

```
FOREIGN KEY
    [index_name] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]
```

където:

index_name	(незадължителен) е името на индекса в реферящата таблица
index_col_name	е името на колоната-индекс в реферящата таблица
REFERENCES	Задава името на реферираната таблица и колоната (уникален индекс или първичен ключ) с които се създава връзка
ON DELETE	Дефинира операцията в реферящата таблица когато се изтрие запис в реферираната таблица. Възможни стойности са RESTRICT, CASCADE, SET NULL и NO ACTION
ON UPDATE	Дефинира операцията в реферящата таблица когато се актуализира запис в реферираната таблица. Възможни стойности са RESTRICT, CASCADE, SET NULL и NO ACTION

Да разгледаме един пример:



Схемата показва връзката между реферираща таблица city и реферираната таблица country, създадени с SQL командите:

```
CREATE TABLE IF NOT EXISTS city (
id INTEGER PRIMARY KEY AUTO_INCREMENT,
Name VARCHAR(20),
CountryCode CHAR(3),
INDEX(CountryCode),
FOREIGN KEY(CountryCode)
        REFERENCES Country(Code)
        ON UPDATE CASCADE
        ON DELETE CASCADE
) ENGINE=InnoDB"
```

Пример 124 Код за създаване на реферираща таблица city

```
CREATE TABLE IF NOT EXISTS country (
  Code CHAR(3) NOT NULL PRIMARY KEY,
  Name VARCHAR(20)
) ENGINE=InnoDB
```

Пример 125 Код за създаване на реферираната таблица country

Когато се създаде релация между таблици от тип InnoDB, MySQL сървър поема задължението да поддържа интегритета на БД, съответстващ на релацията, който се изразява в следното:

1) Реферираща таблица да съдържа само записи, които имат стойност на външния ключ, която съществува като стойност на свързания индекс в реферираната (главна) таблица. Например в таблицата city от примера може да има само кодове на държави, които се съдържат в реферираната таблица country.

2) Ако свързаният индекс в реферираната таблица бъде променен, съответният му външен ключ в реферираща таблица също трябва да получи новата му стойност (cascading update).
3) Ако запис с даден индекс в реферираната таблица бъде изтрит, свързаните с него записи (със същата стойност на външния ключ) в реферираща таблица също трябва да бъдат изтрити (cascading delete)

Значение на ограниченията RESTRICT, CASCADE, SET NULL и NO ACTION

Ограниченията, които се задават като параметър на клаузите ON DELETE и ON UPDATE имат следното значение

1) RESTRICT (подразбиращо се) Забранява изтриването на запис и променянето на външен ключ на запис в родителската (реферираната) таблица ако има поне един свързан запис в свързаната (реферираща) таблица.
2) CASCADE – Изтриването или променянето на запис в реферираната таблица автоматично изтрива или променя съответните таблици в свързаната (реферираща) таблица.
3) SET NULL - Изтриването или променянето на запис в реферираната таблица автоматично записва стойност NULL в свързаната (реферираща) таблица.
4) NO ACTION – има същото значение както RESTRICT.

23.9. Библиотеки на PHP за операции с MySQL бази от данни

PHP предоставя две разширения за операции с MySQL бази от данни – разширение mysql и разширение mysqli. Файловете на разширенията (php-mysql.dll и php-mysqli.dll) се инсталират в поддиректорията \ext на PHP. В четвърта версия на PHP разширението mysql е било вградено в PHP, но от 5-та версия то се инсталира като опция.

Операциите с MySQL бази от данни се изпълняват по същия сценарий както операциите с SQLite бази от данни: създаване на

връзка, изпълнение на операции и затваряне на връзката. Има обаче съществена разлика – при операции с MySQL се създава връзка между клиент (PHP модула) и сървър (MySQL сървър) и поради това е необходимо да се зададат допълнителни параметри на връзката.

Разширението mysqli е написано изцяло наново за да се предостави на програмистите достъп до новите възможности на MySQL версия 4.1 и следващи. Подобренията спрямо разширението mysql са:

2. Вградена функционалност за свързване, подготовка и изпълнение на компилирани конструкции.
3. Поддръжка на курсор
4. Кодове за грешки SQLSTATE
5. Включване на няколко конструкции в една заявка.
6. Анализатор на индекси

Разширението mysqli предоставя два интерфейса за операции с MySQL сървър – обектно ориентиран и процедурно ориентиран. Разликата е до голяма степен синтактична, тъй като функциите от процедурно ориентирания интерфейс също получават като параметри обекти и връщат обекти. По-нататък ще се ограничим с разглеждането само на процедурно ориентирания интерфейс, който е подобен на интерфейса, предоставян от по-старото разширение mysql, което продължава да се използва.

23.10. Създаване на връзка с MySQL сървър – функция mysqli_connect

Функцията mysqli_connect създава връзка с MySQL сървър и връща обект от клас mysqli при успешно свързване или стойност FALSE при генериране на грешка. Тя има следното описание:

mysqli mysqli_connect ([string host [, string username [, string passwd [, string dbname [, int port [, string socket]]]]]])

където:

host е името или IP адреса на компютъра, на който се изпълнява MySQL сървър. Стойност NULL или localhost указва, че MySQL е инсталиран на същия компютър, на който работи Web сървъра.

	Поставянето на префикс “p:”пред параметъра host за PHP5.3 и следващи версии се създава постоянна (persistent) [*]връзка. Тази връзка не се затваря автоматично след изпълнението на кода на PHP модула и при повторна заявка за свързване със същите параметри с MySQL сървър се използва същата връзка.
username	е името на потребителя, с чийто акаунт се осъществява връзката
passwd	е паролата на потребителя
dbname	Задава подразбиращата се база от данни
port	Задава порта, чрез който се осъществява комуникацията. Подразбиращата се стойност е 3306
socket	Задава сокет или именуван канал за комуникация

Обекта от клас mysqli, който връща функцията при успешно свързване, се използва като параметър на функциите, чрез които се извършват операции с MySQL.

23.11. Получаване на информация за грешка при свързване – функции mysqli_connect_error

Функцията връща текст с информация за последната възникнала грешка при опит за свързване с MySQL сървър. Тя има следното описание:

string mysqli_connect_error (void)

Подобна е функцията mysqli_connect_errno ( ) , която връща номер на грешката от последния опит за свързване с MySQL сървър.

23.12. Изпълнение на заявка към MySQL сървър – функция mysqli_query

Функцията mysqli_query е основната функция от разширението mysqli, чрез която се изпраща заявка към MySQL сървър и се

---

[*] В разширението mysql имаше отделна функция mysql_pconnect за създаване на постоянна връзка. В mysqli тя е премахната.

получава резултат от нейното изпълнение. Тя има следното описание:

mixed mysqli_query (mysqli link, string query [, int resultmode])

където:

link	е обект от клас mysqli , който представя връзката с MySQL сървър, създадена с функция mysqli_connect.
query	е низ, съдържащ текста на команда (по правило SQL команда) която се изпраща към сървъра за изпълнение.
resultmode	Определя типа на резултата. Допустими стойности са декларираните константи: MYSQLI_STORE_RESULT – (подразбираща се) – получаване на буферирана извадка от записи при изпълнение на SQL команда SELECT MYSQLI_USE_RESULT – получаване на небуферирана извадка от записи при изпълнение на SQL команда SELECT

Функцията връща стойност TRUE или обект от клас mysqli_result при успешно изпълнение or FALSE при грешка. Обект от клас mysqli_result се връща когато се изпълняват команди като SELECT, SHOW, DESCRIBE или EXPLAIN, които връщат резултат във вид на извадка или динамична таблица.

Буферираните извадки дават възможност за двупосочно движение на указателя на текущия запис (курсора) и за търсене в извадката по зададен критерий.

Небуферираните извадки дават възможност само за последователно четене на записите, но не изискват допълнителна памет за съхраняване на извадката. PHP модулът получава поредния запис веднага след извличането му.

Важно е да се запомни, че при изпълнение на небуферирана извадка не може да се изпраща друга команда към MySQL сървър чрез същата връзка, докато не се извлекат всички записи или докато не се прекрати извличането чрез функцията mysql_free_result.

23.13. Изпълнение на заявка, която не връща извадка от записи – функция mysqli_real_query

Разширението mysqli дава възможност за изпълнение на друг сценарий за операции, при който се получава извадка от записи: Функцията mysqli_real_query може да изпълни заявка, която извлича записи от MySQL база от данни, но връща само логически резултат TRUE при успешно изпълнение и FALSE при генериране на грешка. Функцията има следното описание:

bool mysqli_real_query (mysqli link, string query)

където:

link е обект от клас mysqli , който представя връзката с MySQL сървър, създадена с функция mysqli_connect.

query е низ, съдържащ текста на команда (по правило SQL команда) която се изпраща към сървъра за изпълнение.

След изпълнението на тази функция за да се осъществи достъп до извадката е необходимо да се получи обект от клас mysqli_query (който може да бъде получен и чрез изпълнение на функция mysqli_query). Важно е да се отбележи, че такъв обект се получава само ако е изпълнена SQL команда, която извлича записи от БД.

23.14. Получаване на обект mysqli_result – функции mysqli_store_result и mysqli_use_result

При успешно изпълнение на mysqli_real_query програмистът има две възможности:

1) Да получи резултата от изпълнението на заявката в буфериран вид посредством функцията mysqli_store_result( ) Функцията има следното описание:

mysqli_result mysqli_store_result(mysqli link)

където link е обект от клас mysqli , който представя връзката с MySQL сървър. Функцията връща обект от клас mysqli_result, който представя буферираната извадка или FALSE ако се генерира грешка.

2) Да получи резултата от изпълнението на заявката в небуфериран вид посредством функцията mysqli_use_result( ) Функцията има следното описание:

mysqli_result mysqli_use_result(mysqli link)

където link е обект от клас mysqli , който представя връзката с MySQL сървър. Функцията връща обект от клас mysqli_result,

който представя небуферираната извадка или FALSE ако се генерира грешка. В тоя случай обектът mysqli_result дава възможност само за последователно четене на записи от извадката.

23.15. Извличане на запис в масив – функции mysqli_fetch_row и mysqli_fetch_assoc

Разширението mysqli предоставя две функции, които извличат от обект mysqli_result поредния запис в масив и преместват указателя на следващия запис. Ако няма повече записи за извличане, функциите връщат стойност NULL Функциите имат идентично описание:

```
array mysqli_fetch_row ( mysqli_result result )
array mysqli_fetch_assoc ( mysqli_result result )
```

където result е обект от клас mysqli_result, който представя извадката.

Разликата между резултата, получаван от двете функции е в ключовете на масивите, в който се записват стойностите от полетата, прочетени от записа:

a) Функцията mysqli_fetch_row връща масив в който ключовете са последователни целочислени индекси, започващи от 0 (zero-based indexing)
b) Функцията mysqli_fetch_assoc връща масив в който като ключове се използват имената на полетата от прочетения запис

Да разгледаме един пример:

```
$link = mysqli_connect("localhost", "root", "admin")
or die('Could not connect: ' . mysqli_connect_error());
 if ($rs = mysqli_query($link,"SELECT * FROM db1.tb1",
    MYSQLI_USE_RESULT))
 {
      $t="<table border>";
      while($a=mysqli_fetch_assoc($rs)) //Цикъл през всички записи
      {
            $t.="<tr>\n";
            // Цикъл през всички полета от записа
            foreach($a as $val)
```

```
            {
                //Добавяне на клетка със стойност на поле
                $t.="<td>$val</td>\n";
            }
            $t.="<tr>\n";
        }
        $t.="</table>"; //Край на кода на HTML таблицата
        echo $t; //Извеждане на таблицата
        mysqli_free_result($rs);
        mysqli_close($link);
    }else{
        echo "No records in table";
    }
```

Пример 126 Извеждане на съдържанието на MySQL таблица

В примера се създава връзка към MySQL сървър с функцията mysqli_connect. Ако връзката не може да се осъществи се прекратява изпълнението на PHP модула с функцията die(). При успешно създадена връзка се извличат всички записи от таблицата чрез функцията mysqli_query с параметър SQL команда SELECT. Функцията връща обект $rs от клас mysqli_result, който представя прочетените записи. Обекта $rs е за небуферирана извадка, тъй като за прочитане на таблицата е достатъчно последователно четене на записите.

Последователното прочитане на зависите се осъществява чрез функция mysqli_fetch_assoc($rs), която се извиква в цикъл while. След прочитане на всички записи функцията връща стойност NULL и изпълнението на цикъла се прекратява. Итерацията на масива $a, в който са извлечени полетата от записа, се извършва с конструкция foreach. Кодът на HTML таблицата се натрупва в променливата $t и таблицата се извежда с конструкция echo.

23.16. Сортиране и ограничаване броя на записите в извадката – клаузи ORDER BY и LIMIT

Както беше посочено в описанието на командата SELECT в глава SQL, клаузата ORDER BY дава възможност за сортиране на

записите в извадката. По правило сортирането се прави по съдържанието на поле или на списък от полета от таблицата, от която се извличат записите. Допуска се обаче вместо по съдържанието на поле сортирането да се извърши по стойността на израз, зададен като параметър на клаузата ORDER BY. Ако в дадения по горе пример за извличане на записите се използва SQL командата

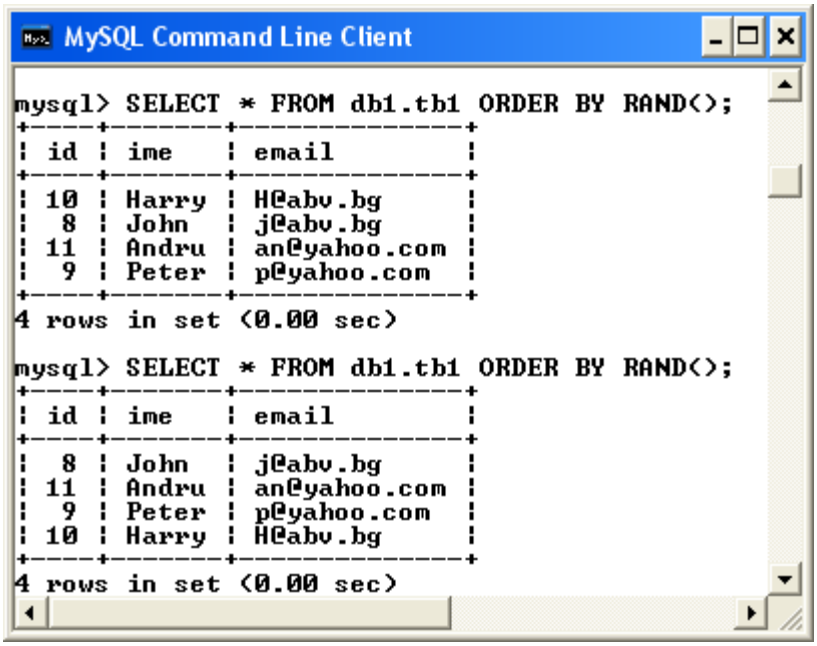
```
SELECT * FROM db1.tb1 ORDER BY ime DESC LIMIT 5, 100
```

Пример 127 Сортиране за записи в извадка с клаузата ORDER BY

записите ще бъдат подредени по съдържанието на полето ime в намаляващ ред. По подразбиране (ако параметърът DESC липсва) сортирането се извършва в нарастващ ред. Клаузата LIMIT ограничава броя прочетени записи (в примера до 100 от начален запис 5). Ако първият параметър (5) липсва, се извлича от запис 0.

SQL, поддържан от MySQL сървър, предоставя функцията RAND( ) – генератор на псевдослучайни числа с равномерно разпределение. Ако се извиква без параметър, функцията връща псевдослучайно число от интервала (0,1). Използването на функцията RAND( ) като параметър на клаузата ORDER BY, както е показано в следващия пример, има като резултат подреждането на записите в случаен ред.

В примера е изпълнена два пъти SQL командата “SELECT * FROM db1.tb1 ORDER BY RAND( ) “ чрез клиентската програма mysql. Както се вижда от резултатите, изведени в прозореца “MySQL Command Line Client”, подреждането на записите е различно, въпреки че се изпълнява една и съща команда SELECT.



Пример 128 Сортиране на записи в случаен ред посредством SQL функцията RAND( )

23.17. Освобождаване на паметта, използвана за съхраняване на заявка – функция mysqli_free_ result

Функцията освобождава паметта, заделена за съхраняване за обекта от клас mysqli_result, който се получава при изпълнение на заявка, която извлича записи. Тя има следното описание:

void mysqli_free_result (mysqli_result result)

където result е обект от клас mysqli_result, който представя извадката.

Препоръчва се винаги да се освобождава паметта от обекта mysqli_result след като съхраняването на извадката вече не е необходимо.

23.18. Изпращане на няколко SQL команди с едно обръщение към MySQL – функция mysqli_multi_query

Функцията mysqli_multi_query има същото описание както функцията mysqli_real_query

bool mysqli_multi_query (mysqli link, string query)

и, ако заявката съдържа само една SQL команда, връща същия резултат. Функцията обаче дава възможност за изпращане на низ от няколко SQL команди, разделени със “;” като MySQL връща последователно резултатите от изпълнението на командите. Резултата от изпълнението на първата SQL команда се получава (както и за mysqli_real_query) чрез функцията mysqli_use_result( ) (в небуфериран вид) или mysqli_store_result( ) (в буфериран вид). Резултатите от изпълнението на следващите команди се получават чрез функциите mysqli_more_results( ) и mysqli_next_result( ). Функцията mysqli_more_results(mysqli link) връща стойност TRUE ако има още резултати за получаване, а функцията mysqli_next_result(mysqli link) подготвя следващия резултат за получаване. Резултатът отново се получава чрез функцията функцията mysqli_use_result( ) или mysqli_store_result( ).

Функцията mysqli_multi_query връща стойност TRUE ако първата SQL команда се изпълни успешно, и FALSE в противен случай. Резултата от изпълнението на следващите команди се получава след всяко изпълнение на функцията mysqli_next_result( ). Последователността на получаване на резултатите се илюстрира със следния пример:

```
//Изпращане на низа от команди
if (mysqli_multi_query($link, $query)) {
   do {
      // Получаване на поредната буферирана извадка
      if ($result = mysqli_store_result($link)) {
       //Операции с поредната буферирана извадка
          mysqli_free_result($result); //Освобождаване на паметта
      }
      if (mysqli_more_results($link)) {
      // Подготовка за операции със следващата извадка
      }
     //Подготовка на следващия резултат
   } while (mysqli_next_result($link));
}
```

Пример 129 Последователност на операциите при изпълнение на низ от SQL команди чрез функция mysqli_multi_query

Примерът илюстрира с PHP код последователността на операциите при изпълнение на низ от команди чрез функция mysqli_multi_query. Ако при изпълнението на функцията mysqli_multi_query се връща стойност TRUE, започва изпълнението на цикъл do-while, в който се извлича поредния резултат (като буферирана извадка) чрез функцията mysqli_store_result. Подготовка за следващо извличане се извършва ако функцията mysqli_more_results връща стойност TRUE. Подготовка на следващия резултат за извличане се извършва при изпълнение на функцията mysqli_next_result. Ако тя върне стойност TRUE цикълът продължава с извличане на следващия резултат, а в противен случай цикълът на получаване и обработка на резултата завършва.

23.19. Компилирани конструкции

Компилираните конструкции (prepared statements) са конструкции на MySQL сървър, които дават възможност за многократно изпълнение след еднократно компилиране. Това ги прави близки по предназначение със съхранените процедури, които се поддържат от повечето съвременни сървъри за бази от данни. Компилираните конструкции обаче, за разлика от съхранените процедури включват само една параметрична SQL команда, която се компилира и използва само в рамките на сесията на PHP приложението, докато съхранените процедури съдържат блок инструкции и съхраняват се като обект в БД, който може да бъде използван от различни потребители и в различни приложения.

Компилираните конструкции са включени в MySQL от версия 4.1. Те имат следните предимства:

4) По-високо бързодействие при изпълнение поради еднократното компилиране на командите. За разлика от компилираните конструкции SQL командите се компилират всеки път преди изпълнение.
5) Използването на нов (за версия 4.1) двоичен протокол при предаването на данните. При обикновените заявки всичко, което се предава между клиента и сървъра се преобразува в символен низ. При двоичния протокол, който се използва за предаване на данни за предварително компилирани конструкции, данните се предават в оригиналния им вид.

Едно от предимствата на разширението msqli е, че предоставя функции (както и обекти и методи от обектно-ориентирания си интерфейс) за операции с компилирани конструкции. Полезно е обаче да се знае, че MySQL предоставя и SQL команди за подготовка и използване на компилирани конструкции, които могат да се използват и с разширението mysql.

Подготовка на компилирана конструкция. Функция mysqli_prepare

Функцията подготвя компилирана конструкция от SQL команда, зададена в определен формат и връща обект от клас mysqli_stmt, който се използва за операции с компилираната конструкция. Функцията има следното описание:

mysqli_stmt mysqli_prepare (mysqli link, string query)

където:

link е обект от клас mysqli , който представя връзката с MySQL сървър, създадена с функция mysqli_connect.

query е параметрична SQL команда за компилирана конструкция

Пи успешно изпълнение функцията връща обект от клас mysqli_stmt, а при неуспешно изпълнение – стойност FALSE

Параметрите в SQL командата за компилирана конструкция се записват със символа “?”. С параметри могат да се представят само елементи от SQL командата, в където се очаква задаване на данни С параметри не могат да се задават име на БД, таблица, поле или

клаузи от командата. Например като параметър може да се представи условието, което се задава на клауза WHERE или стойностите, които се задават в клаузата VALUES на SQL команда INSERT. Ето един пример за параметрична SQL команда, от която може да получи компилирана конструкция.

```
INSERT INTO db1.tb1 (ime,email) VALUES( ? ,? )
```

Пример 130 Параметрична SQL команда за компилирана конструкция

Свързване на параметри с компилирана конструкция - функция mysqli_stmt_bind_param

Преди всяко изпълнение на компилирана конструкция, която има входни (за MySQL) параметри, тя трябва да се свърже към параметрите, от които ще получи тези входни данни. И за това свързване може да се направи аналогия със съхранените процедури: преди извикване на съхранена процедура на всички входни за процедурата параметри трябва да се зададат *стойности. Функцията има следното описание:

bool mysqli_stmt_bind_param(mysqli_stmt stmt, string types, mixed &var1 [, mixed &...])

където:

stmt — е обект от клас mysqli_stmt, представящ компилираната конструкция

types — е низ от символи, с които се задава типа на предаваните параметри. Могат да се използват следните символи:
- i за параметър от тип integer
- d за параметър от тип double
- s за параметър от тип string
- b за параметър от тип BLOB

var1, var2, … — Променливи, от които се получават входните данни при изпълнение.

---

* Да си припомним, че параметрите, с които се извиква една функция е прието да се наричат фактически параметри, а параметрите, с които тя е декларирана - формални параметри.

Функцията връща стойност TRUE при успешно изпълнение, и FALSE в противен случай.

Изпълнение на компилирана конструкция - функция mysqli_stmt_execute

След като компилираната конструкция е подготвена и свързана с променливите, от които се получават стойностите на входните параметри, тя може да бъде изпълнена с функцията mysqli_stmt_execute. Тя има следното описание:

bool mysqli_stmt_execute (mysqli_stmt stmt)

където stmt е обект от клас mysqli_stmt, представящ компилираната конструкция.

Функцията връща стойност TRUE при успешно изпълнение, и FALSE в противен случай. Да разгледаме един пример за подготовка и изпълнение на компилирана конструкция.

```
$link = mysqli_connect("localhost", "root", "admin")
or die('Could not connect: ' . mysqli_connect_error());
$query="INSERT INTO db1.tb1 (ime,email) VALUES( ? ,? )";
//Подготовка за изпълнение на компилираната конструкция
$stmt=mysqli_prepare($link, $query);
// Свързване на параметри
mysqli_stmt_bind_param($stmt,”ss”, $v1, $v2);
$v1=”John”;
$v2=”j@abv.bg”;
//Изпълняване на командата
mysqli_stmt_execute($stmt);
$v1=”Peter”;
$v2=”p@yahoo.com”;
//Изпълняване на командата с втори набор от параметри
mysqli_stmt_execute($stmt);
mysqli_close($link);
```

Пример 131 Подготовка и изпълнение на компилирана конструкция с входни параметри

В примера се подготвя компилирана конструкция от SQL команда INSERT, която съдържа входни параметри в клаузата VALUES. Входните параметри се свързват с променливите $v1 и $v2 чрез функцията mysqli_stmt_bind_param. След като на променливите се зададе стойност компилираната конструкция се изпълнява с функцията mysqli_stmt_execute. След като на променливите се зададат нови стойности компилираната конструкция се изпълнява отново. Така при изпълнение на примера към таблицата tb1 се добавят два записа със стойности на полетата ime с email, получени от свързаните променливи.

Получаване на данни от компилирана конструкция - функции mysqli_stmt_bind_result и mysqli_stmt_fetch

Компилираните конструкции дават възможност за използване и на изходни параметри, чрез които да се получават данни, извлечени от изпълнението на конструкцията. За извличане на данните е необходимо съответните променливи да се свържат с изходните параметри на конструкцията чрез функцията mysqli_stmt_bind_ result. Тя има следното описание:

bool mysqli_stmt_bind_result (mysqli_stmt stmt, mixed &var1 [, mixed &...])

където:

stmt	е обект от клас mysqli_stmt, представящ компилираната конструкция
var1, var2, …	Променливи, от които се получават изходните данни от изпълнението на конструкцията.

Компилираните конструкции , които имат изходни параметри са на практика SQL команди SELECT. Функцията mysqli_stmt_ bind_result асоциира зададените (в реда, в който са в списъка на командата) променливи с полета от извадката.

Извлечените чрез компилираната конструкция записи се обработват последователно, като стойностите на полетата от текущия запис се извличат в изходните променливи с функцията mysqli_stmt_fetch. Тя има следното описание:

bool mysqli_stmt_fetch (mysqli_stmt stmt)

където:

stmt — е обект от клас mysqli_stmt, представящ компилираната конструкция

Функцията връща стойност TRUE при успешно изпълнение, и FALSE в противен случай. Да разгледаме един пример:

```
$link = mysqli_connect("localhost", "root", "admin")
        or die('Could not connect: ' . mysqli_connect_error());
$query="SELECT ime,email FROM db1.tb1";
// Подготовка на конструкцията
$stmt=mysqli_prepare($link, $query)
        or die("Can't prepare".mysqli_error($link));
//Изпълняване на командата
mysqli_stmt_execute($stmt)
        or die("Can't execute".mysqli_error($link));
//Свързване на променливи с полетата
mysqli_stmt_bind_result($stmt, $col1, $col2)
        or die("Can't bind".mysqli_error($link));
echo "<table border>";
// Цикъл през всички записи от извадката
while( mysqli_stmt_fetch($stmt) )
{ // Извеждане съдържанието на ред от таблицата
      echo "<tr><td>$col1</td><td>$col2</td>\n";
}
echo "</table>"; //край на HTML таблицата

mysqli_stmt_close($stmt); // Затваряне на връзката с командата
mysqli_close($link); // Затваряне на връзката с MySQL
```

Пример 132 Подготовка и изпълнение на компилирана конструкция с изходни параметри

В примера се подготвя компилирана конструкция от SQL команда SELECT, изпълнява се конструкцията и след това чрез функцията mysqli_stmt_bind_result се свързват изходни променливи $col1 и $col2 с полета от извадката. За извеждане на съдържанието на записите от получената извадка е организиран цикъл с конструкция while. Извличането на стойностите на полетата от текущия запис в променливите $col1 и $col2 става с функцията

mysqli_stmt_fetch. Съдържанието на записите се извежда в HTML таблица.

При извличане на данни чрез компилирана конструкция не е задължително извличането да е с последователни извиквания на функцията mysqli_stmt_fetch. Данните могат да се извличат по всяко време докато има още неизвлечени записи от извадката и докато не е прекратена връзката с компилираната конструкция.

Прекратяване на връзката с компилирана конструкция - функция mysqli_stmt_close

Компилираните конструкции се изтриват автоматично когато се прекрати връзката с MySQL сървър, така че не е необходимо обезателно програмно да се прекратява. Но ако програмистът иска по-рано да освободи паметта, ангажирана с конструкцията, може да използва функцията mysqli_stmt_close. Тя прекратява връзката с компилираната конструкция и унищожава обекта от клас *mysqli_stmt, който представя конструкцията. Функцията има описание:

bool mysqli_stmt_close (mysqli_stmt stmt)

където:

stmt — е обект от клас mysqli_stmt, представящ компилираната конструкция

23.20. Заключвания при операции с MYSQL бази от данни

Заключването при операциите с БД е механизъм, който предотвратява възникването на проблеми при едновременен достъп до данни от много потребители. Пример за конфликт е операцията актуализиране на запис от двама потребители. Ако двама потребители N1 и N2 искат да добавят стойност към числово поле от записа, последователността от операциите може да бъде следната:

---

* По-точно унищожава се указателят към обекта, а самият обект се унищожава в последствие.

1) N1 чете съдържанието на полето
2) N2 чете съдържанието на полето
3) N1 добавя стойност и записва
4) N2 добавя стойност и записва

Проблемът, който възниква при актуализацията на данните в случая е: кое изменение на съдържанието на полето ще бъде запазено? При тази последователност от операции – очевидно на потребителя N2, но потребителят N1 няма да знае, че актуализирането, което той е направил, е анулирано.

Описаният конфликт между потребителите N1 и N2 може да бъде избегнат при използването на заключване. Тогава последователността от операции добива вида:

1) N1 заключва записа и чете съдържанието на полето
2) N1 добавя стойност, записва резултата и отключва записа
3) N2 заключва записа и чете съдържанието на полето
4) N2 добавя стойност, записва резултата и отключва записа

В тоя случай конфликтът е избегнат, но може да се наложи потребителят N2 да изчака, докато потребителят N1 приключи актуализирането на записа.

Не всички опити за едновременен достъп водят до конфликти. Типът заключване, който е необходим за избягване на конфликти зависи от това, дали потребителят (клиентската програма) ще извършва четене или запис в БД. Да разгледаме двата случая:

a) Ако потребителят осъществява четене на данни, другите потребители, които четат данни, няма да предизвикат конфликт. В този случай те могат да работят едновременно. Ако обаче някой от другите потребители иска да извърши запис, той трябва да изчака докато първият потребител завърши операцията четене.
b) Ако потребителят осъществява запис на данни, всички останали потребители трябва да изчакат, независимо дали искат да извършват операция четене или запис.

Накратко казано, за да няма конфликти една четяща програма трябва да блокира записващите програми, а една записваща програма – всички – и четящи и записващи.

В MySQL (както и в другите сървъри за БД) могат да се разграничат два типа заключвания: явни и неявни.

Неявно заключване

Неявното заключване с извършва по инициатива на MySQL сървър, за да се осигури безконфликтно изпълнение на командите, при условие , че не е направено заключване от клиента. Например ако клиентската програма изпрати към сървъра команда SELECT, се извършва заключване за четене, (т.е. блокират се заявките за запис), а ако клиентската програма изпрати команда INSERT, се извършва заключване за запис (т.е. блокират се всички заявки). Неявното заключване се извършва само за времето, през което се изпълнява единичната команда.

Такъв тип заключване, при който записът се заключва само докато се изпълнява единична команда от сървъра за БД се нарича *оптимистично заключване*.

Явно заключване

Явното заключване се извършва по инициатива на клиентската програма чрез SQL командата LOCK TABLES, а отключването – с командата UNLOCK TABLES. Явното заключване е необходимо когато клиентската програма трябва да извърши операция, изискваща изпълнението на няколко SQL команди, и когато това изпълнение не трябва да се прекъсва с изпълнение на команди на други потребители. Даденият по-горе пример за безконфликтно извършване на актуализация на запис от потребители N1 и N2 изисква точно такъв тип заключване.

Такъв тип заключване, при който обектът от БД се заключва през цялото време докато се изпълнява блок от команди на потребителя (клиентската програма) се нарича *песимистично заключване*.

Заключване, прилагано при различните типове таблици

Явното заключване в MySQL е винаги на ниво таблица. Неявното заключване може да бъде на различно ниво в зависимост от вида на таблицата (и съответно машината за съхранение).

1) За таблиците от тип MyISAM, MEMORY и MERGE заключването е на ниво таблица.
2) За таблиците от тип BDB заключването е на ниво страница.
3) За таблиците от тип InnoDB заключването е на запис

Заключванията на ниво страница и на ниво запис създават по-малко изчакване при многопотребителски достъп до БД от заключването на ниво таблица, тъй като позволяват на няколко потребители едновременно да оперират с данните в различни части (страници или записи) в рамките на една таблица.

Заключването на ниво таблица не създава блокиране от тип "мъртва хватка", докато при частичните заключвания то е възможно. Такова блокиране се получава тогава, когато две програми заключат (и държат заключени) различни записи и след това всяка от тях се опитва да модифицира записа, заключен от другата програма.

Явното заключване ускорява операциите на заключващата програма по две причини:

1) Заключването и отключването се правят еднократно в началото и края на блока от команди, а не за всяка команда поотделно.
2) Обновяването на индексите при изпълнение на команди за модифициране на данни се прави еднократно при отключване, а не след всяка модифицираща команда като INSERT или DELETE.

Заключване на таблици - команда LOCK TABLES

Командата LOCK TABLES дава възможност за заключване една или повече на таблици. Тя има следния синтаксис:

LOCK TABLES tbl_name [[AS] alias] lock_type [, tbl_name [[AS] alias] lock_type] ...

където:

tbl_name е името на таблицата, която се заключва

alias (незадължителен) е псевдоним на таблицата

lock_type е типът на извършваното заключване. Възможните стойности са:

READ [LOCAL] – Заключване за четене. Клиентът, който притежава заключването и всички останали клиенти могат да четат от таблицата, но не могат да извършват запис, т.е. не могат да се изпълняват команди INSERT, DELETE или UPDATE.
Модификаторът LOCAL разрешава неконкурентно изпълнение на команди INSERT. За таблици от тип InnoDB този модификатор е без значение.

[LOW_PRIORITY] WRITE - Заключване за запис. Клиентът, който притежава заключването може да чете и да записва в таблицата. Другите клиенти нямат достъп до таблицата. Заявките за заключване от други клиенти също се блокират.
Модификаторът LOW_PRIORITY задава нисък приоритет на заявката за заключване за запис. Това означава, че се изчаква приключването на текущите операции, и ако има клиенти, които изчакват разрешение за заключване за четене, те ще го получат преди клиентът да получи разрешение за заключване за запис.

Прекратяване на заключването - команда UNLOCK TABLES

Само клиентът, извършил заключването може да го прекрати. Но ако се прекъсне връзката с клиента, заключванията, осъществени от него се прекратяват след изтичане на контролно време.

Командата UNLOCK TABLES е без параметри. Тя отключва всички таблици, заключени от потребителя (клиентската програма), която изпълнява командата.

Консултативно заключване

Констултативното заключване е средство за синхронизиране на заключванията между няколко потребители, които ползват за целта

един и същ символен низ (ключ). MySQL предоставя две функции, с които се осъществява синхронизацията между потребителите: функция GET_LOCK за получаване на ключа и функция RELEASE_LOCK за освобождаване на ключа.

За да получи право на достъп до общия ресурс клиентската програма трябва да изпълни функцията GET_LOCK. Функцията има следния синтаксис:

GET_LOCK(‘switch’,timeout)

където:

switch’ е името на ключа

timeout е максималното време в секунди, което се изчаква от програмата за да получи право за операция.

Ако програмата получи правото за операция в рамките на timeout, функцията връща 1, при изтичане на времето без право на операция връща 0, а при генериране на грешка – NULL.

След като завърши операцията с общия ресурс клиентската програма трябва да освободи ключа чрез функцията RELEASE_LOCK(‘switch’).

Както се вижда и от описанието им, функциите GET_LOCK и RELEASE_LOCK нито заключват, нито отключват обекти в MySQL БД. Те служат само за синхронизиране между клиентските програми, които ползват общ ключ.

Да разгледаме един пример:

```
$link = mysqli_connect("localhost", "root", "admin")
        or die('Could not connect: ' . mysqli_connect_error());
$query="SELECT GET_LOCK(‘my_switch’,10)";
// Изпращане на заявка за ключа
$rs = mysqli_query($link, $query);
$a=mysqli_fetch_row($rs);
//Проверка дали ключа е получен
if($a[0]==1)
{
    echo “Операции с общия ресурс”;

    RELEASE_LOCK(‘my_switch’); //Освобождаване на ключа
}else{
    echo “Не е получено разрешение за операция”;
```

```
}
mysqli_close($link);  //Затваряне на връзката с MySQL
```

Пример 133 Консултативно заключване с функция GET_LOCK

В примера се създава връзка с MySQL сървър и се изпраща заявка за получаване на разрешение за достъп до общ ресурс чрез функция GET_LOCK. Ако заявим този модул от локално инсталиран Web сървър, тъй като MySQL сървър няма друг клиент, функцията GET_LOCK ще върне стойност 1, която ще бъде получена в извадка след изпълнение на SQL команда SELECT. В прозореца на браузъра ще бъде изведено съобщението“Операции с общия ресурс”.

Можем обаче да създадем още един клиент на MySQL като стартираме в прозореца Command Prompt клиентската програма mysql

```
>mysql -u root --password=admin
```

и въведем командата

```
mysql> SELECT GET_LOCK('my_switch',100);
```

където “mysql>” е системното съобщение на клиентската програма mysql. Функцията GET_LOCK изпраща заявка получаване на разрешение за достъп чрез ключа 'my_switch' с timeout 100 секунди. Ако функцията върне стойност 1 ще се изведе резултат

```
+--------------------------+
| GET_LOCK('my_switch',100) |
+--------------------------+
|                        1 |
+--------------------------+
1 row in set (0.00 sec)
```

Ако след това заявим отново PHP модула (например като щракнем върху бутона Refresh на браузъра), тъй като ключът 'my_switch' не е освободен, функцията PHP модула ще изведе съобщение “Не е получено разрешение за операция”. Едва след като излезем от клиентската програма mysql с команда quit или exit, след Refresh на браузъра ще получим отново съобщението “Операции с общия ресурс”.

23.21. Изпълнение на операции в транзакция в MySQL

Транзакциите., както беше посочено при разглеждането на разширение SQLite, са средство за групово изпълнение на команди при операции над БД, при което или се запазва резултата от изпълнението на всички команди в транзакцията, или се анулират всички направени в транзакцията изменения.

MySQL версия 3.23-max и всички версии след 4.0 поддържат транзакции с таблиците (и съответно машините за съхранение) от тип InnoDB и BDB. Използването на транзакции е особено необходимо, когато се изисква по-надеждна защита от загуба на данни, например при финансови операции.

InnoDB модел на транзакциите

Innobase е финландска софтуерна фирма, която е разработила машина за съхранение на данни и съответстващ тип таблици, които поддържат транзакции и външни ключове. Системите, поддържащи транзакции често се определят като съвместими с ACID, т.е. притежаващи следните свойства:

1) Atomic (неделимост) – пакетно изпълнение или анулиране на измененията.
2) Consistent (съгласуваност) – Всяка БД, която е в съгласувано състояние при започване на транзакция е в такова състояние и при завършването ù.
3) Isolated (изолация) – транзакциите не оказват влияние помежду си.
4) Durable (стабилност) – всички модификации, направени от изпълнена успешно транзакция се записват правилно в БД. Всички направени промени се съхраняват.

По подразбиране MySQL изпълнява командите в режим на автоматично потвърждаване (autocommit), което означава, че всяка команда се изпълнява в отделна транзакция. Програмистът има две възможности да зададе изпълнение на група команди в транзакция:

1. Изключване на режима autocommit. Това може да бъде направено:

а) Чрез функцията mysqli_autocommit. Тя има следната декларация:

bool mysqli_autocommit (mysqli link, bool mode)

където:

link е обект от клас mysqli , който представя връзката с MySQL сървър, създадена с функция mysqli_connect.

mode при стойност FALSE изключва, а при стойност TRUE включва режима autocommit.

Пример:

```
mysqli_autocommit ( $link, FALSE);
```

Пример 134 Изключване на режима autocommit с функция mysqli_autocommit

b) Чрез изпълнение на SQL командата *SET AUTOCOMMIT=0

```
mysqli_query($link, “SET AUTOCOMMIT=0”);
```

Пример 135 Изключване на режима autocommit с SQL командата SET AUTOCOMMIT=0

2. Втората възможност за изпълнение на група команди в транзакция е стартирането на транзакция с SQL командата START TRANSACTION или с SQL командата BEGIN

```
mysqli_query($link, “START TRANSACTION”);
```

Пример 136 Стартиране на транзакция с SQL командата START TRANSACTION

Транзакцията може да завърши с потвърждаване на измененията с SQL командата COMMIT или с анулиране на всички изменения с SQL командата ROLLBACK.

При инцидентно прекратяване на операциите с БД по време на транзакция, сървърът за базата от данни анулира всички изменения като изпълнява автоматично ROLLBACK.

---

* Да си припомним, че SQL не прави разлика между малки и главни букви. В учебника изписването на SQL командите е с главни букви само за да се разграничават по-добре от останалия текст.

Важно е да се запомни, че за да се поддържат транзакции при операции с таблица от БД, тя трябва да бъде от тип InnoDB или BDB, което означава, че SQL командата CREATE TABLE трябва да завършва с параметър ENGINE=InnoDB или ENGINE= BDB. По-добрият избор е типът InnoDB и затова от MySQL 5 това е подразбиращият се тип таблици.

Да разгледаме един пример:

```
$link = mysqli_connect("localhost", "root", "admin")
          or die('Could not connect: ' . mysqli_connect_error());
$sql_table =
 "CREATE TABLE IF NOT EXISTS db1.tb1
 (
id INTEGER PRIMARY KEY,
 ime VARCHAR(20),
 email VARCHAR(20)
) ENGINE=InnoDB ";
mysqli_query($link,$sql_table)
                  or die("Could not create table". mysqli_error($link));
// Стартиране на транзакция
mysqli_query($link,"START TRANSACTION");
// Äîáàâÿíå íà çàïèñ
$sql_insert ="INSERT INTO db1.tb1 (ime, email) VALUES
('John','j@abv.bg')";
mysqli_query($link, $sql_insert)
                  or die("Could not insert". mysqli_error($link));
// Получаване на параметъра save
if(isset($_GET["save"]))$save=$_GET["save"]; else $save="no";
//Проверка дали да се съхранява записа
if($save=="yes"){
  mysqli_query($link, "COMMIT"); //Потвърждаване на измененията
}else{
  mysqli_query($link, "ROLLBACK"); //Анулиране на измененията
}
mysqli_close($link); // Прекратяване на връзката с MySQL
include("Show_MySQL_Table.php"); //Извеждане на таблицата
```

Пример 137 Добавяне на запис в транзакция

В примера се създава таблица tb1 от тип InnoDB, стартира се транзакция с SQL командата START TRANSACTION и се добавя запис към таблицата с SQL командата INSERT. Командите се изпращат за изпълнение от MySQL сървър с функцията mysqli_query. PHP модулът проверява дали е получен по метода GET параметър save. Ако е получен и има стойност “yes”, транзакцията се прекратява със запазване на измененията чрез SQL команда COMMIT. Ако параметърът не е получен или има друга стойност, транзакцията се прекратява със анулиране на измененията чрез SQL команда ROLLBACK. За извеждане на съдържанието на таблицата с конструкция include е вмъкнат PHP модула (примерът е даден след описанието на функцията mysqli_fetch) който извежда съдържанието на таблицата tb1 от база от данни tb1.

23.22. MySQL съхранени процедури

Съхранените процедури (подпрограми и функции) — са обекти на БД, които капсулират блок от SQL команди, който се компилира еднократно и се съхранява в сървъра на БД . Съхранените процедури, както и процедурите в програмните езици могат да притежават входни и изходни параметри и локални променливи. В тях могат да се изпълняват заявки към БД, числови и символни операции, резултатите от които могат да се присвояват на променливи и параметри.

Разликата между съхранените процедури подпрограми и функции е същата, както и в програмните езици с общо предназначение: подпрограмите се извикват самостоятелно чрез оператор (CALL) и могат да връщат стойности само чрез изходните си параметри, докато функциите връщат стойност чрез името си и могат да се включват непосредствено в SQL изрази.

Предимства на съхранените процедури (СП):

1) Разделяне на логиката СП се съхраняват в БД и могат да се ползват от много PHP приложения. Така се улеснява поддръжката им и се опростява кода на приложенията.

2) По-бързо изпълнение При изпълнение на блок от SQL команди СП трябва да получи само входните параметри и да върне крайния резултат. Така се намалява трафика между приложението и сървъра на БД. Освен това в СП може да се използва разширен набор от SQL команди.
3) Локално обработване на изключения При възникване на грешки при изпълнение на СП те могат да бъдат обработени в нея по подходящ начин за да се върне резултат на извикващия модул.
4) Повишена сигурност – От съображения за сигурност операциите с БД могат да се извършват само чрез съхранени процедури, в който да се контролират допълнително правата на приложенията за извикването им.

Всяка съхранените процедури и функции е привързана към определена БД както MySQL таблиците и изгледите.

Създаване на съхранени процедури и функции

Съхранените процедури и функции се създават с SQL конструкциите CREATE PROCEDURE и CREATE FUNCTION. Те има следния синтаксис:

```
CREATE PROCEDURE име ([параметри])
[характеристики]
тяло
```

и

```
CREATE FUNCTION име ([параметри])
RETURNS тип_данни
[характеристики]
тяло
```

където:

параметри - е набор от параметри на процедурата в следния формат: *[IN | OUT | INOUT] име_на_параметър тип* съответно за входен, изходен и входно-изходен параметър.

характеристики - може да съдържа няколко параметри, от които най-съществен е параметърът SQL SECURITY с възможни стойности DEFINER и INVOKER. При стойност DEFINER съхранената процедура се

	изпълнява с правата на създателя, а при стойност INVOKER - с правата на извикващия я потребител.
тип_данни	- задава типа на връщаните данни от съхранена функция.

Като пример нека създадем една съхранена процедура, която да добавя запис към таблица от БД и да прочита съдържанието на таблицата.

```
DELIMITER $$
DROP PROCEDURE IF EXISTS db1.insert_read_products $$
CREATE PROCEDURE db1.insert_read_products
(IN new_name VARCHAR(20), IN new_price FLOAT)
BEGIN
INSERT INTO products (name, price) values (new_name,new_price);
SELECT * FROM products;
END $$
DELIMITER ;
```

Пример 138 SQL код за създаване на съхранена процедура, която добавя запис към таблица и прочита всички записи

Съхранената процедура insert_read_products има два входни параметъра - new_name и new_price. Чрез тези параметри процедурата получава стойностите, които се записват в полетата name и price при добавяне на запис към таблицата с SQL команда INSERT.

Обърнете внимание на командата “DELIMITER $$”. Чрез нея се променя символа, с който трябва да завършват SQL командите. Това е необходимо, тъй като в тялото на съхранената процедура се съдържат инструкции, които трябва да завършват с “;”, но това не трябва да се приема от MySQL като край на кода, с който се създава съхранената процедура. Кода за създаване на съхранената процедура завършва с “$$” след командата END и след това с “DELIMITER ;”.се възстановява символа “;” за край на SQL командите.

Даденият по-долу пример съдържа код на PHP модул, който извиква съхранената процедура insert_read_products. Процедурата добавя запис към таблицата products от БД db1 с SQL команда

INSERT и прочита съдържанието на таблицата с SQL команда SELECT. PHP модулът извежда съдържанието на таблицата, прочетено от съхранената процедура.

```
<?php
$link = mysqli_connect("localhost", "root", "admin")
or die('Could not connect: ' . mysqli_connect_error());
$sql = "call db1.insert_read_products('keyboard',23)";
$rs=mysqli_query($link,$sql);
  if ($rs){
        $t="<table border>";
        //Цикъл през всички записи
        while($a=mysqli_fetch_assoc($rs))
        {
              $t.="<tr>\n";
              //Цикъл през всички полета от записа
              foreach($a as $val){ $t.="<td>$val</td>\n"; }
        }
        $t.="</table>"; //край на кода на HTML таблицата
        echo $t; //Извеждане на таблицата
        mysqli_free_result($rs);
        mysqli_close($link);
  }else{echo "Няма прочетени записи";}
?>
```

Пример 139 Извикване на MySQL съхранена процедура

24. Операции с бази от данни чрез ODBC

Open **D**ata**B**ase **C**onnectivity (ODBC) е програмен интерфейс (API), дефиниран от Microsoft, който предоставя унифициран достъп до различни видове бази от данни. По такъв начин ODBC предоставя ниво на абстракция между сървърите за бази от данни (БД) с техните специфични интерфейси и приложните програми, които изпълняват операции с БД.

24.1. ODBC драйвери

В архитектурата на ODBC, приложението, което извършва операции с БД се свързва към диспечера на ODBC драйвери (ODBC Manager), който от своя страна използва конкретен ODBC драйвер (например драйвер за Microsoft SQL ODBC) за свързване към източник на данни. Фирмите, които разработват системи за управление на бази от данни (СУБД), предоставят и ODBC драйвери за тях. Например MySQL AB предоставя MySQL Connector/ODBC – стандартизиран драйвер за Windows, Linux, Mac OS X, и Unix операционни системи. Очевидно е, че за PHP приложения при наличието на разширения mysql и mysqli е безсмислено да се използва ODBC драйвер за операции с MySQL база от данни. За приложения, създадени по други технологии обаче, MySQL Connector/ODBC може да бъде удачно решение за свързване с MySQL сървър.

24.2. ODBC източници на данни

В терминологията на базите от данни, източникът на данни е комбинация от специфичен набор от данни и информация, необходима за достъп до тези данни, които се представят чрез името на източника (data-source name). Информацията за свързване по правило включва местоположението на сървъра за БД, името на базата данни, име на потребител и парола и различни опции на ODBC драйвера, които описват начина на свързване към източника на данни.

Диспечера на ODBC драйверите поддържа три типа източници на данни според начина на създаването им и правата за достъп до тях: системни (машинни) източници на данни, файлови източници на данни и низове за свързване.

Машинни източници на данни

Машинните източници на данни съхраняват информация за свързването в системния регистър на Windows на конкретния

компютър. Те се създават от Control Panel-Administrative Tools с ODBC Data Source Administrator и могат да се използват само на компютъра, на който са дефинирани. Има два типа машинни източници на данни – потребителски и системни. Потребителските източници на данни могат да се използват само от текущия потребител и са видими само за него. Системните източници на данни могат да се използват от всички потребители на даден компютър и са видими за всички потребители на компютъра и общосистемните услуги. Машинният източник данни е особено полезен, когато искате да осигурите допълнителна защита на достъпа до БД, тъй като само потребителите, притежаващи акаунт на компютъра могат да видят машинен източник на данни и той не може да се копира от отдалечен потребител на друг компютър.

Файлови източници на данни

В терминологията на ODBC файловите източници на данни са текстови файлове с разширение ".dsn", които съдържат информация за свързването със сървъра и БД. За разлика от системните източници на данни, файловите източници на данни (DSN файловете) могат да бъдат копирани и използвани на друг компютър, ако на него е инсталиран необходимият ODBC драйвер.

Низове за свързване

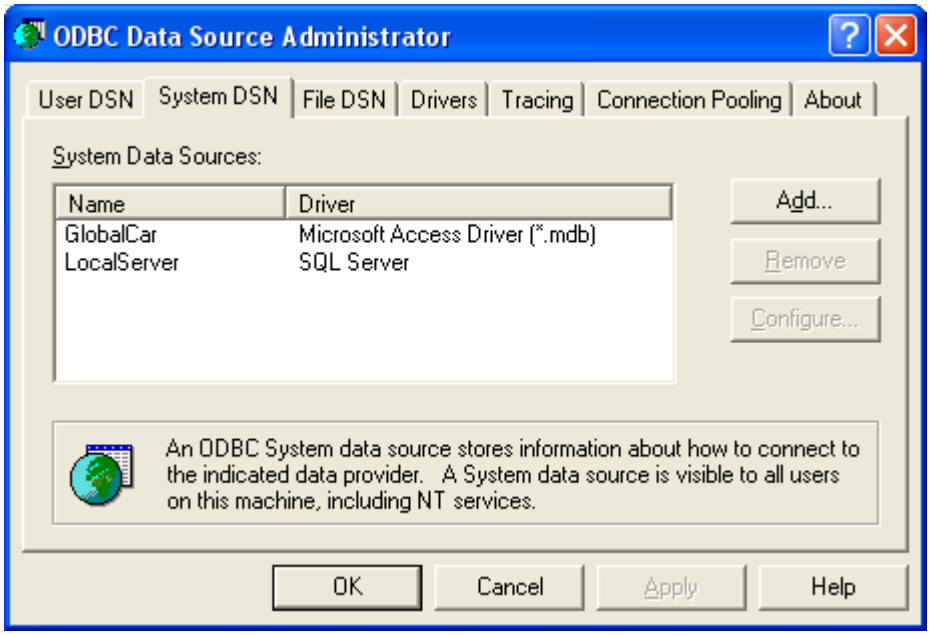
Низът за свързване е форматиран низ, който може да се предаде непосредствено на диспечера на ODBC драйверите за създаване на връзка с БД. С използването на низ за свързване се премахва необходимостта от предварително подготвяне на ODBC системен или файлов източник на данни. Информация за низовете за свързване към различни сървъри за БД можете да намерите на http://www.devlist.com/ConnectionStringsPage.aspx

24.3. Създаване на ODBC източник на данни

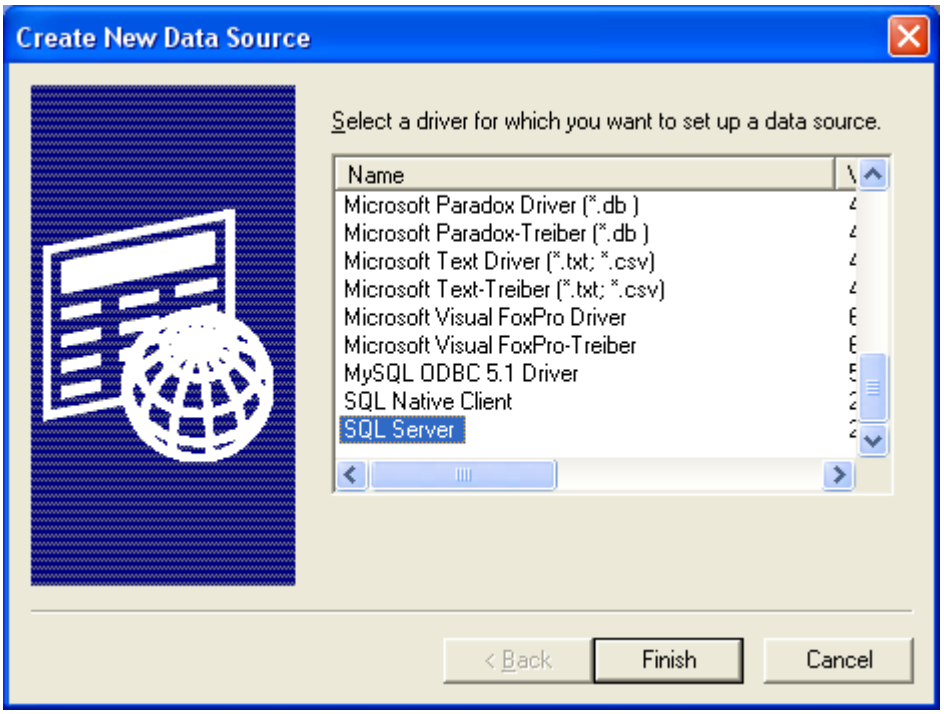
Да разгледаме един пример за създаване на системен ODBC източник на данни. Ще използваме ODBC Data Source

Administrator, който можем да стартираме от Control Panel-Administrative Tools.

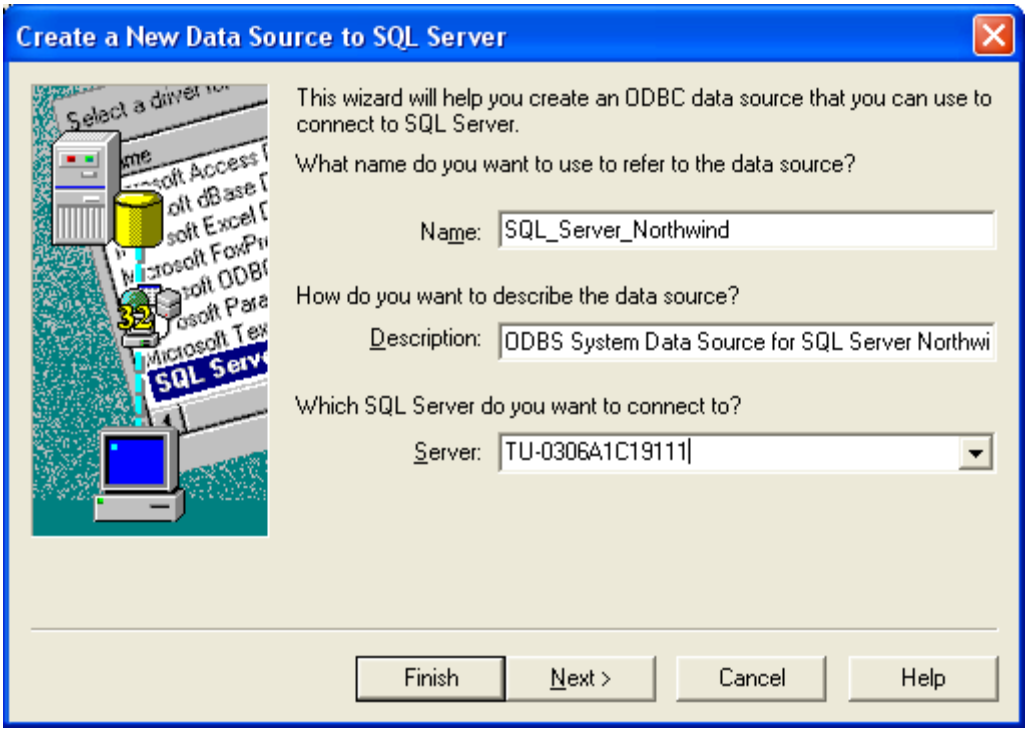
Извежда се прозореца на ODBC Data Source Administrator, от който избираме страницата “System DSN”.



За да добавим нов системен източник на данни трябва да щракнем върху бутона “Add”. Извежда се диалогова кутия със списък от инсталираните ODBC драйвери.



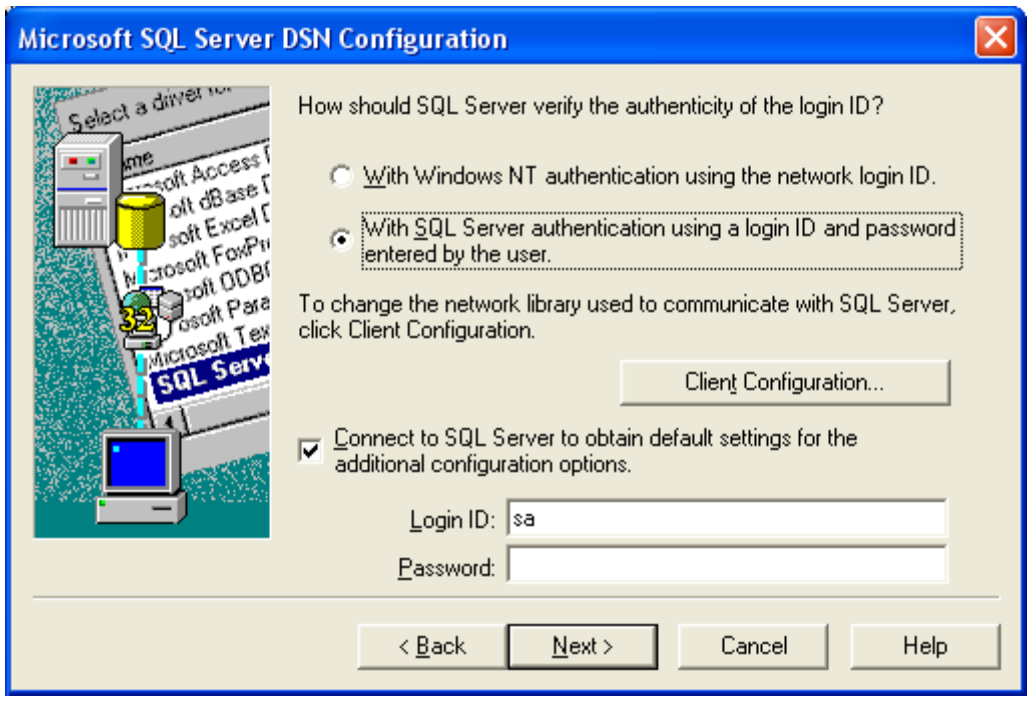
Избираме ODBC драйверът за SQL сървър. Спецификацията на .dll файла на драйвера се съдържа във файла ODBCINST.INI, който е записан в Windows директорията. След като щракнем върху бутона “Finish” се стартира помощната програма (Wizard) за създаване на ODBC системен източник на данни за SQL сървър.



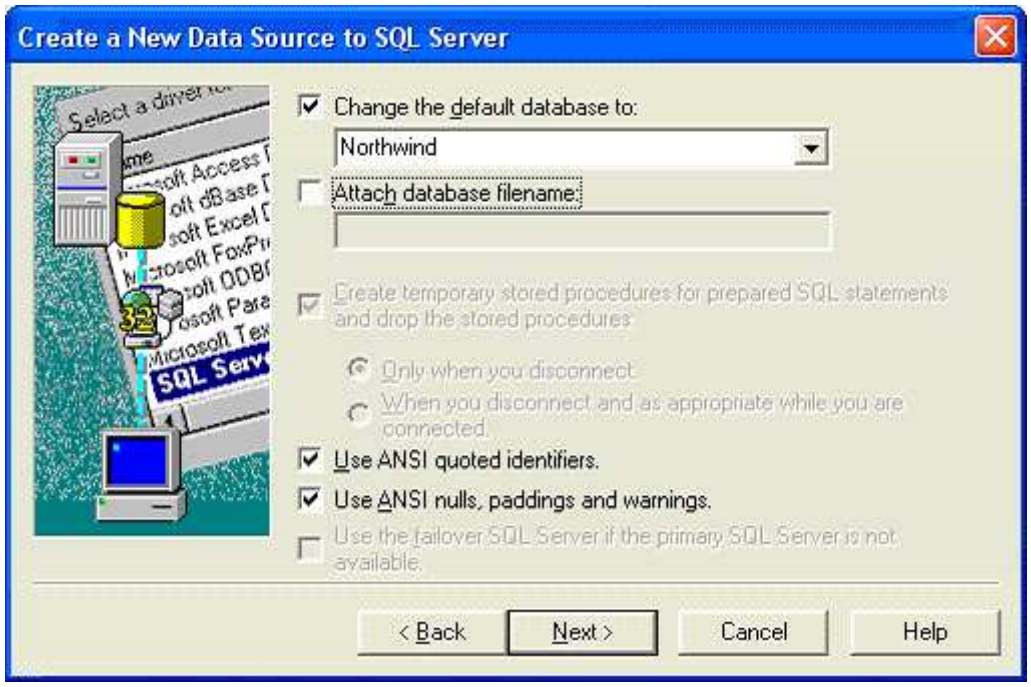
В първия прозорец от помощната програма трябва да се зададе име и описание на системния източник и да се избере или въведе името на Microsoft SQL сървъра, с който ще се създава връзка. Ако SQL сървър е инсталиран на същия компютър, на който се изпълнява PHP приложението, името на сървъра може да бъде копирано от прозореца на SQL Server Service Manager.

След като въведем данните щракваме върху бутона Next и се извежда следващият прозорец на помощната програма. Трябва да

се избере идентификацията “With SQL server authentification using login ID and password entered by user”. Ако сървърът е инсталиран

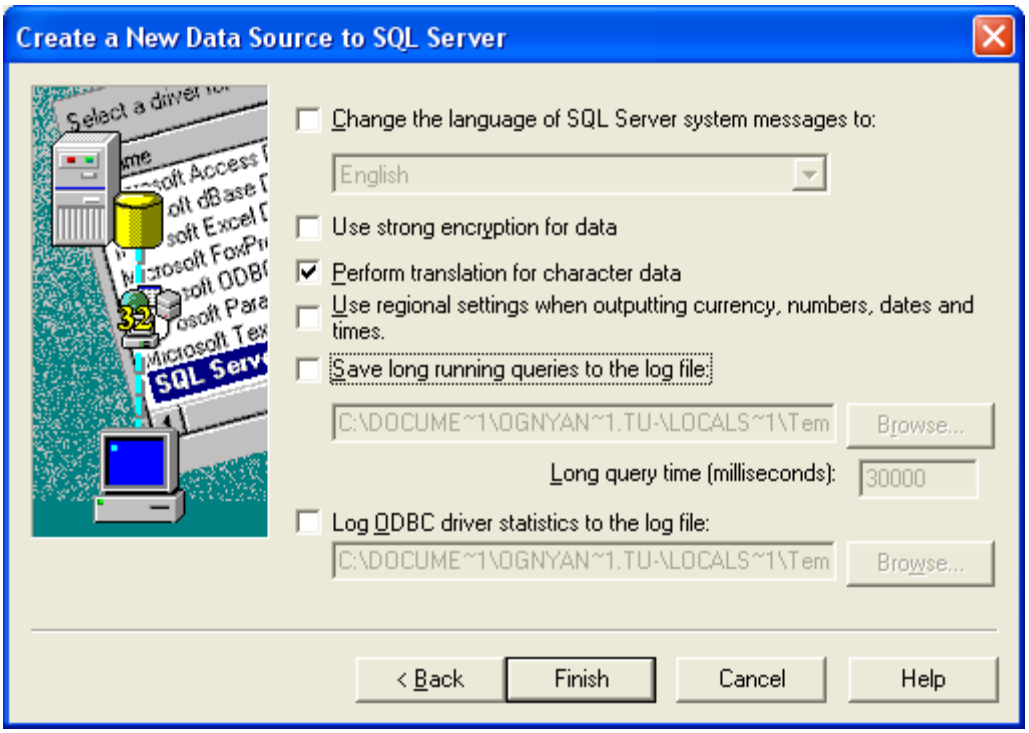


на същия компютър, на който се изпълнява PHP приложението по подразбиране се създава потребител “sa” и празна парола. Трябва да бъде избрана опцията “Connect to SQL Server to obtain default settings for the additional configuration options”. След щракване върху

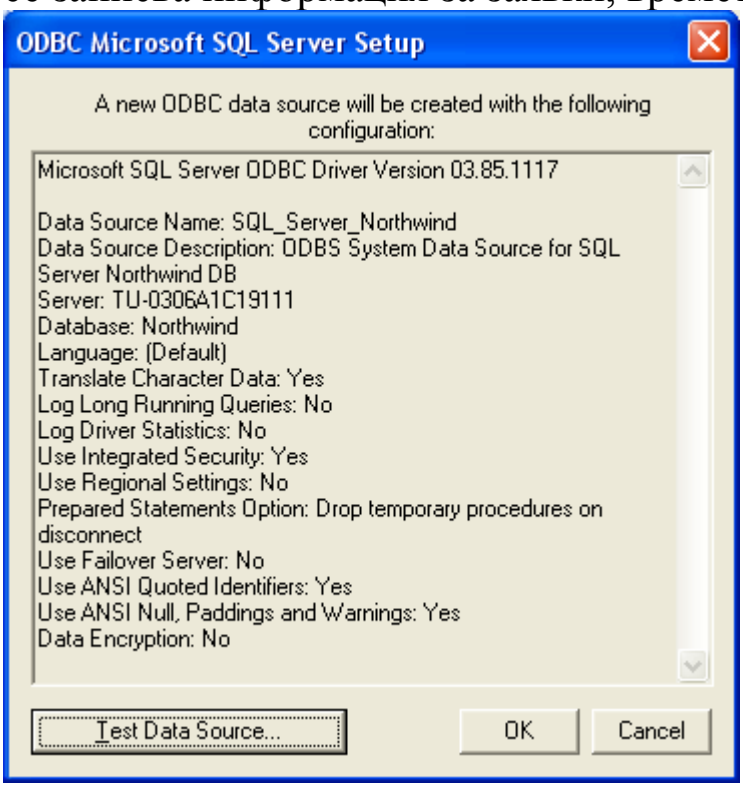


бутона “Next” се извежда следващият прозорец. В него трябва да се маркира опцията “Change the default database to:” и да се избере от

падащия списък името на БД, с която ще се извършват операции. В случая това е БД Northwind. При щракване върху бутона “Next” се извежда следващия прозорец



В него могат да се зададат спецификации на log файлове, в които да се записва информация за заявки, времето за изпълнение на които



надвишава зададена стойност, или информация от ODBC драйвера.Ако не са ви нужни записи в log файлове, можете да щракнете върху бутона “Finish” без да променяте нищо. Извежда се прозорец с информация за източника на данни, който ще бъде създаден. Можете да тествате източника на данни като щракнете върху бутона “Test Data Source”. Ако конфигурирането е правилно ще

се изведе диалогова кутия със съобщение “TESTS COMPLETED SUCCESSFULLY!”.

24.4. PHP функции за операции с БД чрез ODBC

PHP предоставя набор от функции за операции с бази от данни чрез използването на ODBC драйвер. Функциите са подобни по наименование и предназначение на функциите от разширенията mysql и mysqli за операции с MySQL бази от данни, които бяха разгледани в предната глава. Поради ограничения обем на курса не са разгледани някои функции, например функциите за подготовка и изпълнение на компилирани (параметрични) команди. Независимо от това, с разгледаният набор от функции може да бъдат изпълнени всички необходими операции чрез ODBC, тъй като те дават възможност за изпълнение на SQL команди по същия сценарий, който се използва и при операции с MySQL бази от данни.

Създаване на връзка с ODBC източник – функция odbc_connect

Функцията създава връзка с ODBC източник на данни и връща ресурс – идентификатор на източника, при успешно свързване или стойност FALSE при генериране на грешка. Тя има следното описание:

resource odbc_connect (string $dsn, string $user, string $password [, int $cursor_type])

където:

$dsn	е низ, съдържащ името на източника на данни.
$user	е името на потребителя
$password	е паролата на потребителския акаунт
$cursor_type	(незадължителен) задава типът на курсора, Приема следните стойности: SQL_CUR_USE_IF_NEEDED SQL_CUR_USE_ODBC SQL_CUR_USE_DRIVER SQL_CUR_DEFAULT

Незадължителният параметър $cursor_type обикновено не се използва, но може да бъде полезен за избягване на някои специфични грешки за определени ODBC драйвери. Например някои ODBC драйвери при изпълнение на комплексни съхранени процедури генерират грешка от вида: ”Cannot open a cursor on a stored procedure that has anything other than a single select statement in it”.

Задаването на стойност SQL_CUR_USE_ODBC на параметъра $cursor_type дава възможност да се избегне тази грешка. Също така, по подразбиране някои драйвери не поддържат незадължителния параметър, задаващ номер на записа във функцията odbc_fetch_row(). Тази функционалност също се получава със задаване на стойност SQL_CUR_USE_ODBC на параметъра $cursor_type.

Следващият пример създава връзка към база от данни на SQL сървър чрез системния ODBC източник на данни SQL_Server_db1:

```
$conn=odbc_connect('SQL_Server_db1','sa',")or die("Can't connect");
```

Пример 140 Създаване на връзка към ODBC източник на данни

Функцията odbc_pconnect има същото предназначение и набор от параметри както odbc_connect, но се различава от нея по това, че връзката с източника не се затваря след изпълнение на PHP модула. При ново извикване на функция odbc_connect или odbc_pconnect със същите параметри се използва вече създадената връзка към ODBC източника вместо да се създава нова.

Изпълнение на команда над ODBC източник – функция odbc_exec

Функцията изпълнява SQL команда и връща ресурс – идентификатор на резултат, при успешно свързване или стойност FALSE при генериране на грешка. Тя има следното описание:

resource odbc_exec (resource $connection_id, string $query_)

където

$connection_id е идентификатор на източника, получен от функция odbc_connect или odbc_pconnect

$query_string е низ, съдържащ SQL командата

Функцията изпраща SQL командата $query_string към сървъра на БД като използва идентификатора на източника $connection_id. Полученият при успешно изпълнение идентификатор на резултат се използва като параметър на функциите, с които се получават данни от източника.

Извличане на запис в масив – функция odbc_fetch_array

Функцията извлича съдържанието на запис от идентификатор на резултат, получен от изпълнение на SQL команда SELECT чрез функцията odbc_exec. Функцията има следното описание:

array odbc_fetch_array (resource $result [, int $rownumber])

където:

$result е идентификатор на резултат, който представя извадка от записи

$rownumber (незадължителен) е номера на записа, чието съдържание се извлича. Ако параметърът не е зададен се извлича текущият запис.

Функцията връща съдържанието на записа в асоциативен масив, в който ключове на елементите са имената на полетата, а стойности – стойностите на полетата от извлечения запис. След извличането на записа указателят за четене се премества на следващия запис. Ако няма повече записи за извличане, функцията връща стойност FALSE.

Да разгледаме един пример, който извежда съдържанието на таблица от БД на SQL сървър като използва ODBC източник

```
$conn=odbc_connect('SQL_Server_db1','sa','')or die("Can't connect");
echo "Connection Successful<br>";
$sql="SELECT * FROM tb1";
$rs=odbc_exec($conn,$sql) or die("Can't retrieve records
".odbc_error($conn));
//
$t="<table border>";
$title_flag=TRUE; //Флаг за извеждане на заглавен ред
```

```
while($a=odbc_fetch_array($rs)) {
        if($title_flag){
          //Извеждане на заглавен ред
          $t.="<tr>";
          //Цикъл през всички полета
          foreach($a as $key=>$value){
              $t.="<th>$key</th>";
          }
          $t.="</tr>";
          $title_flag=FALSE;
        }
       //Извеждане съдържанието на текущия запис
       $t.="<tr>";
       foreach($a as $value){
          $t.="<td>$value</td>";
       }
       $t.="</tr>";
} //край на цикъла while
$t.="</table>";
echo $t; //Извеждане на таблицата
odbc_free_result($rs);
odbc_close($conn);
```

Пример 141 Извеждане на съдържанието на таблица от ODBC източник

В примера се създава връзка към БД на SQL сървър чрез ODBC източника, 'SQL_Server_db1' , чието получаване беше разгледано в началото на тази глава. Потребителският акаунт се идентифицира с име ‘sa’ и празна парола ‘’. Чрез функцията odbc_exec се изпълнява SQL команда SELECT, с която се извличат всички записи от таблицата tb1 от базата от данни db1 на източника.В цикъл while се извличат чрез odbc_fetch_array последователно записите и съдържанието им се записва в кода на HTML таблица, който се натрупва в променливата $t. След извличане на първия запис първо в HTML таблицата се записва заглавен ред с имената на полетата.

Получаване на информация за грешка – функция odbc_error

Функцията връща информация (шестнадесетичен номер) на последната генерирана грешка при изпълнение на операция над ODBC източник на данни. Тя има следното описание:

string odbc_error ([resource $connection_id])

където незадължителният параметър $connection_id трябва да съдържа идентификатор на връзката. Ако параметърът е зададен, функцията връща последната генерирана грешка при операция с тази връзка, а ако липсва – последната генерирана грешка при операция с кой да е ODBC източник,

Освобождаване на паметта от извадката – функция odbc_free_result

Функцията освобождава оперативната памет, заета от извадката, получена при изпълнение на SQL команда SELECT чрез ODBC източник. Това има смисъл да се прави само ако извадката (представена от идентификатора на резултат) заема твърде много памет. В противен случай това не е необходимо, тъй като след завършване на изпълнението на PHP модула паметта, заета от извадки от записи се освобождава. Все пак признак за добър стил на програмиране е след като дадена извадка от записи вече не е необходима, да се извика функцията odbc_free_result за да се освободи паметта, заета от нея. Функцията има следното описание:

bool odbc_free_result (resource $result_id)

където $result_id е идентификатора на резултата, който представя извадката

Ако функцията се извика в незавършена транзакция автоматично се отказват всички изменения (извършва се ROLLBACK на транзакцията).

Затваряне на връзката към ODBC източник – функция odbc_close

Функцията затваря връзката сървъра за БД асоцииран със зададен идентификатор на връзка. Тя има следното описание:

void odbc_close (resource $connection_id)

където $connection_id е идентификатор на източника, получен от функция odbc_connect или odbc_pconnect

Функцията ще върне грешка, ако с връзката, представена от $connection_id, е била стартирана транзакция, която не е завършена. В този случай връзката остава отворена.

25. Операции с бази от данни в ОС Windows чрез ADO

При изпълнение в ОС Windows PHP предоставя класа COM, който дава възможност за създаване на COM обекти от ActiveX библиотеки, инсталирани на Web сървъра. Така в ОС Windows PHP модулите могат да използват функционалността на COM обектите, и в частност добре отработената ADO (ActiveX Data Objects) технология за операции с бази от данни.

25.1. Създаване на COM обекти – класът COM

Класът COM дава възможност за създаване на екземпляр на OLE съвместим COM обект и достъп до неговите атрибути и методи:

Конструктор:

com COM::COM (string module_name [, mixed server_name [, int codepage [, string typelib]]])

където:

module_name е програмен идентификатор (ProgID) на класа. Програмният идентификатор се задава като символен низ, който съдържа името на библиотеката, разделител “.” и името на класа, от който се създава обекта, напр. Word.Application

server_name (незадължителен) е името на DCOM сървър на който обектът ще се зареди и ще се изпълнява. При стойност NULL, обектът се изпълнява на подразбиращият се сървър – по правило на Web

	сървъра, на който се изпълнява PHP модула
codepage	(незадължителен) задава кодовата страница, която се използва при преобразуване на низове в unicode и обратно. Това преобразуване се прави когато PHP обменя символни низове с COM обекта.
typelib	(незадължителен) задава спецификация на .tlb файл, който съдържа типова библиотека за COM обекта.

Ще разгледаме един пример, в който се използва обекта Connection от библиотеката ADO за извеждане на съдържанието на таблица от БД на Access.

```
<?php
//Създаване на обект Connection от библиотека ADO
$conn = new COM("ADODB.Connection") or die("Cannot start ADO");
//define connection string, specify database driver
$connStr = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source= D:\\
db1.mdb";
  $conn->open($connStr); //Създаване на връзка към БД
//Извличане на записите от таблицата с SQL команда SELECT
$rs = $conn->execute("SELECT * FROM Address");
$num_columns = $rs->Fields->Count( ); //Получаване на броя полета
echo "<table border>";
//Цикъл през всички записи от извадката
while (!$rs->EOF)
{
   echo "<tr>";
   for ($i=0; $i < $num_columns; $i++) {
      echo "<td>" . $rs->Fields($i)->value . "</td>";
   }
   echo "</tr>";
   $rs->MoveNext(); //Преместване на следващия запис
}
echo "</table>";
// Затварят се обектите Recordset и Connection
$rs->Close();
$conn->Close();
// Освобождаване на ресурсите
```

```
$rs = null;
$conn = null;
?>
```

Пример 142 Извеждане на съдържанието на БД на Access чрез обект ADODB.Connection

В примера чрез конструктора на класа COM се създава обект Connection от библиотека ADODB. Чрез метода execute на обект Connection се изпълнява SQL команда SELECT, чрез която се извличат записите от таблицата Address на БД db1. Методът връща обект Recordset, който представя динамичната таблица, съдържаща извлечените записи.

Атрибутът Fields на обект Recordset връща колекция от обекти Field, които представят полетата в извлечените записи. Методът Count() на колекцията връща броя на полетата в един запис.

Атрибутът EOF на обект Recordset връща логическа истина (TRUE) когато указателят на текущия запис на Recordset е след последния запис в динамичната таблица. Поради това инвертираната стойност на атрибута се използва като условие за изпълнение на цикъла while(!$rs->EOF), с който се извършва итерация на извлечените записи. С вътрешния цикъл for с индекс $i се извършва итерация през полетата на текущия запис, като стойностите на полетата се извеждат в клетки от реда на HTML таблицата.

След извеждането на съдържанието на таблицата Address се затваря обекта Recordset и връзката към БД и се освобождават ресурсите, който те използват. Това всъщност е необходимо ако програмистът иска да използва създадения обект Connection за създаване на връзка към друг източник на данни. В противен случай, дори и да не се изпълнят методите Close() на Recordset и Connection след приключване на изпълнението на PHP модула връзката се затваря и обектите се унищожават.

Методът Execute на обект ADODB.Connection дава възможност за изпълнение и на SQL команди, които не извличат записи, като INSERT, DELETE и UPDATE, следователно чрез този метод може да се извършва и актуализация на данните в таблиците от БД.

Детайлното разглеждане на библиотеката ADO излиза извън рамките на настоящия курс. Полезно е да се отбележи, че основният обект за изпълнение на SQL команди в библиотеката е обектът Command, чрез който могат да се създават бази от данни и да се изпълняват параметрични заявки и съхранени процедури с параметри.

Използвана литература

1. Анди Гутманс, Стиг Баккен, Дерик Ританс, PHP 5 Професионално програмиране, Софтпрес, 2005
2. Колектив на НИСОФТ, PHP в лесни стъпки, “НИСОФТ”, С. , 1998
3. MySQL Documentation Team, MySQL 5.5 Reference Manual
4. Пол Дюбоа, Стефан Хинц, Карстен Педерсън , MySQL 5 Официално ръководство за сертифициране. “СОФТПРЕС”., 2006г.
5. О. Железов, Б. Градинарова, Д. Илиева, Web Дизайн, печ. база на ТУ-Варна, 2006г.
6. Ерик Нюкъмър, Web услуги XML, WSDL, SOAP, UDDL, , СофтПрес, 2004

Информация в интернет сайтове:

1. http://www.php.net - Официален сайт на разработчиците на PHP.
2. http://www.w3schools.com/php Сайт с учебни материали за PHP
3. http://www.tizag.com/phpT/ Сайт с учебни материали за PHP
4. http://www.freewebmasterhelp.com/tutorials/phpmysql Сайт с учебни материали за PHP и MySQL
5. http://www.heidisql.com/ Безплатна програма за администриране и поддържане на MySQL бази от данни
6. http://www.mysql.com - Официален сайт на разработчиците на MySQL.
7. http://www.sunvirtual.com/internet_resource_links/mysql_database_tools_resources.htm Ресурси и помощни средства за MySQL
8. http://www.sqlite.org/lang.html SQLite документация
9. http://www.dlldll.com/ - Адрес за изтегляне на .dll файлове за разширения на php
10. http://www.myphpquiz.com/question.php Online Test

Web сайт с учебни материали, поддържан от автора:
http://rusalka-bg.com/dt