Introducere în *PHP* şi MySQL

Obiectivul acestui curs este de a învăța cum să creem site-uri web dinamice, în concordanță cu cerințele actuale. Site-urile web pot fi dezvoltate utilizând *HTML* simplu, însă această abordare are foarte multe limitări, deoarece conținutul unui astfel de site este static, nemodificându-se decât dacă îl actualizăm fizic; utilizatorii nu pot interacționa cu site-ul în mod semnificativ.

Utilizarea unui limbaj precum *PHP* şi a unei baze de date (*MySQL*) permite obținerea unor site-uri web dinamice, care pot fi personalizate şi conțin informație în timp real.

Ce este *PHP*?

PHP este un limbaj de tip *scripting* pe partea de server, proiectat special pentru Web. Într-o pagină *HTML* poate fi încapsulat cod *PHP* care va fi executat de fiecare dată când pagina este vizitată. Codul *PHP* este interpretat pe serverul web şi generează cod *HTML* (sau un alt output) care va fi văzut de către vizitator.

PHP a fost proiectat în 1994 şi, la origine, a fost contribuția unui singur om, Rasmus Lendorf. Apoi, limbajul a fost adoptat şi de către alți specialişti şi a trecut prin 4 rescrieri majore care l-au adus la versiunea complexă de astăzi. În noiembrie 2007, era instalat pe mai mult de 21 milioane de domenii din lumea întreagă, număr aflat în continuă creştere.

PHP este un proiect *open source*, ceea ce înseamnă că avem acces la codul sursă şi îl putem utiliza, modifica şi redistribui gratuit.

La început, acronimul *PHP* reprezenta *Personal Home Page*, dar a fost modificat conform convenției recursive de numire *GNU* (Gnu's Not Unix) şi acum reprezintă *PHP Hypertext Processor*.

Versiunea 7 este versiunea curentă majoră a limbajului *PHP*.

Ce este *MySQL*?

MySQL este un *SGBD* relațional recunoscut pentru rapiditate şi robustețe. Serverul *MySQL* controlează accesul concurent la date de către mai mulți utilizatori, care trebuie să fie autorizați pentru a obține accesul respectiv. Prin urmare, serverul *MySQL* este *multiuser* şi *multi threaded*. El utilizează *SQL*, limbajul standard de gestiune a bazelor de date.

MySQL este disponibil din 1996, însă dezvoltarea sa are originile în anul 1979. Este cel mai popular *SGBD open source*.

MySQL este disponibil sub o schemă de licențiere dublă:

- se poate utiliza sub licență *open source* (*GPL*) în mod gratuit dacă sunt acceptați termenii licenței respective;
- dacă se doreşte distribuirea unei aplicații non-*GPL* care include *MySQL*, poate fi achiziționată o licență comercială.

De ce se utilizează *PHP* şi *MySQL*?

Pentru crearea unui site web pot fi utilizate mai multe produse. În privința deciziilor care trebuie luate, trebuie ales:

- hardware pentru serverul web
- sistemul de operare
- software pentru serverul web
- un *SGBD*
- un limbaj de programare sau de *scripting*.

Unele dintre aceste decizii sunt dependente de celelalte (nu toate sistemele de operare rulează pe orice tip de hardware, nu toate serverele web suportă toate limbajele de programare etc.).

Una dintre trăsăturile importante atât ale *PHP* cât şi *MySQL* este aceea că rulează pe orice sistem de operare important şi multe dintre cele secundare.

Majoritatea codului *PHP* poate fi scris astfel încât să fie portabil între sistemele de operare şi serverele web. Există câteva funcții *PHP* care se referă la sistemul de fişiere şi sunt dependente de sistemul de operare.

Caracteristici ale *PHP*

Principalii concurenți ai *PHP* sunt *Perl*, *Microsoft ASP.NET*, *Ruby*, *Java Server Pages* (*JSP*) şi *ColdFusion*.

Comparativ cu aceste produse, *PHP* are multe avantaje, inclusiv:

- performanță;

- scalabilitate;
- interfețe cu multe *SGBD*-uri;
- biblioteci predefinite pentru multe *task*-uri web comune;
- cost redus;
- facilitatea învățării şi utilizării;
- suport pentru orientarea pe obiecte;
- portabilitate;
- flexibilitate în dezvoltare;
- disponibilitatea codului sursă;
- disponibilitatea suportului şi a documentației.

Performanță. *PHP* este foarte rapid. Utilizând un singur server, necostisitor, putem servi milioane de accesări pe zi.

Scalabilitate. Aşa cum Rasmus Lendorf a spus, *PHP* are o arhitectură „*shared-nothing*". Aceasta presupune că se poate implementa scalarea orizontală cu un număr mare de servere, în mod eficient şi ieftin.

Integrarea bazelor de date. *PHP* are posibilități native de conectare la multe sisteme de baze de date. Pe lângă *MySQL*, se poate conecta direct la *PostgreSQL*, *Oracle*, *DB2*, *Informix*, *InterBase*, *Sybase* etc. Utilizând standardul *ODBC* (*Open Database Connectivity*) ne putem conecta la orice bază de date care furnizează un driver *ODBC*. Aceasta include, printre altele, produsele *Microsoft*.

Pe lângă bibliotecile native, *PHP* vine cu un nivel de abstractizare a accesului la baze de date denumit *PHP Database Objects* (*PDO*), care permite accesul consistent şi promovează practici sigure de scriere a codului.

Biblioteci predefinite. Deoarece *PHP* a fost proiectat pentru *Web*, deține multe funcții *built-in* pentru efectuarea multor *task*-uri referitoare la web. Se pot genera imagini, realiza conexiuni la servicii web şi alte servicii în rețea, parsa cod *XML*, trimite *email*-uri, lucra cu *cookie*-uri şi genera documente *PDF*, toate acestea scriind doar câteva linii de cod.

Cost. *PHP* este gratuit, ultima versiune poate fi descărcată de la adresa http://www.php.net.

Uşurința învățării PHP. Sintaxa *PHP* se bazează pe cea a altor limbaje de programare, în principal *C* şi *Perl*.

Suport pentru orientarea pe obiecte. Versiunea 5 a *PHP* are caracteristici OO bine proiectate: moştenire, atribute şi metode *private* şi *protected*, clase abstracte şi metode, interfețe, constructori şi destructori. De asemenea, există şi câteva trăsături mai puțin comune, cum ar fi iteratorii.

Portabilitate. *PHP* este disponibil pentru multe sisteme de operare: sisteme gratuite *Unix-like* (*Linux*, *FreeBSD*), versiuni *Unix* comerciale (*Solaris*, *IRIX*, *OS X*) sau pe diferite versiuni de *Microsoft Windows*. Un cod *PHP* bine scris va rula fără modificări pe un sistem diferit, care rulează *PHP*.

Flexibilitate în dezvoltare. *PHP* permite implementarea *task*-urilor simple cu uşurință, şi în egală măsură se adaptează la implementarea aplicațiilor mari care utilizează un *framework* bazat pe modele de proiectare (*design patterns*) cum ar fi *Model-View-Controller* (MVC).

Codul sursă. Codul sursă al *PHP* poate fi accesat; este permis să realizăm modificări sau adăugări în limbaj.

Disponibilitatea suportului şi a documentației. Compania din spatele motorului *PHP*, *Zend Technologies* (www.zend.com) oferă suport comercial şi produse software înrudite. Documentația şi comunitatea *PHP* constituie resurse consistente şi mature de informații.

Caracteristici ale MySQL

Principalii concurenți ai *MySQL* sunt *PostgreSQL*, *Microsoft SQL Server* şi *Oracle*. Dintre avantajele *MySQL*, enumerăm:

- performanța ridicată;
- costul scăzut;
- uşurința configurării şi învățării;
- portabilitatea;
- disponibilitatea codului sursă;
- disponibilitatea suportului.

Performanță. Referințele dezvoltatorilor arată că MySQL este cu mult mai rapid decât competitorii săi (http://web.mysql.com/whymysql/benchmarks).

Cost scăzut. *MySQL* este disponibil gratuit sub licență *open source* sau la un cost scăzut sub licență comercială. Este necesară această licență dacă dorim

redistribuirea lui *MySQL* ca parte a unei aplicații. Cele mai multe aplicații web nu trebuie distribuite, astfel încât nu este necesară licența.

Uşurința folosirii. Majoritatea bazelor de date moderne folosesc *SQL*, deci adaptarea la cel implementat în *MySQL* nu ar trebui să fie o problemă. De asemenea, *MySQL* este mai simplu de configurat decât produsele similare.

Portabilitate. *MySQL* poate fi folosit pe diferite sisteme *Unix* şi *Microsoft Windows*.

Codul sursă. Similar limbajului *PHP*, codul sursă al *MySQL* poate fi accesat şi modificat.

Disponibilitatea suportului. Spre deosebire de alte produse *open source*, *MySQL AB* (www.mysql.com) oferă suport, training, consultanță şi certificare.

Limbajul PHP

Pentru început, vom prezenta pe scurt sintaxa şi construcțiile de programare ale limbajului. De asemenea, vom introduce primele exemple pentru a arăta modul în care sunt utilizate variabilele, operatorii şi expresiile în PHP.

Crearea unei aplicații exemplu: magazin de componente auto

Una dintre cele mai comune aplicații ale oricărui limbaj de *scripting* pe partea de server este procesarea formularelor *HTML*. Vom implementa un formular de comandă pentru o companie de piese de schimb pentru maşini. Pe lângă comenzile propriu-zise, proprietarul doreşte să determine prețul total al comenzilor şi taxele aferente acestora.

Inițial, formularul de comandă are următorul conținut: http://193.226.51.37/web/lab1/exemplu1.html

Etapele pentru crearea acestei aplicații sunt scrierea formularului *HTML* şi procesarea acestui formular.

1. Codul *HTML* parțial pentru acest formular se găseşte la adresa: http://193.226.51.37/web/lab1/exemplu1.txt

Observații:

- Acțiunea formularului referă scriptul *PHP* care va procesa comanda clientului (*processorder.php*). În general, valoarea atributului *action* este *URL*-ul care va fi încărcat atunci când utilizatorul acționează butonul *Submit*. Datele pe care utilizatorul le-a introdus în acest formular vor fi trimise către *URL*-ul respectiv prin metoda specificată în atributul *method*, care poate fi *get* (adăugare la sfârşitul *URL*-ului) sau *post* (trimitere ca mesaj separat).
- Numele câmpurilor din formular (*tireqty*, *oilqty*, *sparkqty*) vor fi folosite şi în script-ul *PHP*. Utilizați nume cât mai sugestive! Se recomandă adoptarea unor convenții de scriere a codului, care să prevadă un anumit format pentru numele câmpurilor utilizate în site.

2. Pentru procesarea formularului, trebuie creat scriptul menționat în cadrul *atributului action* din *tag*-ul *form*, *processorder.php*. Codul corespunzător primei versiuni a acestui script se află la adresa: http://193.226.51.37/web/lab1/processorder1.txt

Înlocuind extensia .txt cu .php, vom obține următorul rezultat: http://193.226.51.37/web/lab1/processorder1.php

3. Încapsularea *PHP* în *HTML*.

Sub *heading*-ul <h2> din fişier, adăugăm următoarele linii:

```
<?php
   echo '<p>Order processed.</p>';
?>
```

Fişierul devine: http://193.226.51.37/web/lab1/processorder2.txt

Invocarea acestui fişier va conduce la afişarea următorului formular: http://193.226.51.37/web/lab1/processorder2.php

Observație:

- Dacă vizualizăm sursa paginii în browser, vom observa că nu sunt vizibile liniile *PHP* deoarece interpretorul *PHP* a rulat scriptul şi l-a înlocuit cu *output*-ul acestuia. Aceasta înseamnă că, din *PHP*, putem produce cod *HTML* care poate fi vizualizat cu orice *browser*; cu alte cuvinte, *browser*-ul utilizatorului nu trebuie să compileze *PHP*.

Exemplul anterior ilustrează conceptul de *scripting* pe partea de server. Codul *PHP* a fost interpretat şi executat pe serverul web, spre deosebire de *JavaScript* şi alte tehnologii *client-side* interpretate şi executate în cadrul unui *browser* web pe maşina clientului.

În acest moment, codul pe care îl avem în fişier este de următoarele 4 tipuri:

- *HTML*
- *tag*-uri *PHP*
- instrucțiuni *PHP*
- spații libere.

De asemenea, pot fi adăugate comentarii.

Tag*-urile *PHP sunt <?php şi ?>. Acestea anunță serverul web unde începe şi se termină codul *PHP*, iar orice text dintre aceste 2 *tag*-uri este interpretat ca *PHP*. Textul din afara *tag*-urilor *PHP* este tratat ca *HTML* normal.

Există 4 stiluri diferite de *tag*-uri *PHP*. Următoarele fragmente de cod sunt echivalente:

- stilul XML:

```
<?php echo '<p>Order processed.</p>'; ?>
```

Acesta este stilul de tag *PHP* recomandat. Administratorul serverului nu îl poate dezactiva, deci se poate garanta că va fi disponibil pe toate serverele. Acest stil de tag poate fi utilizat în documentele XML.

- stilul prescurtat:

```
<? echo '<p>Order processed.</p>'; ?>
```

Pentru a utiliza acest stil, este nevoie fie să activăm setarea *short_open_tag* în fişierul de configurare, fie să compilăm *PHP* cu *short tag*-urile activate. Stilul nu este activat implicit!

- stilul *script*:

```
<script language='php'>
   echo '<p>Order processed.</p>'; </script>
```

Stilul este familiar celor care au utilizat *JavaScript* sau *VBScript*. Poate fi folosit dacă editorul *HTML* are probleme cu celelalte stiluri.

- stilul *ASP*:

```
<% echo '<p>Order processed.</p>'; %>
```

Poate fi folosit dacă a fost activată setarea de configurare *asp_tags*. Implicit, stilul este dezactivat.

Observații:

- Instrucțiunile PHP se scriu între tag-urile PHP şi sunt separate prin „;”. Un prim exemplu-, apărut mai sus, de instrucțiune PHP este *echo*.
- Spațiile libere (*whitespaces*) includ *newline*, *blank*-uri şi *tab*-uri. *Browser*-ele ignoră aceste spații în *HTML*, iar motorul *PHP* procedează similar. Aceste spații sunt utile pentru a obține un cod sursă mai lizibil.
- Comentariile în *PHP* pot fi în stil *C* (/* */), *C++* (//) sau *shell script* (#).

Adăugarea conținutului dinamic

Motivul principal pentru care utilizăm un limbaj de *scripting* pe partea de server este furnizarea de conținut dinamic pentru utilizatorii unui site.

De exemplu, modificăm codul *PHP* din *processorder1.php* astfel încât să obținem data şi ora la care a fost procesată comanda: http://193.226.51.37/web/lab1/processorder3.txt

Rezultatul este: http://193.226.51.37/web/lab1/processorder3.php

Observații:

- Operatorul de concatenare este „.”.
- Funcția predefinită *date*() acceptă ca argument un şir de caractere ce reprezintă formatul datei. Dintre formatele existente, amintim: *H* este ora în formatul cu 24 de ore, *i* reprezintă numărul de minute cu „0” la început, dacă este necesar, *j* este ziua din lună fără „0” la început, *s* reprezintă sufixul ordinal (*th*), *F* este numele întreg al lunii.

Accesarea variabilelor din formulare

Scopul utilizării formularelor este de a obține informații furnizate de către clienți. Obținerea acestor detalii este simplă în *PHP*, dar metoda exactă depinde de versiunea de *PHP* utilizată şi de o setare din fişierul *php.ini*.

Variabile

În cadrul *script*-ului *PHP*, fiecare câmp poate fi accesat ca o variabilă *PHP* al cărei nume este în legătură cu numele câmpului respectiv. Numele variabilelor sunt uşor de recunoscut, deoarece încep cu semnul „$”.

În funcție de versiunea de *PHP* şi de configurare, datele din formular pot fi accesate în 3 moduri. Pentru a accesa conținutul câmpului *tireqty*, aceste modalități sunt:

- *$tireqty*
- *$_POST['tireqty']*
- *HTTP_POST_VARS['tireqty']*

Pentru a alege una dintre cele 3 modalități, trebuie avute în vedere următoarele considerente:

- Stilul „scurt” (*$tireqty*) este convenabil, însă necesită ca setarea de configurare *register_globals* să fie activată. Din motive de securitate, această setare este implicit dezactivată. Acest stil facilitează introducerea unor erori care determină insecuritatea codului, motiv pentru care nu este o abordare recomandată (pe lângă aceasta, se pare că opțiunea va dispărea într-o versiune viitoare).
- Stilul „mediu” (*$_POST['tireqty ']*) constituie abordarea recomandată. Dacă vom crea versiuni scurte ale numelor variabilelor, pe baza stilului „mediu”, nu mai apare o problemă de securitate.
- Stilul „lung” (*$HTTP_POST_VARS['tireqty']*) oferă cele mai multe informații. Oricum, acest stil este depreciat şi probabil că va fi eliminat ulterior. Acest stil a fost cel mai portabil, însă acum poate fi dezactivat prin intermediul directivei de configurare *register_long_arrays*, care îmbunătățeşte performanța.

Revenind la stilul „mediu”, acesta presupune regăsirea variabilelor din formular într-unul dintre tablourile *$_POST*, *$_GET* sau *$_REQUEST*. Unul dintre tablourile *$_GET* sau *$_POST* reține detalii despre toate variabilele din formular. Tabloul folosit depinde de tipul metodei de trimitere a

informațiilor din formular (*GET* sau *POST*). O combinație a tuturor datelor trimise prin *GET* sau *POST* se află în tabloul *$_REQUEST*.

Aceste tablouri sunt superglobale.

Pentru uşurința folosirii, se pot crea copii ale variabilelor utilizând operatorul de atribuire (=). Următoarea instrucțiune creează o nouă variabilă denumită *$tireqty* în care este copiat conținutul lui *$_POST['tireqty ']*:

```
$tireqty = $_POST['tireqty'];
```

Pentru a trata cu mai multă uşurință datele dintr-un formular, se poate plasa la începutul *script*-ului un fragment de cod de forma următoare:

```
<?php
// crearea variabilelor “scurte”
$tireqty = $_POST['tireqty'];
$oilqty = $_POST['oilqty'];
$sparkqty = $_POST['sparkqty '];
?>
```

Pentru ca script-ul să producă un rezultat vizibil, afişăm valorile variabilelor în script (http://193.226.51.37/web/lab1/processorder4.txt). Rezultatul este vizibil rulând o versiune modificată a primului script (exemplu1.html) care permite selectarea formularului de procesare a comenzii (deocamdată, 1-4): http://193.226.51.37/web/lab1/exemplu2.php (cod sursă disponibil în exemplu2.txt).

Concatenarea şirurilor de caractere

Aşa cum am menționat într-o observație anterioară, operatorul de concatenare este „.”.

Variabilele simple pot fi plasate şi în interiorul şirurilor situate între ghilimele("). De exemplu instrucțiunea:

```
echo "$tireqty tires<br />";
```

este echivalentă cu cea din script-urile anterioare.

Procesul de înlocuire a conținutului unei variabile într-un şir de caractere este cunoscut sub numele de interpolare. Aceasta este o trăsătură a şirurilor de caractere plasate între ghilimele, nu apostrofuri!

Variabile şi literale

Variabilele *simbolizează* date, iar şirurile de caractere *sunt* date. Atunci când datele însele sunt utilizate ca atare în programe, ele se numesc

literale, spre a fi deosebite de variabile. *$tireqty* este o variabilă, iar *' tires
 '* este un literal.

Avem două tipuri de şiruri de caractere: cu ghilimele şi cu apostrofuri. *PHP* încearcă să evalueze şirurile de caractere dintre ghilimele, aşa cum am arătat mai sus. Spre deosebire de acestea, şirurile de caractere aflate între apostrofuri sunt tratate ca literale.

Mai există un al treilea mod de specificare a şirurilor de caractere utilizând sintaxa *heredoc* (<<<), familiară utilizatorilor *Perl*. Aceasta permite specificarea şirurilor lungi de caractere, adăugând un *marker* de final care va fi utilizat pentru a încheia şirul.

Următorul exemplu creează şi afişează un şir de caractere de 3 linii:

```
echo <<<theEnd
line 1
line 2
line 3
theEnd
```

Similar şirurilor între apostrofuri, şi cele *heredoc* pot fi interpolate.

Identificatori

Identificatorii sunt nume de variabile, funcţii şi clase.

Regulile pentru definirea de identificatori valizi sunt următoarele:

- Identificatorii pot avea orice lungime şi pot conţine litere, cifre şi *underscore*.
- Identificatorii nu pot începe cu o cifră.
- În *PHP*, identificatorii sunt *case-sensitive*. Funcţiile fac excepţie de la această regulă, numele lor putând fi scrise oricum.
- O variabilă poate avea acelaşi nume cu o funcţie (nu se recomandă acest lucru, deoarece poate produce confuzie).

Pe lângă variabilele corespunzătoare formularelor *HTML*, pot fi declarate şi utilizate propriile variabile.

O trăsătură a limbajului *PHP* este că nu necesită ca variabilele să fie declarate înainte de a fi utilizate. O variabilă este creată atunci când i se atribuie prima dată o valoare.

Presupunem că dorim să se calculeze numărul de articole comandate şi totalul de plată. Putem crea două variabile care să stocheze aceste valori, pe care le iniţializăm cu 0 la începutul script-ului *PHP*:

```
$totalqty = 0;
$totalamount = 0.00;
```

Tipurile de date ale variabilelor

PHP oferă următoarele tipuri de date de bază:

- *Integer* – numere întregi
- *Float* (double) – numere reale
- *String* – şiruri de caractere
- *Boolean* – valori logice
- *Array* – stochează date multiple
- *Object* – stochează instanțe ale claselor.

De asemenea, sunt disponibile două tipuri speciale: *NULL* şi *resource*.

Variabilele cărora nu le-a fost atribuită o valoare sau care au primit valoarea specifică *NULL* sunt de tip *NULL*.

Anumite funcții predefinite (de exemplu, funcțiile referitoare la baza de date) returnează variabile al căror tip este *resource*. Acestea reprezintă resurse externe(cum ar fi conexiunile la baza de date). O astfel de variabilă nu va fi prelucrată direct; de obicei, acestea sunt returnate de anumite funcții şi trebuie transmise ca parametri altor funcții.

Puterea tipurilor de date

PHP este un limbaj cu tipuri de date dinamice (sau slabe). În majoritatea limbajelor de programare, variabilele pot reține un singur tip de date, iar tipul de date trebuie declarat înainte ca variabila să îl poată folosi. În *PHP*, tipul unei variabile este determinat de variabila atribuită lui.

În atribuirile precedente, *$totalqty* este o variabilă de tip întreg, iar *$totalamount* este o variabilă de tip real. Pe de altă parte, se poate adăuga ulterior următoarea linie în script:

```
$totalamount = 'Hello';
```

Variabila *$totalamount* va avea apoi tipul de date *string*. *PHP* modifică tipul de date al variabilei în funcție de ceea ce reține respectiva variabilă la un moment dat.

Conversia tipurilor de date

Putem considera că o variabilă sau valoare este de tip diferit utilizând conversia tipurilor de date. Această caracteristică lucrează identic modului existent în *C*. Tipul de date temporar este plasat între paranteze în fața variabilei.

De exemplu, putem avea următoarele declarații:

```
$totalqty = 0;
$totalamount = (float)$totalqty;
```

Variabile „variabile" (*variable variables*)

PHP furnizează un nou tip: variabilele variabile. Acestea permit modificarea numelui unei variabile în mod dinamic.

O astfel de variabilă utilizează valoarea unei variabile ca nume pentru o altă variabilă. De exemplu, putem avea atribuirea:

```
$varname = 'tireqty';
```

iar apoi putem utiliza $$*varname* în locul lui $*tireqty*. De exemplu, putem seta valoarea lui $*tireqty* după cum urmează:

```
$$varname = 5;
```

Aceasta este echivalentă cu:

```
$tireqty = 5;
```

În loc de a lista şi utiliza fiecare variabilă din formular separat, se poate utiliza o ciclare şi o variabilă pentru a le procesa pe toate automat.

Constante

O constantă stochează o valoare, asemănător unei variabile, însă această valoare este stabilită o singură dată şi nu poate fi modificată ulterior.

În aplicația exemplu, putem stoca prețurile pentru fiecare item de vânzare sub forma unei constante. Constantele se declară utilizând funcția *define*.

```
define('TIREPRICE', 100);
define('OILPRICE', 10);
define('SPARKPRICE', 4);
```

Scrierea cu majuscule a constantelor este doar o convenție care face mai uşoară deosebirea dintre variabile şi constante.

Există câteva deosebiri importante între variabile şi constante:

- referirea unei constante se face fără semnul „$" în față;
- constantele pot stoca doar date de tip scalar (*boolean*, *integer*, *float*, *string*).

Pe lângă constantele definite de utilizator, *PHP* oferă un număr mare de constante predefinite, care pot fi listate cu ajutorul funcției *phpinfo*();

Domeniul de vizibilitate al variabilelor

Regulile de bază referitoare la domeniile de vizibilitate din PHP sunt:

- Variabilele predefinite superglobale sunt vizibile oriunde în *script*.
- Constantele, odată declarate, sunt întotdeauna vizibile global (atât în funcții, cât şi în afara acestora).
- Variabilele globale declarate într-un *script* sunt vizibile în tot *script*-ul, dar nu şi în interiorul funcțiilor.
- Variabilele din interiorul funcțiilor care sunt declarate globale referă variabilele globale având acelaşi nume.
- Variabilele create în interiorul funcțiilor şi declarate statice sunt invizibile din afara funcției dar îşi păstrează valorile de la o execuție a funcției la alta.
- Variabilele create în interiorul funcțiilor sunt locale respectivei funcții şi încetează să existe atunci când funcția îşi încheie execuția.

Tablourile *$_GET* şi *$_POST*, precum şi alte variabile speciale au propriile lor reguli privind vizibilitatea: Acestea sunt cunoscute ca superglobale sau autoglobale şi sunt vizibile de oriunde.

Lista completă a variabilelor superglobale este următoarea:

- *$GLOBALS* – tablou conținând toate variabilele globale; permite accesarea variabilelor globale din interiorul unei funcții (*$GLOBALS[' myvariable ']*.);
- *$_SERVER* – tablou conținând variabilele de mediu ale serverului;
- *$_GET* – tablou cu variabilele transmise script-ului prin intermediul metodei *GET*;
- *$_POST* – tablou cu variabilele transmise script-ului prin intermediul metodei *POST*;
- *$_COOKIE* – tablou cu variabilele din *cookie*-uri;
- *$_FILES* – tablou cu variabilele referitoare la încărcările de fişiere;
- *$_ENV* – tablou cu variabilele de mediu;
- *$_REQUEST* – tablou cu toate intrările din partea utilizatorului, incluzând *$_GET*, *$_POST*, şi *$_COOKIE*;
- *$_SESSION* – tablou cu variabilele sesiunii.

Operatori

1) Operatori aritmetici: +, -, *, /, % (modulo).
2) Operatori pe şiruri de caractere: .
3) Operatorul de atribuire: =

Observație: Pot fi formate expresii de forma: `$b = 6 + ($a = 5);`

4) Operatori de atribuire compuşi:

Operator	Utilizare	Echivalent cu
+=	$a += $b	$a = $a + $b
-=	$a -= $b	$a = $a - $b
*=	$a *= $b	$a = $a * $b
/=	$a /= $b	$a = $a / $b
%=	$a %= $b	$a = $a % $b
.=	$a .= $b	$a = $a . $b

5) Pre- şi Post- incrementare şi decrementare: ++, - - sunt similari operatorilor += şi -=, cu diferența că atribuirea are loc anterior, respectiv ulterior, operației propriu-zise.

6) Operatorul de referință: &
   Când o variabilă este atribuită unei altei variabile, se face o copie a sa şi se stochează în memorie. Liniile următoare au acest efect (se creează o copie a lui *$*a şi se stochează în *$b*):
```
$a = 5;
$b = $a;
```
   În acest exemplu, o modificare ulterioară a valorii lui *$a* nu are niciun efect asupra valorii lui *$b*. Pentru a nu fi realizată o copie, se foloseşte operatorul referință:
```
$a = 5;
$b = &$a;
$a = 7; // $a si $b au acum valoarea 7
```
   Atât *$a* cât şi *$b* referă aceeaşi locație de memorie. Acest lucru poate fi modificat cu ajutorul funcției *unset*() (de exemplu, *unset*(*$a*)).

7) Operatori de comparare:
   - Operatorul de egalitate: == (valorile stocate în cele 2 variabile sunt egale?)
     Atenție! Confuzia între = şi == poate conduce la următoarea situație, deoarece valorile diferite de 0 sunt evaluate *true*, iar 0 este evaluat *false*:
```
$a = 5;
$b = 7;
$a = $b // true; valoarea este cea din membrul stang, adica 7 (=> true)
```
   - Operatorul de identitate: ===returnează *true* doar dacă operanzii sunt egali şi au acelaşi tip de date (0=='0' true, dar 0 === '0' false).

- Operatorii !=, !==, <>, <, >, <=, >=

8) Operatori logici: AND (&&, *and*), OR (||, *or*), XOR (*x or*), NOT (!).
Observație: Operatorii and şi or au precedență mai mică decât && şi ||.

9) Operatori pe biți: AND (&), OR (|), NOT (~), XOR (^), left shift (<<), right shift (>>).

10) Operatorul ternar:
```
condition ? value_if_true : value_if_false
```

11) Operatorul de suprimare a erorilor: @
Exemplu: `$a = @(57/0);`

12) Operatorul de execuție: `` - ceea ce este plasat între apostrofurile inverse este executat ca o comandă în linia de comandă a serverului.
Exemplu: (în Windows)
```
$out = `dir c:`;
echo '<pre>'.$out.'</pre>';
```

13) Operatori pe tablouri: reuniunea (+), egalitatea (==), identitatea (===), inegalitatea (!=, <>, non-identitatea (!==).

14) Operatorii pentru instanțiere: *new*, ->

15) Operatorul pentru tipuri: *instanceof* (utilizat în programarea orientată pe obiecte) – permite să aflăm dacă un obiect este o instanță a unei anumite clase.

Efectuarea totalurilor în formular

Pentru a calcula cantitatea totală de articole comandate, precum şi taxele aferente acestora, procesarea din aplicația exemplu se modifică astfel: http://193.226.51.37/web/lab1/processorder5.txt

Scriptul anterior utilizează operatori (+, *, .) şi funcția *number_format*() – pentru a formata totalurile ca şiruri de caractere cu două zecimale.

Precedența şi asociativitatea

În general, operatorii au o anumită precedență sau ordine în care sunt evaluați. De asemenea, ei au şi o asociativitate, care se referă la ordinea în

care sunt evaluați operatorii de aceeaşi precedență. În general, ordinea poate fi de la stânga spre dreapta (*left*), de la dreapta spre stânga (*right*) sau *not relevant*.

Tabelul următor arată precedența şi asociativitatea operatorilor în *PHP*. Operatorii sunt ordonați în ordinea crescătoare a precedenței.

Associativitate	**Operatori**
left	,
left	or
left	xor
left	and
right	print
left	=
+=	-=
*=	/=
.=	%=
&=	|=
^=	~=
<<=	>>=
left	?
:	
left	||
left	&&
left	|
left	^
left	&
n/a	==
!=	===
!==	
n/a	<
<=	>
>=	
left	<<
>>	
left	+
-	.
left	*
/	%
right	!
~	++
--	(int)
(double)	(string)
(array)	(object)
@	
right	[]
n/a	new
n/a	()

Observații:

- Operatorul având cea mai mare precedență este reprezentat de paranteze (). Acestea ridică precedența operațiilor pe care le conțin.
- Operatorul *print* este echivalent cu comanda *echo*. Nici *print*, nici *echo* nu sunt funcții, însă pot fi apelate ca funcții cu parametri în paranteze. De asemenea, amândoi pot fi tratați ca operatori: şirul de caractere se specifică după cuvântul cheie respectiv. Apelul lui *print* ca funcție returnează valoarea 1.

Utilizarea funcțiilor pentru variabile

PHP furnizează o bibliotecă de funcții care permit prelucrarea şi testarea variabilelor în diferite moduri.

Testarea şi setarea tipurilor de variabile

Cele mai multe funcții pentru variabile se referă la testarea tipului unei variabile. Funcțiile cele mai generale sunt *gettype*() şi *settype*(), ale căror prototipuri sunt:

```
string gettype(mixed var);
bool settype(mixed var, string type);
```

Observație: Documentația *php.net* face referință la tipul de date „*mixed*”. Acest tip nu există, dar pentru că *PHP* este atât de flexibil în privința tipurilor, multe funcții pot accepta mai multe tipuri de date (sau orice tip de dată) pentru argumente. Astfel de argumente sunt afişate având pseudo-tipul „*mixed*”.

Exemplu:

```
$a = 56;
echo gettype($a).'<br />'; //integer
settype($a, 'double');
echo gettype($a).'<br />';//double
```

PHP furnizează şi câteva funcții specifice pentru testarea tipurilor. Fiecare dintre acestea are ca argument o variabilă şi returnează *true* sau *false*. Aceste funcții sunt:

- *is_array*() – verifică dacă variabila este un tablou;
- *is_double*(), *is_float*(), *is_real*() – verifică dacă variabila este reală;

- *is_long*(), *is_int*(), *is_integer*() – verifică dacă variabila este întreagă;
- *is_string*() – verifică dacă variabila este de tip şir de caractere;
- *is_bool*() – verifică dacă variabila este de tip Boolean;
- *is_object*() – verifică dacă variabila este un obiect;
- *is_resource*() – verifică dacă variabila este o resursă;
- *is_null*() – verifică dacă variabila este *null*;
- *is_scalar*() – verifică dacă variabila este scalară.
- *is_numeric*() – verifică dacă variabila este un număr (întreg sau real) sau un şir de caractere ce reprezintă un număr;
- *is_callable*() – verifică dacă variabila reprezintă numele unei funcții valide.

Testarea stării variabilelor

PHP oferă funcții pentru testarea stării variabilelor.

- `bool isset(mixed var);[;mixed var[,...]])` – funcția primeşte ca argument numele unei variabile şi returnează *true* dacă aceasta există şi *false* altfel. De asemenea, se poate transmite o listă de variabile, caz în care funcția returnează *true* dacă toate aceste variabile sunt definite.
- `void unset(mixed var);[;mixed var[,...]])`– elimină variabila transmisă ca argument.
- `bool empty(mixed var)` – verifică dacă variabila există şi are o valoare nevidă, diferită de 0.

Exemplu: Presupunem că este introdus următorul cod în script-ul considerat drept exemplu. Valorile returnate de funcții sunt scrise în comentariu.

```
echo 'isset($tireqty): '.isset($tireqty).'br />'; // true
echo 'isset($nothere): '.isset($nothere).'<br />'; //false
echo 'empty($tireqty): '.empty($tireqty).'<br />';
                            //depinde de ce a fost introdus
echo 'empty($nothere): '.empty($nothere).'<br />';//true
```

Reinterpretarea variabilelor

Echivalentul conversiei variabilelor se poate obține cu ajutorul unor funcții:

- `int intval(mixed var[, int base]);`
- `float floatval(mixed var);`
- `string strval(mixed var);`

Fiecare dintre aceste funcții primeşte ca argument o variabilă şi returnează valoarea variabilei convertită la tipul respectiv. Funcția *intval* permite specificarea bazei pentru conversie atunci când variabila este convertită la un string. Astfel, se poate converti, de exemplu, un şir hexazecimal în număr întreg.

Instrucțiuni *PHP*

Structurile de control sunt acele structuri din cadrul unui limbaj care permit controlul fluxului de execuție într-un program sau *script*. Aceste structuri pot fi:

- condiționale
- repetitive

Instrucțiuni condiționale

Instrucțiunea *if – elseif - else*

Condițiile exprimate în instrucțiunea *if* trebuie să fie cuprinse între paranteze ().

De exemplu, dacă vizitatorul site-ului nu comandă niciun produs, probabil că butonul *Submit* a fost apăsat din greşeală; prin urmare pagina ar trebui să afişeze un mesaj corespunzător în loc de „Order processed”:

```
if( $totalqty == 0 )
   echo 'You did not order anything on the previous page!<br />';
```

Blocuri de cod

Pot exista mai multe instrucțiuni care trebuie executate în funcție de condițiile unei instrucțiuni condiționale. În acest caz, instrucțiunile vor fi grupate în blocuri, cuprinse între acolade {}.

```
if ($totalqty == 0) {
      echo '<p style="color:red">';
      echo 'You did not order anything on the previous page!';
      echo '</p>';
}
```

Ramura *else* a instrucțiunii *if* permite definirea unei alternative care se execută atunci când condiția exprimată în instrucțiunea *if* este *false*. De

exemplu, afişăm un mesaj dacă un client nu comandă nimic, iar în caz contrar afişăm numărul de produse comandate din fiecare categorie:

```
if ($totalqty == 0) {
   echo "You did not order anything on the previous page!<br />";
} else {
   echo $tireqty." tires<br />";
   echo $oilqty." bottles of oil<br />";
   echo $sparkqty." spark plugs<br />";
}
```

În cazul multor decizii există mai mult de 2 opțiuni. Se poate creea o secvență corespunzătoare acestor opțiuni utilizând clauza *elseif*. În acest caz, programul verifică fiecare condiție până când găseşte una care este adevărată.

Exemplu: În magazin se aplică reduceri pentru comenzile mari de cauciucuri, astfel:

- mai puțin de 10 cauciucuri – nicio reducere
- 10-49 cauciucuri – 5% reducere
- 50-99 cauciucuri – 10% reducere
- >= 100 cauciucuri – 15% reducere

Secvența de cod care stabileşte valoarea reducerii este următoarea:

```
if ($tireqty < 10) {
      $discount = 0;
} elseif (($tireqty >= 10) && ($tireqty <= 49)) {
      $discount = 5;
} elseif (($tireqty >= 50) && ($tireqty <= 99)) {
      $discount = 10;
} elseif ($tireqty >= 100) {
      $discount = 15;
}
```

Observație: Formele *elseif* şi *else if* sunt ambele corecte.

Instrucțiunea *switch*

Această instrucțiune lucrează într-un mod similar instrucțiunii *if*, dar permite condiției să ia mai mult de 2 valori. Într-o instrucțiune *switch*, condiția poate lua orice număr de valori diferite, atât timp cât este evaluată la un tip simplu (*integer*, *string* sau *float*). Trebuie furnizată o clauză *case* care să trateze fiecare valoare căreia îi va corespunde o acțiune şi, opțional, o clauză *case* implicită care să trateze cazurile neprevăzute explicit.

De exemplu, dorim să cunoaştem formele de publicitate care au adus clienți, deci putem adăuga o întrebare în formular, astfel încât acesta va arăta astfel: http://193.226.51.37/web/curs2/exemplu1.html

```
<tr>
<td>How did you find us?</td>
<td><select name="find">
<option value = "a">I'm a regular customer</option>
<option value = "b">TV advertising</option>
<option value = "c">Phone directory</option>
<option value = "d">Word of mouth</option>
</select>
</td>
</tr>
```

Codul HTML precedent adaugă o nouă variabilă formularului (*find*) ale cărei valori vor fi *a*, *b*, *c* sau *d*. Variabila poate fi prelucrată cu ajutorul mai multor clauze *elseif* sau printr-o instrucțiune *switch*:

```
switch($find) {
case "a" :
echo "<p>Regular customer.</p>";
break;
case "b" :
echo "<p>Customer referred by TV advert.</p>";
break;
case "c" :
echo "<p>Customer referred by phone directory.</p>";
break;
case "d" :
echo "<p>Customer referred by word of mouth.</p>";
break;
default :
echo "<p>We do not know how this customer found us.</p>";
break;
}
```

Observații:

- Exemplul anterior presupune că variabila *$find* a fost extrasă din tabloul *$_POST*.
- Comportamentul instrucțiunii *switch* este diferit de cel al lui *if-elseif*: instrucțiunea *if* afectează o singură clauză, în vreme ce selectarea unei clauze *case* a instrucțiunii *switch* conduce la executarea de către PHP a instrucțiunilor până când este întâlnită instrucțiunea *break*, care determină trecerea la următoarea instrucțiune după *switch*. Dacă

instrucțiunea *break* lipseşte, *switch* va executa tot codul care urmează clauzei *case* a cărei condiție a fost adevărată.

Instrucțiuni iterative

Exemplu: Dorim afişarea unui tabel care să prezinte costul de transport corespunzător comenzilor. Costul depinde de distanța la care are loc livrarea şi poate fi deteminat pe baza unei formule.

Tabelul este următorul: http://193.226.51.37/web/curs2/transport.html

Codul HTML care afişează acest tabel este următorul: http://193.226.51.37/web/curs2/transport.txt

Se observă că acest cod este lung şi repetitiv. În locul acestuia, putem folosi instrucțiuni PHP de ciclare.

Instrucțiunea *while*

Forma sintactică a acestei comenzi este următoarea:

```
while( condition ) expression;
```

Exemplu: Să se afişeze primele 5 numere întregi pozitive.

```
$num = 1;
while ($num <= 5 ){
      echo $num."<br />";
$num++;
}
```

Exemplu: Afişarea costurilor de transport corespunzătoare comenzilor se poate realiza cu ajutorul instrucțiunii *while* (http://193.226.51.37/web/curs2/transport.php), codul sursă corespunzător fiind http://193.226.51.37/web/curs2/transport_php.txt

Instrucțiunile *for* şi *foreach*

Forma sintactică a comenzii de ciclare *for* este:

```
for( expression1; condition; expression2)
      expression3;
```

- *expression1* este evaluată o singură dată, la început, stabilind valoarea inițială a contorului;
- expresia *condition* este testată înainte de fiecare iterație; dacă returnează *false*, iterația se încheie;

- *expression2* este executată o dată la sfârşitul fiecărei iteraţii, actualizând valoarea contorului;
- *expression3* se execută pentru fiecare iteraţie, fiind reprezentată de un bloc de cod.

Exemplu: Calculul costurilor de transport cu ajutorul instrucţiunii *for* este http://193.226.51.37/web/curs2/transport_for.txt

Observaţie: Variabilele variabile pot fi utilizate împreună cu instrucţiunea *for* pentru a itera printr-o mulţime de câmpuri ale unui formular. Dacă numele câmpurilor sunt *name1*, *name2*, ... acestea pot fi gestionate astfel:

```
for ($i=1; $i <= $numnames; $i++){
   $temp= “name$i”;
   echo $$temp.’<br />’; // sau o alta procesare
}
```

Pe lângă instrucţiunea *loop*, mai există o instrucţiune de iterare specifică lucrului cu tablouri, *foreach*.

Instrucţiunea *do...while*

Forma sintactică a acestei instrucţiuni este următoarea:

```
do
     expression;
while( condition );
```

Spre deosebire de instrucţiunea *while*, condiţia este testată la sfârşit, astfel că instrucţiunea sau blocul de cod din *do...while* se execută cel puţin o dată.

Instrucţiuni de ieşire

Dacă dorim oprirea execuţiei une secvenţe de cod, avem la dispoziţie 3 abordări.

- Instrucţiunea *break* opreşte execuţia unei comenzi de ciclare.
- Instrucţiunea *continue* conduce la trecerea la următoarea iteraţie.
- Instrucţiunea *exit* conduce la terminarea întregului script PHP. Acest lucru este util atunci când se realizează verificarea erorilor.

Utilizarea structurilor alterative de control

Pentru toate structurile de control prezentate, există o formă sintactică alternativă. Aceasta constă în înlocuirea acoladei deschise ({) cu „:” iar a celei închise (}) cu un cuvânt cheie care va fi *endif*, *endswitch*, *endwhile*,

endfor sau *endforeach*, în funcție de tipul instrucțiunii. Nu există o sintaxă alternativă pentru instrucțiunea *do...while*.

Exemplu: Codul sursă:

```
if ($totalqty == 0) {
   echo "You did not order anything on the previous page!<br />";
   exit;
}
```

poate fi scris sub forma următoare:

```
if ($totalqty == 0) :
   echo "You did not order anything on the previous page!<br />";
   exit;
endif;
```

Utilizarea cuvântului cheie declare

Forma sintactică a acestei structuri de control din PHP este următoarea:

```
declare (directive)
{
   // block
}
```

Această structură este folosită pentru stabilirea directivelor de execuție pentru un bloc de cod, adică a unor reguli asupra modului în care codul urmează să fie executat. Până acum a fost implementată o singură directivă de execuție, denumită *ticks*. Aceasta se setează *ticks* = *n* şi permite ca o anumită funcție să fie executată la fiecare *n* linii de cod din interiorul unui bloc, lucru care este util în activitatea de *debug*.

Stocarea şi regăsirea datelor

Datele introduse prin intermediul unui formular trebuie stocate pentru prelucrarea lor ulterioară. Vom învăța cum să salvăm aceste date într-un fişier iar apoi într-o bază de date *MySQL*. În privința fişierelor, vom studia operațiile care pot fi realizate asupra acestora.

Salvarea datelor

În exemplul din această secțiune vom scrie comenzile clienților într-un fişier text, câte o comandă pe fiecare linie. Scrierea comenzilor în acest mod este foarte simplă, însă are nişte limite considerabile. Atunci când este gestionat un volum considerabil de informații, va trebui utilizată o bază de date.

Scrierea şi citirea în/din fişiere este foarte similară multor limbaje de programare.

Stocarea şi regăsirea comenzilor

Vom utiliza următoarea versiune (uşor modificată) a formularului pe care am lucrat până acum: http://193.226.51.37/web/curs2/orderform2.html

Formularul a fost modificat astfel încât include un mod rapid de regăsire a adresei de livrare. Noul câmp se numeşte *address* şi îi corespunde o variabilă care poate fi accesată ca *$_REQUEST['address']*, *$_POST['address']* sau *$_GET'['address']*, în funcție de metoda aleasă.

Vom scrie toate comenzile lansate de către clienți într-un fişier, apoi vom construi o interfață web pentru vizualizarea acestor comenzi.

Procesarea fişierelor

Scrierea datelor într-un fişier presupune 3 paşi:

1. Deschiderea fişierului; dacă acesta nu există, va trebui creat.
2. Scrierea datelor în fişier.
3. Închiderea fişierului.

Citirea dintr-un fişier presupune:

1. Deschiderea fişierului; trebuie prevăzută eroarea care apare în cazul în care deschiderea nu se poate realiza (de exemplu, fişierul nu există).
2. Citirea datelor din fişier.
3. Închiderea fişierului.

Deschiderea fişierelor

Se realizează cu ajutorul funcției *fopen*(). La deschiderea unui fişier, trebuie specificat şi modul în care acesta va fi utilizat. Modul furnizează sistemului de operare un mecanism prin care determină cum să trateze accesul din partea altor persoane sau script-uri, precum şi o metodă de a verifica dacă avem acces şi drepturi asupra unui anumit fişier.

La deschiderea unui fişier trebuie făcute 3 alegeri:

1. Fişierul poate fi deschis doar pentru citire, doar pentru scriere, sau pentru ambele operații.
2. Dacă scriem într-un fişier, putem opta pentru suprascrierea conținutului sau adăugarea noilor date la sfârşitul fişierului. O altă posibilitate este aceea de a încheia programul dacă fişierul există deja, în loc de a-l suprascrie.
3. Dacă scriem într-un fişier dintr-un sistem care face diferență între fişierele text şi cele binare, va trebui specificat acest lucru.

Funcția *fopen*() suportă combinații ale acestor 3 opțiuni. Deschiderea unui fişier pentru scrierea comenzilor din exemplul nostru se realizează prin:

```
$fp = fopen("$DOCUMENT_ROOT/../orders/orders.txt", 'w');
```

Primul parametru din apelul anterior este calea către fişier; a fost utilizată variabila PHP predefinită $_SERVER['DOCUMENT_ROOT']. Calea furnizată este una relativă: utilizăm „..” pentru crearea fişierului în directorul părinte al celui specificat în $DOCUMENT_ROOT, deoarece nu dorim ca acest fişier să fie accesibil pe web.

Pentru ca linia anterioară să fie corectă, este necesar să scriem la începutul script-ului:

```
$DOCUMENT_ROOT = $_SERVER['DOCUMENT_ROOT'];
```

Pot fi utilizate căi absolute sau putem să nu specificăm nicio cale, caz în care fişierul va fi creat sau căutat în directorul în care se află script-ul.

Într-un mediu Unix se utilizează caracterul slash (/) în specificarea căilor către directoare. În Windows, se poate utiliza atât slash (/) cât şi backslash (\). În cazul caracterului backslash, acesta trebuie marcat drept caracter special, prin adăugarea încă a unui astfel de caracter (\):

```
$fp = fopen("$DOCUMENT_ROOT\\..\\orders\\orders.txt", 'w');
```

Al doilea parametru al funcției *fopen*() reprezintă modul de deschidere a fişierului. Acesta este un *string* şi poate lua una dintre valorile

- *r* (*read*) – deschiderea fişierului pentru citire, de la începutul fişierului;
- *r+* – deschiderea fişierului pentru citire şi scriere, de la începutul fişierului;
- *w* (*write*) – deschiderea fişierului pentru scriere, de la începutul acestuia; dacă fişierul există, va fi şters conținutul său;
- *w+* – deschiderea fişierului pentru scriere şi citire, de la începutul acestuia; dacă fişierul există, va fi şters conținutul său;
- *x* (*cautious write*) – deschiderea fişierului pentru scriere, de la începutul acestuia; dacă fişierul există, nu va fi deschis, ci *fopen*() returnează *false*, iar PHP generează un *warning*;
- *x+* - deschiderea fişierului pentru scriere şi citire, de la începutul acestuia; dacă fişierul există, nu va fi deschis, ci *fopen*() returnează *false*, iar PHP generează un *warning*;
- *a* (*append*) – deschiderea fişierului doar pentru adăugare, începând de la sfârşitul conținutului curent; dacă fişierul nu există, va fi creat;
- *a+* – deschiderea fişierului pentru adăugare şi citire, începând de la sfârşitul conținutului curent; dacă fişierul nu există, va fi creat;
- *b* (*binary*) – se utilizează în conjuncție cu unul dintre celelalte moduri; este util pentru sistemele care fac diferența între fişierele text şi cele binare (Windows face această diferență, Unix nu). Se recomandă utilizarea opțiunii pentru asigurarea portabilității. Aceasta este valoarea implicită.
- *t* (*text*) – opțiune existentă doar în Windows.

Al treilea parametru al funcției *fopen* este opțional şi se utilizează dacă dorim să căutăm în *include_path* (parametru de configurare al lui PHP). În caz afirmativ, valoarea acestui parametru va fi 1 şi nu va mai fi necesar să specificăm un director sau o cale pentru fişierul respectiv.

```
$fp = fopen('orders.txt', 'ab', true);
```

Al patrulea parametru este, de asemenea, opțional. Funcția *fopen*() permite ca numele fişierelor să fie prefixate de un protocol (de exemplu, http://) şi deschise la o locație distantă. Unele protocoale necesită un parametru. Dacă funcția *fopen*() deschide fişierul cu succes, este returnată o resursă (pointer către fişier) care trebuie stocată într-o variabilă ($fp).

Deschiderea fişierelor prin *FTP* sau *HTTP*

Pe lângă deschiderea fişierelor locale pentru citire sau scriere, pot fi deschise fişiere prin intermediul FTP, HTTP sau al altor protocoale. Această posibilitate poate fi eliminată prin dezactivarea directivei *allow_url_fopen* din fişierul php.ini.

Scrierea într-un fişier

Pentru scrierea într-un fişier în PHP se pot itiliza funcțiile *fwrite*() sau *fputs*(). Aceasta din urmă este un alias al lui *fwrite*(). Un apel la funcția *fwrite* este de forma următoare:

```
fwrite($fp, $outputstring);
```

şi determină scrierea şirului de caractere stocat în *$outputstring* în fişierul la care pointează $*fp*.

O alternativă a lui *fwrite*() este funcția *file_put_contents()*. Aceasta are următorul prototip:

```
int file_put_contents ( string filename,
                        string data
                        [, int flags
                        [, resource context]])
```

Această funcție scrie şirul conținut în parametrul *data* în fişierul numit *filename* fără a fi nevoie de invocarea funcțiilor *fopen*() şi *fclose*(). Parametrii opționali *flags* şi *context* sunt folosiți la scrierea la distanță a fişierelor.

Parametrii funcției *fwrite*

Prototipul funcției *fwrite* este următorul:

```
int fwrite ( resource handle, string string [, int length])
```

Al treilea parametru reprezintă numărul maxim de octeţi care urmează să fie scrişi. Specificarea acestuia determină scrierea lui *string* în fişier până când sunt scrişi *length* octeţi sau până când este scris tot şirul (dacă acesta este mai scurt decât *length*).

Lungimea unui şir de caractere poate fi obţinută cu ajutorul funcţiei *strlen*:

```
fwrite($fp, $outputstring, strlen($outputstring));
```

Formatul fişierelor

Fie următorul şir de caractere, care reprezintă o înregistrare în fişier:

```
$outputstring = $date. "\t".$tireqty. " tires \t".$oilqty. " oil\t"
.$sparkqty. " spark plugs\t\$".$totalamount
. "\t". $address. "\n";
```

Fiecare înregistrare va fi scrisă pe o linie nouă (\n) iar câmpurile sunt separate prin tab (\t).

Închiderea unui fişier

După terminarea lucrului cu un fişier este nevoie să îl închidem:

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

Această funcţie returnează *true* dacă închiderea fişierului a avut loc cu succes sau *false* altfel.

Forma finală a fişierului *processorder.php* este dată în http://193.226.51.37/web/curs2/processorder2.txt

Vectori în *PHP*

Spre deosebire de variabilele de tipuri scalare, o variabilă de tip vector stochează o mulțime sau o secvență de valori. Elementele unui vector pot avea tipuri scalare sau, la rândul lor, pot fi vectori (în acest caz, obținem vectori multidimensionali).

PHP permite definirea a două tipuri de vectori:

- indexați numeric
- asociativi.

Vectorii asociativi permit utilizarea unor valori mai utile drept index; acestea pot fi chiar şiruri de caractere.

În exemplele din acest curs, vom folosi vectori pentru a lucra mai uşor cu informațiile având caracter repetitiv (de exemplu, comenzile clienților).

Ce este un vector?

O variabilă scalară este o locație cu nume, în care poate fi stocată o valoare. Similar, un vector este o locație cu nume în care poate fi stocată o mulțime de valori.

Exemplu: Lista de produse oferite de site.

Stocarea informațiilor într-un vector permite:

- efectuarea aceloraşi acțiuni asupra fiecărei valori din vector, utilizând instrucțiuni de ciclare;
- tratarea unitară a întregului set de informații; astfel, toate valorile unui vector pot fi transmise unei funcții (de exemplu, *sort*()).

Valorile stocate într-un vector se numesc elemente ale vectorului. Fiecare element are un indice asociat (sau cheie), care este utilizat pentru a accesa elementul respectiv. În majoritatea limbajelor de programare, vectorii au indici numerici, care pot începe de la 0 sau 1.

PHP permite utilizarea atât a numerelor, cât şi a şirurilor de caractere ca indici ai tabloului. Tablourile pot fi indexate numeric sau se pot stabili valori cât mai sugestive sau semnificative ale cheilor (similar structurilor de tip *map*, *hash* sau dicționar din alte limbaje de programare).

Tablouri indexate numeric

Acest tip de vector este furnizat de către majoritatea limbajelor de programare. În PHP, indicii încep implicit de la 0.

Inițializarea tablourilor indexate numeric

Pentru a crea un vector ce conține cele 3 produse vândute prin intermediul site-ului exemplu, linia de cod PHP este următoarea:

```
$products = array('Tires', 'Oil', 'Spark Plugs' );
```

Ca şi *echo*, *array*() este mai degrabă o construcție a limbajului, şi nu o funcție.

Elementele unui vector pot fi copiate în alt vector utilizând operatorul de atribuire (=).

Dacă dorim ca un vector să rețină o secvență crescătoare de numere, funcția *range*() permite crearea automată a acestui vector. Funcția acceptă un parametru opțional care stabileşte distanța dintre valori.

Exemplu: Crearea unui vector cu elemente de la 1 la 10 se realizează astfel:

```
$numbers = range(1,10);
```

Pentru crearea unui vector cu numerele impare între 1 şi 10 se obține prin:

```
$odds = range(1, 10, 2);
```

Funcția poate fi folosită şi pentru caractere:

```
$letters = range('a', 'z');
```

Elementele unui vector pot fi încărcate din fişiere sau din baze de date. Există funcții care ne permit extragerea unei părți a unui vector sau reordonarea tabloului.

Accesarea conținutului tabloului

Conținutul unui vector poate fi accesat folosind numele variabilei şi un index (sau cheie), care se plasează între paranteze drepte după nume.

Folosind această notație, putem actualiza valorile elementelor din vector sau putem adăuga altele noi:

```
$products[0] = 'Fuses'; //modificare
$products[3] = 'Fuses'; //adaugare
```

Similar altor variabile PHP, tablourile nu trebuie inițializate sau create în avans, ci sunt create automat atunci când sunt folosite prima dată. Următoarele linii de cod creează acelaşi vector, pentru care anterior s-a folosit construcția *array*():

```
$products[0] = 'Tires';
$products[1] = 'Oil';
$products[2] = 'Spark Plugs';
```

Tablourile sunt redimensionate în mod dinamic pe măsură ce le sunt adăugate elemente, funcționalitate care nu există în cele mai multe limbaje de programare.

Utilizarea instrucțiunilor de ciclare pentru accesarea vectorilor

Pentru un vector indexat printr-o secvență de numere, parcurgerea se poate realiza simplu cu ajutorul unei bucle *for*:

```
for ($i = 0; $i<3; $i++) {
     echo $products[$i]." ";
}
```

Pe lângă instrucțiunea *for* se poate utiliza *foreach*, concepută special pentru lucrul cu vectori:

```
foreach ($products as $current) {
     echo $current." ";
}
```

Fiecare element este stocat în variabila $*current* şi apoi prelucrat (tipărit).

Tablouri cu indici diferiți

PHP permite definirea de vectori în care se poate asocia fiecărei valori orice cheie sau index.

Inițializarea unui vector

Următorul fragment de cod creează un vector în care numele produselor sunt chei şi prețurile acestora reprezintă valorile.

```
$prices = array('Tires'=>100, 'Oil'=>10, 'Spark Plugs'=>4);
```

Accesarea elementelor vectorului

Elementele vectorului pot fi accesate într-un mod similar vectorilor în care indicii sunt numerici (`$prices['Tires']`).

Vectorul anterior poate fi creat şi în alte moduri:

```
$prices = array('Tires'=>100 );
$prices['Oil'] = 10;
$prices['Spark Plugs'] = 4;
```

sau:

```
$prices['Tires'] = 100;
$prices['Oil'] = 10;
$prices['Spark Plugs'] = 4;
```

Utilizarea instrucțiunilor de ciclare

Deoarece indicii nu sunt numere, nu se poate folosi un simplu contor în instrucțiunea *for* pentru a lucra cu vectorul. Se poate utiliza instrucțiunea *foreach* sau construcțiile *list*() şi *each*().

Forma instrucțiunii *foreach* este puțin diferită atunci când sunt utilizați vectori asociativi. Instrucțiunea poate fi folosită ca în exemplul anterior sau poate încorpora şi cheile:

```
foreach ($prices as $key => $value) {
      echo $key." – ".$value."<br />";
}
```

Folosind construcția *each*(), afişarea conținutului tabloului se realizează astfel:

```
while ($element = each($prices)) {
      echo $element['key'];
      echo " – ";
      echo $element['value'];
      echo "<br />";
}
```

Rezultatul execuției acestui *script* este:
http://193.226.51.37/web/curs3/each.php

Funcția *each*() returnează elementul curent dintr-un vector şi trece la următorul element. În fragmentul anterior, variabila $*element* este un vector. Locația *key* sau 0 conține cheia elementului curent, iar locațiile *value* sau 1 conțin valoarea elementului curent.

Un alt mod, mai obişnuit, utilizează construcţia *list*() pentru a împărţi vectorul într-un număr de valori. Astfel, cele două valori returnate de funcţia *each*() pot fi despărţite astfel:

```
while (list($product, $price) = each($prices)) {
      echo "$product - $price<br />";
}
```

Funcţia *list*() atribuie cele două elemente returnate de *each*() la două variabile, $*product* şi $*price*.
Observaţie: Atunci când este utilizat *each*(), elementul curent al tabloului se modifică. Dacă vectorul este folosit încă o dată în script, atunci trebuie resetat elementul curent la începutul tabloului, utilizând funcţia *reset*(). Pentru a parcurge din nou vectorul $*prices*, putem scrie:

```
reset($prices);
while ( list( $product, $price ) = each( $prices ) )
echo “$product - $price<br />”;
```

Operatori asupra tablourilor

Operator	*Nume*	*Exemplu*	*Rezultat*
+	Reuniune	$a + $b	Tabloul $b este adăugat lui $a, mai puţin elementele având chei deja existente în $a.
==	Egalitate	$a == $b	Adevărat dacă $a şi $b conţin aceleaşi elemente.
===	Identitate	$a === $b	Adevărat dacă $a şi $b conţin aceleaşi elemente, au aceleaşi tipuri şi se află în aceeaşi ordine.
!=, <>	Inegalitate	$a != $b	Adevărat dacă $a şi $b nu conţin aceleaşi elemente.
!==	Non-identitate	$a !== $b	Adevărat dacă $a şi $b nu conţin aceleaşi elemente, nu au aceleaşi tipuri sau nu se află în aceeaşi ordine.

Tablouri multidimensionale

Tablourile pot să nu fie doar simple liste de chei şi valori. O locație dintr-un vector poate reține un alt vector. Astfel, se poate crea un vector bidimensional, care poate fi gândit ca o matrice.

Presupunem că dorim să stocăm mai multe date despre produsele disponibile, folosind un vector bidimensional în care fiecare linie reprezintă un produs, iar fiecare coloană reprezintă un atribut al unui produs. Considerăm că un produs are 3 atribute: *cod*, *descriere*, *pret*. Următorul fragment de cod creează vectorul bidimensional corespunzător:

```
$products = array( array('TIR', 'Tires', 100 ),
                   array('OIL', 'Oil', 10 ),
                   array('SPK', 'Spark Plugs', 4 ) );
```

Astfel, vectorul $*products* va conține, la rândul lui, alți 3 vectori. Pentru a accesa un element al tabloului bidimensional, furnizăm doi indici: al liniei şi al coloanei. Pentru afişarea tabloului precedent, se poate proceda astfel:

```
for ($row = 0; $row < 3; $row++) {
      for ($column = 0; $column < 3; $column++) {
            echo '|'.$products[$row][$column];
      }
      echo '|<br />';
}
```

În browser vom obține: http://193.226.51.37/web/curs3/bidim.php

Putem folosi nume drept chei ale coloanelor, în loc de numere. Presupunem că dorim să stocăm mulțimea de produse anterioară folosind numele atributelor corespunzătoare drept chei pentru coloane. Fragmentul de cod este:

```
$products = array( array('Code' => 'TIR',
                         'Description' => 'Tires',
                         'Price' => 100
                        ),
                   array('Code' => 'OIL',
                         'Description' => 'Oil',
                         'Price' => 10
                         ),
                   array('Code' => 'SPK',
                         'Description' => 'Spark Plugs',
                         'Price' =>4
                         )
              );
```

În acest fel, putem avea nume semnificative pentru linii şi coloane. Pe de altă parte, se pierde posibilitatea de a folosi instrucțiunea *for* simplă pentru a parcurge coloanele.

Putem proceda astfel:

```
for ( $row = 0; $row < 3; $row++){
      echo '|'.$products[$row][ 'Code'].
           '|'.$products[$row][ 'Description'].
          '|'.$products[$row][ 'Price']. '|<br />';
}
```

Pentru a cicla şi în tablourile care utilizează indici descriptivi, putem utiliza funcțiile *each*() şi *list*() într-o instrucțiune *while*:

```
for ( $row = 0; $row < 3; $row++){
      while ( list( $key, $value ) = each( $products[$row])){
            echo "|$value";
      }
      echo '|<br />';
}
```

Putem avea vectori de dimensiune mai mare decât 2. Similar modului în care elementele tablourilor pot reține noi vectori, acei noi vectori pot reține, la rândul lor, alți vectori.

Un vector tridimensional are înălțime, lățime şi adâncime. Putem gândi un astfel de vector ca pe o „stivă” de vectori bidimensionali, în care un element este referit prin nivel, linie şi coloană.

Presupunem că produsele sunt împărțite în categorii (corespunzătoare maşinilor, camionetelor şi camioanelor) şi folosim un vector tridimensional pentru a le stoca. Tabloul tridimensional corespunzător se creează astfel:

```
$categories = array( array ( array('CAR_TIR', 'Tires', 100 ),
                             array('CAR_OIL', 'Oil', 10 ),
                             array('CAR_SPK', 'Spark Plugs', 4 )
                           ),
                     array ( array('VAN_TIR', 'Tires', 120 ),
                             array( 'VAN_OIL', 'Oil', 12 ),
                             array( 'VAN_SPK', 'Spark Plugs', 5 )
                           ),
                     array ( array('TRK_TIR', 'Tires', 150 ),
                             array('TRK_OIL', 'Oil', 15 ),
                             array('TRK_SPK', 'Spark Plugs', 6 )
                           )
                   );
```

Deoarece acest vector are doar indici numerici, se poate utiliza instrucțiunea *for* pentru a afișa conținutul său:

```
for ($layer = 0; $layer < 3; $layer++) {
      echo "Layer $layer<br />";
      for ($row = 0; $row < 3; $row++) {
            for ($column = 0; $column < 3; $column++) {
                  echo '|'.$categories[$layer][$row][$column];
            }
            echo '|<br />';
      }
}
```

Nu există vreo limită a limbajului asupra numărului de dimensiuni ale unui vector.

Sortarea tablourilor

Sortarea datelor dintr-un vector este o operație foarte utilă, care poate fi realizată cu uşurință în PHP.

Funcția *sort*()

Următorul fragment de cod are ca rezultat ordonarea alfabetică a vectorului:

```
$products = array('Tires', 'Oil', 'Spark Plugs' );
sort($products);
```

Evident, dacă vectorul conține valori numerice, acesta este ordonat crescător:

```
$prices = array( 100, 10, 4 );
sort($prices);
```

Funcția *sort*() este *case sensitive*. Această funcție acceptă un parametru opțional: putem specifica una dintre constantele *SORT_REGULAR* (valoarea implicită a parametrului), *SORT_NUMERIC* sau *SORT_STRING*. Acest parametru este util atunci când avem de comparat şiruri de caractere care pot conține numere.

Funcțiile *asort*() şi *ksort*()

În cazul în care este folosit un vector cu chei descriptive pentru a stoca articolele şi prețurile acestora, va trebui să folosim forme diferite ale funcției de sortare pentru a menține cheile şi valorile împreună, atunci când

sunt sortate. Următorul fragment de cod creează un vector ce conține cele 3 produse şi prețurile asociate lor, iar apoi sortează vectorul în ordinea crescătoare a prețurilor:

```
$prices = array('Tires'=>100, 'Oil'=>10, 'Spark Plugs'=>4 );
asort($prices);
```

Funcția *asort*() ordonează vectorul conform valorii fiecărui element. În vector, valorile sunt prețurile, iar cheile sunt descrierile produselor. Dacă dorim să sortăm după descriere, va trebui să utilizăm funcția *ksort*(), care sortează după chei. Următorul fragment de cod are ca efect ordonarea alfabetică a cheilor vectorului:

```
$prices = array('Tires'=>100, 'Oil'=>10, 'Spark Plugs'=>4 );
ksort($prices);
```

Sortarea în ordine inversă

Cele trei funcţii prezentate anterior (*sort*, *asort*, *ksort*) se referă la sortarea în ordine crescătoare. Pentru fiecare dintre acestea există câte o funcție corespunzătoare pentru sortarea în ordine inversă: *rsort*, *arsort*, respectiv *krsort*. Acestea se utilizează într-un mod similar funcțiilor pentru ordonarea crescătoare.

Sortarea tablourilor multidimensionale

Sortarea tablourilor cu mai mult de o dimensiune, sau altfel decât în ordine alfabetică sau numerică, este mai complicată. PHP poate compara două numere sau două şiruri de caractere, dar într-un vector multidimensional un element este, al rândul lui, un vector. Două vectori nu pot fi comparați, decât dacă este creată o metodă pentru compararea lor.

Sortarea definită de utilizator

Considerăm definirea tabloului bidimensional utilizat anterior:

```
$products = array( array('TIR', 'Tires', 100 ),
                   array('OIL', 'Oil', 10 ),
                   array('SPK', 'Spark Plugs', 4 ) );
```

Dacă sortăm acest vector, în ce ordine vor apărea valorile? Am putea dori ca produsele să fie stocate în ordine alfabetică după descriere sau în ordine numerică după preț. Pentru a obține rezultatul dorit, trebuie să scriem propria funcție de comparare şi să utilizăm funcția *usort*() pentru a furniza un mod de comparare a elementelor.

Următorul exemplu sortează vectorul în ordine alfabetică pe baza celei de-a doua coloane din vector (descrierea):

```
function compare($x, $y) {
      if ($x[1] == $y[1]) {
            return 0;
      } else if ($x[1] < $y[1]) {
            return -1;
      } else {
            return 1;
      }
}
usort($products, 'compare');
```

Funcția de comparare trebuie să returneze 0 dacă $x = $y, un număr negativ dacă $x < $y, sau un număr pozitiv dacă $x > $y.

Funcția *usort*() are ca argumente vectorul care trebuie sortat şi numele funcției de comparare.

Există versiuni *uasort*() şi *uksort*(). Funcția *uasort*() este utilizată pentru vectori indexați numeric, după valoare, în cazul în care valorile sunt vectori. Similar, funcția *uksort*() se utilizează atunci când sortăm un vector indexat non-numeric după cheie, iar cheile sunt vectori.

Sortare utilizator inversă

Sortările definite de utilizator nu au variante pentru ordonarea inversă, însă putem ordona invers un vector multidimensional scriind o funcție de comparare care să întoarcă valori opuse celor pentru ordonarea crescătoare. De exemplu:

```
function reverse_compare($x, $y) {
      if ($x[2] == $y[2]) {
            return 0;
      } else if ($x[2] < $y[2]) {
            return 1;
      } else {
            return -1;
      }
}
```

Apelul `usort($products, 'reverse_compare')` va conduce la ordonarea inversă după preț.

Reordonarea vectorilor

În anumite aplicații, sunt utile şi alte funcții referitoare la ordonare. Funcția *shuffle* reordonează elementele vectorului în mod aleator. Funcția *array_reverse* furnizează o copie a vectorului cu toate elementele în ordine inversă.

Funcția *shuffle*()

Presupunem că dorim ca pe pagina principală a site-ului să apară un număr mic de produse (de exemplu, 3 articole selectate aleator). Acest lucru se poate realiza simplu dacă produsele se află într-un vector. Script-ul următor afişează 3 imagini alese aleator, reordonând vectorul aleator şi selectând primele 3 elemente:
http://193.226.51.37/web/curs3/front_page.php.

De fiecare dată când pagina este încărcată, vor apărea imagini diferite.

Funcția *array_reverse*()

Această funcție ia ca argument un vector şi creează un vector nou, cu acelaşi conținut însă în ordine inversă.

Exemplu: Crearea unui vector conținând numerele de la 10 la 1.

1) Folosind funcția *range*() se creează secvența crescătoare, care apoi se ordonează descrescător folosind *array_reverse*() sau *rsort*().

```
$numbers = range(1,10);
$numbers = array_reverse($numbers);
```

2) Creând vectorul element cu element:

```
$numbers = array();
for($i=10; $i>0; $i--) {
      array_push($numbers, $i);
}
```

În exemplul anterior, a fost creat un vector vid, în care au fost adăugate elemente cu ajutorul funcției *array_push*(). Aceasta permite adăugarea unui element la sfârşitul unui vector. Funcția care realizează acțiunea inversă, de ştergere a unui element de la sfârşitul vectorului, este *array_pop*().

3) Dacă datele constituie un domeniu de numere întregi, acesta se poate crea direct în ordine inversă astfel:

```
$numbers = range(10, 1, -1);
```

Încărcarea vectorilor din fişiere

În cursul anterior, am stocat comenzile lansate de clienți într-un fişier. Pentru a procesa respectivele comenzi, ar putea fi nevoie să reîncărcăm datele din fişier într-un vector.

Conținutul curent al comenzilor se află în fişierul: http://193.226.51.37/web/curs3/vieworders.php.

Script-ul utilizează funcția *file*() care încarcă fişierul într-un vector. Fiecare linie din fişier devine element în vector. Funcția *count*() furnizează numărul de elemente din vector.

În continuare, putem încărca fiecare secțiune a liniilor corespunzătoare comenzilor în elemente separate ale vectorului: http://193.226.51.37/web/curs3/vieworders2.txt

Fişierul este încărcat în vector şi este utilizată funcția *explode*() pentru a separa fiecare linie astfel încât aceasta să poată fi procesată (în exemplu, pusă într-o celulă separată a unui tabel).

Rezultatul este următorul: http://193.226.51.37/web/curs3/vieworders2.php

Funcția *explode* are următorul prototip:

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

Parametrul opțional *limit* poate limita numărul maxim de elemente returnate.

Reutilizarea codului. Funcții în *PHP*

Reutilizarea codului

PHP furnizează două instrucțiuni foarte simple care permit reutilizarea codului. Utilizând instrucțiunile *require*() sau *include*(), se poate încărca un fişier într-un script PHP. Fişierul inclus poate conține orice componentă ce poate exista într-un script (instrucțiuni PHP, text, *tag*-uri HTML, funcții PHP, clase PHP).

Aceste instrucțiuni lucrează similar incluziunilor de pe partea de server oferite de multe servere web, sau directivelor *#include* din C sau C++.

Instrucțiunile *require*() şi *include*() sunt aproape identice, singura diferență având loc atunci când acestea eşuează, caz în care *require*() conduce la o eroare fatală, iar *include*() întoarce doar un *warning*.

Există două variante ale lui *require*() şi *include*(), denumite *require_once*() şi, respectiv, *include_once*(). Scopul acestor instrucțiuni este cel de a asigura ca un fişier să poată fi inclus o singură dată. Această funcționalitate devine utilă atunci când *require*() şi *include*() sunt folosite pentru a include biblioteci de funcții. Astfel, se evită includerea de două ori a aceleiaşi biblioteci de funcții, lucru care ar conduce la redefinirea funcțiilor şi la apariția unei erori. Pe de altă parte, se recomandă utilizarea instrucțiunilor *require*() şi *include*(), deoarece acestea se execută mai repede.

Extensiile fişierelor incluse

Următoarea linie de cod este stocată în fişierul *reusable.php*:

```
<?php
     echo 'Aici este o instructiune PHP simpla.<br/>';
?>
```

Următorul fragment de cod se află în fişierul *main.php*:

```
<?php
     echo 'Fisierul principal.<br/>';
     require('reusable.php');
     echo 'Incheiere script.<br/>';
?>
```

La încărcarea script-ului *reusable.php* va apărea mesajul „*Aici este o instructiune PHP simpla.*”. La încărcarea script-ului *main.php*, apare: http://193.226.51.37/web/curs4/main.php

La execuția script-ului *main.php* instrucțiunea *require*('*reusable.php*') este înlocuită cu conținutul fișierului *reusable.php*, iar script-ul este apoi executat.

Nu este importantă extensia fișierului referit în instrucțiunea *require*(). Atât timp cât acest fișier nu este încărcat direct, el poate avea orice nume dorim. Dacă includem un *html* printr-o instrucțiune *require*(), va fi procesat codul PHP din interiorul lui. Se recomandă utilizarea unei convenții în privința extensiilor, cum ar fi *.inc* sau *.php*.

Observație: Dacă este inclus un fișier cu o extensie non-standard (de exemplu, *.inc*), acesta este stocat în ierarhia de documente web iar utilizatorii vor putea vizualiza conținutul acestuia direct în browser, ca text, prin urmare ar putea avea acces la parole sau alte informații sensibile. Se recomandă fie ca aceste fișiere să fie stocate în afara ierarhiei de documente, fie să utilizeze extensii standard.

Codul din fișierul inclus trebuie plasat între tag-uri PHP, pentru a putea fi tratat ca atare. Altfel, codul va fi tratat ca text sau ca HTML şi nu va fi executat.

Utilizarea instrucțiunii *require*() pentru *template*-urile site-urilor web

Dacă paginile web ale unei companii au un aspect consistent, putem adăuga *template*-ul şi elementele standard utilizând *require*().

De exemplu, presupunem că site-ul are un număr de pagini, toate cu aspectul următor: http://193.226.51.37/web/curs4/home.php. Atunci când este adăugată o pagină nouă, programatorul poate deschide o pagină existentă, înlocui textul şi salva fişierul cu un nume nou.

Să presupunem că site-ul web are un număr mare de pagini, toate urmând acelaşi stil. Dorim să modificăm acest aspect standard, lucru care devine foarte costisitor pentru un număr mare de pagini.

Reutilizarea directă a secțiunilor HTML comune tuturor paginilor constituie o abordare mai bună şi mai eficientă. Codul sursă al paginii *home* (*home.html*) este: http://193.226.51.37/web/curs4/home_html.txt

Se poate observa un număr de secțiuni distincte de cod în fişierul anterior. *Head*-ul HTML conține definiții CSS (*cascading style sheet*)

utilizate de către pagină. Secțiunea etichetată „*page header*” afișează numele companiei şi logo-ul, „menu” creează bara de navigare a paginii, iar „page content” conține textul specific paginii. Fişierul mai conține *footer*-ul paginii.

Fişierul poate fi împărțit în 3 fişiere: *header.php*, *home.php* şi *footer.php*. Fişierele *header.php* şi *footer.php* conțin cod care va fi reutilizat în alte pagini.

Fişierul *home.php* înlocuieşte *home.html* cuprinzând conținutul unic al paginii şi două instrucțiuni *require*(), care încarcă *header.php* şi *footer.php*: http://193.226.51.37/web/curs4/home_php.txt

Fişierul *header.php* conține definițiile CSS utilizate de către pagină, tabelele care afişează numele companiei şi meniurile de navigare: http://193.226.51.37/web/curs4/header_php.txt

Fişierul *footer.php* conține tabelul care afişează *footer*-ul în partea de jos a paginii: http://193.226.51.37/web/curs4/footer_php.txt

Această abordare ne permite să obținem un site web cu un aspect consistent, în care o pagină nouă, având acelaşi stil, se poate obține astfel:

```
<?php require('header.php'); ?>
     Here is the content for this page
<?php require('footer.php'); ?>
```

Cel mai important lucru este acela că, după ce au fost create mai multe pagini care utilizează acelaşi *header* şi *footer*, fişierele pot fi modificate cu uşurință. Indiferent cât de mare este modificarea adusă, aceasta trebuie efectuată o singură dată.

Exemplul anterior utilizează doar cod HTML în corpul, antetul şi subsolul fişierului, însă acest lucru nu este necesar, putându-se folosi instrucțiuni PHP pentru generarea dinamică a unor părți din pagină.

Dacă dorim ca un fişier să fie tratat ca fişier text sau HTML şi să nu determine execuția de cod PHP, putem folosi funcția *readfile*(). Aceasta afişează conținutul unui fişier fără a-l parsa. Utilizarea acestei funcții poate fi o măsură importantă de precauție în cazul în care sunt folosite intrări furnizate de către utilizator.

Utilizarea opțiunilor *auto_prepend_file* şi *auto_append_file*

Adăugarea *header*-ului şi a *footer*-ului în fiecare pagină se poate realiza şi în alt mod, cu ajutorul opțiunilor *auto_prepend_file* şi *auto_append_file* din fişierul *php.ini*. Setând valorile acestora la fişierele corespunzătoare *header*-ului, respectiv *footer*-ului, ne asigurăm că acestea vor fi încărcate înainte şi după fiecare pagină. Fişierele incluse în acest mod

se vor comporta ca şi cum ar fi fost adăugate cu ajutorul instrucțiunii *include*(); dacă fişierul lipseşte, va apărea un *warning*.

De exemplu, setarea acestor opțiuni în *Windows* se realizează astfel:

```
auto_prepend_file = "c:/Program Files/Apache Software
Froundation/Apache2.2//include/header.php"
auto_append_file = "c:/Program Files/Apache
Group/Apache2/include/footer.php"
```

Dacă este utilizat un server web *Apache*, se pot utiliza diferite opțiuni de configurare pentru directoare individuale. Pentru aceasta, este necesar ca server-ul să permită suprascrierea fişierului său principal de configurare. Pentru stabilirea opțiunilor anterioare pentru un director, trebuie creat un fişier denumit *.htaccess* în directorul respectiv. Acest fişier va trebui să conțină următoarele două linii (cu o sintaxă uşor diferită față de cea din *php.ini*):

```
php_value auto_prepend_file "/home/username/include/header.php"
php_value auto_append_file "/home/username/include/footer.php"
```

Stabilirea opțiunilor în fişierul *.htaccess*, în loc de *php.ini*, oferă multă flexibilitate. Astfel, pot fi afectate doar setările care se referă la directoarele proprii de pe un calculator partajat. Server-ul web nu trebuie repornit şi nu sunt necesare drepturi de administrator. Un dezavantaj al metodei care utilizează fişiere *.htaccess* este acela că fişierele sunt citite şi parsate de fiecare dată când un fişier din acel director este accesat, şi nu doar o singură dată la început.

Utilizarea funcțiilor în PHP

O funcție reprezintă un modul independent de cod care furnizează o interfață pentru apel, efectuează anumite acțiuni şi, opțional, returnează un rezultat.

Apelarea funcțiilor

Cel mai simplu apel este cel la o funcție (*function_name*) care nu are parametri şi ignoră valorile care ar putea fi returnate este:

```
function_name();
```

Funcția *phpinfo*() este apelată în acest mod şi este adesea utilă pentru testare deoarece afişează versiunea instalată de PHP, informații despre PHP, server-ul web şi valorile diferitelor variabile PHP sau ale server-ului.

Funcțiile care necesită parametri sunt apelate plasând datele sau variabilele între paranteze. Exemple de apeluri:

```
function_name('parameter'); //parametru sir de caractere
function_name(2);
function_name(7.993);
function_name($variable); //variabila poate avea orice tip,
                          //inclusiv vector sau obiect
```

Un parametru poate avea orice tip de date, dar funcțiile prevăd de obicei tipuri particulare de date pentru parametri.

Prototipul unei funcții furnizează numărul de parametri ai unei funcții şi tipurile de date ale acestora. De exemplu, prototipul funcției *fopen* este următorul:

```
resource fopen ( string filename, string mode
                 [, bool use_include_path [, resource context]])
```

În fața numelui funcției este specificat tipul de date al rezultatului. Parametrii funcției se află între paranteze şi sunt specificați împreună cu tipurile lor de date. Pentru parametrii opționali, acestora fie le sunt furnizate valori, fie sunt ignorați, caz în care sunt utilizate valorile lor implicite. Dacă o funcție are mai mult de un parametru opțional, pot fi ignorați doar parametrii situați cel mai la dreapta (în cazul funcției *fopen*, nu putem furniza valoare pentru *context* şi ignora parametrul *use_ignore_path*).

Apelarea unei funcții nedefinite

Dacă este apelată o funcție care nu există, va apărea un mesaj de eroare: http://193.226.51.37/web/curs4/undefined.php

Motivele pentru care o astfel de eroare poate apărea sunt următoarele:

- numele funcției nu a fost scris corect
- funcția nu există în versiunea de PHP utilizată
- funcția apelată face parte dintr-o extensie PHP care nu a fost încărcată.

Case-sensitivity

Apelurile la funcții nu sunt *case-sensitive*. Spre deosebire de acestea, numele variabilelor sunt *case-sensitive*.

Funcţii definite de utilizator

O declaraţie de funcţie începe cu cuvântul cheie *function*, specifică numele funcţiei, parametrii acesteia şi codul care va fi executat la fiecare apel al funcţiei.

Exemplu:

```
function my_function() {
      echo 'My function was called';
}
```

Funcţiile definite de utilizator sunt valabile doar în script-ul în care au fost declarate. Se recomandă să existe un fişier sau set de fişiere ce conţin funcţiile utilizate, iar apoi să avem o instrucţiune *require*() în script-urile în care funcţiile sunt apelate, astfel încât acestea să devină disponibile.

Corpul funcţiei (specificat între {}) poate conţine orice este permis într-un script PHP, incluzând apeluri de funcţii, declaraţii de variabile noi, funcţii, instrucţiuni *require*() sau *include*(), declaraţii de clase şi cod HTML. Dacă dorim să ieşim din PHP în cadrul unei funcţii şi să scriem cod HTML, acest lucru se poate realiza în acelaşi mod ca în script, cu un tag PHP de încheiere urmat de codul HTML.

Exemplu: Codul următor este o modificare a celui din exemplul precedent şi furnizează acelaşi rezultat.

```
<?php
function my_function() {
?>
      My function was called
<?php
}
?>
```

Numele funcţiilor

Numele funcţiilor trebuie să fie scurte, însă descriptive. Restricţiile care au loc asupra numelor funcţiilor sunt următoarele:

- două funcţii nu pot avea acelaşi nume
- numele funcţiei poate conţine doar litere, cifre şi _
- numele funcţiei nu poate începe cu o cifră.

Multe limbaje permit supraîncărcarea funcţiilor. În PHP acest lucru nu este posibil. Pe de altă parte, funcţiile definite de utilizator există doar în script-urile în care au fost declarate, prin urmare putem reutiliza numele unei

funcţii într-un alt fişier, însă acest lucru ar conduce la confuzie şi ar trebui evitat.

Exemplu: Următoarele nume de funcţii sunt permise:

```
name()
name2()
name_three()
_namefour()
```

Următoarele nume de funcţii nu sunt legale:

```
5name()
name-six()
fopen() // exista deja
```

Observaţie: Deşi *$name* nu este un nume de funcţie valid, apelul

```
$name();
```

poate fi corect, în funcţie de valoarea lui *$name*. PHP ia valoarea stocată în *$name*, caută o funcţie cu acel nume şi o apelează (funcţie variabilă).

Utilizarea parametrilor

Următoarea funcţie cere un parametru (un vector unidimensional) şi îl afişează sub forma unui tabel.

```
function create_table($data) {
      echo "<table border=\"1\">";
      reset($data); // Pointeaza la inceput
      $value = current($data);
      while ($value) {
            echo "<tr><td>".$value."</td></tr>\n";
            $value = next($data);
      }
      echo "</table>";
}
```

Apelul funcţiei:

```
$my_array = array('Line one.', 'Line two.', 'Line three.');
create_table($my_array);
```

produce următorul rezultat: http://193.226.51.37/web/curs4/create_table.php

Funcţiile definite de utilizator pot avea parametri multipli, iar unii dintre aceştia pot fi opţionali. Modificăm funcţia anterioară astfel încât să putem specifica bordura sau alte atribute ale tabelului:

```
<?php
function create_table2($data, $border=1, $cellpadding=4,
                       $cellspacing=4 ) {
      echo "<table border=\"".$border."\"
      cellpadding=\"".$cellpadding."\"
```

```
      cellspacing=\"".$cellspacing."\">";
      reset($data);
      $value = current($data);
      while ($value) {
            echo "<tr><td>".$value."</td></tr>\n";
            $value = next($data);
      }
      echo "</table>";
}
```

Această funcție poate fi apelată în următoarea secvență de cod:

```
$my_array = array('Line one.','Line two.','Line three.');
create_table2($my_array, 3, 8, 8);
```

Primul parametru al funcției este obligatoriu, iar următoriii 3 sunt opționali deoarece le-au fost specificate valori implicite. Rezultatul apelului anterior este : http://193.226.51.37/web/curs4/create_table2.php

Se pot declara funcții cu număr variabil de parametri. Putem afla câți parametri au fost transmişi, precum şi valorile acestora, cu ajutorul a 3 funcții: *func_num_args*(), *func_get_arg*() şi *func_get_args*().

Fie funcția:

```
function var_args() {
      echo "Numarul de parametri:";
      echo func_num_args();
      echo "<br />";
      $args = func_get_args();
      foreach ($args as $arg) {
            echo $arg."<br />";
      }
}
```

Această funcție arată numărul de parametri care i-au fost transmişi şi afişează valorile acestora. Funcția *func_num_args*() returnează numărul de argumente transmise. Funcția *func_get_args*() întoarce un vector de argumente. Alternativ, argumentele pot fi accesate unul câte unul cu ajutorul funcției *func_get_arg*(), căreia îi transmitem numărul argumentului pe care dorim să îl accesăm (începând cu 0).

Domeniu de vizibilitate

Atunci când anumite variabile trebuie folosite în interiorul unui fişier inclus, acestea trebuie declarate în script înainte de instrucțiunea *require*() sau *include*(). Când folosim o funcție, transmitem explicit acele variabile funcției.

Domeniul unei variabile controlează modul în care o variabilă este vizibilă şi utilizabilă. Limbajele de programare au diferite reguli privind domeniul de vizibilitate al variabilelor. Regulile din PHP sunt următoarele:

- Variabilele declarate în interiorul unei funcţii sunt vizibile începând cu instrucţiunea care le declară, până la încheierea funcţiei (}). Acesta se numeşte domeniul funcţiei, iar variabilele de acest tip se numesc variabile locale.
- Variabilele declarate în afara funcţiilor sunt vizibile începând cu instrucţiunea care le declară, până la sfârşitul fişierului, dar nu în interiorul funcţiilor. Un astfel de domeniu se numeşte domeniu global, iar variabilele de acest tip se numesc variabile globale.
- Variabilele speciale superglobale sunt vizibile atât în interiorul cât şi în exteriorul funcţiilor.
- Utilizarea instrucţiunilor *require*() şi *include*() nu afectează domeniul de vizibilitate. Dacă instrucţiunea este folosită în interiorul unei funcţii, se aplică domeniul funcţiei. În caz contrar, se aplică domeniul global.
- Cuvântul cheie *global* poate fi folosit pentru a obţine explicit dacă o variabilă definită sau utilizată în cadrul unei funcţii va avea domeniu global.
- Variabilele pot fi şterse manual apelând *unset*(*$nume_variabila*). O variabilă nu se mai află în domeniu dacă a fost aplicată funcţia *unset*().

Următorul cod nu produce nicio ieşire. În cadrul funcţiei este declarată o variabilă *$var*, al cărei domeniu va fi, astfel, domeniul funcţiei. Referirea la *$var* în exteriorul funcţiei creează de fapt o nouă variabilă *$var*, al cărei domeniu este global şi care va fi vizibilă până la sfârşitul fişierului.

```
function fn() {
      $var = "contents";
}
      fn();
echo $var; //$var nu a primit nicio valoare
```

Exemplul următor declară o variabilă în exteriorul funcţiei şi încearcă să o utilizeze în interiorul acesteia.

```
<?
function fn() {
      echo "inside the function, \$var = ".$var."<br />";
      $var = "contents 2";
      echo "inside the function, \$var = ".$var."<br />";
}
```

```
$var = "contents 1"; //domeniu global
fn(); //creează variabila locala $var
echo "outside the function, \$var = ".$var."<br />";
```

Fragmentul de cod anterior va afişa:

```
inside the function, $var =
inside the function, $var = contents 2
outside the function, $var = contents 1
```

Dacă dorim ca o variabilă creată în interiorul unei funcții să fie globală, putem utiliza cuvântul cheie *global*:

```
function fn() {
      global $var;
      $var = "contents";
      echo "inside the function, \$var = ".$var."<br />";
}
fn();
echo "outside the function, \$var = ".$var."<br />";
```

În acest fel, variabila *$var* va exista în exteriorul funcției, după ce aceasta este apelată. Fragmentul de cod anterior va afişa:

```
inside the function, $var = contents
outside the function, $var = contents
```

Cuvântul cheie *global* poate fi utilizat la începutul unui script, la prima utilizare a unei variabile, pentru a declara că domeniul acesteia este întregul script. Acesta este un mod de utilizare mai obişnuit al cuvântului cheie *global*.

Transfer prin referință şi transfer prin valoare

Considerăm că avem următoarea funcție pentru incementarea unei valori:

```
function increment($value, $amount = 1) {
      $value = $value +$amount;
}
```

Această funcție nu realizează nimic. Următorul fragment de cod va afişa valoarea 10:

```
$value = 10;
increment ($value);
echo $value;
```

Conținutul variabilei *$value* nu a fost modificat din cauza regulilor referitoare la domeniu. Codul precedent creează variabila *$value*, care conține valoarea 10, apoi apelează funcția *increment*(). Variabila *$value* din

funcție este creată atunci când funcția este apelată. Valoarea lui *$value* devine 11 în interiorul funcției, până la sfârşitul acesteia. După aceasta, se revine în codul apelant, în care *$value* este o variabilă diferită, cu domeniu global, şi prin urmare nemodificată.

Un mod de a rezolva această problemă constă în declararea lui *$value* ca variabilă globală în interiorul funcției, dar aceasta ar însemna că, pentru a utiliza această funcție, variabila pe care dorim să o incrementăm trebuie să se numească *$value*.

Acesta este transferul prin valoare al parametrilor funcției: la transferul unui parametru, este creată o nouă variabilă ce conține valoarea transferată. Valoarea poate fi modificată în funcție, însă valoarea variabilei originale, din exteriorul funcției, rămâne nemodificată.

Transferul prin referință al parametrilor constituie o abordare mai bună. Atunci când un parametru este transferat unei funcții, în loc de a crea o nouă variabilă, funcția primeşte o referință la variabila originală. Această referință are un nume de variabilă şi este utilizată exact ca o variabilă. Diferența este că, în loc de a avea o valoare proprie, doar referă valoarea unei variabile. Orice modificare adusă referinței afectează şi variabila originală.

Faptul că un parametru este transferat prin referință este specificat prin specificarea caracterului & în fața numelui parametrului. Nu va apărea nicio modificare în apelul funcției.

Pentru ca exemplul anterior să funcționeze corect, funcția trebuie modificată astfel:

```
function increment(&$value, $amount = 1) {
      $value = $value +$amount;
}
```

Utilizarea cuvântului cheie *return*

Cuvântul cheie *return* poate fi folosit pentru a întoarce o valoare furnizată de funcție sau pentru a opri execuția unei funcții. Atunci când o funcție se încheie, controlul va trece la următoarea instrucțiune după apelul funcției.

O condiție referitoare la o eroare este un motiv obişnuit pentru a încheia execuția unei funcții. De exemplu, funcția următoare returnează maximul dintre două numere, însă execuția sa se opreşte dacă vreunul dintre numere lipseşte:

```
function larger( $x, $y ) {
      if ((!isset($x)) || (!isset($y))) {
            echo "This function requires two numbers.";
            return;
```

```
      }
      if ($x>=$y) {
            echo $x."<br/">;
      } else {
            echo $y."<br/">;
      }
}
```

Funcția predefinită *isset*() spune dacă o variabilă a fost creată şi i-a fost acordată o valoare.

Funcția anterioară poate fi rescrisă astfel încât să returneze (nu să afişeze) valoarea cea mai mare:

```
function larger ($x, $y) {
      if ((!isset($x)) || {!isset($y))) {
            return false;
      } else if ($x>=$y) {
            return $x;
      } else {
            return $y;
      }
}
```

Observație: Valoarea returnată de un apel la funcția anterioară trebuie testată cu ===, pentru a nu se crea o confuzie între 0 şi false.

Recursivitate

PHP pemite definirea funcțiilor recursive. Astfel de funcții sunt utile pentru navigarea în structurile de date dinamice, cum ar fi listele înlănțuite sau arborii.

Este posibil ca recursivitatea să fie utilizată în locul iterațiilor, însă trebuie avut în vedere faptul că funcțiile recursive sunt mai lente şi utilizează mai multă memorie decât iterațiile, prin urmare ar trebui utilizate acestea din urmă.

Exemplu: Inversarea unui şir de caractere (2 variante)

```
<?php
function reverse_r($str) {
      if (strlen($str)>0) {
            reverse_r(substr($str, 1));
      }
      echo substr($str, 0, 1);
      return;
}

function reverse_i($str) {
      for ($i=1; $i<=strlen($str); $i++) {
            echo substr($str, -$i, 1);
```

```
        }
        return;
}
```

Spații de nume (*namespaces*)

Un spațiu de nume este un container abstract care reține un grup de identificatori; în PHP, aceasta înseamnă că spațiile de nume pot conține funcțiile, constantele şi clasele pe care le definim. Există câteva avantaje ale spațiilor de nume:

- Toate funcțiile, clasele şi constantele din cadrul unui *namespace* sunt prefixate automat cu numele acestuia.
- Numele claselor, funcțiilor şi ale constantelor sunt rezolvate la *runtime*, la prima căutare care are loc în *namespace*, înainte de a se trece la domeniul global.

Tratarea excepțiilor

Ideea de bază privind tratarea excepțiilor se referă la execuția codului care poate genera erori în interioul unui bloc *try*. Acesta este o secțiune de cod de forma următoare:

```
try
{
// cod sursa
}
```

Dacă apare o eroare în interiorul unui bloc *try*, putem declanşa o excepție. Anumite limbaje, precum *Java*, declanşează excepții în mod automat în anumite cazuri. În PHP, excepțiile trebuie declanşate manual, astfel:

```
throw new Exception('message', code);
```

Cuvântul cheie *throw* declanşează mecanismul de tratare a excepțiilor. Acesta este o construcție a limbajului şi nu o funcție, însă trebuie să îi transferăm o valoare, care este un obiect. În cazul cel mai simplu, se instanțiază clasa predefinită *Exception*, al cărei constructor primeşte doi parametri: un mesaj şi un cod. Ambii parametri sunt opționali.

La sfârşitul blocului *try*, este nevoie de cel puțin un bloc *catch*, a cărui formă este:

```
catch (typehint exception)
```

```
{
      // handle exception
}
```

Putem avea mai multe blocuri *catch*, fiecare tratând câte un tip diferit de excepție, asociate unui singur bloc *try*. Obiectul transferat în blocul *catch* este cel trimis în cadrul instrucțiunii *throw* care a declanşat excepția. Atunci când este declanşată o excepție, PHP caută blocul *catch* corespunzător. Dacă avem mai multe blocuri catch, *obiectele* transferate fiecăruia dintre acestea ar trebui să fie de tipuri diferite.

Un bloc *catch* poate declanşa, la rândul lui, alte excepții.

Exemplu:

```
<?php
try {
      throw new Exception("A terrible error has occurred", 42);
}
catch (Exception $e) {
      echo "Exception ". $e->getCode(). ": ".
                        $e->getMessage()."<br />".
            " in ". $e->getFile(). " on line ".
                    $e->getLine(). "<br />";
}
?>
```

Exemplul anterior produce următorul rezultat:
http://193.226.51.37/web/curs4/basic_exception.php

În exemplul anterior au fost utilizate câteva metode ale clasei *Exception*, care va fi discutată în continuare.

Clasa *Exception*

Constructorul acestei clase acceptă 2 parametri: mesajul şi codul erorii. Pe lângă acesta, clasa furnizează următoarele metode predefinite:

- *getCode*() – returnează codul erorii;
- *getMessage*() – returnează mesajul erorii;
- *getFile*() – returnează calea absolută a fişierului în care a apărut excepția;
- *getTrace*() – returnează un vector care conține un *backtrace*, acolo unde excepția a fost declanşată (funcțiile care se executau la momentul apariției excepției);
- *getTraceAsString*() – aceeaşi informație ca *getTrace*(), formatată ca string;

- *__toString*() – permite afişarea unui obiect *Exception*, furnizând informaţiile din metodele de mai sus.

Excepţii definite de utilizator

Înclauza *throw* poate fi transmis orice obiect. Acest lucru poate fi util la *debug*, dacă un anumit obiect creează probleme.

De cele mai multe ori, se extinde clasa *Exception*. Pentru aceasta, este necesar să îi cunoaştem structura:
http://www.php.net/manual/en/class.exception.php

Cele mai multe dintre metodele publice ale clasei *Exception* sunt finale, adică nu pot fi suprascrise. Putem crea propria subclasă, dar nu putem schimba comportamentul metodelor de bază. Funcţia *__toString*() poate fi modificată, astfel încât se poate modifica modul de afişare al excepţiei.

Următoarea clasă implementează o excepţie definită de utilizator:

```
<?php
class myException extends Exception
{
      function __toString()
      {
        return "<table border=\"1\">
                  <tr>
                  <td><strong>Exception ".$this->getCode()."
                  </strong>: ".$this->getMessage()."<br />"."
                        in ".$this->getFile()." on line ".
                        $this->getLine()."
                  </td>
                  </tr>
                </table><br />";
      }
}
try
{
      throw new myException("A terrible error has occurred", 42);
}
catch (myException $m)
{
      echo $m;
}
?>
```

Diferenţa dintre clasa anterioară şi *Exception* constă în faptul că afişarea se face într-un mod mai plăcut:
http://193.226.51.37/web/curs4/user_defined_exception.php

Exemple de excepții în site-ul web exemplu (componente auto)

Anterior, am studiat modul în care comenzile lansate prin intermediul site-ului web pot fi stocate într-un fişier. Operațiile I/O asupra fişierelor reprezintă o sursă de apariție frecventă a excepțiilor.

În codul corespunzător procesării formularului, pot apărea 3 probleme referitoare la fişiere:

- fişierul nu poate fi deschis
- nu poate fi obținută o blocare asupra fişierului
- nu se poate scrie în fişier.

Creem câte o clasă corespunzătoare fiecărui tip de excepție prevăzut; clasele vor fi salvate în fişierul *file_exceptions.php*:

http://193.226.51.37/web/curs4/file_exceptions_php.txt

Fişierul *processorder.php* este rescris astfel încât să utilizeze aceste excepții: http://193.226.51.37/web/curs4/processorder_php.txt

Au fost incluse doar două blocuri *catch*: unul care tratează excepțiile *fileOpenException*, iar celălalt tratează excepțiile de tip *Exception*. Deoarece celelelate două tipuri de excepții nu sunt tratate explicit, însă moştenesc clasa *Exception*, vor fi tratate de către al doilea bloc *catch*.

Dacă este declanşată o excepție pentru care nu există un bloc *catch* corespunzător, PHP va raporta o eroare fatală.

MySQL

În acest curs vom crea şi integra o bază de date într-un script PHP. Avantajele stocării datelor într-o bază de date, spre deosebire de salvarea datelor în fişiere, includ faptul că sistemele de gestiune a bazelor de date:

- pot furniza un acces mai rapid la date decât fişierele;
- pot fi interogate cu uşurință pentru a extrage mulțimi de date care îndeplinesc anumite criterii;
- dețin mecanisme predefinite pentru tratarea accesului concurent;
- furnizează acces aleator la date;
- furnizează mecanisme predefinite de asigurare a securității.

O bază de date permite obținerea rapidă a unor statistici utile (aflarea clientului care a cheltuit cel mai mult, a produsului celui mai bine vândut).

SGBD-ul pe care îl vom utiliza în acest curs este MySQL. Cursul se bazează pe noțiunile prezentate în cadrul cursurilor de „Baze de date” şi „Sisteme de gestiune a bazelor de date”, la care facem trimitere pentru conceptele şi terminologia bazelor de date relaționale, proiectarea şi arhitectura acestora.

Arhitectura unei baze de date *web*

Pornind de la arhitectura internă a unei baze de date, trecem la studiul arhitecturii externe a unui sistem de baze de date *web* şi vom discuta metodologia pentru dezvoltarea unui astfel de sistem de baze de date.

Operația principală a unui server *web* este prezentată în Figura 1. Sistemul constă din 2 obiecte: un *browser web* şi un *server web*, între care este necesară o legătură de comunicație. *Browser*-ul *web* lansează o cerere către server, iar server-ul trimite înapoi un răspuns. Această arhitectură este adecvată unui server care trimite pagini statice. Arhitectura care furnizează un website care lucrează cu o bază de date este mai complexă.

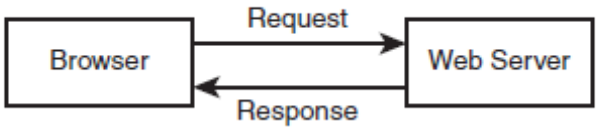

Figura 1

Aplicațiile cu baze de date *web* pe care le vom prezenta urmează o structură generală (Figura 2), care conține un *browser web*, un server *web*, motorul de *scripting* şi server-ul de baze de date.

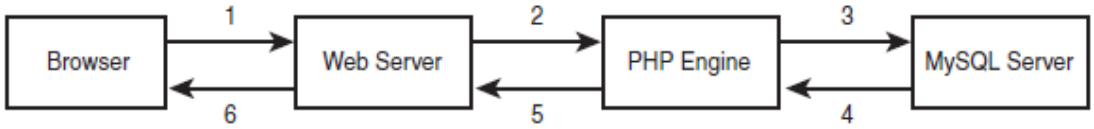

Figura 2

O tranzacție obişnuită într-o bază de date *web* constă din următoarele etape. Pentru o aplicație care lucrează cu o bază de date pentru o bibliotecă, aceste etape pot fi exemplificate astfel:

1. Un utilizator lansează, prin intermediul unui *browser web*, o cerere HTTP pentru o anumită pagină *web*. De exemplu, cu ajutorul unui formular HTML, utilizatorul solicită căutarea tuturor cărților scrise de Liviu Rebreanu. Pagina de căutare a rezultatelor se numeşte *results.php*.
2. Serverul *web* primeşte cererea pentru *results.php*, regăseşte fişierul şi îl transmite motorului PHP pentru procesare.
3. Motorul PHP începe parsarea script-ului. Scriptul conține o comandă pentru conectarea la baza de date şi execută o cerere (căutare de cărți). PHP deschide o conexiune către server-ul MySQL şi trimite cererea corespunzătoare.
4. Serverul MySQL primeşte cererea pentru baza de date, o procesează şi trimite rezultatul (lista de cărți) înapoi motorului PHP.
5. Motorul PHP încheie execuția script-ului, care de obicei presupune şi formatarea rezultatului cererii în HTML. Apoi, returnează rezultatul HTML către server-ul web.
6. Server-ul web trimite codul HTML browser-ului, unde utilizatorul va putea vizualiza lista de cărți solicitată.

Procesul este, în general, acelaşi indiferent de motorul de *scripting* sau de serverul de baze de date folosit. Adeseori, serverul *web*, motorul PHP şi

serverul de baze de date rulează toate pe aceeaşi maşină. Însă, nu este mai puțin obişnuit ca serverul de baze de date să ruleze pe o maşină diferită, din motive de securitate, capacitate crescută, sau distribuire a activităților. Din punct de vedere al dezvoltării, această abordare este similară, însă oferă avantaje din punct de vedere al performanțelor.

Pe măsură ce o aplicație creşte în dimensiune şi complexitate, ea va fi separată în niveluri (*tiers*), de cele mai multe ori un nivel al bazei de date care asigură interfața cu MySQL, un nivel al logicii aplicației care conține nucleul acesteia şi un nivel de prezentare care gestionează ieşirile în format HTML. Arhitectura din figura 2 se păstrează, diferența fiind aceea că secțiunea PHP devine mai structurată.

Crearea bazei de date web în *MySQL*

În cele ce urmează, vom prezenta modul în care se configurează o bază de date MySQL pentru a fi utilizată pe un site *web*. Exemplele vor face referință la un magazin de cărți online, care are o bază de date ale cărei scheme relaționale sunt:

```
Customers(CustomerID, Name, Address, City)
Orders(OrderID, CustomerID, Amount, Date)
Books(ISBN, Author, Title, Price)
Order_Items(OrderID, ISBN, Quantity)
Book_Reviews(ISBN, Reviews
```

Instalarea lui *MySQL* pe server-ul *web* include:

- instalarea fişierelor
- stabilirea unui utilizator care să ruleze MySQL ca administrator
- configurarea căii
- rularea script-ului *mysql_install_db*
- stabilirea parolei *root*
- ştergerea utilizatorului anonim şi testarea bazei de date
- pornirea server-ului MySQL prima dată şi configurarea ca acesta să pornească automat.

Crearea bazei de date şi a utilizatorilor

MySQL poate avea mai multe baze de date. De obicei, vom avea câte o bază de date pentru fiecare aplicație. În exemplul nostru, aceasta se va numi *books*.

Crearea bazei de date se efectuează simplu:

```
mysql> create database <dbname>;
```

Crearea utilizatorilor şi stabilirea privilegiilor

Utilizatorul *root* ar trebui utilizat doar în scopuri administrative, din motive de securitate. Pentru fiecare utilizator care doreşte să folosească sistemul trebuie creat un cont şi o parolă. Nu este necesar ca acestea să fie aceleaşi cu numele de utilizatori şi parolele din afara MySQL (cele din sistemul de operare). Acelaşi principiu se aplică lui *root*.

Atribuirea de parole utilizatorilor nu este obligatorie, dar se recomandă. În scopul configurării unei baze de date *web*, se recomandă stabilirea cel puțin a câte unui utilizator pentru fiecare aplicație web. Motivul pentru care se face această recomandare are legătură cu privilegiile.

Este necesar să avem un utilizator care, prin intermediul *script*-urilor PHP, se conectează la MySQL. Pentru aceasta se aplică principiul privilegiului celui mai mic: se acordă doar privilegiile necesare operațiilor pe care respectivul utilizator va trebui să le efectueze.

Presupunem că utilizatorul va trebui să execute doar comenzi LMD. Prin urmare, stabilirea privilegiilor acestuia se realizează astfel:

```
mysql> grant select, insert, delete, update
-> on books.*
-> to student identified by 'student123';
```

Dacă se utilizează un serviciu de web hosting, de obicei este acordat acces şi la alte privilegii asupra bazei de date pe care serviciul o creează. De obicei, primim acelaşi nume de utilizator şi parolă pentru a fi utilizate în linie de comandă (crearea tabelelor ş.a.m.d) şi pentru conexiunile prin intermediul script-ului (interogarea bazei de date). Utilizarea aceluiaşi nume de utilizator şi a aceleiaşi parole este mai puțin sigură. Putem configura un utilizator cu nivelul corespunzător de privilegii după cum urmează:

```
mysql> grant select, insert, update, delete, index, alter,
create, drop
-> on books.*
-> to student identified by 'student123';
```

Utilizarea bazei de date corespunzătoare

Primul pas după conectare constă în specificarea bazei de date pe care dorim să o utilizăm:

```
mysql> use dbname;
```

Comanda *use* poate fi evitată dacă specificăm baza de date la conectare:

```
mysql -D dbname -h hostname -u username -p
```

Script-ul pentru crearea tabelelor pentru aplicația referitoare la un magazin online de cărți este http://193.226.51.37/web/curs5/books.sql.

Un fişier SQL poate fi rulat cu ajutorul comenzii (trebuie furnizată cale completă către fişierul books.sql):

```
> mysql -h host -u student -D books -p < books.sql
```

Redirecționarea fişierelor presupune că va fi posibil ca acestea să poată fi editate într-un editor de text înainte de a fi executate.

Observații:

- *AUTO_INCREMENT* este o caracteristică specială a MySQL care poate fi utilizată asupra coloanelor întregi. Aceasta presupune că, dacă respectiva valoare nu este furnizată atunci când sunt inserate linii în tabel, MySQL va genera un identificator unic, mai mare cu 1 decât valorile existente în coloană. Putem avea o singură astfel de coloană în fiecare tabel.
- *UNSIGNED* specificat după un tip întreg presupune că acesta poate avea doar valori mai mari sau egale decât 0.
- Tipul de date *char* specifică şiruri de caractere de dimensiune fixă. Tipul de date *varchar* utilizează doar spațiul necesar, deci furnizează o optimizare în termeni de spațiu, însă tipul de date *char* este mai rapid.
- Tipul de date *int* reține numere întregi, iar *float* stochează numere reale în virgulă mobilă.
- Pentru datele calendaristice se utilizează tipul *date*.
- Tipul de date *TINYINT UNSIGNED* reține valori întregi între 0 şi 255.
- Tipul de date *text* permite stocarea textelor de dimensiuni mari.

Accesarea bazei de date *MySQL* utilizând *PHP*

Vom relua cei 6 paşi corespunzători arhitecturii unei baze de date *web*.

Mai întâi, se creează formularul de căutare HTML: http://193.226.51.37/web/curs5/search_html.txt.
Acesta produce http://193.226.51.37/web/curs5/search.html

Script-ul care va fi apelat atunci când este acționat butonul Search este http://193.226.51.37/web/curs5/results_php.txt

Rezultatul în urma unei căutări cu ajutorul script-ului se poate observa introducând în căutare „*Java 2 for Professional Developers*".

În continuare, vom explica ce face acest script şi modul în care lucrează el.

Interogarea unei baze de date de pe *web*

Într-un script utilizat pentru a accesa o bază de date de pe *web*, au loc următoarele acțiuni:

1. Verificarea şi filtrarea datelor care au fost introduse de către utilizator.
2. Stabilirea unei conexiuni la baza de date corespunzătoare.
3. Interogarea bazei de date.
4. Recuperarea rezultatelor.
5. Prezentarea rezultatelor către utilizator.

Aceştia sunt paşii care au fost urmăriți în fişierul *results.php*.

Verificarea şi filtrarea datelor de intrare

Script-ul începe prin a elimina orice spațiu liber pe care utilizatorul l-ar fi putut introduce din greşeală la începutul sau sfârşitul termenului său de căutare:

```
$searchterm=trim($_POST['searchterm']);
```

Următorul pas este de a căuta dacă utilizatorul a introdus un termen de căutare şi a selectat un tip de căutare:

```
if (!$searchtype || !$searchterm) {
      echo "You have not entered search details. Please go back
      and try again.";
      exit;
}
```

Filtrarea intrărilor din partea utilizatorului include filtrarea caracterelor de control. În acest sens, sunt utile funcțiile *addslashes*(), *stripslashes*() şi *get_magic_quotes_gpc*(). În MySQL trebuie introduse secvențe *escape* asupra datelor atunci când acestea provin din partea utilizatorului.

În acest caz, se utilizează valoarea returnată de funcția *get_magic_quotes_gpc*(), care verifică dacă introducerea apostrofurilor se realizează automat. În caz contrar, se utilizează funcția *addslash*() pentru a introduce secvențele *escape* corespunzătoare:

```
if (!get_magic_quotes_gpc()) {
```

```
        $searchtype = addslashes($searchtype);
        $searchterm = addslashes($searchterm);
}
```

Funcția *stripslashes*() se utilizează asupra datelor care se întorc din baza de date. Dacă este activată opțiunea *magic quotes*, datele vor conține caractere *slash* atunci când se întorc din baza de date, deci trebuie eliminate.

În exemplul nostru, utilizăm *htmlspecialchars*()pentru a codifica acele caractere care au semnificații speciale în HTML. Datele curente de test nu includ caractere &, <, > sau „, însă unele titluri de cărți pot conține &. Prin utilizarea acestei funcții, se elimină posibilele erori.

Stabilirea unei conexiuni

Biblioteca PHP pentru conectarea la MySQL se numeşte *mysqli* (*i* = improved). Atunci când se utilizează această bibliotecă, folosim fie o sintaxă procedurală, fie una orientată pe obiecte.

Pentru a realiza o conexiune la server-ul MySQL, vom avea următoarea linie de cod în script:

```
@ $db = new mysqli('localhost', 'student', 'student123',
'books');
```

Această linie de cod instanțiază clasa *mysqli* şi creează o conexiune la *localhost* cu numele de utilizator şi parola specificate. Conexiunea este configurată pentru a utiliza baza de date *books*.

Utilizând această abordare orientată pe obiecte, putem invoca metode asupra acestui obiect pentru a accesa baza de date. Dacă este preferată abordarea procedurală, linia de cod corespunzătoare este următoarea:

```
@ $db = mysqli_connect('localhost', 'bookorama', 'bookorama123',
'books');
```

Această funcție returnează o resursă, care reprezintă conexiunea la baza de date. Resursa va fi apoi transferată tuturor celorlalte funcții *mysqli*, similar modului în care lucram cu fişiere.

Majoritatea funcțiilor *mysqli* au o interfață orientată-obiect şi una procedurală. În general, numele funcțiilor din versiunea procedurală încep cu *mysql_* şi necesită ca argument resursa obținută cu ajutorul funcției *mysqli_connect*(). Conexiunile la baza de date constituie o excepție de la această regulă deoarece ele se pot realiza de către constructorul obiectului *mysqli*.

Rezultatul încercării de conectare trebuie verificat, deoarece nimic nu va funcționa dacă nu s-a obținut o conexiune validă. Verificarea se poate realiza astfel, atât în versiunea procedurală cât şi în cea orientată obiect:

```
if (mysqli_connect_errno()) {
      echo 'Error: Could not connect to database. Please try
            again later.';
      exit;
}
```

Funcția *mysql_connect_errno*() returnează 0 dacă nu au apărut erori la conectare, altfel returnează numărul erorii.

Remarcăm că linia de cod care realizează conectarea la baza de date începe cu @ pentru suprimarea erorilor. În acest mod, putem trata erorile corespunzător. Acelaşi lucru se poate realiza şi cu ajutorul excepţiilor.

Există o limită a conexiunilor MySQL care pot fi active la un moment dat. Parametrul MySQL *max_conections* determină această limită. Scopul acestui parametru şi al lui *MaxClients* din *Apache* este acela de a determina server-ul să respingă noile conexiuni, prevenind ca resursele să fie utilizate complet în perioade aglomerate.

Ambele valori ale acestor parametri pot fi modificate în fişierele de configurare: *httpd.conf* (pentru *MaxClients*) şi *my.conf* (pentru *max_connections*).

Alegerea bazei de date

În linia de comandă specificăm baza de date care va fi utilizată astfel:

```
use books;
```

Acest lucru este necesar şi atunci când conexiunea se realizează pe *web*. Baza de date va fi specificată ca parametru al constructorului *mysqli* sau al funcției *mysqli_connect*(). Dacă dorim să schimbăm baza de date, putem utiliza funcția *mysqli_select_db*(), care poate fi accesată:

```
$db->select_db(dbname)
```

sau

```
mysqli_select_db(db_resource, db_name)
```

Interogarea bazei de date

Pentru a efectua o interogare se utilizează funcția mysql_query(). Înainte de aceasta, se stabileşte cererea care va fi executată.

```
$query = "select * from books where ".$searchtype." like
'%".$searchterm."%'";
```

În acest caz este căutată valoarea introdusă de către utilizator ($searchterm) în câmpul specificat de către utilizator ($searchtype).

Cererea care este trimisă către MySQL nu trebuie să conţină „;”, spre deosebire de cele lansate direct în MySQL.

Cererea poate fi executată astfel:

```
$result = $db->query($query);
```

sau

```
$result = mysqli_query($db, $query);
```

Versiunea OO returnează un obiect, iar cea procedurală o resursă. În cazul unei erori, funcţia returnează *false*.

Regăsirea rezultatelor cererii

Există funcţii care permit obţinerea de informaţii din obiectul sau resursa rezultată.

În exemplul nostru, sunt numărate liniile returnate de cerere şi, de asemenea, este utilizată funcţia *mysqli_fetch_assoc*().

Atunci când este utillizată abordarea OO, numărul de linii returnate este stocat în atributul *num_rows* al obiectului rezultat şi poate fi accesat astfel:

```
$num_results = $result->num_rows;
```

În varianta procedurală, funcţia *mysqli_num*() returnează această informaţie pentru resursa furnizată ca parametru:

```
$num_results = mysqli_num_rows($result);
```

Această valoare ne permite să procesăm sau să afişăm rezultatul cererii, utilizând o instrucţiune de ciclare:

```
for ($i=0; $i <$num_results; $i++) {
      // process results
}
```

Instrucţiunea de ciclare nu se va executa dacă nu a fost returnată nicio linie. În fiecare iteraţie a acestei instrucţiuni de ciclare , va trebui apelat `$result -> fetch_assoc()` (sau `mysqli_fetch_assoc()`). Această funcţie ia fiecare linie din mulţimea rezultat şi o returnează ca vector, în care fiecare atribut este cheie şi îi corespunde valoarea de pe linia respectivă:

```
$row = $result->fetch_assoc();
```

sau

```
$row = mysqli_fetch_assoc($result);
```

Dat acest vector, putem parcurge şi afişa fiecare câmp:

```
echo "<br />ISBN: ";
```

```
echo stripslashes($row['isbn']);
```

Pot fi utilizate câteva variante pentru a obține informații din rezultat. În locul unui vector cu chei, rezultatele pot fi regăsite într-un vector cu enumerare utilizând *mysql_fetch_row*():

```
$row = $result->fetch_row($result);
```

sau

```
$row = mysqli_fetch_row($result);
```

Valorile atributelor se află în fiecare dintre valorile tabloului *$row*[0], *$row*[1] etc.

De asemenea, o înregistrare poate fi regăsită în cadrul unui obiect cu ajutorul funcției *mysql_fetch_object*():

```
$row = $result->fetch_object();
```

sau

```
$row = mysqli_fetch_object($result);
```

Putem accesa fiecare atribut prin *$row -> title*, *$row -> author* etc.

Deconectarea de la o bază de date

Putem elibera mulțimea rezultat utilizând:

```
$result->free();
```

sau

```
mysqli_free_result($result);
```

Apoi, pentru închiderea bazei de date scriem:

```
$db->close();
```

sau

```
mysqli_close($db);
```

Utilizarea acestei funcții nu este obligatorie deoarece script-ul va închide conexiunea la terminarea execuției.

Introducerea de informații în baza de date

Inserarea de informații noi în baza de date este asemănătoare modului în care acestea sunt obținute din baza de date. Sunt urmați aceiaşi paşi: crearea unei conexiuni, trimiterea unei cereri şi verificarea rezultatelor.

Cererea care este trimisă corespunde, de data aceasta, unei comenzi INSERT.

Considerăm formularul
http://193.226.51.37/web/curs5/newbook.html.
Codul acestuia este
http://193.226.51.37/web/curs5/newbook_html.txt.

Rezultatele acestui formular sunt transmise script-ului *insert_book.php*, care preia informațiile, realizează câteva validări şi încearcă să le scrie în baza de date.

Acest script este: http://193.226.51.37/web/curs5/insert_book_php.txt
Adăugăm corect o carte şi observăm ce se obține.

Observăm că script-ul *insert_book.php* este similar celui care regăsea date din baza de date. Verificăm dacă au fost completate toate câmpurile formularului şi le formatăm corect pentru inserarea în baza de date cu *addslashes*():

```
if (!get_magic_quotes_gpc()) {
      $isbn = addslashes($isbn);
      $author = addslashes($author);
      $title = addslashes($title);
      $price = doubleval($price);
}
```

Deoarece prețul este stocat în baza de date ca număr real, nu trebuie să îi adăugăm caractere „\”. Putem obține acelaşi efect de filtrare a caracterelor necorespunzătoare asupra unui câmp numeric utilizând funcția *doubleval*() (de exemplu, aceasta va trata simbolurile monetare care ar fi putut fi adăugate din greşeală).

Conexiunea la baza de date se realizează instanțiind obiectul *mysqli* şi va fi creată o cerere care va fi trimisă la baza de date şi executată:

```
$query = "insert into books values
('".$isbn."', '".$author."', '".$title."', '".$price."')";
$result = $db->query($query);
```

O diferență între utilizarea lui SELECT şi a lui INSERT este dată de folosirea funcției *mysqli_affected_rows*() sau a metodei corespunzătoare:

```
echo $db->affected_rows." book inserted into database.";
```

Această funcție se utilizează în cazul instrucțiunilor de modificare a bazei de date; în cazul unei cereri SELECT, funcția corespunzătoare este *mysqli_num_rows*().

Utilizarea instrucțiunilor „pregătite”

Biblioteca *mysqli* permite utilizarea cererilor pregătite. Acestea permit o execuție mai rapidă atunci când este executată de mai multe ori aceeaşi cerere cu date diferite. De asemenea, aceste cereri constituie un mijloc de prevenire a atacurilor *SQL Injection*.

Conceptul de bază al unei cereri pregătite este acela că trimitem un *template* al cererii pe care vom dori să o executăm server-ului MySQL, iar apoi trimitem datele separat. Putem trimite mai multe date unei cereri, funcționalitate utilă în cadrul inserărilor de tip *bulk*.

În cadrul script-ului *insert_books.php* putem utiliza instrucțiuni pregătite astfel:

```
$query = "insert into books values(?, ?, ?, ?)";
$stmt = $db->prepare($query);
$stmt->bind_param("sssd", $isbn, $author, $title, $price);
$stmt->execute();
echo $stmt->affected_rows.' book inserted into database.';
$stmt->close();
```

La crearea cererii, în loc de a furniza valori ale atributelor se utilizează caracterul „?” pentru fiecare dată. Linia de cod *$db -> prepare* (*mysqli_stmt_prepare*()) construieşte un obiect sau resursă instrucțiune, care va fi utilizată pentru a realiza procesarea efectivă.

Obiectul instrucțiune are o metodă numită *bind_param*() (*mysql_stmt_bind_param*()). Scopul acestei metode este de a comunica lui PHP variabilele care ar trebui înlocuite caracterelor „?”. Primul parametru este un şir de caractere care specifică formatul, similar funcției *printf*. Valoarea „sssd” specifică faptul că avem 4 parametri, tipurile de date ale primilor 3 fiind *string* iar al celuilalt *double*. Alte caractere posibile în acest şir care specifică formattul sunt : *i* (numere întregi), *b* (*blob*).

Apelul la *$stmt -> execute* (*mysqli_stmt_execute*()) efectuează cererea. Ulterior, poate fi accesat numărul de linii afectate şi se poate încheia instrucțiunea.

De ce este utilă o astfel de instrucțiune pregătită? Putem modifica valorile celor 4 variabile legate şi reexecuta instrucțiunea fără a o mai pregăti. Această funcționalitate este utilă la ciclarea în cazul inserărilor *bulk*.

Similar legării parametrilor, se pot lega rezultatele. Pentru cererile de tip SELECT se poate utiliza *$stmt -> bind_result()* (sau *mysqli_stmt_bind_result*()) pentru a furniza lista variabilelor în care recuperăm coloanele din rezultat. La fiecare apel al *$stmt -> fetch*() (sau

mysqli_stmt_fetch()), valorile coloanelor din următoarea linie din mulțimea rezultat sunt atribuite acestor variabile legate. De exemplu, în script-ul anterior avem:

```
$stmt->bind_result($isbn, $author, $title, $price);
```

Această instrucțiune leagă cele 4 variabile de cele 4 coloane care vor fi returnate de către cerere.

După apelul

```
$stmt->execute();
```

se poate apela în instrucțiunea de ciclare:

```
$stmt->fetch();
```

La fiecare apel al acesteia , următoarea linie rezultat va fi regăsită în cele 4 variabile legate.

Putem utiliza funcțiile *mysqli_stmt_bind_param*() şi *mysqli_stmt_bind_result*() în cadrul aceluiaşi script.

MySQL

Elemente de programare avansată *MySQL*

Instrucțiunea *LOAD DATA IN FILE*

Această instrucțiune permite să încărcăm datele dintr-un fişier în baza de date. Comanda are mai multe opțiuni, însă utilizarea cea mai frecventă are următoarea formă:

```
LOAD DATA INFILE "newbooks.txt " INTO TABLE books;
```

Această linie citeşte date din fişierul *newbooks.txt* în tabelul *books*. Implicit, câmpurile trebuie separate în fişier prin *tab*-uri şi cuprinse între apostrofuri, iar fiecare înregistrare trebuie separată prin *newline* (*n*). Caracterele speciale trebuie tratate cu *backslash* (\). Toate aceste caracteristici sunt configurabile cu ajutorul diferitelor opțuni ale instrucțiunii *LOAD*.

Motoare de stocare

MySQL suportă diferite motoare de stocare, uneori denumite tipuri de tabele. Aceste motoare permit alegerea implementării interne a tabelelor. Fiecare tabel din baza de date poate utiliza un motor de stocare diferit şi putem realiza conversii între acestea.

Putem alege tipul de tabel în comanda de creare a tabelului astfel:

```
CREATE TABLE table TYPE=type ....
```

Tipurile de tabele disponibile sunt:

- *MyISAM* – acest tip este cel implicit şi se bazează pe tipul *ISAM* (*Indexed Sequential Access Method*), o metodă standard pentru stocarea înregistrărilor şi a fişierelor. *MyISAM* adaugă un număr de avantaje acestui tip. Comparativ cu celelalte motoare de stocare, acesta are cele mai multe instrumente pentru verificarea şi repararea tabelelor. Tabelele *MyISAM* pot fi compresate şi suportă căutarea de text. Nu sunt sigure la tranzacții şi nu suportă chei externe.

- *MEMORY* (anterior denumit *HEAP*) – tabelele de acest tip sunt stocate în memorie, iar indecşii lor sunt de tip *hash*. Aceste caracteristici determină ca tabelele de acest tip să fie extrem de rapide dar, în cazul unei căderi, datele vor fi pierdute. Prin urmare, tabelele de acest tip sunt ideale pentru stocarea datelor temporare sau derivate. Va trebui specificată opțiunea *MAX_ROWS* în instrucțiunea *CREATE TABLE*. Aceste tabele nu pot avea coloane *BLOB*, *TEXT* sau *AUTO INCREMENT*.
- *MERGE* – aceste tabele permit tratarea unui set de tabele *MyISAM* ca pe un singur tabel în scopul interogării acestuia.
- *ARCHIVE* – aceste tabele stochează cantități mari de date, suportă doar operații *INSERT* şi *SELECT*. Nu sunt utilizați indecşi.
- *CSV* – aceste tabele sunt stocate pe server într-un singur fişier ce conține valori separate prin virgule. Beneficiul acestor tabele apare atunci când dorim să lucrăm cu datele respective într-o altă aplicație (de exemplu, Microsoft Excel).
- *InnoDB* – aceste tabele sunt sigure la tranzacții; aceasta înseamnă că furnizează funcționalități de *COMMIT* şi *ROLLBACK*. De asemenea, aceste tabele suportă cheile externe. Deşi sunt mai lente decât tabelele *MyISAM*, posibilitatea de a folosi tranzacții în aplicație constituie un compromis benefic.

În cele mai multe aplicații web, se folosesc fie tabele *MyISAM*, fie *InnoDB*, sau o combinație a celor două.

Tabelele *MyISAM* ar trebui folosite atunci când utilizăm un număr mare de operații *INSERT* sau *SELECT* pe un tabel (nu amestecate) deoarece furnizează cel mai rapid mod de a efectua aceste operații (de exemplu, pentru cataloage). De asemenea, *MyISAM* ar trebui folosite atunci când sunt necesare funcționalitățile sale de căutare de text. Tabelele *InnoDB* ar trebui utilizate atunci când tranzacțiile sunt importante în aplicație, cum ar fi tabelele care stochează date financiare sau pentru situațiile în care avem operații *INSERT* şi *SELECT* laolaltă, cum ar fi mesajele online sau forumurile.

Tabelele MEIMORY se folosesc în cazul tabelelor temporare sau pentru a implementa vizualizări, iar tabelele *MERGE* se utilizează dacă este nevoie de tabele *MyISAM* foarte mari.

Tipul unui tabel poate fi modificat după creare sa astfel:

```
alter table orders type=innodb;
alter table order_items type=innodb;
```

Utilizarea tranzacțiilor

Implicit, *MySQL* rulează în mod *autocommit*. Aceasta înseamnă că fiecare instrucțiune care este executată va fi scrisă imediat în baza de date (salvată). Dacă utilizăm un tabel care permite siguranța tranzacțiilor, nu dorim acest comportament.

Pentru a dezactiva modul *autocommit* în sesiunea curentă lansăm comanda:

```
set autocommit=0;
```

Dacă modul *autocommit* este *on*, o tranzacție poate fi începută cu ajutorul comenzii:

```
start transaction;
```

Dacă modul *autocommit* este *off*, această comandă nu este necesară deoarece este începută automat o tranzacție atunci când lansăm o instrucțiune *SQL*.

După ce au fost introduse comenzile care alcătuiesc tranzacția, putem salva în baza de date efectul acestor comenzi prin intermediul instrucțiunii *commit*. Anularea tranzacției se realizează cu *rollback*. Efectele unei tranzacții nu sunt vizibile celorlalți utilizatori sau celorlalte sesiuni până când aceasta este salvată.

Proceduri stocate

O procedură stocată este un subprogram creat şi stocat în *MySQL*. Aceasta poate conține instrucțiuni *SQL* şi structuri speciale de control.

Exemplu:

```
# O procedura simpla
delimiter //

create procedure total_orders (out total float)
BEGIN
     select sum(amount) into total from orders;
END
//
delimiter ;
```

Instrucțiunea *delimiter*// modifică delimitatorul pentru sfârşitul instrucțiunii de la valoarea curentă (de obicei, caracterul „;”) la „//”. Acest lucru se realizează pentru a putea utiliza delimitatorul „;” în cadrul procedurii stocate pe măsură ce codul este scris fără ca *MySQL* să încerce să îl execute.

Parametrii procedurii pot avea modul *IN* (valoarea parametrului este transmisă procedurii), *OUT* (parametrul este returnat) sau *INOUT* (valoarea este transmisă, dar poate fi modificată de către procedură).

Specificarea parametrilor este diferită față de cea din sistemul *Oracle*:

mod_parametru nume_parametru tip_de_date

Corpul procedurii stocate este cuprins între cuvintele cheie *BEGIN* şi *END*. O instrucțiune *SELECT* în corpul procedurii stocate va conține o clauză suplimentară, *INTO*.

După declararea procedurii, aceasta pate fi apelată cu ajutorul instrucțiunii *call*:

```
call total_orders(@t);
```

Această instrucțiune apelează funcția şi transmite o variabilă în care va fi stocat rezultatul. Pentru a vizualliza rezultatul, afişăm valoarea acestei variabile:

```
select @t;
```

O funcție stocată se creează similar unei proceduri, cu deosebirea că aceasta acceptă doar parametri de intrare şi returnează o singură valoare.

Exemplu:

```
# O functie simpla
delimiter //
create function add_tax (price float) returns float
return price*1.1;
//
delimiter ;
```

În funcție, nu este nevoie să specificăm modul parametrilor (aceştia sunt toți de mod *IN*). Clauza *returns* specifică tipul de date al valorii returnate. În corpul funcției, valoarea va fi returnată utilizând instrucțiunea *return*.

Cuvintele cheie *BEGIN* şi *END* nu sunt necesare dacă există o singură comandă în corp. Apelul unei funcții se poate realiza astfel:

```
select add_tax(100);
```

Vizualizarea procedurii şi a funcției create se poate realiza cu:

```
show create procedure total_orders;
show create function addtax;
```

Suprimarea acestora are loc cu ajutorul comenzilor:

```
drop procedure total_orders;
drop function add_tax;
```

Subprogramele stocate au capacitatea de a utiliza structuri de control, variabile, cursoare şi *handler*-e *DECLARE*.

Variabile locale

Variabilele locale pot fi declarate în cadrul unui bloc *begin ... end* utilizând o instrucțiune *declare*, în care se specifică numele variabilei, tipul de date al acesteia şi, opțional, valoarea inițială.

Exemplu: Modificăm funcția *add_tax*, astfel încât acesta să utilizeze o variabilă locală, care să rețină valoarea TVA-ului:

```
delimiter //
create function add_tax (price float) returns float
begin
     declare tax float default 0.10;
     return price*(1+tax);
end
//
delimiter ;
```

Cursoare şi structuri de control

Presupunem că dorim să construim o procedură stocată care determină comanda cea mai scumpă şi returnează codul acesteia. Procedura corespunzătoare este:

```
delimiter //
create procedure largest_order(out largest_id int)
begin
  declare this_id int;
  declare this_amount float;
  declare l_amount float default 0.0;
  declare l_id int;
  declare done int default 0;
  declare continue handler for sqlstate '02000' set done = 1;
  declare c1 cursor for select orderid, amount from orders;
  open c1;
  repeat
     fetch c1 into this_id, this_amount;
     if not done then
          if this_amount > l_amount then
               set l_amount=this_amount;
               set l_id=this_id;
          end if;
     end if;
  until done end repeat;
  close c1;
  set largest_id=l_id;
```

```
end
//
delimiter ;
```

Fragmentul de cod anterior utilizează structuri de control (condiționale şi de ciclare), cursoare şi *handler*-e declarative.

Linia de cod:

```
declare continue handler for sqlstate '02000' set done = 1;
```

se numeşte *handler* declarativ. Acesta este similar unei excepții în procedurile stocate. Există *handler*-e *continue* şi *exit*. Un *handler continue*, de genul celui din exemplul anterior, realizează acțiunea specificată şi apoi continuă execuția procedurii. *Handler*-ele *exit* determină ieşirea din cel mai apropiat bloc *begin ... end*.

Următoarea parte a *handler*-ului declarativ specifică momentul în care *handler*-ul va fi apelat. În exemplu, acesta va fi apelat atunci când *sqlstate* ajunge la valoarea '02000', ceea ce înseamnă că va fi apelat atunci când nu mai sunt găsite înregistrări (mulțimea rezultat fiind procesată linie cu linie). Echivalent, se mai poate specifica opțiunea *FOR NOT FOUND*. Alte opțiuni disponibile sunt *SQLWARNING* şi *SQLEXCEPTION*.

Cursorul regăseşte mulțimea rezultat a unei cereri (returnată de *mysqli_query*()) şi permite procesarea a câte unei linii (similar modului în care am realiza acest lucru utilizând *mysqli_fetch_row*()).

Linia de cod:

```
declare c1 cursor for select orderid, amount from orders;
```

conține doar definiția cursorului, cererea nefiind încă executată. Cererea este executată atunci când are loc:

```
open c1;
```

Pentru a obține fiecare linie, trebuie executată câte o instrucțiune *fetch*, într-o instrucțiune repetitivă.

Instrucțiunile de ciclare care pot fi utilizate în procedurile stocate sunt de următoarele tipuri:

- ```
  repeat
  ...
  until done end repeat;
  ```

- ```
  while condition do
  ...
  end while;
  ```

- ```
  loop
  ...
  end loop
  ```

Aceasta din urmă nu specifică o condiție, ieşirea realizându-se prin intermediul instrucțiuni *leave*.

Se observă că nu există instrucțiuni repetitive *for*.

Instrucțiunea:

```
fetch c1 into this_id, this_amount;
```

regăseşte o linie din cursor. Cele două atribute regăsite de cerere sunt stocate în cele două variabile locale.

Variabilelor locale le sunt atribuite valori cu ajutorul instrucțiunii *set*.

Instrucțiunile condiționale care pot fi utilizate în procedurile stocate pot avea următoarele forme:

- ```
  if condition then
  ...
  [elseif condition then]
  ...
  [else]
  ...
  end if
  ```

- ```
  case value
  when value then statement
  [when value then statement ...]
  [else statement]
  end case
  ```

Referitor la exemplul anterior, după ce instrucțiunea de ciclare se încheie, fiind procesate toate liniile regăsite de cursor, acesta trebuie închis cu ajutorul comenzii:

```
close c1;
```

În finalul procedurii anterioare, i se atribuie parametrului OUT valoarea calculată. Acest parametru nu poate fi utilizat ca variabilă temporară, ci doar pentru a stoca valoarea finală.

Procedura poate fi apelată în modul următor:

```
call largest_order(@l);
select @l;
```
```

Tehnici *PHP* avansate

Generarea imaginilor

PHP are funcții predefinite pentru prelucrarea imaginilor. De asemenea, putem utiliza biblioteca *GD2* pentru a crea imagini noi sau pentru a le prelucra pe cele existente. Vom prezenta modul în care pot fi utilizate aceste funcții, făcând referire la generarea butoanelor unui site *web* şi la desenarea histogramelor pe baza datelor dintr-o bază de date *MySQL*.

Vom utiliza biblioteca *GD2*. Pe lângă aceasta, mai există o bibliotecă *PHP* pentru prelucrarea imaginilor: *ImageMagick*. Aceasta nu face parte din standardul *PHP*, dar se instalează uşor din *PHP Extension Class Library* (*PECL*). Aceste două biblioteci au un număr mare de trăsături similare, dar în anumite privințe *ImageMagick* duce lucrurile mai departe. Dacă dorim să creăm imagini *GIF* (chiar animate), atunci se recomandă să utilizăm *ImageMagick*.

Configurarea suportului pentru imagini în *PHP*

Unele funcții pentru imagini sunt disponibile oricum, însă majoritatea necesită biblioteca *GD2*. Începând cu versiunea 4.3, *PHP* are propria versiune de bibliotecă *GD2*, care este mai uşor de instalat şi mai stabilă. Sub *Windows*, imaginile *PNG* şi *JPEG* sunt suportate automat dacă a fost înregistrată extensia *php_gd2.dll*. Acest lucru se poate realiza copiind fişierul *php_gd2.dll* din directorul de instalare *PHP* (subdirectorul *\ext*) în directorul sistemului (*C:\Windows\system*). De asemenea, în fişierul *php.ini* trebuie eliminat comentariul (;) de la începutul liniei:

```
extension=php_gd2.dll
```

Biblioteca *GD* suportă formatele *JPEG*, *PNG* şi *WBMP*.

Crearea imaginilor

Cei patru paşi de bază în crearea unei iamgini în *PHP* sunt:

1. Crearea unei imagini *canvas* pe care urmează să lucrăm.
2. Desenarea de forme sau tipărirea de text pe acel canvas.
3. Generarea imaginii finale.
4. Curățarea resurselor.

Exemplu:

```
<?php
  // set up image
  $height = 200;
  $width = 200;
  $im = imagecreatetruecolor($width, $height);
  $white = imagecolorallocate ($im, 255, 255, 255);
  $blue = imagecolorallocate ($im, 0, 0, 64);

// draw on image
  imagefill($im, 0, 0, $blue);
  imageline($im, 0, 0, $width, $height, $white);
  imagestring($im, 4, 50, 150, 'Sales', $white);

// output image
  Header ('Content-type: image/png');
  imagepng ($im);

// clean up
  imagedestroy($im);
?>
```

Rezultatul acestui script este următorul:
http://193.226.51.37/web/curs6/simplegraph.php

Crearea unei imagini *canvas*

Dacă dorim să construim sau să modificăm o imagine în *PHP*, trebuie să creăm un identificator al acesteia. Aceasta se poate realiza în două moduri:

- Crearea unui canvas vid, care se poate realiza cu ajutorul funcției *imagecreatetruecolor*() (ca în exemplul anterior). Funcția acceptă doi parametri care specifică lățimea, respectiv înălțimea noii imagini, şi returnează identificatorul acesteia.
- Citirea unui fişier imagine existent care poate fi apoi filtrat, redimensionat sau căruia i se pot adăuga elemente noi. Utilizarea funcțiilor *imagecreatefrompng*(), *imagecreatefromjpeg*() şi *imagecreatefromgif*(), în funcție de formatul imaginii pe care o citim, permite acest lucru.
  Aceste funcții acceptă ca parametru numele fişierului.

Desenarea sau tipărirea de text pe o imagine

Pentru a desena sau a tipări text pe o imagine trebuie să parcurgem două etape:

- Selectarea culorilor cu care dorim să desenăm. Culorile sunt alcătuie din diferite proporții de roşu (*R*), verde (*G*) şi albastru (*B*). Formatele de imagine utilizează o paletă de culori care constă dintr-o submulțime a tuturor combinațiilor posibile a celor 3 culori. Pentru a utiliza o culoare într-o imagine, trebuie să adăugăm culoarea în paleta imaginii.
  Culorile pot fi selectate cu ajutorul funcției *ImageColorAllocate*(), căreia trebuie să îi transmitem ca parametri identificatorul imaginii şi valorile *RGB* ale culorii, şi care returnează un identificator al culorii. În exemplul anterior au fost alocate 2 culori, alb şi albastru.
- Pentru a desena în imagine, se pot utiliza funcții predefinite pentru ceea ce dorim să desenăm: linii, arce, poligoane sau text.
  Aceste funcții necesită, în general, următorii parametri:
  - Identificatorul imaginii
  - Coordonatele de început (uneori şi cele de sfârşit) ale figurii pe care o vom desena
  - Culoarea cu care vom desena
  - Informații referitoare la font, în cazul textelor.

Exemplul anterior a utilizat trei funcții pentru desenare. Mai întâi, a fost creat un fundal albastru pe care desenăm cu ajutorul lui *imagefill*(). Parametrii acestei funcții sunt identificatorul imaginii, coordonatele de început (colțul din stânga sus) şi culoarea cu care va fi umplut.

Apoi, este desenată o linie albă începând de la colțul din stânga sus până la cel din dreapta jos:

```
imageline($im, 0, 0, $width, $height, $white);
```

Adăugarea unei etichete acestui element grafic are loc cu ajutorul funcției:

```
imagestring($im, 4, 50, 150, 'Sales', $white);
```

Prototipul funcției *imagestring*() este următorul:

```
int imagestring (resource im, int font, int x, int y, string s,
int col)
```

Parametrii acesteia sunt identificatorul imaginii, fontul, coordonatele *x* şi *y* de unde începe scrisul, textul şi culoarea. Fontul este un număr între 1 şi 5. Aceste numere reprezintă o mulțime de fonturi în codificarea *latin2*, numerele mai mari reprezentând fonturi mai mari. O alternativă a utilizării acestor fonturi o constituie fonturile *TrueType* sau cele *PostScript Type* 1.

Fiecare dintre aceste mulțimi de fonturi are o mulțime corespunzătoare de funcții.

Afişarea imaginii finale

O imagine poate fi afişată fie direct în *browser*, fie într-un fişier. În exemplul anterior, imaginea este afişată în *browser*. Acest proces se defăşoară, de asemenea, în două etape.

Trebuie să comunicăm *browser*-ului dacă vom afişa o imagine, utilizând funcția *Header*() pentru a specifica tipul *MIME* al imaginii (*Multipurpose Internet Mail Extensions* – inițial conceput pentru mail, devenit apoi metoda standard pe web pentru identificarea tipului de fişier trimis unui browser) :

```
Header ('Content-type: image/png');
```

În mod normal, atunci când un fişier este regăsit în *browser*, tipul *MIME* este primul lucru pe care îl trimite serverul. Pentru o pagină HTML sau PHP, primul lucru trimis este:

```
Content-type: text/html
```

Aceasta comunică *browser*-ului cum să interpreteze datele care urmează. De data aceasta, dorim trimiterea unei imagini.

Funcția *Header* trimite şiruri de caractere reprezentând *header*-e *HTTP*. O altă aplicație tipică a acestei funcții este redirecționarea *HTTP*. Astfel, putem spune *browser*-ului să încarce o pagină diferită de cea solicitată (de obicei, atunci când pagina a fost mutată). De exemplu:

```
Header (' Location:
http://www.domain.com/new_home_page.html ');
```

Atunci când este utilizată funcția *Header*(), trebuie să avem în vedere că aceasta nu poate fi executată în cazul în care conținutul a fost deja trimis paginii. *PHP* va trimite automat un *header HTTP* atunci când este afişat ceva în *browser*. Prin urmare, dacă avem vreo instrucțiune *echo* sau chiar un spațiu liber înainte de *tag*-ul *PHP* de deschidere, *header*-ele vor fi trimise şi vom primi un avertisment din partea *PHP* că încercăm să apelăm *Header*(). Pot fi trimise mai multe *header*-e *HTTP* cu apeluri multiple la funcția *Header*() în cadrul aceluiaşi script, deşi toate acestea trebuie să apară înainte ca un *output* să fie trimis la *browser*.

După ce au fost trimise datele *header*, datele imagine vor fi afişate cu ajutorul unui apel la `imagepng ($im);`

Dacă dorim să afişăm date în format *JPEG*, vom utiliza funcția *imagejpeg*(), în condițiile în care este trimis *header*-ul corespunzător:

```
Header ('Content-type: image/jpeg');
```

Pentru a scrie imaginea într-un fişier, trebuie adăugat al doilea parametru (opțional) în apelul funcției: *imagepng($im, $filename)*.

Eliberarea resurselor

Atunci când se încheie lucrul cu o imagine, trebuie returnate serverului resursele care au fost utilizate, distrugând imaginea creată cu ajutorul funcției *imagedestroy($id_imagine)*.

Utilizarea imaginilor generate automat în alte pagini

Deoarece un *header* poate fi trimis o singură dată şi acesta este singurul mod de a transmite *browser*-ului că primeşte o imagine, încapsularea imaginilor în paginile obişnuite se poate realiza astfel:

- Întreaga pagină constă din *output*-ul imaginii, ca în exemplul anterior
- Putem scrie imaginea într-un fişier, şi ulterior să ne referim la el cu ajutorul *tag*-ului <*IMG*>
- Plasăm *script*-ul care produce imaginea într-un *tag* imagine.

Primele două metode au fost deja prezentate. Pentru a putea utiliza cea de-a treia metodă, includem imaginea *inline* în *HTML* prin intermediul unui tag imagine de forma următoare:

```
<img src="simplegraph.php" height="200" width="200" alt="Sales going down" />
```

În loc de a pune direct un *PNG*, *JPEG* sau *GIF*, punem scriptul care generează imaginea în *tag*-ul *src*. Acesta va fi regăsit şi *output*-ul său va fi adăugat *inline*: http://193.226.51.37/web/curs6/simplegraph.html

Utilizarea textelor şi a fonturilor pentru a crea imagini

Este foarte util să putem crea butoane sau alte imagini pentru un site. În continuare, vom genera butoane utilizând un *template* pentru un buton vid. Acesta permite să avem caracteristici care sunt uşor de obținut în utilitarele grafice (*Photoshop*, *GIMP* etc.). Biblioteca pentru imagini din *PHP* permite să pornim cu o imagine de bază, pe care să desenăm elemente noi. De asemenea, vom utiliza fonturi *TrueType*.

Vom avea ca intrare un text şi vom genera butoane conținând textul respectiv, centrat atât orizontal cât şi vertical, de dimensiunea celui mai mare font care încape în buton.

Considerăm o interfață pentru testarea şi experimentarea generatorului de butoane: http://193.226.51.37/web/curs6/design_button.html.

Rezultatul este un buton, care este generat de script-ul make_button.php: http://193.226.51.37/web/curs6/make_button_php.txt

Script-ul începe cu o verificare de bază a erorilor şi stabileşte *canvas*-ul pe care vom lucra. Pornim cu o imagine existentă pentru buton, furnizând opțiuni pentru diferite culori (*red-button.png*, *green-button.png*, *blue-button.png*).

Apelul la *imagecreatefrompng*() creează imaginea doar în memorie. Pentru a salva imaginea într-un fişier sau a o afişa într-un *browser*, trebuie apelată funcția *imagepng*().

Încadrarea textului în buton

Am reținut textul introdus de către utilizator în variabila $*button_text*. Vom dori să tipărim textul respectiv pe buton, cu cel mai mare font care se încadrează. Vom realiza acest lucru prin încercări şi erori repetate.

Dimensiunile imaginii sunt stabilite în variabilele $*width_image*, $*height_image*. Mai departe, stabilim marginea având în vedere că marginile pe fiecare parte au aproximativ 18 pixeli:

```
$width_image_wo_margins = $width_image - (2 * 18);
$height_image_wo_margins = $height_image - (2 * 18);
```

Stabilim dimensiunea inițială a fontului (32), cea mai mare care ar putea încăpea într-un buton. În *GD2* trebuie să configurăm locația în care se află fonturile stabilind valoarea variabilei de mediu *GDFONTPATH*:

```
putenv('GDFONTPATH=C:\WINDOWS\Fonts');
```

Fontul pe care vom dori să îl folosim va fi căutat în locația precedentă, cu extensia *.ttf* (*truetype font*).

Iterăm, scăzând dimensiunea fontului la fiecare pas, până când textul se încadrează în buton:

```
do
{
  $font_size--;
  // find out the size of the text at that font size
  $bbox=imagettfbbox ($font_size, 0, $fontname, $button_text);
```

```
  $right_text = $bbox[2]; // right co-ordinate
  $left_text = $bbox[0]; // left co-ordinate
  $width_text = $right_text - $left_text; // how wide is it?
  $height_text = abs($bbox[7] - $bbox[1]); // how tall is it?
}
while ( $font_size>8 &&
       ( $height_text>$height_image_wo_margins ||
         $width_text>$width_image_wo_margins )
       );
```

În fragmentul de cod anterior, testăm dimensiunea textului în funcție de *bounding box* (cel mai mic obiect *box* care poate fi trasat în jurul textului) cu ajutorul funcției *imagegetttfbbox*() (al doilea parametru reprezintă unghiul cu care este înclinat textul, ceilalți parametri sunt evidenți). După ce deducem dimensiunea, putem tipări textul utilizând fontul respectiv şi funcția *imagettftext*().

Funcția *imagegetttfbbox*() returnează un vector ce conține coordonatele colțurilor *bounding box*-ului, în următoarea ordine: coordonata *x* a colțului din stânga jos, *y* a colțului din stânga jos, *x* a colțului din dreapta jos, *y* a colțului din dreapta jos, *x* a colțului din dreapta sus, *y* a colțului din dreapta sus, *x* a colțului din stânga sus, *y* a colțului din stânga sus. Aceste coordonate sunt relative la linia de bază.

Poziționarea şi scrierea textului

În exemplul anterior, stabilim poziția de bază a textului la mijlocul spațiului disponibil, în variabilele $*text_x* şi $*text_y*. Din cauza complicațiilor care apar din cauza sistemului de coordonate relativ la linia de bază, trebuie adăugați câțiva factori de corecție:

```
if ($left_text < 0)
  $text_x += abs($left_text); // add factor for left overhang
  $above_line_text = abs($bbox[7]); // how far above the
                                    // baseline?
$text_y += $above_line_text; // add baseline factor
$text_y -= 2; // adjustment factor for shape of our template
```

Stabilim culoarea textului:

```
$white = ImageColorAllocate ($im, 255, 255, 255);
```

Scrierea textului pe buton se realizează cu ajutorul funcției *imagettftext*():

```
imagettftext ($im, $font_size, 0, $text_x, $text_y, $white,
$fontname, $button_text);
```

Parametrii acestei funcții sunt: identificatorul imaginii, dimensiunea fontului exprimată în puncte, unghiul la care este scris textul, coordonatele *x* şi *y* de început ale textului, culoarea textului, fişierul pentru font şi textul.

Ulterior, butonul va putea apărea în browser:

```
Header ('Content-type: image/png');
imagepng ($im);
```

La final, eliberăm resursele:

```
imagedestroy ($im);
```

Tehnici *PHP* avansate

Desenarea figurilor şi a graficelor

Presupunem că dorim un *poll* pe site-ul web pentru a testa pe cine ar vota alegătorii la nişte alegeri fictive. Rezultatele acestui *poll* vor fi stocate într-o bază de date MySQL şi vom afişa o histogramă utilizând funcții care operează asupra imaginilor.

Desenarea graficelor este un alt obiectiv al acesor funcții. Putem construi orice grafice dorim: pentru vânzări, vizitări ale site-ului etc.

Baza de date va conține un tabel denumit *poll_results*, în care se vor afla numele candidaților (coloana *candidate*) şi numărul de voturi pe care le-au primit (coloana *num_votes*). Presupuneam că numele acestei baze de date, al utilizatorului şi parola acestuia, au toate valoarea *poll*. Script-ul pentru crearea bazei de date este: http://193.226.51.37/web/curs7/pollsetup.sql

Crearea bazei de date necesare votului se poate realiza astfel:

```
mysql -u root -p < pollsetup.sql
```

Interfața pentru votare este furnizată de pagina http://193.226.51.37/web/curs7/vote.html. Atunci când un utilizator dă click pe buton, votul său va fi adăugat în baza de date, se vor returna toate voturile din baza de date şi se va afişa histograma rezultatelor curente. Aceasta este creată cu ajutorul unor linii, dreptunghiuri şi item-uri de tip text plasate pe un canvas.

Script-ul care generează această imagine este lung şi îl vom diviza în 4 părți.

Prima parte actualizează baza de date şi regăseşte noile rezultate: http://193.226.51.37/web/curs7/show_poll_php.txt (până la linia *$result -> data_seek(0);*).

După obținerea acestor informații, se pot efectua calculele pentru trasarea graficului. Acest lucru este realizat în a doua parte a script-ului, în

care se stabilesc valorile variabilelor utilizate în trasarea graficului. Valorile sunt stabilite conform modului în care dorim să arate imaginea finală, estimând proporţiile dorite.

Variabila $*width* reprezintă lăţimea totală a *canvas*-ului pe care vom lucra. De asemenea, sunt stabilite valori pentru marginea din stânga şi, respectiv, cea din dreapta ($*left_margin*, $*right_margin*), înălţimea barelor şi distanţele dintre acestea ($*bar_height*, $*bar_spacing*), precum şi fontul, dimensiunea acestuia şi poziţia etichetelor ($*font*, $*title_size*, $*main_size*, $*small_size*, $*text_indent*).

Se mai calculează distanţa din histogramă care reprezintă o unitate ($*bar_unit*) şi înălţimea totală necesară pe canvas ($*height*).

Partea a treia a script-ului creează graficul, pregătit pentru a-i fi adăugate date. Este stabilită imaginea de bază, sunt alocate culori şi apoi este desenat graficul.

Fundalul graficului este umplut cu ajutorul funcţiei:

```
imagefilledrectangle($im,0,0,$width,$height,$bg_color);
```

Această funcţie acceptă ca prim parametru identificatorul imaginii, apoi valorile *x*, *y* ale colţului din stânga sus şi ale celui din dreapta jos. Ultimul parametru furnizează culoarea cu care va fi umplut dreptunghiul. În cazul script-ului nostru, întregul *canvas* este umplut cu culoarea albă.

Pentru a avea un dreptunghi negru trasat împrejurul *canvas*-ului, se poate utiliza funcţia:

```
imagerectangle($im,0,0,$width-1,$height-1,$line_color);
```

Pentru centrarea şi scrierea titlului graficului, sunt folosite aceleaşi funcţii precum în script-ul anterior.

Linia de referinţă a histogramei va fi desenată la final:

```
imageline($im, $x, $y-5, $x, $height-15, $line_color);
```

Această funcţie trasează o linie pe imaginea specificată prin identificatorul său ($*im*) între coordonatele *x*, *y* specificate, cu culoarea $*line_color*.

În partea a patra vom completa graficul conform datelor din baza de date. Candidaţii din baza de date sunt regăsiţi unul câte unul, este determinat procentul de voturi şi sunt desenate barele din histogramă şi etichetele pentru fiecare candidat.

Adăugarea etichetelor se realizează cu *imagettftext*(), iar barele sunt desenate ca dreptunghiuri pline, cu ajutorul funcţiei:

```
imagefilledrectangle($im, $x, $y-2, $bar_length, $y+$bar_height,
$bar_color);
```

Apoi, sunt desenate dreptunghiurile goale pentru a reprezenta 100%:

```
imagerectangle($im, $bar_length+1, $y-2, ($x+(100*$bar_unit)),
$y+$bar_height, $line_color);
```

După desenarea tuturor acestor elemente, se afişează imaginea utilizând funcția *imagepng*() şi se eliberează resursele prin *imagedestroy*().

Controlul sesiunilor în *PHP*

HTTP este un protocol *stateless*, ceea ce înseamnă că nu are un mod predefinit de menținere a stării între două tranzacții. Atunci când utilizatorul solicită o pagină, urmată de o alta, *HTTP* nu furnizează un mod de a comunica faptul că ambele cereri provin de la acelaşi utilizator.

Ideea de control al sesiunilor este legată de posibilitatea de a urmări un utilizator pe parcursul unei sesiuni pe un site *web*. Dacă acest lucru poate fi realizat, se pot furniza uşor funcționalitățile de conectare a utilizatorului şi afişare a conținutului site-ului conform nivelului de autorizare al acestuia sau preferințelor personale. Se poate urmări comportamentul utilizatorului respectiv şi se pot implementa anumite lucruri precum coşurile de cumpărături.

Începând cu versiunea 4, PHP include funcții native pentru controlul sesiunilor. Abordarea controlului sesiunilor s-a schimbat puțin atunci când au fost introduse variabilele superglobale ($_*SESSION*).

Funcționalitatea de bază a unei sesiuni

Sesiunilor din PHP le corespund câte un identificator unic, un număr aleator criptat. Acest identificator este generat de către PHP şi stocat pe partea client, pe durata de viață a sesiunii. Poate fi păstrat fie pe calculatorul utilizatorului într-un *cookie*, fie transmis prin URL-uri.

Identificatorul sesiunii are rol de cheie care permite înregistrarea variabilelor particulare ca variabile ale sesiunii. Conținutul acestor variabile este stocat pe server. Identificatorul sesiunii constituie singura informație

vizibilă pe partea de client. Dacă, la momentul unei anumite conexiuni la site-ul web, identificatorul sesiunii este vizibil prin *cookie* sau URL-uri, pot fi accesate variabilele sesiunii respective, de pe server. Acestea sunt stocate, implicit, în fişiere simpe (text), lucru care poate fi modificat astfel încât să fie utilizată o bază de date pentru stocarea lor.

Cookie-uri

Un *cookie* este o informaţie pe care script-ul o poate stoca pe maşina client. Acesta poate fi creat (cu numele *NAME* şi valoarea *VALUE*) prin trimiterea unui header HTTP ce conţine datele în formatul următor:

```
Set-Cookie: NAME=VALUE; [expires=DATE;] [path=PATH;]
[domain=DOMAIN_NAME;] [secure]
```

Cu excepţia lui *NAME*, ceilalţi parametri sunt opţionali. Parametrul *expires* stabileşte o dată după care respectivul *cookie* nu va mai fi relevant. Dacă nu specificăm o astfel de dată, el va exista până atunci când va fi şters manual. Parametrii *path* şi *domain* pot fi utilizaţi pentru a specifica URL-urile pentru care este relevant respectivul *cookie*. Cuvântul cheie *secure* determină ca acel *cookie* să nu fie trimis prin conexiuni HTTP obişnuite.

Atunci când un *browser* se conectează la Internet, el va căuta mai întâi *cookie*-urile stocate local. Dacă vreunul dintre acestea este relevant URL-ului la care se conectează, acestea vor fi transmise server-ului.

Crearea *cookie*-urilor din PHP

În PHP, *cookie*-urile pot fi create manual cu ajutorul funcţiei setcookie(). Aceasta are următorul prototip:

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

Parametrii acestei funcţii corespund exact celor din *header*-ul *Set-Cookie* prezentat anterior.

Dacă un cookie este creat astfel:

```
setcookie ('mycookie', 'value');
```

el va putea fi accesat prin intermediul lui $_*COOKIE*[*'mycookie'*] atunci când utilizatorul vizitează următoarea pagină din site sau reîncarcă pagina curentă.

Un *cookie* poate fi şters tot cu ajutorul funcţiei *setcookie*(), cu acelaşi nume al *cookie*-ului şi cu un timp de expirare din trecut. De asemenea, un *cookie* poate fi creat manual cu ajutorul funcţiei *header*() şi cu sintaxa pentru *cookie* dată anterior. De remarcat că *header*-ele pentru *cookie*-uri trebuie trimise înaintea celorlalte *header*-e, altfel nu vor funcţiona.

Utilizarea *cookie*-urilor în cadrul sesiunilor

Cookie-urile au anumite probleme: unele *browser*-e nu le acceptă sau au fost dezactivate de către utilizatori. Acesta este unul dintre motivele pentru care sesiunile PHP utilizează o metodă dual*ă cookie*/URL.

Atunci când sunt utilizate sesiuni PHP, *cookie*-urile nu trebuie create manual. Funcțiile care operează asupra sesiunilor realizează acest lucru.

Funcția *session_get_cookie_params*() permite obținerea conținutului *cookie*-ului stabilit de către controlul sesiunii. Aceasta returnează un vector ce conține elementele *lifetime*, *path*, *domain* şi *secure*.

De asemenea, mai putem folosi funcția:

```
session_set_cookie_params($lifetime, $path, $domain [, $secure]);
```

Salvarea identificatorului sesiunii

PHP utilizează implicit *cookie*-urile în cadrul sesiunilor. Dacă acest lucru este posibil, va fi creat un *cookie* pentru a salva identificatorul sesiunii.

O altă metodă pe care o poate folosi este adăugarea ID-ului sesiunii în URL. Putem determina ca acest lucru să aibă loc automat dacă activăm directiva *session.use_trans_sid* în fişierul *php.ini*. Implicit, aceasta este dezactivată. Activarea ei trebuie făcută cu atenție, deoarece creşte riscurile de securitate ale site-ului. Dacă este activată, un utilizator poate trimite prin email URL-ul care conține ID-ul sesiunii către o altă persoană, acesta poate fi stocat pe un calculator public, sau poate figura în istoricul *bookmark*-urilor unui *browser*.

Altfel, ID-ul sesiunii poate fi încapsulat manual în *link*-uri astfel încât acesta să fie transmis. ID-ul sesiuni este stocat în constanta SID. Pentru a putea fi transmisă, este adăugată la sfârşitul unui *link*, în mod similar unui parametru *GET*:

```
<A HREF="link.php?<?php echo strip_tags(SID); ?>">
```

Funcția *strip_tags*() este utilizată pentru a evita atacurile *cross-scripting*.

Implementarea sesiunilor simple

Paşii de bază pentru utilizarea sesiunilor sunt următorii:

1. Pornirea sesiunii
2. Înregistrarea variabilelor sesiunii
3. Utilizarea variabilelor sesiunii

4. ştergerea variabilelor şi suprimarea sesiunii

Nu este necesar ca aceşti paşi să aibă loc toți în acelaşi script.

Pornirea unei sesiuni

Există două metode pentru a porni o sesiune. Primul şi cel mai simplu este de a începe script-ul cu un apel la funcția *session_start*(). Aceasta verifică dacă există deja o sesiune curentă. Dacă nu există, va crea una, furnizând acces la vectorul superglobal $_*SESSION*. Dacă există deja o sesiune, funcția încarcă variabilele înregistrate ale sesiunii astfel încât acestea să poată fi utilizate.

Este important să apelăm această funcție la începutul tuturor script-urilor care folosesc sesiuni. Dacă nu apelăm funcția, nimic din ceea ce este stocat în sesiune nu va fi disponibil în acel script.

Al doilea mod în care putem începe o sesiune este de a determina PHP să pornească automat o sesiune atunci când cineva intră pe site. Aceasta se realizează cu ajutorul opțiunii *session.auto_start* din fişierul *php.ini*. Această metodă are un mare dezavantaj: nu se pot folosi obiecte ca variabile ale sesiunii. Acest lucru are loc deoarece definiția clasei pentru acel obiect trebuie încărcată înainte de începerea sesiunii pentru a putea fi create obiecte.

Înregistrarea variabilelor sesiunii

Variabilele sesunii se stochează în vectorul superglobal $_*SESSION*. Pentru a crea o variabilă a sesiunii, trebuie creat un element în acest vector:

```
$_SESSION['myvar'] = 5;
```

Variabila creată va fi urmărită până la sfârşitul sesiunii sau până când este suprimată manual. De asemenea, sesiunea poate expira pe baza valorii parametrului *session.gc_maxlifetime* din fişierul *php.ini* (durata de timp, în secunde, până când sesiunea este încheiată de către *garbage collector*).

Utilizarea variabilelor sesiunii

Atunci când un obiect este utilizat ca variabilă a sesiunii, definiția clasei corespunzătoare trebuie să preceadă apelul la *session_start*().

Pe de altă parte, trebuie să avem grijă atunci când verificăm dacă variabilele sesiunii au fost create (*isset*(), *empty*()). Variabilele pot fi stabilite de către utilizator prin GET sau POST. Putem verifica dacă o variabilă este înregistrată în sesiune căutând-o în $_*SESSION*.

Verificarea se pate face astfel:

```
if (isset($_SESSION['myvar'])) ...
```

Ştergerea variabilelor şi suprimarea sesiunii

Atunci când o variabilă nu mai este necesară, ea poate fi suprimată. Putem suprima elementul corespunzător din tabloul $_*SESSION*, astfel:

```
unset($_SESSION[‘myvar’]);
```

Nu se recomandă suprimarea conținutului întregului vector $_*SESSION*, deoarece acest lucru ar dezactiva sesiunile. Pentru a suprima toate variabilele sesiunii, putem proceda astfel:

```
$_SESSION = array();
```

După suprimarea tuturor variabilelor, putem încheia sesiunea apelând:

```
session_destroy();
```

Exemplu::

Vom implementa trei pagini. În prima pagină, pornim o sesiune şi creăm variabila $_*SESSION*['*sess_var*']:
http://193.226.51.37/web/curs7/page1_php.txt

Script-ul variabila şi îi stabileşte o valoare. Rezultatul este http://193.226.51.37/web/curs7/page1.php

Valoarea finală a variabilei în pagină este cea vare va fi disponibilă în paginile următoare. La sfârşitul sesiunii, variabila este serializată până când este reîncărcată prin intermediul următorului apel la *session_start*().

Prin urmare, putem începe următorul script apelând *session_start*(): http://193.226.51.37/web/curs7/page2_php.txt

După apelul la *session_start*(), variabila $_*SESSION*['*sess_var*'] este disponibilă, având valoarea stocată anterior: http://193.226.51.37/web/curs7/page2.php.

După utilizarea variabilei, o suprimăm. Sesiunea încă există, dar variabila nu.

În cea de-a treia pagină avem:
http://193.226.51.37/web/curs7/page3_php.txt

Rezultatul este: http://193.226.51.37/web/curs7/page3.php

Se observă că nu mai avem acces la valoarea persistentă a lui $_*SESSION*['*sess_var*'].

Pentru a elibera identificatorul sesiunii, script-ul se încheie cu apelul *session_destroy*().

Implementarea autentificării în controlul sesiunii

Cea mai frecventă utilizare a controlului sesiunilor este legată de urmărirea acţiunilor utilizatorului după ce aceştia se autentifică. În următorul exemplu, această funcţionalitate este asigurată prin autentificarea cu ajutorul unei baze de date MySQL şi utilizarea sesiunilor.

Baza de date MySQL pentru autentificare este: http://193.226.51.37/web/curs7/createauthdb.sql

Exemplul conţine trei script-uri. Primul, *authmain.php*, furnizează un formular pentru conectare pentru membri. Al doilea, *members_only.php*, afişează informaţii doar membrilor care s-au conectat cu succes. Al treilea, *logout.php*, deconectează un utilizator.

Prima pagină are următorul cod: http://193.226.51.37/web/curs7/authmain_php.txt.

Rezultatul ei este http://193.226.51.37/web/curs7/authmain.php

Dacă nu furnizăm un *username* şi o parolă valide, nu obţinem acces la pagina membrilor. Dacă ne conectăm cu *testuser*/*password*, vom putea accesa această pagină.

Script-ul *authmain.php* afişează formularul pentru *login*, constituie acţiunea corespunzătoare acelui formular şi conţine codul HTML pentru încercarea de conectare reuşită, respectiv nereuşită.

Dacă un utilizator se conectează cu succes, vom înregistra o variabilă la nivel de sesiune, $_*SESSION*[*'valid_user'*], care va conţine identificatorul utilizatorului.

Primul lucru care trebuie realizat în script este apelul *session_start*(), care va încărca variabila *valid_user*, dacă aceasta a fost creată.

Pagina membrilor are următorul cod: http://193.226.51.37/web/curs7/members_only_php.txt.

Acest script porneşte o sesiune şi verifică dacă sesiunea curentă conţine un utilizator înregistrat (dacă variabila $_*SESSION*[*'valid_user'*] a fost creată). Dacă un utilizator este conectat, i se va afişa conţinutul dedicat membrilor, altfel va primi un mesaj corespunzător.

Scriptul *logout.php* are următorul conţinut: http://193.226.51.37/web/curs7/logout_php.txt.

Acest script porneşte mai întâi o sesiune, stochează numele fostului utilizator, şterge variabila *valid_user* şi suprimă sesiunea.