

1 HTML

1.1. Introducere

HTML (*HyperText Markup Language*) este un limbaj de marcare folosit de browser-ele Web pentru a determina modul de afișare a conținutului paginilor Web. [1][2] Elemente de bază ale HTML sunt o serie de câmpuri sau marcaje speciale, numite și tag-uri sau etichete HTML. O altă caracteristică esențială a HTML o constituie posibilitatea realizării de legături (link-uri sau hyperlink-uri) spre alte documente/pagini Web, așa cum rezultă și din denumirea lui (Hypertext – element creând o legătură spre un alt document). Protocolul destinat comunicației în cadrul WWW (*Word Wide Web* sau pe scurt Web) este HTTP (Hypertext Transfer Protocol), asigurând transferul de informație între un server Web și un client browser Web. [3] Etichetele HTML se folosesc ori de câte ori se dorește o formatare a modului de afișare a informației publicate (scrierea unui text cu bold, alegerea unui anumit fundal al paginii, inserarea unei imagini, afișare tabelată, integrare de sunet sau orice altceva legat de formatarea unei pagini Web). Tag-urile sunt coduri speciale care însotesc conținutul propriu-zis al paginii Web, stabilind modul în care acesta este afișat de către clientul apelant. Codul HTML (împreună cu informația propriu-zisă), stocat la distanță pe un server Web, este transferat spre un browser Web (folosind HTTP), fiind interpretat de către acesta. Multitudinea de browsere Web folosite face ca modul de interpretare al unor etichete HTML să difere uneori de la un caz la altul (din acest motiv, testarea unei aplicații Web trebuie realizată de multe ori pe o gamă largă de browsere).

Pentru o primă exemplificare, se consideră eticheta HTML `` reprezentând tag-ul pentru o afișare *bold* a unui text. În cazul în care acest tag pereche încadrează un text (vezi exemplul următor), afișarea textului se va realiza cu *bold*:

`Acest text este bold!`

Trebuie remarcată existența și a unui tag complementar `` atașat la sfârșitul conținutului textual. Acest tag de sfârșit al marajului închide secțiunea de *bold* și comunică browser-ului ca formarea impusă se încheie în acest punct. Cele mai multe tag-uri au un tag complementar de închidere a secțiunii delimitate, după cum se va vedea în continuare (astfel încât se poate vorbi, de regulă, despre aceste etichete ca despre niște perechi de delimitatori ai informației de afișat). Etichetele HTML nu sunt *case-sensitive*.

HTML este independent de platformă (sistem de operare), dar trebuie avută în vedere situația în care se apelează resurse ale sistemului cu particularități diferite

de la unul la altul (spre exemplu, problema *case-sensitive* la referirea numelui de fișiere pe Linux). Un document în format HTML constă, în esență, într-o mulțime de tag-uri împreună cu informația utilă. Învățarea sintaxei acestor tag-uri este sinonimă cu învățarea HTML, ca limbaj de descriere a documentelor. Totuși, fără apariția unor limbaje de programare ca Java, JavaScript, PHP, Perl, limbajul de marcare HTML ar fi asigurat doar o simplă prezentare prozaică pe ecran a unor texte și grafice, totul într-un format static, singura calitate dinamică constituind-o posibilitatea de navigare de la o pagină la alta.

HTML, ca limbaj de marcare pentru *hypertexte*, ajuns momentan la versiunea 5, prezintă o evoluție continuă. HTML 1.0 a introdus concepțile de antet, paragraf, liste, referințe, HTML 2.0 a adăugat posibilitatea de inserare a imaginilor, HTML 3.0 vine cu tabele, text în jurul figurilor frame-uri iar versiunile au evoluat succesiv, înglobând de obicei tot ceea ce există și aducând în plus elemente noi. Începând cu HTML 4 și continuând cu actuala versiune 5 (integrând semnificative facilități multimedia [7]), noi etichete sunt adăugate, specificațiile altora sunt modificate, tendința principală fiind de a propria aspectul și funcționalitățile unei aplicații Web de cele ale unei aplicații desktop. [22]

Evoluția HTML continuă și în momentul de față, integrând o serie de noi tehnologii și plugin-uri dedicate, extinzând substanțial capabilitățile multimedia ale site-urilor Web. Astfel, se vorbește despre DHTML (Dynamic HTML), reprezentând o combinație între HTML, CSS și JavaScript, AJAX - ca o integrare HTML, JavaScript și XML (eXtended Markup Language) și multe altele, toate acestea conducând la o dinamică sporită a paginilor Web. [4][5] Această evoluție este strâns legată și de versiunile succesive de dezvoltare a browserelor utilizate pe piață (Netscape, Mozilla, Microsoft Internet Explorer, Mosaic, Opera, Safari, Google Chrome etc), fiecare preluând uneori într-o manieră personalizată noile standarde HTML (după cum s-a precizat deja, putându-se vorbi de o dependență de browser, adică același document HTML este interpretat în mod diferit de browsere diferite).

1.2. Etichete HTML

1.2.1. Etichete primare

În continuare sunt prezentate câteva dintre cele mai importante și uzuale folosite tag-uri (etichete) care permit crearea unei pagini Web.[1]

Sintaxa completă (dar nu obligatoriu necesară) la crearea unei simple pagini Web conținând doar cod HTML este următoarea:

```
<html>
<head>
<title>Prima pagina HTML</title>
</head>
<body>
```

Continut prima pagina HTML

```
</body>
</html>
```

Pentru realizarea aplicației Web, trebuie deschis un editor de texte, introdus conținutul de mai sus (salvat într-un fișier cu extensia .html sau .htm), iar apoi se poate deschide (accesa) pagina astfel creată dintr-un browser. Va apărea o pagină cu fundal alb, având titlul "Prima pagina HTML" afișat pe bara de titlu și o linie de text simplu afișată în zona de afișare a browser-ului, conținând mesajul "Prima pagina HTML". Cu bold s-au scris tag-urile folosite în această pagină. Se observă structura și poziționarea acestora în cadrul documentului. O scurtă descriere a rolului acestor etichete în cadrul paginii este prezentată în tabelul următor:

Tabel 1.1

<code><html></html></code>	Specifică faptul că documentul este un document HTML. Toate documentele încep și se termină cu acest tag.
<code><head></head></code>	Creează o zonă de informație de tip meta, cum este cazul titlului documentului, cuvinte cheie sau informații care descriu pagina pentru motoarele de căutare.
<code><title></title></code>	Creează un "titlu" pentru acest document. Orice este scris în interiorul acestui tag, va fi afișat de către browser în bara de titlu.
<code><body></body></code>	Creează o zonă specială pentru conținutul documentului (text, imagini etc). Această secțiune este locul unde se vor insera toate elementele "afișabile" ale paginii.

Cel puțin 99% din documentele Web, fie ele mai mici sau mai mari, mai simple sau mai complicate, conțin cel puțin tag-urile prezentate mai sus. Ele descriu cadrul general al oricărui document HTML. Din definiția tag-ului `<body>` se poate vedea că cea mai mare parte din acțiunea unei pagini HTML se desfășoară în această secțiune, de vreme ce acest tag `<body>` cuprinde tot conținutul afișabil al documentului. Utilizarea tag-urilor HTML este *non-case-sensitive* (se pot folosi atât caracter mici, cât și mari).

Observație: Comentariile HTML sunt încadrate de separatorii `<! -- și -->`:

```
<! --Cod HTML comentat sau simplu comentariu -->
```

1.2.2. Setarea unui fundal al paginii

Una dintre primele operații pe care cei mai mulți programatori Web începători doresc să îl facă unei pagini de Web este adăugarea unui fundal (*background*), indiferent dacă e vorba de un background de tip culoare sau imagine. În mod implicit, un document fără background va fi afișat pe un fundal alb. Această

culoare se poate schimba foarte ușor. În acest sens trebuie adăugat următorul cod în interiorul zonei delimitate de tag-ul `<body>`:

```
<body bgcolor="#xxxxxx">
```

unde `xxxxxx` este codul hexa al culorii dorite, sau se poate folosi chiar numele culorii.

De exemplu, codul următor face ca documentul să fie afișat cu un background de culoare roșie:

```
<body bgcolor="#ff0000">
```

sau utilizând chiar numele culorii:

```
<body bgcolor="red">
```

Câteva dintre codurile hexa pentru cele mai uzuale culori sunt prezentate în tabelul următor:

Tabel 1.2

black	#000000
white	#FFFFFF
blue	#0000FF
yellow	#FFFF00
red	#FF0000
green	#008000
lime	#00FF00
silver	#C0C0C0

După ce s-a descris cum se adaugă un background de o anumită culoare, se prezintă modul cum se pot adăuga imagini în fundal.

Folosirea fișierului imagine "backgr15.jpg" pe post de background (fundal), este posibilă cu ajutorul următoarei sintaxe:

```
<body background="backgr15.jpg">
```

Mulți autori preferă să dea documentelor lor, atât o culoare de background cât și o imagine. În acest fel, chiar dacă imaginea nu a fost încă complet descarcată de pe serverul Web, navigatorii vor vedea între timp culoarea de background:

```
<body bgcolor="#000000" background="backgr15.jpg">
```

Observație: Pentru a evita apariția unor probleme datorate dependenței de sistemul de operare în referirea unor resurse ale acestuia, fișierului imagine referit în codul HTML trebuie să folosească tipul/combinația exactă de caractere - mari și/sau mici - utilizată în denumirea lui ca resursă fișier gestionată de platformă (problemă mai puțin relevantă pentru platformele Windows, dar extrem de relevantă pentru Unix, Linux).

1.2.3. Formatare text

Una dintre cele mai importante funcționalități a limbajului HTML, integrată chiar începând cu primele versiuni, o constituie posibilitatea formatării informației de tip text, toate documentele Web conținând cel puțin o astfel de informație primară, ca și conținut propriu-zis. În continuare, fără a avea pretenția de a epuiza întreg subiectul, sunt prezentate câteva dintre tag-urile uzuale destinate stabilirii unui format textual.

- Heading

Un *heading* este un text - de regulă pe post de titlu/subtitlu - scris cu caractere mai mari și îngroșate, scos în evidență față de restul conținutului paginii. Pentru crearea lor se poate folosi eticheta `<h?>`, unde în loc de ? se trece un număr de la 1 la 6 (1 corespunde celor mai mari caractere ale textului, iar 6 se scrie pentru a obține cele mai mici caractere pentru respectivul text). Spre exemplu:

`<h1>Hello!</h1>`

`<h2>Hello!</h2>`

`<h3>Hello!</h3>`

- Paragraful

Un paragraf poate fi creat în HTML prin utilizarea etichetei `<p>`. Această etichetă creează o zonă de text, care este separată de restul documentului printr-o linie vidă deasupra și dedesubtul zonei de text. De exemplu:

`<p>Acesta este primul paragraf. Acesta este primul paragraf. Acesta este primul paragraf.</p>`

`<p> Acesta este al doilea paragraf. Acesta este al doilea paragraf. Acesta este al doilea paragraf.</p>`

De asemenea, se poate controla și modul în care este aliniat textul în cadrul unui paragraf, folosind atributul *align*. Acest atribut acceptă trei valori: *left*, *center* și *right*. Spre exemplificare, pentru a alinia un text la marginea din dreapta a documentului se folosește sintaxa:

`<p align="left"> Acesta este un paragraf aliniat la marginea din stânga a documentului. Acesta este un paragraf aliniat la marginea din stânga a documentului. Acesta este un paragraf aliniat la marginea din stânga a`

documentului. Acesta este un paragraf aliniat la marginea din stânga a documentului. </p>

Tot pentru o structurare a conținutului unui document Web se poate folosi și eticheta `
` (*break row*), asigurând trecerea pe o linie nouă a informației precedate de aceasta etichetă (fără însă o spațiu și aliniază corespunzător conținutul textual, ca în cazul paragrafelor).

- Bold și italic

Caracterele de tipul bold sau italic se pot obține cu ajutorul etichetelor ``, respectiv `<i>`:

` Acesta este bold`
`<i>Acesta este italic</i>`

- Setare culoare, mărime și tip caractere

La fel ca în orice procesor de texte și în HTML se poate specifica culoarea, mărimea și tipul caracterelor ce formează textul. În acest sens se folosesc trei etichete specializate, după cum urmează:

` Acest text este rosu`
`Acest text este mai mare!`
`Acest text este scris cu caractere de tip courier`

Valorile valide pentru culorile fontului sunt identice cu valorile hexa care au fost prezentate în secțiunea referitoare la fundaluri. Mărimea scrisului se specifică folosind valori cuprinse între 1 și 7, unde 7 reprezintă cel mai mare font. Tipul caracterelor se specifică folosind chiar numele acestuia (Courier, Arial, etc.).

Se poate foarte ușor include într-o singură "comandă" mai multe etichete pentru a obține un efect dorit. De exemplu, dacă se dorește un text scris cu bold, font de mărimea 2, italic și de tip Arial, atunci se poate scrie:

`<i> Text</i>`

După cum se vede, HTML este foarte flexibil și permite combinarea mai multor etichete pentru a obține un anumit efect/formatare, în cazul în care acel efect nu se poate obține cu ajutorul uneia singure.

- Centrarea textului

Eticheta `<center>` permite centrarea rapidă a unui text într-o pagină. Un exemplu în acest sens:

`<center>Acest text este centrat</center>`
`<center><h3>Acest header este centrat de asemenea!</h3></center>`

- Crearea de bare orizontale

Textul poate fi foarte bine structurat în cadrul unui document și cu ajutorul unor bare orizontale (linie cu o anumită grosime și lungime) care să delimitizeze mai bine diferențele secțiunii ale acestuia. Aceste bare se obțin cu ajutorul etichetei `<hr>`.

```
<hr width=100% size=2>
```

Pentru a schimba dimensiunile acesteia se pot folosi atributele `width` (lungimea) și respectiv `size`.

Observație: În noua acceptație de design a site-urilor Web, utilizarea acestor etichete destinate unei formatări a informației este de multe ori substituită de tehnica CSS (*Cascading Style Sheets*), mult mai unitară și eficientă. Totuși, pentru formatări punctuale/individuale etichetele anterior referite își păstrează utilitatea.

1.2.4. Inserarea unei imagini

Inserarea imaginilor într-o pagină Web constituie următorul pas pentru o descriere informațională text-imagine. Imaginile pot fi inserate într-un document prin utilizarea etichetei ``. Codul următor inserează o imagine numită "poza.gif":

```
<IMG SRC="poza.gif">
```

De remarcat că etichetele `` nu trebuie închise cu ajutorul specificatorului `"/"` (nu există un tag `perche` de inchidere ``). Este unul dintre cazurile în care închiderea unei etichete în HTML nu este necesară. Eticheta `` dispune de un atribut `src`, necesar pentru a preciza imaginea care trebuie inserată (sursa, în acest caz *poza.gif*), respectiv dispune opțional de două atrbute pentru setarea dimensiunilor afișabile ale imaginii: `width` și `height`.

Un artificiu care face ca imaginile să fie încărcate mai repede este acela de a specifica explicit dimensiunile acestora, folosind atrbutele `width` și respectiv `height`. În acest mod, browser-ele nu mai trebuie să determine informații despre dimensiunile acestora și astfel să crească durata de încărcare (*download*). Pentru a specifica browser-ului acest lucru, se folosește o sintaxă de genul:

```

```

Atributele `width/height` pot face mult mai mult decât a mări viteza de *download* a imaginilor referite. Astfel, aceste atrbute se pot folosi la dimenziunarea dorită pe ecran a imaginilor încărcate. Spre exemplu, pentru a mări imaginea afișată cu eticheta anterioară, se poate scrie:

```

```

Observație: Încărcarea unei/unor imagini de mari dimensiuni poate constitui uneori sursa unei viteză de reacție scăzute din partea paginii Web (fiind vorba de un

transfer al unui volum mai mare de informație). Din acest motiv, dimensiunea acestora trebuie să fie rezonabilă, recomandându-se utilizarea fișierelor imagine în format comprimat (*jpg, png, tif*).

1.2.5. Hiperlegături

Tot la categoria caracteristici fundamentale ale HTML se încadrează și posibilitatea de a crea hyperlink-uri, asigurând conexiunile/legăturile spre alte pagini Web și implicit "navigarea" pe internet. Ce ar fi WWW fără aceste legături (sau link-uri, hyperlink-uri)? Legăturile reprezintă esența WWW, evoluția și expansiunea lui fiind asigurată tocmai prin legarea împreună a milioane și milioane de pagini Web (și nu numai) de pe tot globul.

O asemenea legătură (hyperlink) este creată cu ajutorul etichetei ancoră `<a>`. Aceasta necesită un atribut `href`, care specifică URL-ul (*Uniform Resource Locator*) întâi care trebuie urmat după ce se face click pe respectiva legătură.

Exemplul următor prezintă o legătură de tip text care va conduce ("naviga") spre site-ul *Yahoo*:

```
<a href="http://www.yahoo.com">Click here for Yahoo</a>
```

Legătura de mai sus este spre un document extern de pe WWW. Se pot de asemenea crea foarte ușor legături și spre documente locale, aflate chiar pe site-ul propriu. Trebuie doar furnizată calea completă și corectă a acestora relativ la punctul de unde se apelează respectiva legătură. Câteva exemple în acest sens:

```
<a href="section3.htm"> Click here for the next section</a>
```

```
<a href="subdir/section3.htm">Click here for the next section</a>
```

Prima legătură presupune că documentul *section3.htm* se află în același director ca și pagina curentă, iar al doilea presupune că *section3.htm* este în directorul "subdir", care este subdirector al directorului curent.

Se pot crea legături folosind și imagini. După ce s-a înțeles modul de creare a legăturilor text, crearea acestora folosind imagini este mai mult decât banală. Trebuie doar substituit textul cu eticheta ``. Spre exemplu, click pe o imagine pentru a asigura legătura (hyperlink-ul) cu site-ul *Yahoo*:

```
<a href="http://www.yahoo.com"></a>
```

Dacă se dorește ca imaginea respectivă să fie încadrată de o bordură:

```
<a href="http://www.yahoo.com"></a>
```

1.2.6. Formulare. Metodele GET și POST

Cele mai multe aplicații Web utilizează script-urile de regulă pentru prelucrarea informației introdusă prin intermediul unor fișiere HTML gen formular. Rolul script-ului (de regulă un CGI) este de a prelua aceste informații, a le prelucra și de a returna un răspuns (o pagină HTML afișabilă de către un browser client) cu informația procesată ca rezultat. Altfel spus, rolul unui script constă în returnarea (pe baza unei procesări de informație realizată pe un server Web) a unor date la ieșirea standard (browser). Aceste date sunt precedate (dacă este cazul) de o serie de antete care vor permite o interpretare corectă a ieșirii scriptului. Astfel de scripturi permit interacțiunea în ambele sensuri între client și server.

Antetul pe care scriptul îl trimite spre serverul Web trebuie să respecte specificațiile MIME (*Multipurpose Internet Mail Extension*). Spre exemplu, un astfel de antet poate să conțină o specificație de forma: Content-type: text/html (pentru documente HTML), Content-type: image/jpeg (pentru imagini JPEG) etc., care va precede informația direcționată spre ieșirea standard, precizând tipul de date care constituie flux de ieșire.

Cele mai multe apeluri ale unui script CGI se fac prin intermediul unui document HTML gen formular, la apăsarea unui buton (de regulă- SUBMIT).

Se consideră următorul formular exemplu (fig.1.1):

```
<form METHOD="GET" ACTION="exemplu.php">
Ani de munca: <input NAME="ani" size="2" ><p>
Salar anual: <input NAME="salar" size="8"><p>
<input TYPE="SUBMIT" VALUE="Run Query">
<input TYPE="RESET" VALUE="Sterge">
</form>
```

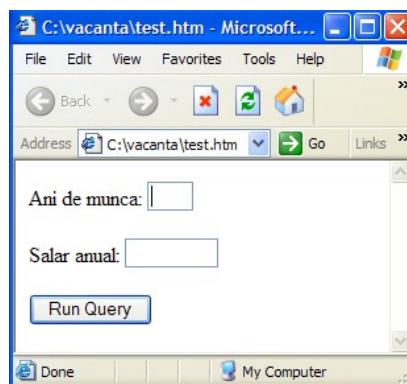


Fig. 1.1. Formular pentru transmiterea parametrilor spre script

Se observă că în cadrul acestui formular (FORM) se folosește metoda GET pentru transmiterea parametrilor spre scriptul indicat prin proprietatea ACTION (în cazul de față, un script PHP). Prin apăsarea butonului de tip SUBMIT "Run Query", presupunând că inițial s-a tastat 5, respectiv 1000, spre scriptul apelat este transmis un sir (string) de forma:

ani=5&salar=1000.

Acest sir de parametrii poate fi remarcat și în linia de apelare a scriptului, de altfel parametrii putând fi transmiși și direct spre script, printr-un apel (cu precizarea completă a căii spre scriptul PHP) cu parametri de forma:

http://localhost/scripts/exemplu.php?ani=5&salar=1000

De remarcat că, variabilele sunt delimitate prin caracterul &, iar caracterele speciale (\ sau ') sunt înlocuite de codul lor numeric (hexa), precedat de caracterul procent %. Spațiile se substituie prin semnul +. Atributul SIZE al unei etichete casetă INPUT setează dimensiunea vizibilă a acesteia pe ecran.

Utilizând metoda GET, acest string este disponibil într-o variabilă de mediu QUERY_STRING. Funcție de mediu folosit pentru crearea scriptului CGI, aceste date utilizabile ca parametrii de intrare pot fi preluate, dacă este cazul, din cadrul acestei variabile de mediu..

Același formular putea folosi ca metodă și metoda POST. În cazul metodei POST datele sunt disponibile, putând fi accesate de la intrarea standard (și nu prin intermediul unui string conținut în variabila de mediu QUERY_STRING, lungimea acestora (în octeți) fiind disponibilă în variabila de mediu CONTENT_LENGTH. Evident acest sir (văzut ca parametru de intrare cu GET) nici nu apare în linia de apelare a scriptului, aşa cum se întâmplă la metoda GET. Se recomandă utilizarea metodei POST în situația transmiterii unui volum mare de date (gen <TEXTAREA> sau a unui fișier – printr-o acțiune de upload). Un exemplu de utilizare a etichetei TEXTAREA:

```
<textarea rows="3" cols="30" name='nume'>
Ceva text pe primul rand.
Alt rand
</textarea>
```

Eticheta INPUT este cel mai des utilizată cu atributul type='text', valoare de altfel implicită. Deci se poate scrie echivalent:

```
<input NAME="ani" type='text' size="2" maxlength=2 value=20>
```

folosindu-se în acest exemplu și alte două atrbute deseori utile:

- maxlength – pentru a se limita numărul de caractere care pot fi introduse de la tastatură (size limitează doar numărul de caractere afișabile, putându-se introduce oricâte, dar zona afișabilă a casetei permite doar două caractere vizibile – ultimele două).

- value – pentru a seta o valoare inițială implicită (20 în cazul de față).

De asemenea, tipurile SUBMIT, respectiv RESET sunt strict legate de formularul FORM, transformând fiecare casetă aferentă într-un buton și activând transferul datelor primă metoda setată, respectiv resetând (ștergând) conținutul casetelor din formular.

Atributul TYPE permite o listă de valori specifice, enumerându-se aici doar câteva dintre cele mai uzuale:

- button – folosit în special pentru activarea unui script/secvențe JavaScript la apariția unui eveniment (apăsare buton în exemplul următor):

```
<input TYPE="button" VALUE="Go" onclick="mesaj_js()">
```
- checkbox – folosit pentru limitarea opțiunilor selectate (predefinite și în număr limitat), permitând selecția uneia sau mai multora (selecție multiplă).
- radio – folosit pentru limitarea opțiunilor selectate, permitând selecția doar a uneia sigure.
- password – asigură ascunderea conținutului introdus în casetă (cazul parolelor)
- file – generează o casetă, respectiv un buton Browse, necesare operației *Upload* de fișiere.
- image – definește un buton SUBMIT de tip imagine, atributul VALUE fiind înlocuit cu SRC pentru referirea unei imagini, ca și în exemplul următor:

```
<input TYPE="IMAGE" src ="imagine.jpg">
```
- hidden – ascunde caseta, fiind util la transferul unor valori implicate din FORM spre o altă pagină, valori setate prin atributul VALUE.

Un exemplu de formular folosind etichete INPUT cu atributul TYPE setat pe valorile anterior referite are codul următor, iar efectul rulării lui este prezentat în figura 1.2.

```
<form METHOD="GET" ACTION="exemplu.php">
<input type="file"><p>
Utilizator: <input NAME="nume" type='text size="10"'><p>
Parola: <input NAME="parola" type="password"><p>
Informatie ascunsa: <input NAME="ascuns" type="hidden" value="ascuns"><p>

Autoturism <input type="checkbox" name="vehicol" value="Dacia" /> Autoturism Dacia
<input type="checkbox" name="vehicol" value="Ford" />
Autoturism Ford <p>
Sex:<p>

<input type="radio" name="sex" value="M"> Masculin<br>
<input type="radio" name="sex" value="F"> Feminin <br>

Descriere: <textarea rows="3" cols="30" name='date'>
Ceva text pe primul rand.
Alt rand
</textarea>
<input TYPE="IMAGE" src="imagine.jpg">
</form>
```

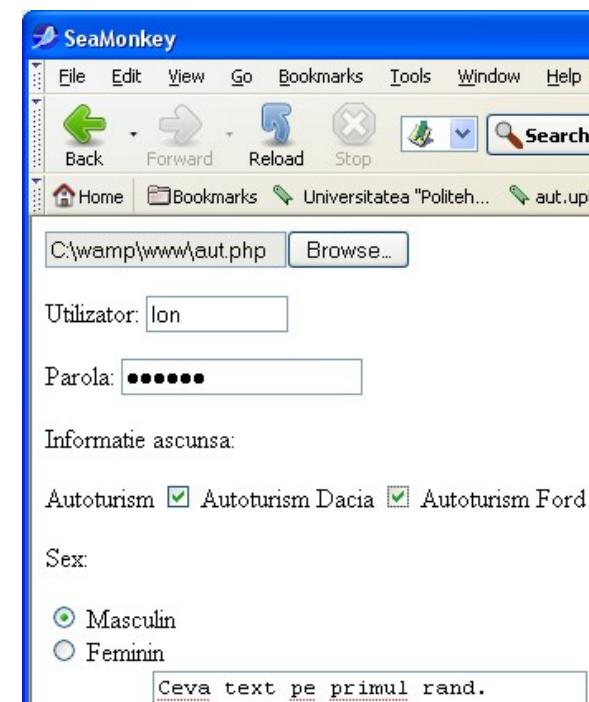


Fig. 1.2. Formular pentru transmisarea parametrilor spre script

Se poate remarcă atribuirea unor nume identice casetelor de tip checkbox, respectiv radio, în timp ce toate celelalte casete INPUT (inclusiv TEXTAREA) au nume distincte, acestea fiind practic parametri transmiși de formular spre o altă pagină Web. Dacă se folosește metoda GET, string-ul cu parametri are următoarea formă:

`exemplu.php?nume=Ion&parola=parola&ascuns=ascuns&vehicol=Dacia&vehicol=Ford&sex=M&date=Ceva+text+pe+primul+rand.%0D%0AAlt+rand%0D%0A&x=25&y=28`

Se remarcă în cadrul lui regulile menționate deja: delimitare parametri prin caracterul &, înlocuirea caracterelor speciale cu codul lor hexa precedat de caracterul procent %, înlocuirea spațiilor cu +.

Observatie: Eticheta FORM dispune de atributul TARGET care permite afișarea paginii apelate (spre care se trimit parametri) într-un alt frame (vezi paragraful destinat frame-urilor). Spre exemplu (pentru un frame cu numele frame1):

```
<form METHOD="GET" ACTION="exemplu.php" target="frame1">
```

O altă etichetă HTML care permite o selecție multiplă (nefăcând parte din categoria etichetelor de tip `INPUT`) este eticheta `SELECT` (practic echivalentul unui element de tip `checkbox` specific altor limbaje de programare vizuală).

Iată un exemplu de cod relevant privind utilizarea acestei etichete `SELECT` (cu sub-etichetele `OPTION`), efectul rulării lui fiind prezentat în fig. 1.3:

```
<form METHOD="GET" ACTION="apelat.php">
Persoane cu vîrstă:
<SELECT NAME="vîrstă" size="1">
<OPTION VALUE="0" > mai mare ca 0
<OPTION VALUE="10" > mai mare ca 10
<OPTION VALUE="20" > mai mare ca 20
<OPTION VALUE="30" > mai mare ca 30
</SELECT>

<SELECT MULTIPLE NAME="meserie[]">
<OPTION selected VALUE="Inginer" > Inginer
<OPTION VALUE="Sofer" > Sofer
<OPTION VALUE="Dentist" > Dentist
</SELECT>

Autoturism <input type="checkbox" name="vehicol[]" value="Dacia" /> Autoturism Dacia
<input type="checkbox" name="vehicol[]" value="Ford" />
Autoturism Ford <p>

<input TYPE="SUBMIT" VALUE="Caută">
</form>
```

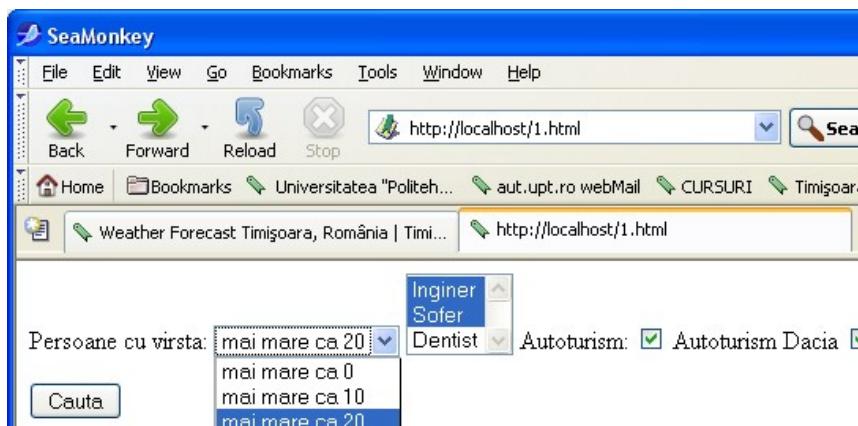


Fig.1.3 Selecții multiple

Primul tag `<SELECT...>` numit *vîrstă*, prin modul lui de definire permite selectarea (la un moment dat) doar a unei singure opțiuni, fiind practic o casetă cu derulare: `<SELECT NAME="vîrstă" size="1">`

Dacă parametrul `size` are valoarea 1, este vizibilă o singură opțiune la momentul inițial; dacă are o valoare n , la momentul inițial vor fi vizibile n opțiuni (indiferent însă de valoarea lui n , o singură opțiune poate fi selectată).

Al doilea tag `<SELECT...>` numit *meserie*, permite realizarea unor selecții multiple (vezi și figura 1.3):

Cazul unor etichete care permit o selecție multiplă ridică anumite probleme privind preluarea valorilor parametrilor transmiși de către acestea. Problema se datorează faptului că, un același parametru este pasat simultan cu mai multe valori diferite (selecții multiple), fiind întâlnită atât la etichetele `SELECT`, cât și la `INPUT` de tip `checkbox` (acesta este motivul pentru care s-a inserat în exemplul anterior și o secvență de cod specific unei etichete `INPUT checkbox`). Problema preluării acestor parametri cu valori multiple, în ambele situații, se rezolvă stabilind un nume al etichetei într-o sintaxă specifică unui *array* (șir): `meserie[]`, `vehicol[]`. Apoi scriptului apelat, spre care sunt pasăți parametri, îi revine sarcina de a prelua aceste variabile de tip *șir* (*array*) și de extrage, succesiv, valorile elementelor lor. În capitolul următor se va reveni asupra problemei, prezentându-se modul în care un script PHP tratează această situație specifică unor selecții multiple.

1.4. Tabele

1.4.1. Elemente introductive

O tabelă permite organizarea datelor pe linii (rânduri) și coloane, ca modalitate de prezentare a unei informații (text sau grafică) într-o pagină. Tabelele sunt extrem de utilizate ca mod de formatare a informației returnate prin interogarea unei baze de date.^{[1][4]}

Cele mai uzuale marcaje (cuvinte cheie, *tag-uri*, etichete) specifice proiectării unei pagini Web utilizând tabele (afișări tabelate) sunt: `<TABLE>`, `<TR>`, `<TD>`, `<TH>` (lista putând continua cu `<CAPTION>`, `<COL>`, `<COLGROUP>`, ``, etc)

Definiția oricărui tabel presupune utilizarea perechii de etichete `<TABLE>` `</TABLE>`. În definiția tablei se poate folosi (optional) o etichetă `<CAPTION>`, prin care se atașează un titlu tabelului.

După cum se va prezenta în continuare, tabelele sunt construite succesiv, linie cu linie (o linie la un moment dat, apoi altă linie, s.a.m.d.). Linile unei tabele sunt definite prin perechi de etichetele `<TR>` `</TR>`. În interiorul acestor etichete trebuie inclus cel puțin un *header* (cap de tabel) sau element de tip date (articol, informație) utilizând perechea de etichete `<TH>` `</TH>`, respectiv `<TD>` `</TD>` (etichete controlând definirea unei coloane sau mai precis un element coloană al

unei linii). De subliniat faptul că, construcția unei tabele începe tot timpul cu definiția unei linii (<TR>), iar în cadrul definițiilor liniilor sunt generate celulele aferente coloanelor (<TH> pentru celule cap tabel, <TD> pentru celulele date)

Exemplu:

```
<TABLE border=1>
  <CAPTION ALIGN="top">Clasament</CAPTION>
  <TR>
    <TH>Cool</TH>
    <TH>Sad</TH>
  </TR>
  <TR>
    <TD>Belle & Sebastian</TD>
    <TD>Michael Bolton</TD>
  </TR>
  <TR>
    <TD>Bentley Rhythm Ace</TD>
    <TD>Mariah Carey</TD>
  </TR>
</TABLE>
```

Efectul pe ecran este următorul:

Clasament

Cool	Sad
Belle & Sebastian	Michael Bolton
Bentley Rhythm Ace	Mariah Carey

Analizând codul, se poate remarcă modul în care s-a construit succesiv fiecare linie (capul de tabel, urmat de două linii cu informație), iar în cadrul fiecărei linii modul în care s-au realizat celulele (practic împărțirea liniei pe coloane).

1.4.2. Controlul global al unui tabel

Există numeroase metode pentru a controla designul unui tabel. În primul rând, se poate adăuga o bordură (margină) fiecărei celule a tabelului; această operație creând practic un caroaj pentru tabel. Se poate specifica poziția titlului tabelului (sus sau jos) și lățimea tabelului. Se poate, de asemenea, spația conținutului tabelului, adăuga spații între celulele tabelului sau crea margini (contururi) în jurul conținutului fiecărei celule. Toate aceste efecte pot fi realizate prin adăugarea uneia sau mai multora dintre atributele următoare, specifice etichetei de control global al designului unui tabel <TABLE>.

BORDER=value - includerea acestui atribut permite realizarea unei borduri în jurul tuturor celulelor tabelului. Lățimea bordurii se precizează în pixeli. Dacă atributul nu este utilizat, nu apare nici o bordură, totuși un spațiu liber va rămâne în jurul fiecărei celule ca și cum o bordură ar fi prezentă.

ALIGN=left/right - folosit pentru a afișa tabelul în stânga, dreapta sau centrat în fereastră. (pentru *Nestcape*, valabil începând cu versiunea 4, și continuând cu Mozilla).

CELLSPACING=value - permite specificarea dimensiunilor (în pixeli) a spațiilor inserate între celulele individuale ale tabelului.

CELLPADDING=value - permite specificarea dimensiunii (în pixeli) a spațiului inserat între marginea și conținutul unei celule din tabel.

WIDTH=value/percent - permite precizarea lățimii tabelului (în pixeli sau în % referitoare la lățimea documentului (paginii)).

Observație: Cel mai compactat tip de tabel trebuie să folosească atributele BORDER, CELLSPACING și CELLPADDING setate pe zero.

Spre exemplificare, definiția TABLE din exemplul precedent poate fi completată cu aceste atribută, astfel:

```
<TABLE BORDER=5 WIDTH="75%" CELLPADDING=6 CELLSPACING=2
ALIGN="LEFT">
```

1.4.3. Controlul unei celule

Pe lângă un control global, asupra tuturor elementelor (celulelor) unui tabel, se pot modifica și individual atributele fiecărei celule.

Modificările care pot fi făcute afecteză alinierea (pe orizontală sau verticală), lățimea și înălțimea celulelor, aranjarea textului ca informație. De asemenea se pot adăuga culori sau grafice (vezi paragrafele următoare).

Următoarele atrbute pot fi aplicate unei întregi linii, prin includerea lor între etichetele de tip <TR>, sau unei celule individuale prin includerea între etichetele <TD></TD>.

ALIGN=left/center/right - permite alinierea întregului text al unei linii sau celule.

VALIGN=top/middle/bottom/baseline vertical align - utilizat pentru alinierea întregului text al unei linii sau celule (sus, la mijloc, jos) și de asemenea pentru a specifica faptul că toate celulele încep (sunt aliniate) de la aceeași linie de bază.

Următoarele atrbute pot fi, de asemenea, aplicate oricărei celule prin includerea lor între etichete de tip <TD> sau <TH>.

WIDTH=value/percent - permite setarea lățimii celulei (în pixeli sau % din lățimea tabelului). Fixarea unei lățimi particulare pentru o celulă, conduce la fixarea acelei lățimi pentru tot tabelul.

HEIGHT=value/percent - permite setarea înălțimii celulei (în pixeli sau % din lățimea tabelului). Fixarea unei înălțimi particulare pentru o celulă, conduce la fixarea acelei înălțimi pentru toată linia.

NOWRAP - specifică faptul că liniile dintre celule nu vor fi întrerupte pentru a seta lățimea celulei.

```
<TABLE BORDER=1>
<TR>
    <TH>Cool</TH>
    <TH>Sad</TH>
</TR>
<TR VALIGN="BOTTOM" ALIGN="CENTER">
    <TD HEIGHT=35>Belle & Sebastian</TD>
    <TD NOWRAP>That Michael used to have such lovely hair, don't you think Gladys?</TD>
</TR>
<TR VALIGN="TOP">
    <TD WIDTH=200 ALIGN="RIGHT">Bentley Rhythm Ace</TD>
    <TD HEIGHT=35>Mariah Carey</TD>
</TR>
</TABLE>
```

1.4.4. Combinarea celulelor

Următorul pas pentru crearea unui tabel cu un aspect dorit, constă în unirea unor celule, atunci când aceasta se impune. Se pot uni celule astfel încât, o celulă să cuprindă două sau mai multe linii sau două sau mai multe coloane. Acest lucru se poate face utilizând atributele COLSPAN și ROWSPAN, astfel:

COLSPAN=value - utilizat pentru a specifica câte coloane sunt cuprinse într-o celulă.

ROWSPAN=value - utilizat pentru a specifica câte linii sunt cuprinse într-o celulă.

Exemplu (însoțit de vizualizarea tabelului creat):

```
<TABLE BORDER=1>
    <TR>
        <TH ROWSPAN=2></TH>
        <TH COLSPAN=2>Rating</TH>
    </TR>
    <TR>
        <TH>Sad</TH>
        <TH>Cool</TH>
    </TR>
    <TR>
        <TH ROWSPAN=2>Artist</TH>
        <TD>Belle & Sebastian</TD>
        <TD>Michael Bolton</TD>
    </TR>
```

```
<TR>
    <TD>Bentley Rhythm Ace</TD>
    <TD>Mariah Carey</TD>
</TR>
</TABLE>
```

Efectul în pagina Web este următorul:

	Rating	
	Sad	Cool
Artist	Belle & Sebastian	Michael Bolton
	Bentley Rhythm Ace	Mariah Carey

1.4.5. Formatarea coloanelor

Prin utilizarea etichetelor **<COL>** sau **<COLGROUP>** se poate specifica aranjarea textului în cadrul coloanelor unui tabel. În mod esențial, aceste atrbute permit formatarea unor celule în cadrul unei coloane, în același manieră în care **<TR>** permite formatarea unor celule în cadrul unei linii. Se utilizează atrbutul **<COLGROUP>** pentru a forma două sau mai multe coloane în același timp și **<COL>** pentru a forma o coloană individuală. Atributul **<COL>** trebuie întotdeauna inclus în cadrul unei perechi de etichete **<COLGROUP>**.

ALIGN=center/justify/left/right - alinierea textului (pe orizontală)

VALIGN=baseline/bottom/middle/top - aliniere pe verticală .

SPAN=number - setează numărul de coloane asupra căror acționează atrbutele **ALIGN** și **VALIGN**.

WIDTH=pixels - setează lățimea coloanei în pixeli.

Exemplu:

```
<TABLE BORDER=1 WIDTH=80%>
    <COLGROUP>
        <COL ALIGN="left">
        <COL ALIGN="right" VALIGN="bottom">
    </COLGROUP>
    <COLGROUP SPAN=2 ALIGN="center" VALIGN="top">
    </COLGROUP>
    <TR>
        <TH>Cool</TH>
        <TH>Groovy</TH>
        <TH>Unique</TH>
        <TH>Sad</TH>
    </TR>
    <TR>
```

```

<TD HEIGHT=35>Belle & Sebastian</TD>
<TD>The Wedding Present</TD>
<TD>Tricky</TD>
<TD>Michael Bolton</TD>
</TR>
<TR>
<TD HEIGHT=35>Bentley Rhythm Ace</TD>
<TD>Leftfield</TD>
<TD>Nine Inch Nails</TD>
<TD>Mariah Carey</TD>
</TR>
</TABLE>

```

Cool	Groovy	Unique	Sad
Belle & Sebastian	The Wedding Present	Tricky	Michael Bolton
Bentley Rhythm Ace	Leftfield	Nine Inch Nails	Mariah Carey

1.4.6. Fundal și margini tabelă

Culorile pot fi incluse într-un tabel prin utilizarea unor atribute adiționale la elementele `<table>`, `<tr>`, `<td>`, `<th>`.

Acstea atribute sunt: `bgcolor`, `bordercolor`, `bordercolorlight`, `bordercolordark`, controlând culorile fundalului sau a bordurilor tabelelor. Pentru ca o colorare a tabelului să afecteze și bordura, atributele `border` trebuie să fie prezente în cadrul unor etichete `<TABLE>`. Toate culorile trebuie exprimate (în mod convențional) fie ca triplete hexa RGB, fie prin "nume" ale culorilor.

Dacă atributele de culoare sunt plasate atașat unui element `<table>` ele se vor referi la întregul tabel. Dacă sunt atașate unor etichete `<tr>` (sau `<th>`, `<td>`), ele vor acționa la nivel de linie sau celulă individuală.

bgcolor="#hextriplet"/"colourname" - se specifică culoarea de fundal a celulei (celulelor).

bordercolor="#hextriplet"/"colourname" - specifică culoarea bordurii celulei (celulelor).

bordercolorlight="#hextriplet"/"colourname" - bordurile pot fi reprezentate tridimensional. Pentru a folosi acest efect, se utilizează o iluminare sau întunecare a bordurii. Se utilizează acest atribut pentru a seta culoarea de iluminare.

bordercolordark="#hextriplet"/"colourname" - se utilizează acest atribut pentru a seta culoarea de întunecare.

Exemplu:

```
<TABLE BORDER BORDERCOLOR="Black" BGCOLOR="Silver" WIDTH=50%>
```

```

<TR>
<TH>Cool</TH>
<TH>Sad</TH>
</TR>
<TR BORDERCOLOR="Red" BGCOLOR="Green">
<TD>Belle & Sebastian</TD>
<TD>Michael Bolton</TD>
</TR>
<TR BORDERCOLOR="Red" BGCOLOR="Green">
<TD BORDERCOLOR="Yellow">Bentley Rhythm Ace</TD>
<TD BGCOLOR="White">Mariah Carey</TD>
</TR>
<TR BORDERCOLOR="Orange" BORDERCOLORDARK="Blue">
<TD BORDERCOLORLIGHT="White">
<TD>The Wedding Present</TD>
<TD>Celene Dion</TD>
</TR>
</TABLE>

```

Efectul pe ecran al scriptului HTML anterior este următorul:

Cool	Sad
Belle & Sebastian	Michael Bolton
Bentley Rhythm Ace	Mariah Carey
The Wedding Present	Celene Dion

Adăugarea unei imagini ca background la un tabel (începând cu Microsoft Internet Explorer 3 și Netscape 4, apoi continuând cu familia Mozilla) se poate face utilizând atributul `background`, atașat unor etichete de genul `<table>` sau `<td>` (exact ca și pentru o etichetă `<body>`). Există câteva deosebiri în această problemă, referitor la modul de lucru cu Mozilla și Internet Explorer. Mozilla repetă graficul de background la început, pentru corpul fiecărei celule, în timp ce Internet Explorer acoperă cu acea imagine întreg tabelul, inclusiv bordurile. În plus, Mozilla nu permite un background individual pentru fiecare celulă, dacă s-a specificat deja un background global, pentru întreg tabelul. Exemplu:

```
<TABLE BACKGROUND="graphics/babies.gif" WIDTH=50%
BORDER=1>
<TR>
<TH BGCOLOR="orange">Cool</TH>
<TH BGCOLOR="orange">Sad</TH>
</TR>
<TR>
<TD>Belle & Sebastian</TD>
<TD>Michael Bolton</TD>
```

```

</TR>
<TR>
    <TD>Bentley Rhythm Ace</TD>
    <TD>Mariah Carey</TD>
</TR>
<TR>
    <TD>The Wedding Present</TD>
    <TD BACKGROUND="graphics/flower_t.jpg"> Celene
Dion</TD>
</TR>
</TABLE>



Observație: Pentru crearea unei celule vide se poate folosi o secvență de genul: <TD>&nbsp;</TD> (&nbsp; - jucând rolul unui spațiu).


```

1.5. Liste

Listele permit ordonarea după anumite criterii și afișarea informației pe ecran sub formă unor articole (*item-uri*) bine structurate.[1][2]

Există mai multe tipuri de liste care pot fi create, cele mai importante fiind listele ordonate și neordonate.

Iată câteva etichete specifice creării unor liste:

,,,<MENU>,<DIR>,<DL>,<DT>,<DD>,

O listă conține o succesiune de item-uri (articole/elemente). Tipul de listă determină cum sunt descrise (afișate) *item-urile*. Pentru cazuri mai complexe se pot crea și liste închinate, pe mai multe nivele.

1.5.1. Liste neordonate

O listă neordonată prevede fiecare element al listei cu un “bullet” (marcaj grafic). O astfel de listă este delimitată de o pereche de etichete .

Atributul prin care se stabilește tipul de marcaj al fiecărui articol (fig.1.7) este: **TYPE=**”disc/circle/square”

```

<UL>
<LI>Pulp</LI>
    <UL>
        <LI>Jarvis Cocker</LI>
        <LI>Candida Doyle</LI>
    </UL>
<LI>The Wedding Present</LI>
    <UL TYPE="square">
        <LI>David Gedge </LI>
        <LI>Simon Smith </LI>
    </UL>
<LI>Spacemen 3</LI>
    <UL>
        <LI>Sonic Boom</LI>
        <LI>Jason Spaceman</LI>
    </UL>
</UL>

```

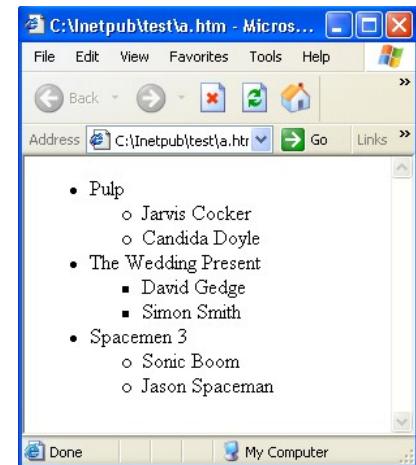


Fig. 1.7 Lista neordonată

1.5.2. Liste ordonate

O listă ordonată asigură o numerotare secvențială automată a articolelor listei. Delimitatorii unei liste ordonate sunt reprezentați de perechea de etichete . Atributele care stabilesc tipul de numerotare sunt:

TYPE =”1/A/a/I/I - numerotare conform tabelului următor.

Tabel 1.3

1	1,2,3,...
A	A,B,C,...
a	A,b,c,...
I	I,II,III,...
i	i,ii,iii,...

START=n - valoarea de start a numerotării. Valoarea este întotdeauna specificată în termeni de 1,2,3, etc., indiferent de orice atribut TYPE.

Exemplu:

```

<OL TYPE="i" START=2 >
    <LI>Element2.</LI>
    <LI>Element3.</LI>
    <LI> Element4.</LI>
</OL>

```

cu rezultatul:

- ii. Element2
- iii. Element3
- iv. Element4

1.5.3. Alte tipuri de liste

- Liste directoare

O listă directoare ar trebui să fie împrejmuită de o pereche de etichete `<DIR>` `</DIR>`. Listele directoare sunt la fel ca listele neordonate cu excepția faptului că nu pot fi închise iar eticheta `` nu are nici un atribut.

Listă directoare:

```
<DIR>
<LI>Mole</LI>
<LI>Foal</LI>
<LI>Vole</LI>
<LI>GOAL.!</LI>
</DIR>
```

- Liste de meniuri

O listă de meniuri ar trebui să fie împrejmuită de o pereche de etichete `<MENU>` `</MENU>`. Listele de meniuri sunt la fel ca listele neordonate, cu excepția că ele pot fi închise.

- Liste definite

Listele definite afișează o listă de articole, fiecare articol fiind o pereche de genul “termen – definiție”. Spre deosebire de celelalte tipuri de liste, listele definite nu se folosesc de elementele `` ``, în schimb perechea de etichete `<DT>` `</DT>` este folosită pentru a indica ‘termenul’, iar etichetele `<DD>` `</DD>` conțin ‘definiția’ corespunzătoare fiecărui termen. O listă definită ar trebui inclusă într-o pereche de etichete `<DL>` `</DL>`.

Some definitions:

```
<DL>
  <DT>plasma</DT>
    <DD>definție pentru plasma</DD>
  <DT>thyristor</DT>
    <DD>definția tiristorului.</DD>
  </DL>
  <DL COMPACT>
    <DT>alt termen</DT>
    <DD>definția noului termen</DD>
  </DL>
```

1.6. Mapări pe imagini

Mapările pe imagini constă practic în delimitarea anumitor zone active pe suprafața unei imagini (crearea unei “hărți”), sub forma unor regiuni de tip *referință-link*, la al căror apelare (*click*) se execută o anumită operație (de regulă

încărcarea unui document HTML, apelul unui script etc.). [1] Etichetele de bază pentru realizarea unei mape pe o imagine sunt : `<MAP>`, `<AREA>`.

Alegând o imagine convenabilă, este necesară în primul rând afișarea graficului pe ecran și indicarea faptului, că referitor la respectiva imagine, există creată o mapă. Pentru a realiza acest lucru se folosește eticheta ``, cu atributul `USEMAP`. În continuare este necesară descrierea coordonatelor care delimită zonele active mapate și a acțiunilor care se execută la apelul acelei mape. Cu alte cuvinte, maparea unei imagini constă în atribuirea unei anumite acțiuni diverselor regiuni ale unei imagini, pe care se face click cu mouse-ul.

Etichetele de bază prin care se realizează o mapă activă pe o imagine sunt prezentate în continuare:

`<MAP>`

Această etichetă conține definirea mapei și numele atribuit acesteia prin proprietatea `NAME: NAME="text"`

`<AREA>`

Această etichetă este folosită pentru a marca o zonă activă a imaginii ca o zonă “activă” (căreia i se mapează o anumită acțiune), lucrând practic ca o sub-etichetă a etichetei MAP. Nu există o limitare a numărului acestor zone `<AREA>` în cadrul unui MAP. Dacă două zone se intersecțează, zona care apare prima în definirea mapei este prioritată în regiunea de suprapunere.

`SHAPE="rect | circle | poly | default"`

Definește forma unei zone active mapate, ca dreptunghi, cerc, poligon sau ca restul imaginii rămas nedefinit (valoarea *default*). Setând valoarea la *default*, înseamnă că eticheta `<AREA>` este aplicată în afara spațiului imaginii corespunzător oricărei zone mapate.

`HREF="URL"`

Este folosit pentru a specifica o legătură destinație (*link-ul*) pentru zona activă. Are aceeași sintaxă ca și atributul HREF al etichetei ``.

`COORDS="x1,y1,x2,y2 | x,y,r | x1,y1,x2,y2,x3,y3..."`

Definește poziția precisă a unei zone mapate (active) folosind coordonatele în pixeli, separate prin virgulă. Un dreptunghi este descris prin coordonatele colțurilor stânga-sus și dreapta-jos. O zonă circulară este descrisă prin coordonatele centrului, urmate de raza sa. Un poligon este definit de coordonatele tuturor colțurilor poligonului. Atributul COORDS este întotdeauna necesar pentru atributul SHAPE, exceptând cazul când SHAPE este setat *default*.

`NOHREF`

Folosit pentru a specifica o zonă care nu e în viață.

`ALT`

Se folosește numai începând cu *Microsoft Internet Explorer 4*. Este folosit pentru a prevedea un mesaj (*hint*) pentru legătura care apare atunci când mouse-ul trece deasupra zonei active.

`TARGET="frame_name | window_name"`

Acest atribut aloca destinația unei legături *link*, pentru a încărca informația referita prin HREF într-un frame predefinit sau într-o fereastră. Dacă nu este nici un frame cu numele specificat, atunci legătura va fi încărcată într-o fereastră nouă (care primește acel nume).

Exemplu (pentru Internet Explorer):

```
<IMG SRC="timis.jpg" USEMAP="#timis" BORDER=3>
<MAP NAME="timis">
  <AREA SHAPE=RECT COORDS="11,121,329,548"
HREF="vest.html" ALT=" vest Timis">
  <AREA SHAPE=RECT COORDS="327,157,700,344"
HREF="ne.html" ALT="nord-est Timis" >
  <AREA SHAPE=RECT COORDS="328,344,685,570"
HREF="se.html" ALT="sud-est Timis" >
</MAP>
```

Observație: Referitor la exemplul anterior, în cazul unei rulării pe un browser Mozilla este posibil ca anumite atribute să nu aibă efectul așteptat (spre exemplu atributul ALT nu este funcțional).

Într-un paragrafele precedente se vorbea despre DHTML, adică de modul în care chiar unele etichete HTML aduc o oarecare dinamică în statismul unei pagini Web. În continuare sunt exemplificate câteva asemenea etichete.

1.7. Dinamică și multimedia. HTML 5

Unul dintre obiectivele majore ale fiecărei noi versiuni HTML a fost îmbunătățirea dinamicii paginilor Web, asigurându-le o interactivitate cât mai mare, respectiv un design cât mai atractiv, în principal printr-o extindere a capabilităților multimedia. [3]

Câteva exemple care surprind aceste aspecte sunt punctate în continuare:

- Crearea unui **blinking** text

Limbajul HTML dispune de o etichetă pentru a crea un text special ca dinamică, care să “sară în evidență” atunci când un navigator accesează o anumită pagină. Aceasta este eticheta **<blink>**, iar textul marcat de ea va înregistra un efect dinamic de “clipire”:

```
<blink>Acesta este un text blink.</blink>
```

Observație: Tag-ul **<blink>** este interpretat numai de browserele Mozilla, celelalte neasigurând o “clipire” a textului.

- Elementul de tip **<marquee>**

Dacă se dorește o deplasare a unui text pe ecran, în genul unei reclame, un alt tag poate fi utilizat rapid și eficient. Doar Internet Explorer suportă eticheta

specială **<marquee>** care asigură o defilare (*scrolling display*) pe ecran a oricărui text încadrat de ea.

Observație: Această etichetă nu este suportată de alte browsere, acestea afișând întregul text, fără însă a-l deplasa pe ecran.

```
<marquee> Dacă folosiți IE, acest text ar trebui să se deplaseze! </marquee>
```

- Etichete META destinate multimedia

Exemple de etichete META care asigură o dinamică a paginii Web, permitând redarea unor secvențe multimedia video/audio:

```
<META HTTP-EQUIV="Refresh" CONTENT="3 clock.avi"></meta>
<META HTTP-EQUIV="Refresh" CONTENT="7 URL=utopia.wav">
<IMG dynsrc=Clock.avi loop=infinite>
```

Cifra care apare în interiorul atributului **CONTENT**, asigură un *refresh* (reîncărcare) a secvenței multimedia după un interval de timp prestabilit.

- HTML 5

Ultima versiune HTML 5 vine cu o serie de facilități care îmbunătățesc performanțele multimedia ale aplicațiilor Web. Astfel, dinamica unei pagini este mult crescută prin introducerea unor noi etichete vizând partea audio, video și grafică, tratarea erorilor este mai simplă și eficientă, iar pentru o serie de etichete HTML anterioare sunt modificate și specificațiile/atributele. [7] Totodată o serie de etichete sunt abrogate. Spre exemplu, frame-urile nu mai sunt funcționale într-un document HTML 5, iar eticheta **<object>** permitând încapsularea unor obiecte *plugin* pentru diverse facilități (în special multimedia) este și ea abrogată, fiind substituită cu noi etichete HTML 5, gen **<audio>**, **<video>**. În momentul actual se poate vorbi de o confruntare în zona Web multimedia între tehnologia HTML 5 funcționând fără *plugin*-uri adiționale (aplicații software adiționale), respectiv tehnologia clasică bazată pe plugin-urile Flash. Actualmente, browser-ele într-o proporție covârșitoare suportă Flash, în timp ce sub jumătate dintre ele suportă HTML 5. [23] HTML 5 implică o utilizare intensivă a altor tehnologii conexe bazate pe JavaScript, CSS, Canvas având o deschidere, conectivitate și capacitate mult mai mare de integrare cu noi tehnologii, ceea ce îl placează ca potențial câștigător în confruntarea cu tehnologia bazată pe clasicele plugin-uri. De menționat totuși că în acest moment se pare ca nici un browser Web nu suportă în totalitate HTML 5, pe cele mai bune poziții situându-se în acest sens Google Crome, Safari și Firefox 4. [7]

Câteva secvențe de cod HTML 5 pentru simple exemplificări sunt punctate în cele ce urmează:

- orice cod HTML 5 este precedat de o declarație de tipul:

```
<!DOCTYPE html>
```

- rularea unei secvențe video (inclusiv tratarea erorii de compatibilitate):

```
<video width="400" height="250" controls="controls">
  <source src="film.mp4" type="video/mp4" />
Browser-ul nu suporta HTML5.
</video>
```

- rularea unui fișier de sunet:

```
<audio controls="controls">
  <source src="sunet.ogg" type="audio/ogg" />
Browser-ul nu suporta HTML5.
</audio>
```

- definirea unui articol ca element afișat:

```
<article>
  <h1>Internet Explorer 9</h1>
  <p>Continut de afisat cu IE 9.</p>
</article>
```

- și exemplele ar putea continua cu o listă lungă de noi etichete (mai mult sau mai puțin suportate de diverse browsere):

- <section> - definește o secțiune a documentului,
- <progress> - indicator pentru progresul unui task,
- <canvas> - utilizat pentru generarea de grafică 2D on-line (utilizând adițional cod JavaScript) etc.

Avantajul HTML 5 privind integrarea facilă cu o serie de alte tehnologii și limbaje de programare Web rulând pe partea de client, poate însă constitui un handicap privind compatibilitatea cu multitudinea de browsere existente. Astfel, utilizarea intensivă a limbajului JavaScript în strânsă corelare cu noile etichete HTML 5 – cu toate problemele de compatibilitate funcție de tipul de browser – constituie un argument relevant în acest sens. Viitorul rămâne să decidă câștigătorul în această confruntare.

1.8. Elemente CSS

Cascading Style Sheets (CSS) este practic un “pseudo-limbaj” de formatare, utilizat pentru a descrie modul de prezentare a documentelor scrise într-un limbaj bazat pe marcaje (HTML, XML etc.). [6] Fișierele CSS (foi de stil în cascadă) permit separarea conținutului HTML propriu-zis al unui document, de stilul de afișare/formatare în pagină al acestuia. [9] Codul HTML se utilizează, de obicei, doar pentru aranjarea conținutului în pagină, iar detaliile care țin de afișare (culori, fonturi, fundaluri, margini etc.) se realizează cu ajutorul CSS-urilor, acestea aplicându-se suplimentar peste codul HTML, în cadrul unui site Web. Cu alte cuvinte, CSS-urile realizează separarea prezentării paginii de structura sa efectivă.

Aplicarea foilor de stil în cascadă asupra codului HTML se poate face în mai multe moduri, putându-se vorbi de:

- stiluri interne;
- stiluri externe;

- stiluri în linie;
- clase CSS.

1.8.1. Stilurile interne

În cazul utilizării stilurilor interne, codul CSS se plasează în interiorul fiecărei pagini HTML pe care se dorește să se aplice stilurile respective, între tagurile `<head> </head>`, după cum se poate vedea în continuare:

```
<head>
<title>Exemplu utilizare stiluri interne</title>
<style type="text/css">Aici se definesc stilurile CSS</style>
</head>
```

Pentru exemplificare, se prezintă un script HTML în care se utilizează stiluri interne (fig. 1.9):

```
<html>
<head>
<title> Exemplu de utilizare a stilurilor interne!!!
</title>
<style type="text/css">
table {
  font-family: "Edwardian Script ITC";
  font-size: 36px;
  color: #FFFFFF;
  background-color: #0099FF;
  border: 4px double #0033FF;
  text-align: center;
}
</style>
</head>

<body>
<br><br>
<table align="center">
<tr>
<td>
Exemplu de utilizare a stilurilor interne!!!
</td>
</tr>
</table>
</body>
</html>
```

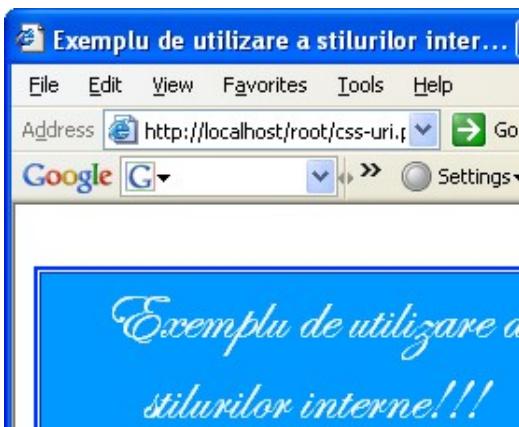


Fig.1.8. Exemplu folosire stiluri interne

Utilizând aceasta metodă de aplicare a CSS-urilor asupra codului HTML, dacă se dorește o schimbare a stilului de afișare (mărimea fontului, culoare, etc) modificarea va trebui realizată în toate paginile care conțin acel stil. Ținând cont de aceste aspect, această metodă este indicat a fi folosită doar în situația în care se dorește "stilizarea" un număr mic de pagini, fiind destul de neproductivă o realizare a acestor modificări pe zeci sau chiar sute de pagini ale unui site WEB.

1.8.2. Stilurile externe

Un fișier CSS extern poate fi scris cu orice editor simplu de text (Notepad, Wordpad, etc) sau cu editoare specializate (gen *Dreamweaver*). Fișierul CSS nu conține cod HTML, ci doar cod CSS și trebuie salvat cu extensia *.css*.

Referirea fișierului extern CSS în paginile HTML se face prin plasarea unui tag *link* (legatură), în secțiunea *<head> </head>* a fiecărei pagini în cadrul căreia se dorește aplicarea stilul respectiv, având forma următoare:

```
<link rel="stylesheet" type="text/css" href="Calea catre fisierul.css" />
```

În continuare, se prezintă un exemplu de referire a unui fișier extern *.css* într-o pagină HTML:

```
<html>
<title> Exemplu de utilizare a stilurilor externe!!!
</title><head>
<link href="cssExt.css" rel="stylesheet" type="text/css">
</head>

<body>
Exemplu de utilizare a stilurilor externe in body!!!

```

```
<table>
<tr>
<td>
Exemplu de utilizare a stilurilor externe in tabel!!!
</td>
</tr>
</table>
</body>
</html>
```

Fișierul CSS referit și utilizat în pagina HTML anterioară (*cssExt.css*), se consideră a avea următorul conținut exemplificativ (efectul utilizării lui putându-se observa în figura 1.8):

```
body
{
    font-family: "Courier";
    font-size: 26px;
    color:#000000;
    background-color:#B3B3B3;
    text-align: center;
}
table {
    font-family: "Edwardian Script ITC";
    font-size: 46px;
    color: #FFFFFF;
    background-color: #0090FF;
    border: 4px double #0033FF;
    text-align: center;
}
```

Aceasta metodă de utilizare a unor fișiere de stil externe, este preferată în momentul în care un site WEB conține un număr mare de pagini utilizând aceleși reguli de stil, motivul fiind evident: atunci când se dorește modificare aspectului întregului site, este suficientă doar o modificare într-o singură locație, și anume - fișierului CSS (efectul rezultându-se asupra tuturor paginilor din site care folosesc foaia de stil respectivă). Astfel, printr-o singură operație, se poate schimba rapid aspectul întregului site, indiferent de dimensiunea lui (număr de pagini).



Fig.1.9. Exemplificare stiluri externe

1.8.3. Stilurile în linie

Stilurile în linie se definesc chiar în codul etichetei HTML aferente elementului care se dorește a fi stilizat, după cum se poate vedea în exemplul următor (fig.1.10):

```
<body>
<p style="color: #00ddff; font-size: 20;">Titlu</p>
<h2 style="font-size: 16;font-weight: bold; color: #ff3300;">Exemplu utilizare stiluri in linie!!! </h2>
</body>
```

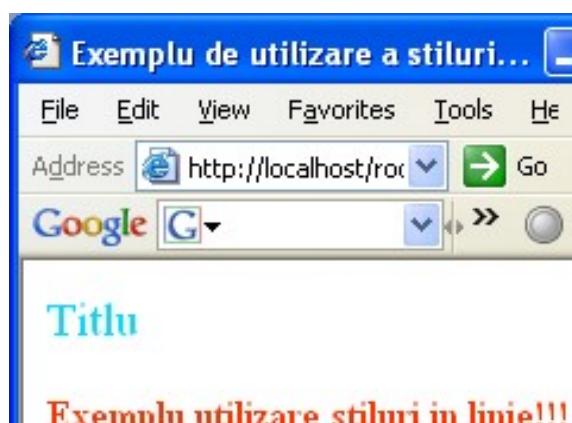


Fig.1.10. Exemplificare stiluri în linie

Stilurile în linie sunt mai puțin utilizate, deoarece ele nu permit schimbări rapide de stil pe mai multe fișiere în același timp, modificările trebuind realizate pe fiecare element în parte, și în fiecare pagină în parte.

Anterior au fost prezentate trei metode de aplicare a CSS-urilor asupra codului HTML. În situația în care, se folosesc două sau chiar trei metode în același timp, se pune întrebarea: care este ordinea/prioritatea folosirii lor pentru o interpretare corectă de către browser? Răspunsul este: metodele se vor executa în cascadă, în ordinea următoare: prima oară -stilurile în linie, urmate apoi de stilurile interne, iar în final - stilurile externe, aceasta fiind și ordinea lor de prioritizare. Evident, un element deja stilizat, spre exemplu, cu un stil linie, nu este restilizat de o regulă de stil existentă într-un fișier CSS extern, acționând imediat ulterior conform regulii de prioritizare anterior enunțate.

1.8.4. Clase CSS

Clasele CSS se utilizează pentru stilizarea în mod diferențiat a unor mulțimi de tag-uri HTML (distribuite în una sau mai multe pagini WEB). Acest mod de lucru este similar cu utilizarea stilurilor în linie, avantajul major fiind acela că atunci când se dorește efectuarea unei modificări de stil pe mai multe elemente/pagini, aceasta nu trebuie efectuată individual la nivelul fiecărui element. [8] Astfel, este suficientă o modificare în cadrul clasei CSS care definește stilurile respective, efectul acestora răspândindu-se asupra tuturor elementelor pe care acționează clasa respectivă. [5]

Definirea unei clase CSS începe cu semnul punct (.), după care urmează numele clasei. Se recomandă folosirea unor denumiri sugestive pentru numele clasei, care să indice ce anume face stilul respectiv. O clasă CSS poate fi folosită în cadrul unui fișier HTML ori de câte ori este nevoie. Iată un exemplu de clasă care stabilește dimensiunea și culoarea unui text:

```
.text20albastru
{
    font-size: 20px;
    color: 00ddff;
}
```

Pornind de la exemplificarea din la paragraful "Stiluri în linie", se prezintă modul de definire și utilizare a unor clase CSS într-un document HTML, clasele fiind stocată într-un fișier de stil extern:

- **Fișierul HTML** utilizând stiluri externe bazate pe clase:

```
<head>
<title> Exemplu de utilizare a stilurilor in linie!!!
</title>
<link href="claseCSS.css" rel="stylesheet" type="text/css">
</head>

<body>
```

```
<p class="text20albastru">Titlu</p>
<h2 class="text16rosu">Exemplu utilizare stiluri in
linie!!! </h2>

</body>
```

- **Fișierul CSS** (*claseCSS.css*) în care sunt definite cele două clase (ambele stilizând texte, dar în mod diferit):

```
.text20albastru {
    font-size: 20px;
    color: 00ddff;
}

.text16rosu {
    font-size: 16px;
    font-weight: bold;
    color: ff3300;
}
```

Stilizarea obținută în cadrul fișierului HTML anterior prezentat este evident identică cu cea obținută prin utilizarea stilurilor în linie (fig. 1.10).

Ca o concluzie, în contextul dezvoltării unor aplicații Web tot mai complexe, conținând un număr tot mai mare de pagini, și implicit de fișiere script, stilurile CSS constituie la ora actuală strategia consacrată, de maximă eficiență, pentru formatarea și designul primar al acestora.

1.8.5. Meniuri cu CSS

Una dintre aplicabilitățile cele mai uzuale ale CSS-urilor constă în crearea de meniuri necesare navigării într-o aplicație, atât foarte simple, cât și cu o complexitate deosebită. Scheletul HTML (ca fundament al unui meniu) pe care sunt aplicate stilurile CSS, consistă în structuri de tip listă (folosindu-se etichete pentru crearea unor liste neordonate **, împreună cu elementele lor constitutive **), încapsulate eventual într-un DIV, respectiv hyperlink-urile aferente (ancore *<a>*). Din păcate dependența de browser se face resimțita și în cazul utilizării de stiluri CSS, astfel încât (mai ales pentru meniuri mai complexe), este foarte posibil ca un meniu care funcționează pe o familie de browsere, să nu fie complet funcțional pe o alta. Într-un astfel de caz, soluția de rezolvare constă în apelarea suplimentară a unor scripturi JavaScript.

Exemplul următor, utilizând doar CSS (fără elemente JavaScript), implementează un meniu simplu (fără submeniuri –vezi fig. 1.11), fiind funcțional

atât pe Mozilla, cât și pe Internet Explorer. Pentru crearea meniului s-a utilizat un fișier extern CSS, în care sunt definite mai multe stiluri.



Fig.1.11. Meniu creat cu CSS

Fișierul HTML (aferent unui meniu cu trei opțiuni) are următorul cod:

```
<link href="meniu-html.css" rel="stylesheet"
type="text/css">
<div id="nav-menu">
<ul>
<li><a href="#">Optiune1</a></li>
<li><a href="#">Optiune 2</a></li>
<li><a href="#">Optiune 3</a></li>
</ul>
</div>
```

Fișierul implementând stilurile externe CSS (*meniu-html.css*) conține următoarele elemente de stilizare:

```
#nav-menu ul
{
padding: 0;
margin: 0;
}
```

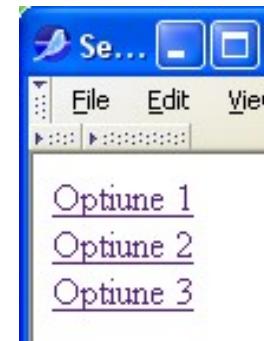


Fig.1.12.a Efect doar cu *#nav-menu ul*

```
#nav-menu li
{
float: left;
margin: 0 0 4px;
background:#B3B3B3;
}
```

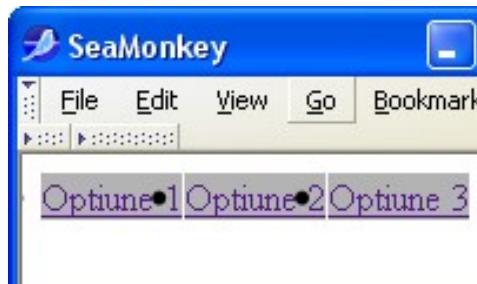


Fig.1.12.b Efect cumulat cu stilul
#nav-menu ul li

```
#nav-menu li a
{
background: url(background.jpg) #fff bottom left repeat-x;
height: 2em;
line-height: 2em;
float: left;
width: 9em;
display: block;
border: 0.1em solid #dcdce9;
color: #FFFFFF;
text-decoration: none;
text-align: center;
}
```

Figurile 1.12.a, respectiv 1.12.b prezintă efectul succesiv cumulat al fiecărui nou element de stil aplicat (ultimul conducând chiar la meniul final din fig. 1.11).

În cadrul fișierului CSS se poate remarca referirea **#nav-menu** specificând *id-ul* elementului DIV (înglobând întreaga construcție a meniului). Cele trei stiluri **#nav-menu ul**, **#nav-menu li**, **#nav-menu li a**, se referă la formatarea listelor (**ul**) având ca părinte DIV-ul (**#nav-menu**), a elementelor listelor (**li**), respectiv a conținutului ancorelor (**a**), având ca părinte elemente ale listei (**li**), care la rândul lor au ca părinte DIV-ul (**#nav-menu**).

Deși rulând codul anterior pe Mozilla, unele stiluri ar putea părea inutile, o simplă rulare pe Internet Explorer este relevantă pentru a dovedi necesitatea lor! Spre exemplificare, eliminarea stilului aferent elementelor din lista (#nav-menu li) conduce pe Internet Explorer la un meniu de genul celui din fig. 1.13.

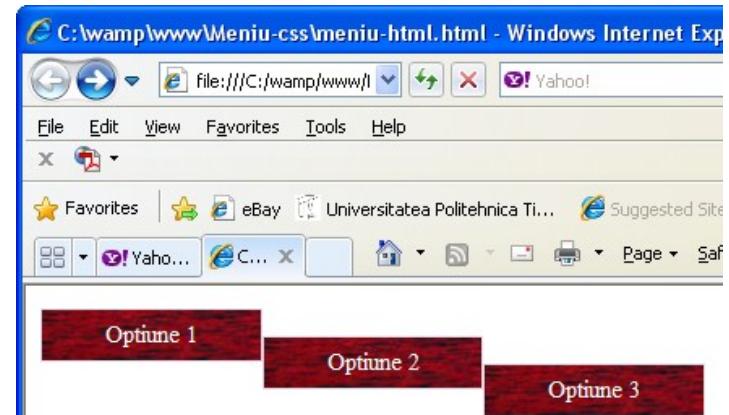


Fig.1.13 Efect fără #nav-menu li – rulare cu Internet Explorer

Un exemplu de meniu mai complex, având și sub-meniuri (*drop-down menu*), este prezentat în cele ce urmează [38]. Întregul cod, atât lista HTML, cât și stilurile CSS (ca stiluri interne), sunt integrate într-un același fișier HTML. Efectul fiecărui nou element CSS succesiv adăugat este prezentat în figurile 1.14.a...1.14.h. (fiecare figură fiind alăturată secvenței de cod CSS nou adăugată).

```
<ul id="menu">
<li><a href="#">unu</a></li>
<li><a href="#">doi</a>
    <ul>
        <li><a href="#">doi-1</a></li>
        <li><a href="#">doi-2</a></li>
        <li><a href="#">doi-3</a></li>
    </ul>
</li>
<li><a href="#">trei</a>
    <ul>
        <li><a href="#">trei-1</a></li>
        <li><a href="#">trei-2</a></li>
    </ul>
</li>
</ul>
```

```
<style type="text/css">
ul
{
    font-family: Arial, Verdana;
    font-size: 14px;
    margin: 0;
    padding: 0;
    list-style: none;
}
```

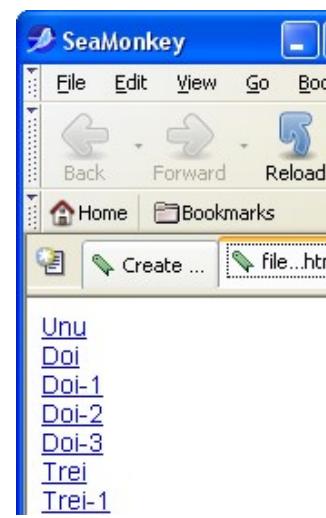
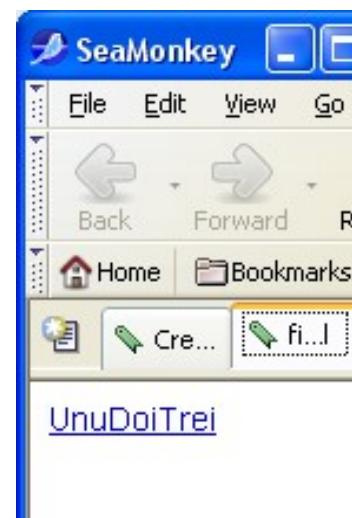


Fig.1.14.a Stilizare liste

```
ul li
{
    display: block;
    position: relative;
    float: left;
}
```

Fig.1.14.b Stilizare elemente liste
(având ca părinte liste)

```
li ul
{
    display: none;
}
```

Fig.1.14.c Stilizare liste (doar cele având ca părinte elemente)
în cazul de față ascunderea lor

```
ul li a
{
    display: block;
    text-decoration: none;
    color: #ffffff;
    border-top: 1px solid #ffffff;
    padding: 5px 15px 5px 15px;
    background: #1e7c9a;
    margin-left: 1px;
    white-space: nowrap;
}
```

Fig.1.14.d Stilizare ancore <a> având ca părinte elemente listă , care la rândul lor au ca părinte liste
(practic toate ancorele meniuului,
inclusiv cele ascunse)

```
ul li a:hover
{
background: #3b3b3b;
}
```

// **hover** – element selector atașat unei etichete HTML, generând, **în momentul în care mouse-ul se deplasează deasupra acesteia**, o stilizare fie a acesteia, fie a altelor etichete.
// **ul li a:hover** - când mouse-ul se deplasează deasupra unei ancore <a> (etichetă selectată), modifică fundal ancorei, având ca părinte un element , care la rândul face parte dintr-o listă .

```
li:hover ul
{
    display: block;
    position: absolute;
}
```

// **li:hover ul** - când mouse-ul se deplasează deasupra unui element , afișează elementul (având ca părinte acel)

```
li:hover li {
    float: none;
    font-size: 20px;
}
```

// **li:hover li** - când mouse-ul se deplasează deasupra unui element , stilizează acel (având ca părinte alt)



Fig.14.e Stilizare elemente (cele vizibile) având ca părinte liste - (setare culoare background/fundal la deplasare mouse deasupra ancorelor <a>)



Fig.14.f Afisare submeniu la o deplasare mouse deasupra unui element având ca părinte o listă (practic anulează ascunderea)

// aici s-ar putea opri dezvoltarea meniului



Fig.14.g La o deplasare deasupra unui element listă - setare scris mai mare

```
li:hover a {
background: #00ff00;
color: #ff00ff;
}
```

// simple schimbări de culori (nu sunt necesare)

```
</style>
```

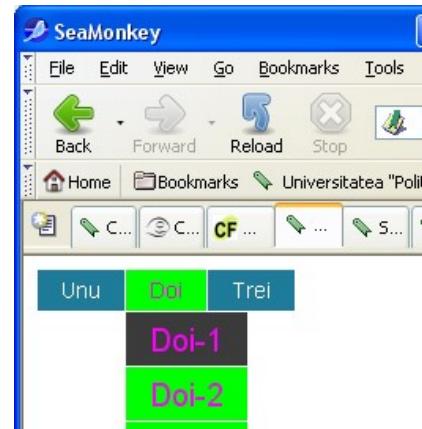


Fig.14.h La o deplasare deasupra unui - setare culoare scris și fundal ancoră <a>

Observație: Din păcate, meniul anterior prezentat funcționează integral doar pe Mozilla, și doar parțial pe Internet Explorer (sub-meniurile nefiind active). Rezolvarea constă în utilizarea de cod suplimentar JavaScript. [38]

Primul exemplu din acest paragraf face referire la o etichetă DIV, definind o diviziune (secțiune) a unui document HTML, des utilizată împreună cu elementele CSS. Oarecum echivalent cu DIV, eticheta SPAN permite și ea o încapsulare a unor elemente într-o secțiune a unei pagini HTML. Deosebirile majore între cele două etichete sunt următoarele:

- DIV - permite formatarea întregii secțiuni delimitate (folosind stiluri CSS);
- SPAN - nu permite o formatare a secțiunii delimitate de el, ci doar a elementelor (de regulă, a textului) incluse în el. În plus, un SPAN este precedat implicit de un paragraf nou, începând pe o linie nouă (mai puțin cazul în care este inclus într-un DIV).

Codul următor (vezi și figura 1.15), este relevant pentru evidențierea caracteristicilor celor două etichete (DIV și SPAN):

```
<div id="mydiv" style="color: blue; background: yellow">Ceva
text in Div 1

<span style="color: red; background: white"> Ceva text in
Span 1 (integrat într-un DIV -fara paragraf nou) </span>
Tot text in Div 1.
</div>

<span style="color: white; background: red">Span extern 2 (se
observă ca începe cu paragraf nou)</span>

<h3>Inainte de Span 3 <span style="color: red; background:
yellow">Continut Span 3 </span> Dupa Span 3</h3>

<span style="color: red; background: yellow; line-height:
10em">Continut Span 4 -stilizare doar continut efectiv
</span>
```



Fig.1.15 Exemplificare DIV și SPAN

1	HTML	1
1.1.	Introducere	1
1.2.	Etichete HTML	1
1.2.1.	Etichete primare	1
1.2.2.	Setarea unui fundal al paginii	2
1.2.3.	Formatare text	3
1.2.4.	Inserarea unei imagini	4
1.2.5.	Hiperlegături	4
1.2.6.	Formulare. Metodele GET și POST	5
1.3.	Frame-uri	Error! Bookmark not defined.
1.3.1.	Frame-uri linie/coloană	Error! Bookmark not defined.
1.3.2.	Frame-uri încubitate	Error! Bookmark not defined.
1.3.3.	Atribute ale frame-urilor	Error! Bookmark not defined.
1.3.4.	Frame-uri legate	Error! Bookmark not defined.
1.4.	Tabele	7
1.4.1.	Elemente introductive	7
1.4.2.	Controlul global al unui tabel	8
1.4.3.	Controlul unei celule	8
1.4.4.	Combinarea celulelor	9
1.4.5.	Formatarea coloanelor	9
1.4.6.	Fundal și margini tabelă	10
1.5.	Liste	11
1.5.1.	Liste neordonate	11
1.5.2.	Liste ordonate	11
1.5.3.	Alte tipuri de liste	12
1.6.	Mapări pe imagini	12
1.7.	Dinamică și multimedia. HTML 5	13
1.8.	Elemente CSS	14
1.8.1.	Stilurile interne	14
1.8.2.	Stilurile externe	15
1.8.3.	Stilurile în linie	16
1.8.4.	Clase CSS	16
1.8.5.	Meniuri cu CSS	17

Elemente HTML 5

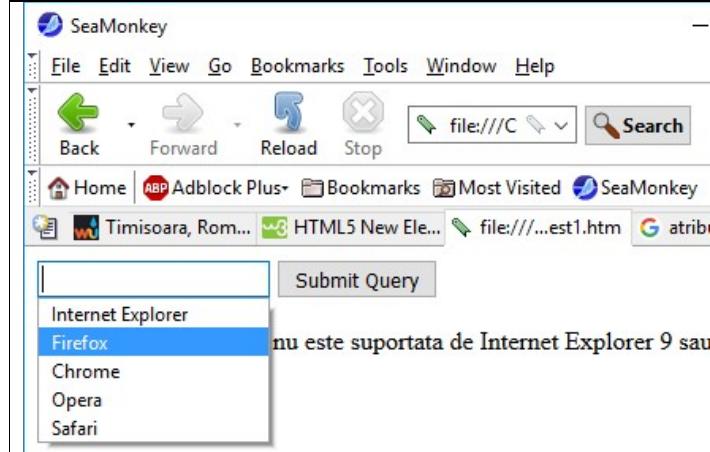
1. Element nou: DATALIST

```
<!DOCTYPE html>
<html>
<body>

<form action="/action_page.php" method="get">
<input list="list1" name="aleg">
<datalist id="list1">
<option value="Internet Explorer">
<option value="Firefox">
<option value="Chrome">
<option value="Opera">
<option value="Safari">
</datalist>
<input type="submit">
</form>
```

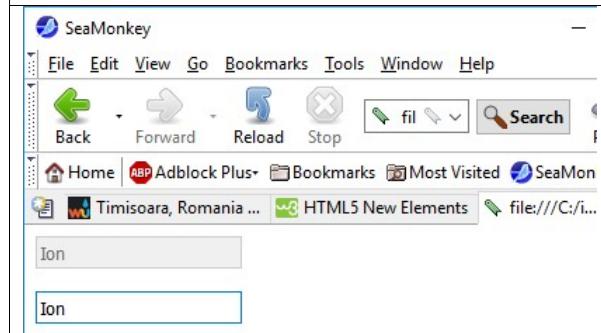
<p>Nota: Eticheta **datalist** nu este suportata de Internet Explorer 9 sau versiuni mai vechi, sau de Safari.</p>

```
</body>
</html>
```



2. Atribut nou pentru elementul INPUT: **DISABLED**

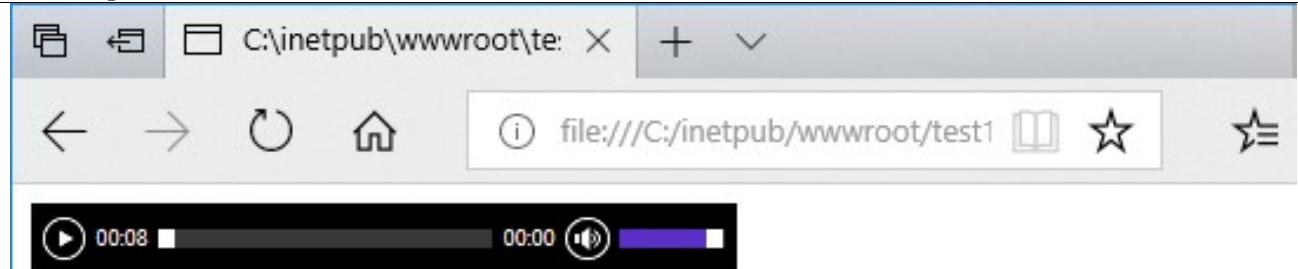
```
<form action="/action_page.php" method="get">
<input type="text" value="Ion" disabled><p>
<input type="text" value=Ion>
</form>
```



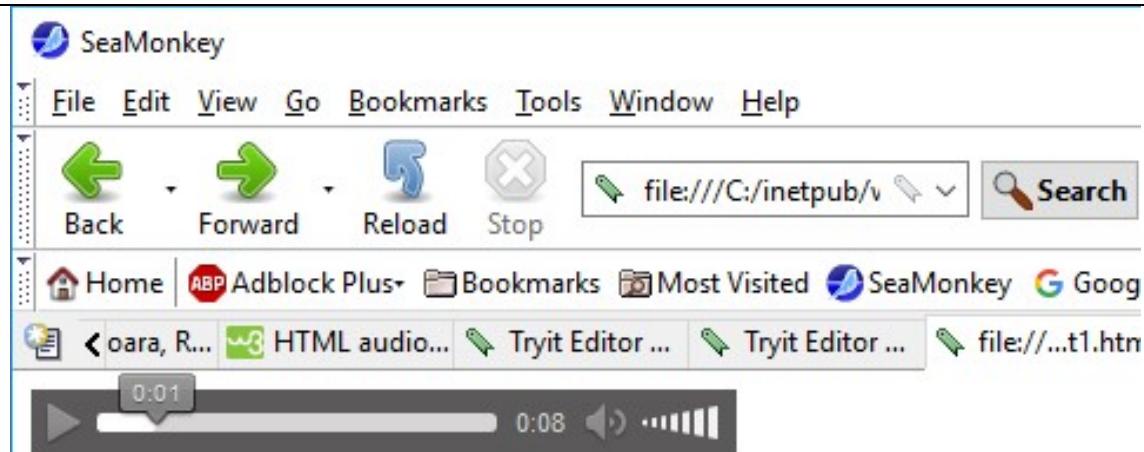
3. AUDIO

```
<audio controls>
<source src="horse.mp3" type="audio/mpeg">
<source src="horse.ogg" type="audio/ogg">
Browser-ul nu suporta acest element!
</audio>
```

<p>Nota: Eticheta AUDIO nu este suportata de Internet Explorer 8 sau versiuni mai vechi.</p>



Nota: Eticheta AUDIO nu este suportata de Internet Explorer 8 sau versiuni mai vechi.
EDGE

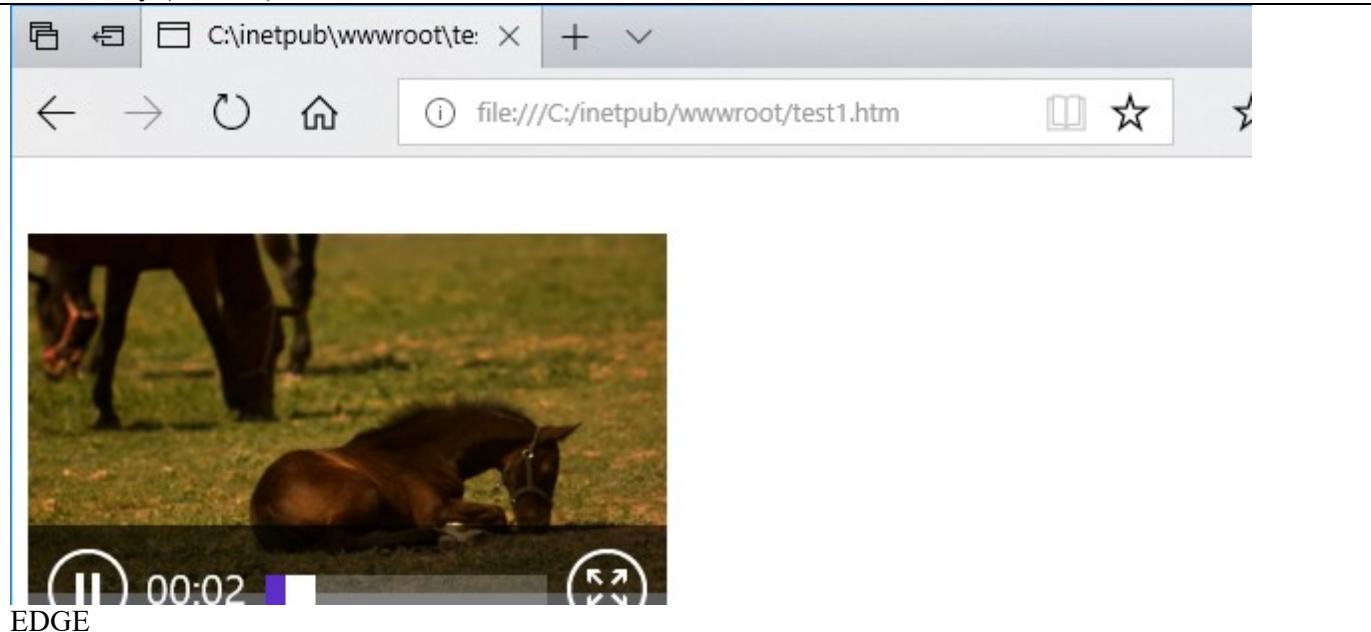
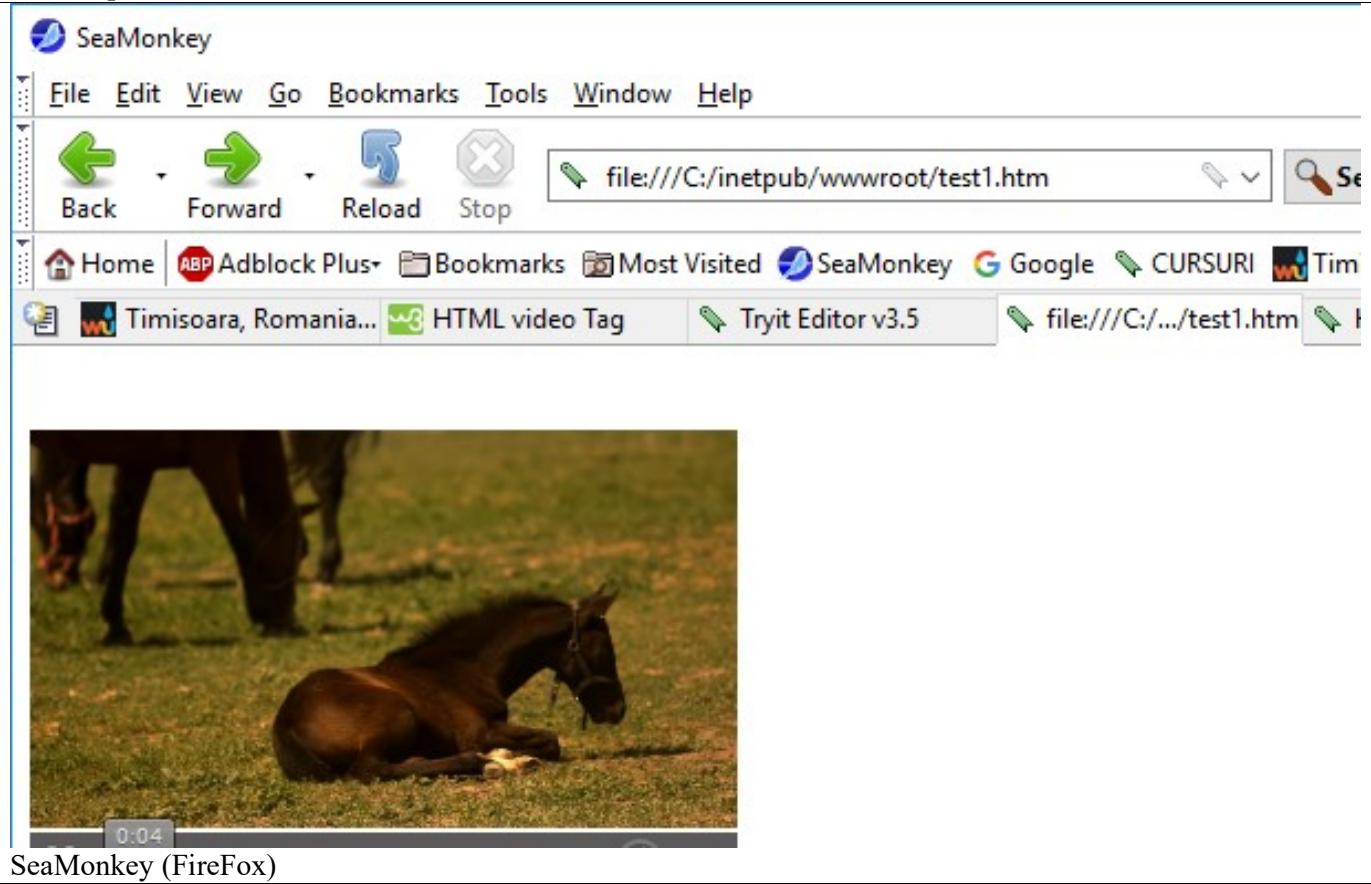


Nota: Eticheta AUDIO nu este suportata de Internet Explorer 8 sau versiuni mai vechi.
SeaMonkey (FireFox)

4. VIDEO

```
<video width="320" height="240" controls>
<source src="horses.mp4" type="video/mp4">
<source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
```

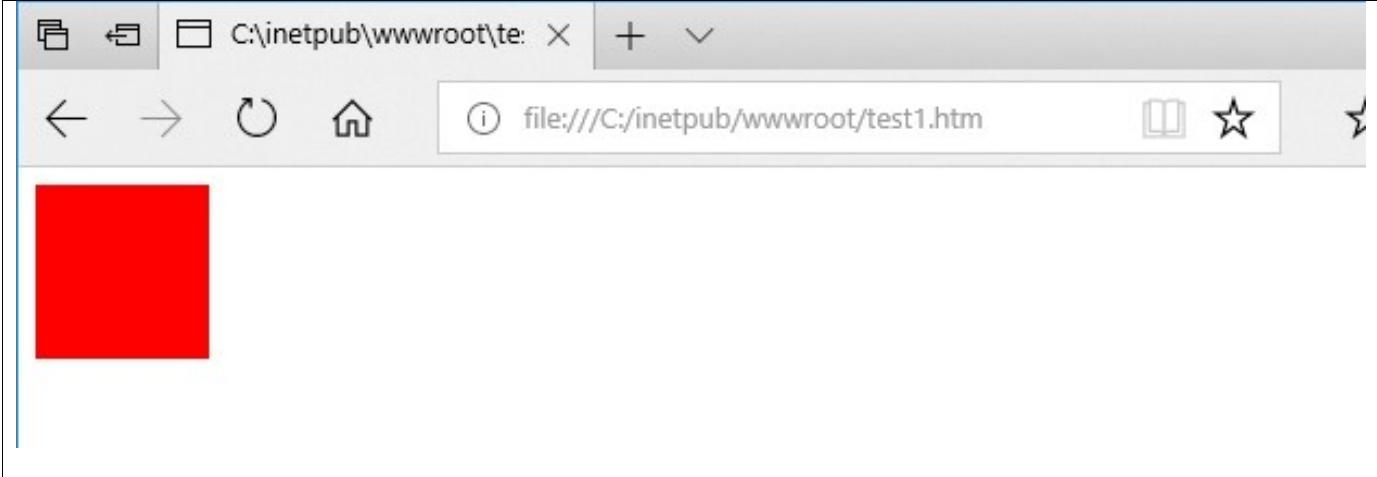
<p>Nota: Eticheta AUDIO nu este suportata de Internet Explorer 8 sau versiuni mai vechi.</p>



5.CANVAS

```
<canvas id="myCanvas"></canvas>

<script>
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
ctx.fillStyle = "#FF0000";
ctx.fillRect(0, 0, 80, 80);
</script>
```



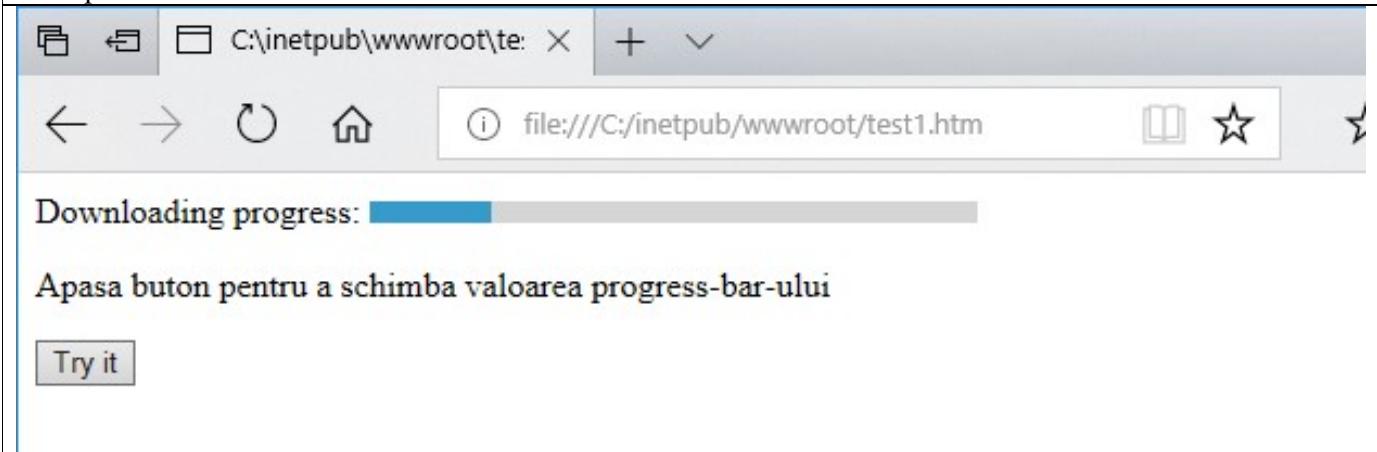
6. PROGRESS

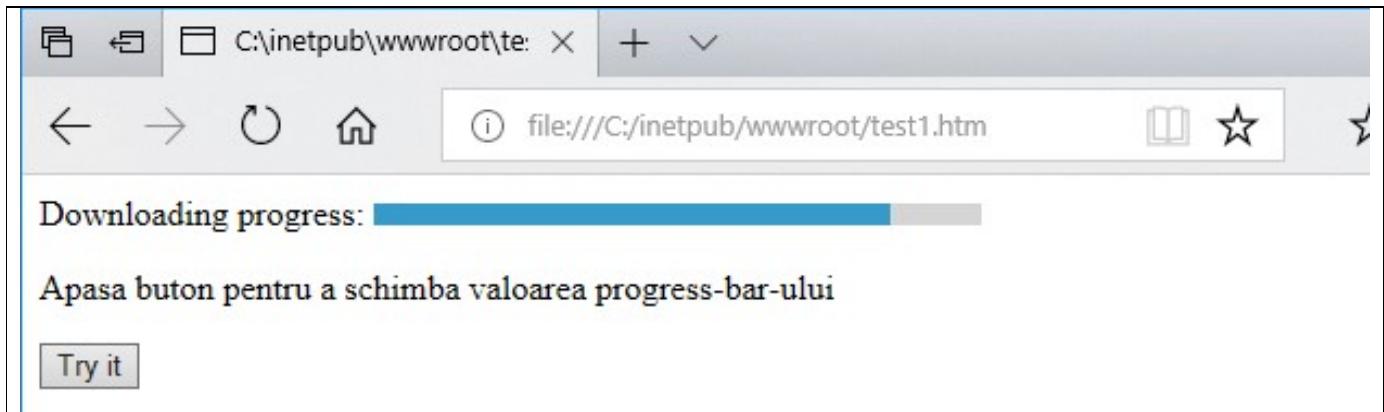
Downloading progress:

```
<progress id="myProgress" value="20" max="100">
</progress>
<p> Apasa buton pentru a schimba valoarea progress-bar-ului</p>

<button onclick="myFunction()">Try it</button>

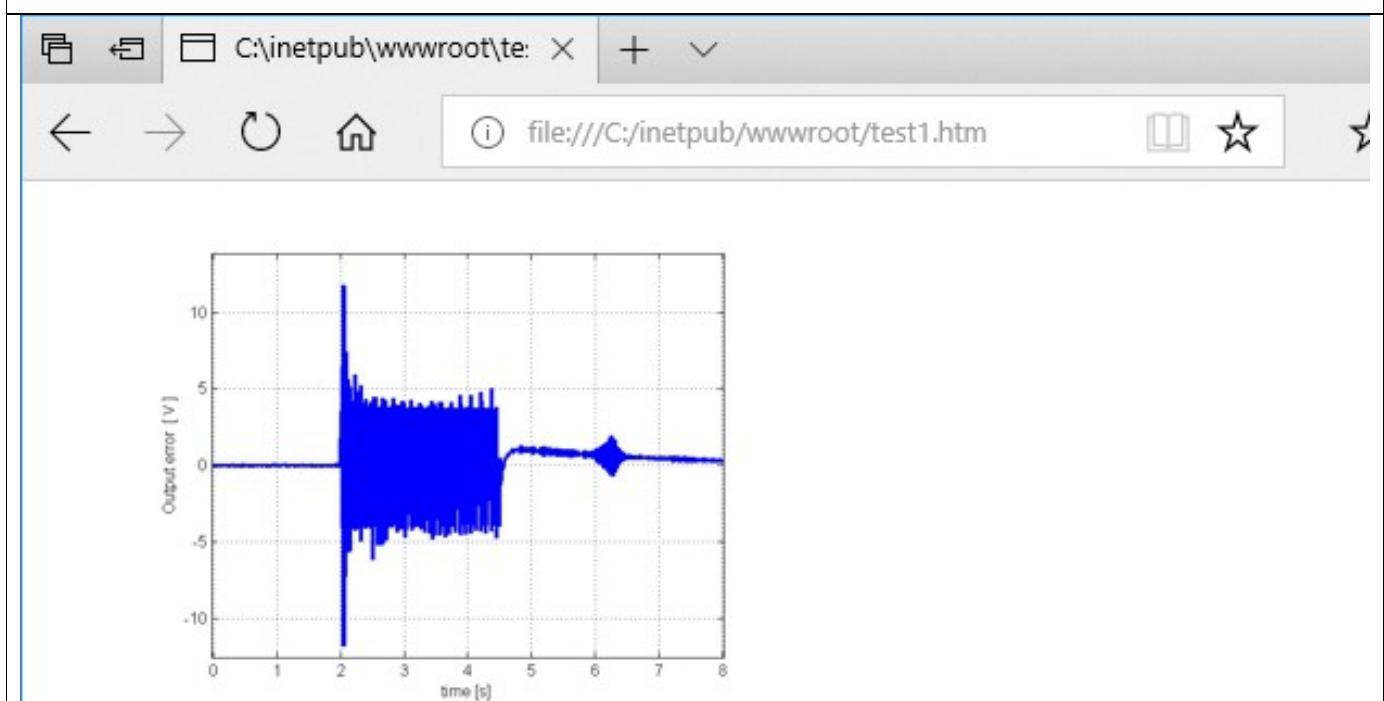
<script>
function myFunction()
{
    document.getElementById("myProgress").value = "85";
}
</script>
```





7. FIGURE si FIGCAPTION

```
<figure>
  
  <figcaption>Fig.1 - Un grafic</figcaption>
</figure>
<p><strong>Note:</strong> The figure tag is not supported in Internet Explorer 8 and earlier versions.</p>
```



PHP

1. Preliminarii

Unul dintre cele mai utilizate limbaje de programare folosit la ora actuală pentru dezvoltarea aplicațiilor Web este PHP-ul (semnificația acronimului PHP fiind - **PHP: Hypertext Preprocessor**). PHP-ul este un limbaj de *scripting* derivat din familia C, rulând multiplatformă (atât pe platformele Windows, cât și Unix-Linux), permitând dezvoltarea rapidă de scripturi server-side de către orice programator familiarizat cu binecunoscutul și popularul limbaj C. [10][20]

Ca software open-source și având o robustețe confirmată în cei aproape 20 de ani de utilizare, PHP-ul prezintă un cert avantaj în puternica competiție pe care o are cu alte limbaje și tehnologii de programare Web server-side, cum ar fi ASP, ASP.Net, JavaServer Pages (JSP) etc. Evident, fiind un limbaj rulând pe partea de server (de aceea se face o comparație a acestuia doar cu acest tip de limbaje de programare Web), scripturile PHP sunt rulate pe un server Web (folosind motorul interpretor PHP), iar utilizatorul (clientul browser) nu poate vedea codul sursă PHP al programului, ci doar codul paginii HTML returnată spre browser-ul client. Astfel spus, motorul limbajului interpretează codul sursă program PHP, generând pe baza acestuia cod HTML (integrat cu datele / informațiile utile returnate/generate), pasat apoi mai departe de către serverul Web spre browser-ul apelant, care știe să interpreze/afișeze corespunzător codul HTML.

Un alt mare avantaj al PHP-ului constă în faptul că suportă o mare varietate de tipuri de baze de date, cum ar fi MySQL, Oracle, PostgreSQL, IBM DB2, InterBase, Sysbase, Microsoft SQL Server (suportul pentru MySQL fiind încorporat nativ în interpretorul PHP). [12] Astfel, se pot crea foarte ușor aplicații Web pentru accesarea bazelor de date, PHP-ul oferind fie suport nativ, fie un suport concretizat prin utilizarea unor biblioteci .dll suplimentare externe (programatorul trebuie să utilizeze selectiv biblioteci specifice tipului respectiv de bază de date). [19] Alt avantaj este faptul că suportă o serie de protocoale de rețea, printre care SMTP, NNTP, POP3 și HTTP. În cadrul lucrării de față s-a folosit versiunea PHP 5.2.3. În momentul redactării, ultima versiune PHP 5.4.4 era disponibilă on-line la adresa www.php.net.

Limbajul de scripting PHP a fost implementat în 1994 de Rasmus Lerdorf, la început fiind vorba de un proiect personal care includea câteva macro-uri și un modul care permitea interpretarea lor, menite să urmărească "activitatea" paginii sale personale. În 1995 este disponibil sub numele *Personal Home Page*. Începând cu versiunea 3, din 1997, a început să fie dezvoltat de o echipă de programatori, iar

începând cu versiunea 4 dispune de *engine*-ul de scripting al firmei Zend Technologies. Există multe clasamente la ora actuală privind popularitatea limbajelor de programare, fiecare având diverse criterii luate în calcul, iar o simplă căutare pe Internet poate găsi o multitudine de asemenea surse vizând aceste clasamente. Din acest motiv, fără a cita explicit o anumită sursă, se poate afirma că PHP este probabil cel mai folosit limbaj de programare Web pe parte de server, surclasându-și momentan toți concurenții prin preț, robustețe și longevitate, simplitate și viteză, posibilitate de integrare cu alte tehnologii (inclusiv baze de date), securitate, portabilitate, instalare facilă etc. [24]

Observație: Limbaje (sau pseudo-limbaje) uzuale folosite în programarea WEB: HTML-CSS, PHP, ASP.NET, JavaServer Pages, JavaScript, Perl etc. Pe partea de client rulează: HTML, JavaScript. Pe partea de server rulează: PHP, ASP.NET, JavaServer Pages, Perl. De asemenea, CSS –*Cascading Style Sheets* – poate fi considerat ca un pseudo-limbaj de formatare, utilizat pentru a descrie modul de prezentare / afișare a documentelor descrise printr-un limbaj bazat pe marcaje (HTML, etc.).

Capitolul de față încearcă o prezentare sintetică, concisă a elementelor de bază ale limbajului PHP, punându-se un accent special pe operarea cu baze de date de tip MySQL, respectiv Oracle, însătoare de exemplificări și aplicații concrete. Scopul este cel de a familiariza programatorul cu caracteristicile esențiale ale limbajului, permitându-i dezvoltarea rapidă a unor aplicații Web, în contextul unei integrări cu HTML și MySQL. Utilizarea PHP, fiind un limbaj server-side, necesită un server de WEB. Toate exemplele prezentate în continuare au fost rulate folosind ca server de WEB, fie *Apache*, fie *Internet Information Services* (IIS), numit uneori formal și *Internet Information Server*.

2. Instalare

Alt avantaj major al PHP îl constituie instalarea extrem de facilă, în contextul unei conexiuni atât cu serverul de Web Apache, cât și cu serverul de baze de date MySQL. Astfel, cel mai simplu mod de a opera cu PHP-ul, în tandem cu MySQL și Apache, este de a folosi pachetul *WAMP*, integrând perfect această tripletă *PHP+MySQL+Apache* și disponibil la adresa www.wampserver.com/en/. Dupa simpla rulare a kit-ului de instalare *WAMP*, totul este deja pregătit pentru a serie și rula scripturi PHP (inclusiv cu apeluri la serverul de baze de date MySQL), fără a mai fi necesară nici o altă setare suplimentară (în fișierele de configurare).

Observație: De remarcat faptul că, la un moment dat, pe un calculator poate rula doar un server Web, astfel încât, dacă un alt server este deja instalat și pornit, este necesară o oprire prealabilă a acestuia.

Pachetul *WAMP* conține și aplicațiile *PHPMyAdmin* și *SQLiteManager*, folosibile pentru gestionarea cu ușurință a bazelor de date MySQL. *WAMP* este pachetul destinat operării sub Windows. [21] În mod similar se poate folosi *LAMP*

pentru operarea sub Linux. Cu aceleași avantaje, se poate folosi și pachetul XAMP (*Xm + Apache + Mysql + Php*), oferind facilități similare.

În cazul în care se utilizează componente disparate (interpretor PHP, server WEB, server de baze de date SQL), iar pe computer nu este instalat un server de WEB, este inițial necesară instalarea unuia. Versiunile mai vechi PHP (până la 3) erau disponibile doar ca pachet separat, integrarea lor cu un server de Web, respectiv server MySQL făcându-se manual, printr-o serie de setări în diverse fișiere de configurare (în principal fișierul *php.ini*), respectiv o setare corespunzătoare a serverului de Web (în special în cazul *IIS*).

De asemenea, realizarea unor anumite aplicații care necesită alte resurse conexe pachetului de bază PHP (biblioteci sau extensii), implică o configurare a PHP-ului prin diverse setări corespunzătoare în fișierul *php.ini*. Spre exemplu, pentru accesarea bazelor de date de tip Oracle este necesar ca extensie fișierul *php_oci8.dll*, iar pentru *InterBase* - fișierul *php_interbase.dll* (aceste biblioteci nefiind preinstalate implicit din motive de evitare a supraîncărării serverul Web, fiind vorba de fișiere *.dll* care rămân rezidente în memorie). Operația de instalare a acestor biblioteci este simplă. Pentru cazul anterior menționat, spre exemplu, este necesară doar identificarea bibliotecilor extensie necesare, urmată de o simplă necomentare (eliminarea caracterului de comentariu ;), în fișierul de configurare *php.ini*, a liniielor pe care apar referite aceste fișiere din secțiunea *Windows Extensions*: (în cazul de față *php_oci8_12c.dll*)

```
;Windows Extensions
```

```
...
```

```
extension=php_oci8_12c.dll
```

Observație: Conținutul fișierului *php.ini* (și implicit starea variabilelor de configurare ale PHP) poate fi vizualizat printr-un apel al funcției PHP ***phpinfo()***.

După o operație de reconfigurare (modificare a fișierului *php.ini*) este necesară o restartare a serverului Apache (sau a oricărui server Web folosit) pentru încărcarea și a acestor noi biblioteci. Toată configurația manuală anterior descrisă poate fi făcută și utilizând opțiunile meniu grafic pus la dispoziție de WAMP după instalare. O serie de alte configurații necesare a fi realizate în fișierul de configurare *php.ini* vor fi precizate pe paragrafele următoare, funcție de contextul folosirii altor biblioteci extensie, respectiv funcție de modul de setare al interpretorului PHP (spre exemplu, pentru realizarea unor configurații de securizare eficiente prin setarea corespunzătoare a unor variabile speciale care, în cele mai multe cazuri, sunt realizate adekvat, implicit la instalare).

Exemplul din paragrafele următoare (operând, cele mai multe, cu baze de date MySQL) au fost rulate utilizând pachetul *WAMPSERVER* versiunea 3.2.0 care conține în principal:

- Apache 2.4.41, PHP 7.3.12, MySQL 5.7.28
- PHPMyAdmin și alte opțiuni grafice de setare/configurare.

Codul PHP fiind interpretat pe partea de server Web (Apache, IIS, etc), nu prezintă nici o dependență față de browser-ul client (Internet Explorer, Mozilla,

Netscape, FireFox, Opera, Chrome, etc.). Verificarea instalării cu succes a pachetului WAMP (și a startării serverului Web) se poate face printr-un simplu apel la adresa: *http://localhost/*. Locația de plasare pe serverul Apache (instalat prin intermediul pachetului WAMP) a surselor cod PHP este *C:\wamp\www*, iar rularea unui fișier *script.php*, plasat aici, se face printr-un apel de forma *http://localhost/script.php* (evident din browser-ul Web).

3. Comenzi PHP

3.1. Sintaxă și elemente de bază

Sintaxa comenziilor elementare PHP este extrem de asemănătoare cu a altor limbi de structurate, precum C, JavaScript, Perl (mai precis cu a comenziilor oricărui limbaj derivat din familia C, împrumutând însă câte ceva și de la limbajul Perl, în special pe partea de operare cu *string-uri* de date). Se vorbește despre comenzi elementare ca fiind acele comenzi care constituie nucleul de bază al oricărui limbaj de programare, dificultatea majoră nefiind operarea cu acestea (în număr relativ limitat), ci mai degrabă constând în cunoașterea multitudinii de biblioteci extensie, a funcțiilor oferite de acestea pentru dezvoltarea diverselor tipuri de aplicații și funcționalități.

Deși PHP dispune și de capabilități specifice programării orientate pe obiecte (în special pentru versiunile mai noi, să cum se va vedea într-un paragraf următor al prezentului capitol), exemplele imediat următoare consideră cazul unei abordări structurate (utilizarea lui ca limbaj structurat fiind de altfel cea consacrată). Un script PHP constă într-o serie de comenzi, executate secvențial, una câte una, de către interpretorul PHP și al căror rezultat (date+cod HTML) este pasat spre un server Web. O eventuală eroare apărută este semnalată de interpretor și afișată de browser (chiar la apelul scriptului), fără blocarea execuției comenziilor următoare. Fiecare comandă se încheie cu caracterul punct-și-virgulă (;). Ca aplicație introductivă, se prezintă codul celui mai simplu program PHP, echivalentul clasicul *"Hello, World"* din C (integrând optional și cod HTML):

```
<html>
<body>
    <?php echo "Hello, World !!!"; ?>
</body>
</html>
```

Se observă prezența unui tag pereche special, *<?php ... ?>*, care încadrează /delimită codul PHP. Funcția *echo* asigură tipărirea pe ecran a sirului specificat ca parametru. PHP permite utilizarea și a altor sintaxe pentru definirea unei zone de cod sau chiar sintaxe diferite pentru referirea parametrului *string* al funcției de afișare *echo*:

```
<?php Echo ("acesta este un simplu test 1\n"); ?>
<?php echo 'acesta este un simplu test 2<br>'; ?>
```

```
<script language="php">
echo ("Un alt exemplu de delimitare cod PHP");
echo "Inca un exemplu de afisare a unui mesaj";
</script>
```

Totuși, cea mai răspândită sintaxă (folosită și în lucrarea de față) este cea utilizând perechea de etichete `<?php ... ?>`, fișierul sursă având extensia .PHP (extensie implicit recunoscută de serverul Apache - instalat integrat cu WAMP-care va pasa automat un fișier script cu aceasta extensie spre interpretorul PHP).

Orice linie de comandă, după cum s-a mai menționat, folosește ca terminator de linie caracterul “;”, la fel ca în C sau Java. Comentariile, de asemenea, împrumută sintaxa C, adică // pentru o linie individuală, respectiv /* ... */ pentru o secvență de cod înglobând mai multe linii.

Se consideră încă un exemplu pentru a evidenția câteva din caracteristicile limbajului (utilizând de data aceasta și funcția echivalentă print în loc de echo, cu același efect de afișare în browser a unei informații /cod HTML):

```
<html>
<head>
<title>Data curenta</title>
</head>
<body>
<b>Data curenta este: </b>
<?php
// se va afisa data calendaristica
echo "<p><b>";
print( date("d F Y, l. ") );
?>
</b></p>
</body>
</html>
```

Efectul rulării acestui script este prezentat în figura 2.1.:

Deși secvența anterioară începe cu cod HTML și conține cod HTML în mare măsură, codul PHP constând practic în doar două linii, fișierul sursă va avea obligatoriu extensia PHP (vezi în fig.1, linia de apel: localhost/data.php) Liniile dintre etichetele delimitatoare `<?php`, respectiv `?>` reprezintă codul PHP. Serverului WEB detectează extensia fișierului, apoi trimite și cere motorului PHP instalat pe același localhost să interpreteze codul dintre cei doi delimitatori. Interpretorul PHP execută funcția date prin care citește data calendaristică de pe server (localhost) și, integrând-o ca pe un simplu string în codul HTML, trimite rezultatul astfel construit (date+cod HTML) spre browser-ul apelant.



Fig. 1 Data curentă (preluată de pe sistemul host al serverului Web)

Browser-ului i se prezintă un cod HTML “curat”, integrând sub forma unei informații de tip sir de caractere (*string*) date curentă (cod HTML vizibil cu opțiunea *View source*):

```
<html>
<head>
<title>Data curenta</title>
</head>
<body>
<b>Data curenta este: </b>
<p><b>02 July 2012, Monday. </b>
</body>
</html>
```

De remarcat că orice “urmă” de cod PHP “dispare”, singura indicație asupra existenței acestuia fiind extensia fișierului script apelat. În locul acestuia, secvența vizibilă este un simplu cod HTML standard. Exemplul scoate în evidență câteva avantaje majore ale script-urilor rulate pe partea de server:

- Eliminarea problemelor de incompatibilitate vizând tipul de browser* (specifice limbajelor client-side). Scriptul este rulat pe partea de server – unde este instalat și interpretorul PHP-, deci nu pot apărea probleme legate de tipul de browser folosit pe partea de client.
- Acces la resursele de pe partea de server*. Dacă aceeași aplicație ar fi fost scrisă în JavaScript, data respectivă ar fi fost data corespunzătoare calculatorului client (pe care rulează browser-ul). Pentru o informație precisă și unitară privind data sau momentul de timp al apelului, informația preluată de pe server este singura demnă de luat în considerare. Un alt exemplu în acest sens, vizează o aplicație Web cu baze de date, unde este foarte clar că baza de date (comună tuturor apelanților aplicației Web) trebuie plasată pe partea de server (chiar dacă, fizic, acest server SQL are alt *remote host* decât cel al serverului Web pe care rulează codul PHP).
- Reducerea încărării clientului*. Rularea unui script pe partea de client (cazul JavaScript) poate fi destul de lentă, depinzând direct de performanțele

calculatorului care găzduiește browser-ul client. În cazul script-urilor rulate pe partea de server, performanțele calculatorului client nu mai prezintă o importanță decisivă privind viteza de accesare a paginii Web.

3.2. Tipuri de date

Principale tipurile de date suportate de PHP sunt: *integer*, *string*, *float*, *boolean*, *array*, *object*, *null*. Spre deosebire de alte limbaje de programare (chiar cele din familia C), nu este necesară o declarare prealabilă a unei variabile (și evident o definire a tipului acesteia). Numele variabilelor sunt precedate de caracterul "\$", făcându-se diferență între literele mari și mici folosite în cadrul numelor de variabile. Exemplul următor este relevant în acest sens :

```
<?php
$var = "Ion";
$Var = "Dan";
echo "$var, $Var"; //se afișează: Ion, Dan
echo $var, $Var; //se afișează: IonDan
echo '$var, $Var'; //se afișează: $var,$Var
?>;
```

Se poate remarcă și faptul că, funcție de tipul de ghilimele simple, duble sau lipsă acestor, comanda `echo` interpretează în mod diferit conținutul variabilelor de afișat (ghilimele simple forțând o interpretare *ad-literam* -ca simplu *string*- a conținutului, chiar dacă caracterul \$ indică prezența unor variabile care ar trebui interpretate ca și conținut și nu ca și nume).

O caracteristică remarcabilă privind tipurile de date în PHP o constituie faptul că tipul unei variabile este decis în momentul execuției unei operații asupra acesteia. Această operație poate fi o simplă atribuire a unei valori (tipul valorii sau modul ei de furnizare dictând tipul variabilei) sau o operație propriu-zisă care poate chiar modifica tipul inițial al variabilei. [25] Se poate ușor concluziona că o variabilă își poate schimba tipul pe parcursul rulării unui script.. De asemenea, nu mai sunt posibile binecunoscutele erori de conversie de tip specifice majorității celorlalte limbaje de programare. Această caracteristică este benefică limbajului PHP prin prisma funcționalităților lui principale de operare cu *string*-uri de date, trimise spre browser pentru construcția și afișarea unei pagini Web (și evident, ne benefică altor limbaje orientate spre cu totul alt gen de aplicații).

Următorul exemplu, vizând cele anterior menționate, este relevant în acest sens:

```
$var = "0"; // $var = "0" string
$var++;
// $var = "1"
$var+=1; // $var = 2 integer
$var += 'c';
// $var = 2 integer
$var = $var + 1.3; // $var = 3.3 float
$var = 5 + "10 obiecte"; // $var =15
$var = 5 + "obiecte"; // $var =5 integer
```

```
$var = "a"; // $var = "a" string
$var++;
// $var = "b"
$var = "nu"; // $var = "nu" string
$var++; // $var = "nu" string
```

Comentariile prezente în dreptul fiecărei linii precizează valoarea variabilei (\$var) după operația efectuată, respectiv tipul acesteia (în multe cazuri fiind vorba de o schimbare de tip, fără a se semnala o eventuală eroare, chiar și pentru cele mai puțin probabile situații de conversie de tip). Ultimele două linii prezintă cazul cel mai defavorabil, în care operația nu este posibilă, situație în care conținutul și tipul variabilei rămâne neschimbate.

Pentru tipul *array* sau sir (elementele unui sir putând fi chiar de tipuri diferite, ceea ce constituie iată o abatere de la regulile specifice altor limbaje puternic tipizate), un exemplu edificator este prezentat în continuare:

```
$sir=array(1=>'ion', 2=>'dan', 3=>225);
echo $sir[1]."<br>"; // afișează 'ion' (string)
echo $sir[2]."<br>"; // afișează 'dan' (string)
echo $sir[3]."<br>"; // afișează 225 (întreg)
$sir[3]++;
echo $sir[3]."<br>"; // incrementează 226 (întreg)
```

Eticheta `
` realizează afișarea pe o linie nouă, iar operatorul '.' (punct) este folosit pentru concatenarea a două *string*-uri). Un element al sirului poate fi referit independent, ca o variabilă, printr-o construcție de genul: `$nume_array[indice_element]`.

Secvența anterioară de cod poate fi rescrisă echivalent:

```
$sir=array('ion', 'dan', 225);
echo $sir[0]."<br>";
echo $sir[1]."<br>";
echo $sir[2]."<br>";
$sir[3]++;
echo $sir[2]."<br>";
```

În acest caz se poate observa că, dacă sirul este inițializat fără o precizare explicită a indicilor elementelor, primul indice -cel de start- are valoarea implicită 0 și NU 1 (situație des întâlnită pentru *array*-uri generate automat de unele comenzi/structuri PHP).

Despre tipul *boolean*, o simplă exemplificare cu cele două valori posibile:

```
$var=TRUE; //sau $var=false;
```

În plus, o eventuală conversie la *boolean* a unei valori numerice 0, string "0" sau *array* vid (fără elemente), conduce automat la valoarea FALSE (restul de valori non-zero, generând TRUE).

Asupra tipului *object* se va reveni într-un paragraf următor dedicat exclusiv abordării din punct de vedere al programării orientate pe obiecte (*new* fiind

cuvântul cheie folosit pentru creare unui obiect nou, obținut prin instanțierea unei clase).

Tipul NULL este un tip mai special, semnificând practic existența unei variabile fără nici o valoare atribuită explicit (printr-un abuz de terminologie, o variabilă "declarată", dar neinitializată). Exemplu:

```
<?php
$var;
echo $var;
?>
```

În contextul unei variabile de acest tip (NULL), este utilă exemplificarea modului de interpretare a conținutului ei de către funcțiile `isset`, `empty`, `is_null` (mai precis, rezultatul `boolean` returnat de aceste funcții):

```
isset($var)           //returnează FALSE (variabilă fără o valoare)
empty($var)          //returnează TRUE (variabilă vidă)
is_null($var)         //returnează TRUE (variabilă nulă)
```

Spre deosebire de o variabilă, având numele precedată de caracterul `$`, o constantă se definește printr-o comandă de genul:

```
define("constanta", valoare);
```

Valoare acesteia nu poate fi modificată ulterior definirii și inițializării, iar referirea la ea se face direct prin numele ei (fără a fi nevoie de precedarea acestuia de caracterul `$`). Spre exemplu:

```
define("increment", 10);
$var=100;
$var=$var+increment;           // $var=110
```

Observație: Cea mai amplă și completă sursă de documentare privind limbajul PHP o constituie manualul on-line al acestuia, accesibil la adresa <http://php.net/manual/en/>. Evident, descrierea sintaxei comenziilor, funcțiilor și a unor structuri de programare tratate în capitolul de față folosește ca sursă primară de documentare acest manual, venind însă în completarea lui printr-o manieră sintetică de prezentare selectivă și exemplificare a unora dintre cele mai importante și uzuale comenzi/structuri. [20] De asemenea, prezentarea făcută are un puternic caracter aplicativ, exemplele fiind noutatea față de alte surse de documentare. [1]

3.3. Structuri condiționale și de ciclare

Ca orice alt limbaj de programare, PHP-ul dispune de o serie de facilități pentru controlul fluxului de desfășurare al unui script. Astfel, limbajul beneficiază de aportul unor instrucțiuni prin care este permisă o deviere de la ordinea firească de derulare a fluxului de comenzi ale scriptului/programului (numite instrucțiuni/structuri de salt condiționat sau de ramificare, respectiv de ciclare). Câteva astfel de structuri de control ale fluxului unui program sunt detaliate în

continuare la nivel de sintaxă, fiind însătoare fiecare de exemple de utilizare: structura `if-else`, structura condițională `switch`, structurile `while` și `do-while`, bucla `for`, respectiv comanda de ciclare `foreach`.

Una din cele mai folosite structuri de control condiționale este `if-else`, în PHP având următoarea sintaxă simplificată:

```
if ( conditie ) {
// secvența care se execută dacă condiția este adevărată.
} else
{
// (Optional) secvența care se execută dacă condiția este falsă.
}
```

Structura `if-else` permite o bifurcare condiționată a programului. Iată un exemplu (în care trebuie remarcat și operatorul de comparare egalitate `==`):

```
if ( $name == "Ion" ) {
echo ("Mesaj de afisat pentru: $name");
} else {
echo ("Persoana neidentificat");
}
```

Exemplificarea anterioară utilizează o sintaxă redusă a structurii condiționale `if-else` (putând să existe, de altfel, și o singură ramură condițională, evident `if`-ul). De subliniat faptul ca parantezele accolade `{ }` sunt obligatorii, chiar dacă linia `else` delimită clar sfârșitul secvenței executate la îndeplinirea condiției. O formă sintactică extinsă permite practic o ramificare nelimitată. Spre exemplificare, o ramificare pe patru ramuri poate avea următorul cod:

```
if (conditie 1) {
    ...cod PHP
} elseif (conditie2) {
    ...cod PHP
} elseif conditie3) {
    ...cod PHP
} else (conditie 4) {
    ...cod PHP
}
```

Fiecare ramură `elseif` este testată doar dacă condițiile impuse pe ramurile anterioare nu sunt îndeplinite. Îndeplinirea unei anumite condiții și execuția unei anumite ramuri `elseif` (sau a ultimei ramuri `else`, ca ramură finală optională) conduce automat la ieșirea din structura condițională. Altfel spus, verificarea condițiilor se face secvențial, iar îndeplinirea uneia conduce automat la ieșirea din structură, verificarea tuturor ramurilor următoare fiind abandonată.

O alternativă des folosită pentru o multi-ramificație bazată pe structura *if-elseif-else* o constituie structura condițională *switch*. Pentru o exemplificare concretă comparativă *switch / if-elseif* se consideră următoarele secvențe:

```
if ($var == 0) {
    echo "mesaj 0";
} elseif ($var == "ceva") {
    echo "ceva";
} elseif ($var == 2) {
    echo "mesaj 2";
}

switch ($var) {
    case 0:
        echo "mesaj 0";
        break;
    case "ceva":
        echo "ceva";
        break;
    case 2:
        echo "mesaj 2";
        break;
}
```

Comanda *break* asigură ieșirea automată din structură, celelalte ramuri ne mai testându-se. Lipsa *break*-ului ar conduce la o verificare și a condițiilor de pe celelalte ramuri (care teoretic nu ar mai trebui îndeplinite, și deci nu și-ar mai avea sens testarea lor). Exemplul prezentat anterior folosește o structură *if-elseif* (fară ramura opțională *else*), evidențiind și faptul că variabila testată poate avea fie valori numerice, fie valori de tip *string* (diferența de abordare constând în modul de încadrare sau nu cu ghilimele a conținutului verificat).

Deși nerecomandată, comanda de salt necondiționat *goto*, este disponibilă și ea în PHP (doar începând cu versiune PHP 5.3 !). Exemplu:

```
goto eticheta;
echo 'Nu se va executa niciodata';
eticheta: echo 'Se executa permanent';
```

Trecând la structurile de ciclare, poate cea mai des utilizată este bucla *while* cu sintaxa clasică:

```
while (conditie) {
// secvența de comenzi executată
// repetat, cât timp condiția este adevărată
}
```

Această buclă permite executarea repetată a unei secvențe de comenzi atât timp cât este îndeplinită o condiție (verificată aprioric execuției corpului de comenzi). Un exemplu de afișare succesivă a numerelor de la 1 la 10, utilizând o buclă *while* are codul următor:

```
$count = 1;
while ($count <= 10) {
echo("$count");
// sau print ('$count');
$count++;
}
```

Oarecum echivalentă cu *while* este structura *do-while*, deosebirea majoră constând în faptul că testarea condiției se face ulterior execuție corpului de instrucțiuni (și nu aprioric ca în cazul *while*):

```
do {
// secvența de comenzi executată repetat,
// (prima execuție fiind implicită) cât timp condiția este adevărată
}
while (conditie);
```

Nu putea lipsi din cadrul structurilor de ciclare clasica buclă *for* cu sintaxa standard împrumutată de la limbajul C:

```
for (initializare_contor; conditie; actualizare_contor)
{
// secvența de comenzi care se execută
// atât timp cât condiția este adevărată
}
```

Un exemplu de implementare a unui numărător printr-o buclă *for*:

```
for ($count = 1; $count <= 10; $count++) {
echo("$count");
}
```

Si în final, foarte des utilizată în contextul operării cu structuri de date de tip *array*, comanda *foreach* permite crearea unei bucle pentru accesarea secvențială a elementelor unui sir - *array*. Dacă o structură de date *array* este convertită într-o structură de tip *object*, *foreach* permite operarea și cu noua structură de date de tip *object*. Sintaxa generală *foreach* vizând tipul *array* este:

```
foreach (nume_array as $element)
{
// secvența de comenzi executată repetat, procesând variabila $element
}
```

Un exemplu concret de extragere și afișare a elementelor unui sir (*array*) folosind *foreach*:

```
$sir = array(1, 2, 3, 4);
foreach ($sir as $element)
{
    print $element.'<br>'; // afișare elemente
}
```

Structurile repetitive implementate cu *foreach* sunt foarte des utilizate în contextul operării cu baze de date. Motivul îl constituie faptul că, de regulă, liniile unei tabele sunt disponibile succesiv sub forma unui *array*, iar accesul la fiecare element al acestuia se poate realiza cu o buclă *foreach*. Atenție însă, la fiecare nou ciclu *foreach*, valoarea anterior extrasă este stearsă (suprascrisă). O alternativă la acest impediment o constituie comanda *list*, permitând extragerea

într-o listă de variabile predefinite a elementelor unui sir (extragerea tuturor elementelor făcându-se simultan, nu succesiv).

Exemplu:

```
$sir = array('unu', 'doi', 3);  
list($var1, $var2, $var3) = $sir;  
echo $var1.' '.$var2.' '.$var3;
```

Dacă numărul de variabile nu coincide cu numărul de elemente ale sirului *array* (indiferent că este mai mare sau mai mic), extragerea se va face doar pentru elementele disponibile, fără semnalarea unui mesaj de avertisment sau de eroare. Dezavantajul față de *foreach* constă în faptul că, dacă se dorește extragerea într-un set de variabile a întregului conținut al sirului, trebuie cunoscut aprioric numărul de elemente ale acestuia.

PHP

4. Dezvoltare de aplicații PHP cu baze de date MySQL

4.1. Funcții PHP pentru operare cu MySQL

Pe lângă setul standard de comenzi aferente limbajului PHP, acesta dispune de o serie de funcții și variabile predefinite care ușurează munca programatorului, unele dintre ele fiind incluse implicit (*built-in*) în interpretor (spre exemplu, suportul pentru MySQL, FTP, XML), altele fiind disponibile prin utilizarea unor fișiere externe ca extensii (*biblioteci .dll*). Aceste fișiere extensie de tip *.dll* (cu numele generic *php_*.dll*) permit limbajului PHP să opereze cu o mare diversitate de tehnologii conexe, inclusiv cu baze de date (în acest ultim caz, ele înglobând funcțiile necesare accesării și manipulării unor baze de date de diverse tipuri).

MySQL este unul dintre cele mai folosite sisteme de gestiune a bazelor de date client-server, fiind cel mai des utilizat în combinație cu scripturi PHP. [11][20] Operarea cu baze de date MySQL nu necesită utilizarea unei biblioteci suplimentare (*.dll*), suportul pentru MySQL fiind inclus funcțional în motorul interpretorului PHP. Prezentul paragraf va prezenta cele mai importante funcții PHP utilizate în lucrul cu bazele de date MySQL (tabel 1), însăși de aplicații exemplificative în paragrafele ulterioare.

Tabel 1

<u>Tip returnat</u>	<u>Funcție(...)</u>	<u>Aceiuni/caracteristici</u>
object mysqli_connect	<ul style="list-style-type: none"> - realizează conectarea la un server MySQL; - funcția necesită patru parametri uzuali: nume server MySQL/ adresă IP server, nume utilizator MySQL, parola de acces și numele bazei de date; - returnează un obiect care reprezintă conexiunea la serverul MySQL, în cazul în care conexiunea a reușit, respectiv valoarea logică FALSE în cazul în care conexiunea a esuat. <p>Stilul procedural: \$mysqli=mysqli_connect('localhost','root','','curs');</p> <p>Stilul orientat pe obiecte: \$mysqli=new mysqli ('localhost','root','','curs'); //se instantiază un obiect al clasei mysqli, prin apelarea constructorului mysqli, având cei 4 parametri.</p>	
bool mysqli_close	<ul style="list-style-type: none"> - primește ca parametru un identificator de acces la o conexiune cu un server MySQL (doar în cazul stilului 	

	<p>procedural) și realizează închiderea acesteia. În cazul în care parametrul nu este specificat, se va închide conexiunea curentă;</p> <ul style="list-style-type: none"> - returnează valoarea logică TRUE în cazul în care închiderea conexiunii s-a realizat cu succes, respectiv valoarea logica FALSE în caz contrar; - funcția este optională în cazul în care conexiunea nu este permanentă, aceasta închizându-se automat la final de script. <p>Stilul procedural: mysqli_close(\$mysqli);</p> <p>Stilul orientat pe obiecte: mysqli->close();</p>
object mysqli_query	<ul style="list-style-type: none"> - trimite/pasează o comandă SQL spre serverul MySQL (sub formă unui <i>string</i> de caractere); - primește ca și parametri stringul aferent comenzi SQL și identificatorul de conexiune, returnat de funcția mysqli_connect (acesta din urmă, doar în cazul stilului procedural). <p>- Pentru comenziile SELECT, SHOW, DESCRIBE OR EXPLAIN executate cu succes, funcția va returna un obiect de tipul mysqli_result (reprezentând setul de date obținut în urma interogării). Pentru alte comenzi executate cu succes, funcția va returna TRUE. În caz de eșec, funcția va returna FALSE.</p> <p>Stilul procedural: \$result=mysqli_query(\$mysqli, \$interrogare);</p> <ul style="list-style-type: none"> - unde: \$interrogare este o comandă SQL, iar \$mysqli este identificatorul de conexiune returnat de funcția mysqli_connect. <p>Stilul orientat pe obiecte: \$result=\$mysqli->query(\$interrogare);</p> <ul style="list-style-type: none"> - unde: \$interrogare este o comandă SQL.
array mysqli_fetch_row	<ul style="list-style-type: none"> - primește ca parametru obiectul de tip mysqli_result returnat de funcția mysqli_query, reprezentând setul

	<p>de date obținut în urma interogării.</p> <ul style="list-style-type: none"> - returnează o înregistrare/linie extrasă dintr-o tabelă interogată (sau juncțiune de tabele), sub forma unui șir (<i>array</i> cu primul indice de element 0). - presupunând \$row numele <i>array</i>-ului returnat, elementele acestuia pot fi referite pe baza indicelui lor: \$row[0], \$row[1], ... <p>Stilul procedural: \$row=mysqli_fetch_row(\$result);</p> <p>Stilul orientat pe obiecte: \$row=\$result->fetch_row();</p>
array mysqli_fetch_assoc	<ul style="list-style-type: none"> - primește ca parametru obiectul de tip <i>mysqli_result</i> returnat de funcția <i>mysqli_query</i>, reprezentând setul de date obținut în urma interogării. - aceeași funcționalitate ca și mysqli_fetch_row, dar elementele șirului returnat sunt referite nu prin indici, ci prin nume asociate lor, identice cu numele coloanelor referite prin interogare: \$row[coloana1], \$row[coloana2], ... <p>Stilul procedural: \$row=mysqli_fetch_assoc(\$result);</p> <p>Stilul orientat pe obiecte: \$row=\$result->fetch_assoc();</p>
array mysqli_fetch_array	<ul style="list-style-type: none"> - primește ca parametru obiectul de tip <i>mysqli_result</i> returnat de funcția <i>mysqli_query</i>, reprezentând setul de date obținut în urma interogării. - aceeași funcționalitate ca și mysqli_fetch_row /mysqli_fetch_assoc, în varianta implicită (fără parametri opționali), permitând ambele moduri de referire a elementelor șirului returnat: fie prin indice, fie prin numele asociat (numele coloanei). - deci se poate face o referire echivalentă a unui element: \$row[0] sau \$row[coloana1] <p>Stilul procedural: \$row=mysqli_fetch_array(\$result);</p>

	<p>Stilul orientat pe obiecte: \$row=\$result->fetch_array();</p>
object mysqli_fetch_object	<ul style="list-style-type: none"> - aceeași funcționalitate ca și mysqli_fetch_row, returnând însă un tip <i>object</i>, datele putând fi referite ca proprietăți ale acestuia: \$row->coloana1, \$row->coloana2, ... <p>Stilul procedural: \$row=mysqli_fetch_object(\$result);</p>
	<p>Stilul orientat pe obiecte: \$row=\$result->fetch_object();</p>
	<p>int mysqli_num_rows</p> <p>int mysqli_num_fields int mysqli_field_count</p> <ul style="list-style-type: none"> - returnează numărul de randuri din setul de date obținut în urma executării unei interogări. <p>Stilul procedural: \$nr=mysqli_num_rows(\$result)</p> <p>Stilul orientat pe obiecte: \$nr=\$result->num_rows;</p> <p>Stilul procedural: \$mysql=mysqli_connect('localhost','root','','curs'); \$result=mysqli_query(\$mysql,"select * from tabela_curs");</p> <p>\$nr=mysqli_num_fields(\$result); sau \$nr=mysqli_field_count(\$mysql); //se ia în considerare setul de date obținut ca urmare a execuției ultimei comenzi SQL de interogare.</p> <p>Stilul orientat pe obiecte: \$mysql=new mysqli ('localhost','root','','curs'); \$result=\$mysql-> query("select * from tabela_curs");</p>

	<pre>\$nr=\$result->field_count; sau \$nr=mysqli>field_count; //se ia in considerare setul de date obtinut ca urmare a executiei ultimei comenzi SQL de interogare.</pre>
int mysqli_affected_rows	<ul style="list-style-type: none"> - primeste ca parametru obiectul de tip mysqli_result returnat de functia mysqli_query, reprezentand setul de date obtainut in urma interogarii. - returneaza numarul de linii afectate de o comanda SQL anterioara de tip UPDATE, INSERT, DELETE, SELECT. Pentru comenziile SELECT, functia mysqli_affected_rows se comporta ca si functia mysqli_num_rows. <p>Stilul procedural:</p> <pre>\$mysqli=mysqli_connect('localhost','root','','curs'); \$result=mysqli_query(\$mysqli,"select * from tabela_curs"); echo \$nr= mysqli_affected_rows(\$mysqli); //se ia in considerare setul de date obtinut ca urmare a executiei ultimei comenzi SQL de interogare.</pre> <p>Stilul orientat pe obiecte:</p> <pre>\$mysqli=new mysqli('localhost','root','','curs'); \$result=\$mysqli->query("select * from tabela_curs"); echo \$mysqli->affected_rows; //se ia in considerare setul de date obtainut ca urmare a executiei ultimei comenzi SQL de interogare.</pre>
object mysqli_fetch_field_direct	<ul style="list-style-type: none"> - permite oferirea de informații asupra unui set de date obtainut in urma unei interogari, returnând ca rezultat un obiect a carui proprietati furnizeaza numele, dimensiunea și tipul campurilor din setul de date la care face referire. <pre>Object mysqli_fetch_field_direct(\$result, int \$field_nr) // 0 - prima coloana, 1 - a doua coloana...</pre> <p>Stilul procedural:</p>

	<pre>\$mysqli=mysqli_connect('localhost','root','','curs'); \$result=mysqli_query(\$mysqli,"select * from tabela_curs"); \$variab = mysqli_fetch_field_direct(\$result, 0); echo \$variab->name; echo \$variab->length; echo \$variab->type;</pre> <p>Stilul orientat pe obiecte:</p> <pre>\$mysqli=new mysqli ('localhost','root','','curs'); \$result=mysqli_query(\$mysqli,"select * from tabela_curs"); \$variab=\$result-> fetch_field_direct(0); echo \$variab->name; echo \$variab->length; echo \$variab->type;</pre>
int mysqli_connect_errno	<ul style="list-style-type: none"> - permite identificarea erorii care s-a produs in timpul realizarii celei mai recente conexiuni la server, daca s-a produs o eroare; - returneaza codul numeric al erorii de conexiune la server, in cazul in care aceasta exista, respectiv 0 in caz contrar. <p>Stilul procedural:</p> <pre>if(mysqli_connect_errno(\$mysqli)) { die ("Eroare"); }</pre> <p>Stilul orientat pe obiecte:</p> <pre>\$mysqli->connect_errno; // proprietate a obiectului mysqli care furnizeaza codul numeric al erorii de conexiune, daca aceasta exista.</pre>
int mysqli_errno	<ul style="list-style-type: none"> - permite identificarea erorii care s-a produs in timpul executiei de catre serverul MySQL a celei mai recente comenzi SQL; - returneaza codul numeric al erorii in cazul in care a aparut o eroare in timpul executarii celei mai recente comenzi SQL, respectiv 0 in caz contrar.

	<p>Stilul procedural: if(mysqli_errno(\$mysqli)) { die ("Eroare"); }</p> <p>Stilul orientat pe obiecte: \$mysqli->errno; // proprietate a obiectului mysqli care furnizeaza codul numeric al erorii, daca aceasta exista</p>
string mysqli_connect_error	<ul style="list-style-type: none"> - permite identificarea erorii care s-a produs în timpul realizarii celei mai recente conexiuni la server, daca s-a produs o eroare; - returnează un mesaj în care eroarea de conexiune este descrisă explicit, în cazul în care aceasta există, respectiv un sir vid în caz contrar. <p>Stilul procedural: if(mysqli_connect_error(\$mysqli)) { die('Eroare: ' . mysqli_connect_error(\$mysqli)); }</p> <p>Stilul orientat pe obiecte: \$mysqli->connect_error; // proprietate a obiectului mysqli care furnizează explicit eroarea de conexiune, daca aceasta există.</p>
string mysqli_error	<ul style="list-style-type: none"> - permite identificarea erorii care s-a produs în timpul executării unei comenzi SQL; - returnează un mesaj în care eroarea este descrisă explicit în cazul în care a apărut o eroare în timpul executării unei comenzi SQL, respectiv un sir vid în caz contrar. <p>Stilul procedural: if(mysqli_error(\$mysqli)) { die('Eroare: ' . mysqli_error(\$mysqli)); }</p>

	<p>Stilul orientat pe obiecte: \$mysqli->error; // proprietate a obiectului mysqli care furnizează explicit eroarea, dacă aceasta există.</p>
	<p>class mysqli_result (obiectul \$result întalnit în cele explicate mai sus este un obiect al clasei mysqli_result, reprezentând setul de date obținut în urma executiei unui query)</p> <p>mysqli_free_result</p>
	<ul style="list-style-type: none"> - reprezintă setul de date obținut în urma executiei unui query. - dispune de o serie de proprietăți și metode pentru a putea accesa elementele din setul de date și pentru a afisa diverse informații legate de setul de date. <p>Stilul procedural: mysqli_free_result(\$result);</p> <p>Stilul orientat pe obiecte: \$result->free();</p>

Observație: De regulă, cele mai utilizate funcții PHP pentru operarea cu o bază de date MySQL sunt: **mysqli_connect** (conectare la serverul MySQL și deschiderea unei baze de date), **mysqli_query** (interrogare prin trimiterea spre server a unui *string* într-o sintaxă validă de comandă SQL), **mysqli_fetch_row** (extragere date returnate), **mysqli_close** (închidere conexiune). Se poate remarcă o mare varietate de comenzi destinate extragerii datelor returnate de o interrogare (**mysqli_fetch_...**), rămânând la latitudinea programatorului selecția comenzi pe care o consideră cea mai potrivită.

Folosind funcțiile anterior prezentate (și nu numai acestea), programatorul poate realiza aplicații PHP complexe în conexiune cu baze de date de tip MySQL, conferind astfel o dinamică sporită aplicației WEB.

4.2. Elemente preliminare MySQL

După instalarea pachetului WAMP, un client consolă MySQL este disponibil chiar din meniul WAMP, sub denumirea *MySQL Console* (fig. 2). [21] User-ul cu rol de administrator al serverului MySQL are numele *root* și nu are o parolă implicit setată. [11] Este indicat ca, după prima logare, să se seteze o parolă pentru acest important user.



Fig. 2 WAMP- MySQL Console

Setarea unei parole pentru user-ul *root* se poate face cu o comandă de genul:

```
mysql> SET PASSWORD FOR 'root'@'localhost' =
PASSWORD('parola');
mysql> SET PASSWORD FOR 'root'@'%' = PASSWORD('parola');
```

De sunt necesare două comenzi? Practic sunt doi useri administrator, cu același nume *root*, unul fiind dedicat conectării exclusiv de pe *localhost*, iar celălalt (cu același nume) pentru realizarea unor conexiuni din exterior (la distanță sau *remote*). Cei doi useri pot fi vizualizați cu comanda (parolele fiind criptate):

```
mysql> select host, user, password from mysql.user;
```

Tot după instalarea serverului de baze de date, două baze sunt implicit create, având numele *mysql* (conținând informațiile pentru administrarea userilor, drepturilor acestora, parolelor de acces, etc.), respectiv *test* (bază de lucru pentru *root*).

Vizualizarea numelor bazelor de date existente pe server, respectiv deschiderea uneia sau alteia și afișarea numelor tabelelor din baza curentă, se poate face cu o secvență de comenzi de forma următoare:

```
mysql> SHOW DATABASES;
mysql> USE test;
mysql> SHOW TABLES;
```

Pentru proiectarea și implementarea unei noi aplicații, se recomandă crearea unei noi baze de date, respectiv crearea unui nou user și atribuirea de drepturi acestuia doar pe această nouă bază de date. Spre exemplu:

```
mysql> CREATE DATABASE persoane;
mysql> CREATE USER 'ion'@'localhost' IDENTIFIED BY
'parola';
```

```
mysql> GRANT ALL ON persoane.* TO 'ion'@'localhost';
```

Se presupune serverul MySQL plasat pe același *host* cu interpretorul PHP, cu serverul Apache și deci, implicit, cu scripturile PHP care integrează rolul de user). Astfel, un singur user MySQL permitând doar o conectare de pe *localhost* este suficient pentru operare (ofierind în plus și o securitate suplimentară, doar un apel conexiune dintr-un script PHP fiind permis, respectiv orice apel extern, de la distanță (*remote*), fiind respins). Folosirea noului user implică deschiderea unei noi console client, aceasta putând fi startată printr-o comandă de rulare directă a aplicației client *mysql.exe* cu parametri (-h urmat de numele/adresa IP a host-ului, -u urmat de numele de user, -p precizând că este necesară și se va cere o parolă, fără a lăsa un spațiu separator între parametru și parolă):

```
C:\wamp\mysql\bin\mysql.exe -h localhost -u ion -p
```

După tastarea parolei adecvate, prompterul *mysql>* va apărea confirmând realizarea conexiunii la server și permitând deschiderea bazei dorite:

```
mysql> USE persoane;
```

În acest moment se poate trece la crearea tabelelor bazei de date și la o operare standard cu comenzi SQL consacrate (CREATE TABLE, INSERT, UPDATE, DELETE, SELECT, etc.).

Tabela *table1*, considerată în continuare în majoritatea exemplificărilor care vor urma (apartenind bazei de date *persoane*), a fost creată folosind comanda:

```
CREATE TABLE table1(
```

```
cnp char(13) PRIMARY KEY,
nume VARCHAR(20),
varsta INTEGER,
localitate VARCHAR(20));
```

Observații: Părăsirea consolei MySQL (și implicit închiderea conexiunii user-ului curent cu serverul) se face cu comanda *exit*. Din motive de comoditate vizând lansarea rapidă a unui ferestrelă consolă MySQL (permítând implicit conectarea doar ca user administrator), exemplele care urmează folosesc userul *root* pentru accesul la baza de date *persoane*.

EXEMPLU PROGRAMARE STRUCTURATA:

```
<?php
// Conexiune cu serverul și baza de date
$conex=mysqli_connect('localhost','root','','persoane') or die("eroare");

// sau: alt mod de verificare conexiune
if (!$conex){
    echo("Connect failed: ".mysqli_connect_error());
    exit();
}
```

```
$query=mysqli_query($conex,"select * from table1");
//numara linii afectate de SELECT
$linii= mysqli_affected_rows($conex);
echo $linii."<br>";

//extragere prima linie
$row=mysqli_fetch_row($query);
echo $row[0].".$row[1].".$row[2].".$row[3];

/* sau:
$row=mysqli_fetch_array($query);
echo $row['cnp'].'.$row['nume'].'.$row['varsta'].'.$row['localitate'];
*/

// Informatii coloana 1 (name, length)
$variab = mysqli_fetch_field_direct($query, 0);
echo $variab->name;
echo "<br>";
echo $variab->length;
echo "<br>";

// Numara coloanele
$col=mysqli_field_count($conex);
echo $col;

//Elibereaza resurse+inchidere conexiune
mysqli_free_result($query);
mysqli_close($conex);
?>
```

```
echo $row[0].".$row[1].".$row[2];

//extragere linie 2
$row=$result->fetch_array();
echo $row['cnp'].'.$row['nume'].'.$row['varsta'].'.$row['localitate'];

$result->free();
$mysqli->close();
?>
```

EXEMPLU POO:

```
<?php
$mysqli=new mysqli('localhost','root','','persoane');
$result=$mysqli->query("select * from table1");

//linii afectate (numarare)
echo $mysqli->affected_rows;

//extragere linie 1
$row=$result->fetch_row();
```

PHP

4. Dezvoltare de aplicații PHP cu baze de date MySQL

4.3. Interrogare neparametrizată a unei tabele a bazei de date

Acest prim exemplu prezintă o modalitate simplă de interrogare fără parametri a unei tabele a unei baze de date MySQL. Sunt exemplificate modurile de utilizare ale următoarelor funcții, respectiv comenzi și structuri:

- funcții legate direct de lucrul cu baze de date MySQL: `mysqli_connect()`, `mysqli_query()`, `mysqli_num_rows()`, `mysqli_num_fields()`, `mysqli_fetch_row()`;
- funcții și comenzi PHP generale: `echo`, `if()`, `while()`, `foreach()`, `die()`.

Exemplul este pur didactic, rolul sau fiind doar acela de a detalia funcționalitățile unor comenzi, funcții și structuri PHP uzuale folosite în operarea cu baze de date, controlul fluxului scriptului și a modului de proiectare și implementare al unui acces la date stocate pe un server MySQL.

Se consideră baza de date MySQL, numita *Admitere*, având tabela *Candidati*, cu urmatoarea structură:

```
CREATE DATABASE Admitere;
```

```
CREATE TABLE Candidati(
CNP char(13) PRIMARY KEY,
Nume varchar(30) NOT NULL,
Sex char(1) NOT NULL,
Stare_civila varchar(20),
Varsta integer NOT NULL,
Adresa varchar(30) NOT NULL,
Email varchar(30) NOT NULL);
```

Tabela *Candidati* va conține următoarele 2 înregistrări:

```
INSERT INTO Candidati VALUES('123', 'Ion Popescu', 'm', 'Necasatorit', 22,
'Str. Lidia, nr. 5, Timisoara', 'ion.popescu@yahoo.com');
INSERT INTO Candidati VALUES('124', 'Roxana Ionescu', 'f', 'Casatorit', 23,
'Str. Mures, nr. 7, Timisoara', 'roxana.ionescu@yahoo.com');
```

Codul sursă PHP al aplicației considerate este următorul (rezultatul apelului acesteia, folosind un browser WEB, fiind prezentat în fig. 3) :

```
<?php
//Conectare la server
//(în cazul de față, server MySQL local, user - root, parola - '', baza de date -
//Clinica_medicala)
$con=mysqli_connect("localhost","root","","Admitere") or
die ("Nu se poate conecta la serverul MySQL");

//Interrogare tabelă (comandă SQL)
$query=mysqli_query($con, "select * from Candidati");

//Calculează nr. înregistrari returnate prin interrogare
$nr=@mysqli_num_rows($query);

if($nr>0){
echo "<center>";
echo "S-au gasit " . $nr . " înregistrari";
echo"</center>";

//Creează capul de tabel (se face o afișare tabelată)
echo "<table align='center' border='1'>";
echo "<tr bgcolor='silver'>";
echo "<th>CNP</th>";
echo "<th>Nume</th>";
echo "<th>Sex</th>";
echo "<th>Stare Civila</th>";
echo "<th>Varsta</th>";
echo "<th>Adresa</th>";
echo "<th>Email</th>";
echo "</tr>";

//Ciclează după nr. înregistrări/linii găsite (realizând o condiție logică și o
//atribuire prin returnarea elementelor unei linii/înregistrări în variabila
//șir (array) $row)
while ($row = mysqli_fetch_row($query)){
//Variabila $row este un șir (array) conținând succesiv, la un moment dat,
//elementele unei înnregistrări (row[0] va conține elemente din coloana 1 a liniei
//curente, etc)

//Creează o linie nouă în tabel
echo" <tr>";
//Ciclează după elementele unei înnregistrări/linii
```

```

foreach ($row as $value) {
//Pune într-o celulă din tabel elementul unei înregistrări (valoarea dintr-un
//camp al înregistrării)
//Creează o coloană nouă în linia tabelului
echo "<td>$value</td>";
}//inchide buclă foreach
echo "</tr>";
}//inchide buclă while
}//inchide if

else
die ("Nu gasesc nici o înregistrare ...");
//Comanda die încide programul și toate conexiunile (ieșire forțată)
//Comanda următoare se va executa numai în caz de căutare reușită
mysql_close($con);
?>

```

S-au gasit 2 înregistrări						
CNP	Nume	Sex	Stare Civilă	Varsta	Adresa	Email
123	Ion Popescu	m	Necasatorit	22	Str. Lidia, nr. 5, Timisoara	ion.popescu@yahoo.com
124	Roxana Ionescu	f	Casatorit	23	Str. Mures, nr. 7, Timisoara	roxana.ionescu@yahoo.com

Fig. 3. Afisare date interogare

Funcția **mysqli_connect** realizează conectarea la serverul MySQL și selectarea bazei de date cu care se va opera. Această funcție are patru parametri de baza:

- Primul parametru reprezintă numele server-ului MySQL sau adresa de IP a computerului pe care rezidă serverul. În cazul nostru, ne vom conecta pe local, indicând fie numele de server, localhost, fie adresa de IP, 127.0.0.1.
- Cel de-al doilea parametru reprezintă numele utilizatorului. În exemplul nostru, acesta este **root** (user-ul implicit MySQL).
- Al treilea parametru reprezintă parola utilizatorului, iar în caz ca aceasta lipsește, se folosește sirul vid pe post de parolă pentru conectarea la server
- Al patrulea parametru reprezinta numele bazei de date care se dorește a fi selectată.

Funcția **mysqli_connect** returnează un obiect care reprezintă conexiunea la serverul MySQL, în cazul în care conexiunea a reușit, respectiv valoarea logică FALSE în cazul în care conexiunea a esuat. Funcția este absolut necesară a fi folosită înaintea oricărei alte funcții operând cu MySQL, asigurând practic realizarea conexiunii/legăturii cu serverul de baze de date.

Funcția **mysqli_query** este cea prin care se asigură interacțiunea efectivă cu elementele (datele) bazei de date, permitând operațiile SQL uzuale (interrogare, adăugare, ștergere, modificare etc). Funcția are doi parametri:

- Primul parametru este reprezentat de identificatorul de conexiune, returnat de funcția **mysqli_connect**;
- Al doilea parametru este reprezentat de stringul aferent comenzii SQL.

In exemplul anterior, s-a facut o interogare pe întreaga tabelă, fără a se impune o condiție de căutare. La fel de bine, căutarea în tabelă se putea face după o valoare a unui camp, bine precizată. De exemplu, pentru a afișa toate persoanele din tabelă care au numele „Popescu Ion”, se putea folosi comanda: „select * from Pacienti where nume='Popescu Ion'”. Există însă situații în care nu se cunoaște exact valoarea cheii de căutare, în acest caz, o alternativă constituind-o folosirea operatorul **like**. Acesta, împreună cu **wildcard-ul %** permit realizarea unei interogări după o mască aproximativă (ceva ce seamănă cu cheia de căutare). Spre exemplu, dacă se dorește afișarea tuturor persoanelor a căror nume conține cuvântul *Popescu*, se poate folosi comanda "select * from Pacienti where nume like '%Popescu%'". În cazul în care se cere afișarea tuturor persoanelor a căror vârstă se termină cu cifra 5, comanda select va arăta în felul următor: "select * from Pacienti where varsta like "%5%".

Observație: Evident, comanda SELECT poate fi și o interogare pe mai multe tabele (joinuri de tabele), spre exemplu: select tabela1.col11, tabela1.col.12, tabela2.col21, tabela2.col22 from tabela1, tabela2 where tabela1.col.12 = tabela2.col21.

Funcția **mysqli_num_rows** returnează numărul de înregistrări rezultat în urma unei interogări SQL SELECT, executate cu ajutorul funcției **mysqli_query**. Dacă **mysqli_query** conține o comandă SQL de forma **SELECT * from Pacienti**, ca și în exemplul anterior, funcția **mysqli_num_rows** va returna numărul total de înregistrări din tabelă referita.

Bucla **while(\$row=mysqli_fetch_row(\$query))**, utilizând funcția **mysqli_fetch_row**, poate fi interpretată astfel: „atât timp cât este găsită o linie (conținutul fiecărei linii găsite fiind returnat succesiv de funcția **mysqli_fetch_row** în variabila **\$row**) execută...”.

Funcția **mysqli_fetch_row** va ‘umple’ succesiv un sir (\$row) cu datele unei linii returnate de interogare. Aceste date sunt prelucrate succesiv (de regulă afișate într-un tabel pe ecran), începând cu primul articol/linie, apoi se trece la al doilea (conținutul sirului \$row fiind astfel suprascris succesiv) și aşa mai departe. Practic, bucla **while()** ciclează după numărul de articole/linii găsite și returnează în urma interogării.

Dacă numărul de linii returnate de interogare este constant, cunoscut aprioric, funcția de extragere a datelor **mysqli_fetch_row** poate fi referită explicit de către ori este nevoie, fără a se mai face apel la o structură de ciclare, gen **while**. Este cazul unei căutări după o cheie primară, numărul de linii găsite putând fi 1 sau 0. În acest caz, comanda de extragere trebuie referită o singură dată, returnând fie un array aferent linie găsite, fie FALSE dacă nu a găsit nici o linie.

După cum s-a mai precizat, PHP-ul oferă o multitudine de funcții alternative pentru extragerea datelor aferente liniilor returnate de interogare. În exemplificarea curentă s-a folosit **mysqli_fetch_row**, dar se puteau folosi (în cadrul condiției buclei **while**) și următoarele variante alternative:

- **\$row = mysqli_fetch_array(\$query)**, elementele array-ului \$row putând fi accesate fie pe baza indexului coloanelor referite prin interogare: \$row[0], \$row[1], \$row[2] etc., fie pe baza numelui coloanelor referite prin interogare: \$row['CNP'], \$row['nume'], \$row['sex'] etc.

- **\$row = mysqli_fetch_assoc(\$query)**, elementele *array*-ului \$row putând fi accesate pe baza numelui coloanelor referite prin interogare: \$row['CNP'], \$row['nume'], \$row['sex'] etc.
- **\$row = mysqli_fetch_object(\$query)**, elementele șirului returnat putând fi referite ca: \$row->CNP, \$row->nume, \$row->sex etc..

Linia **echo "<table align='center' border='1'>"**; trimită spre ecran un șir HTML pentru crearea unui tabel în care vor fi afișate datele găsite. Deci tag-ul <table...> definește un tabel de afișare, tag-ul <tr> se referă la construcția unei linii din tabel, <th> la o linie-header (cap de tabel), <td> la o coloană, toate fiind elemente HTML care permit construcția unui format tabelat de afișare pe ecran.

Observație: Comanda **print** are un efect similar cu al comenzi **echo** (afișare neformatată a unui *string*). Pentru afișarea unor *string*-uri care includ variabile, se poate folosi și o comandă de afișare formatată **printf** (dispunând de câțiva parametri speciali pentru formatarea conținutului variabilelor integrate în *string*-ul de afișat: %s, %d, %f). Un scurt exemplu de utilizare al acesteia este prezentat în continuare:

```
$var="string";
printf("Afisare: %s", $var);           //Afisare: string
$var=10.1234;
printf("Afisare intreg: %d", $var);    //Afisare intreg: 10
printf("Afisare float: %f", $var);     //Afisare float: 10.123400
printf('Afisare float: %.2f ', $var);   //Afisare float: 10.12
```

Comanda **foreach (\$row as \$value)** este practic o buclă internă buclei **while()**, ciclând în cadrul fiecărui articol/linie după numărul de elemente ale articolului/liniei. Pentru fiecare articol succesiv găsit (în ciclul **while()**), comanda **foreach()** ca și buclă, extrage succesiv din șirul \$row către un element și-l trimită spre variabila \$value (care la un moment dat conține o valoare a unui articol corespunzătoare unui anumit câmp, deci practic valoarea unei celule).

Afișarea pe ecran (în celulele tabelului de afișare construit) a conținutului variabilei \$value (**echo "<td>\$value</td>"**), înseamnă practic afișarea element cu element a conținutului fiecărui articol găsit, în celule succesiv create.

Comanda **die** (executată pe ramura **else** a verificării **if()**) închide automat conexiunea și încheie forțat programul (nici o altă comandă care ar mai urma în script nu se mai execută).

Funcția **mysqli_close** închide conexiunea cu serverul MySQL (primind ca parametru un identificator de acces la o conexiune spre un server MySQL). În exemplul de față, această comandă se execută doar dacă comanda **die()** de pe latura **else** nu s-a executat în prealabil.

În aplicația anterior prezentată, capul de tabel pentru afișarea informațiilor din tabela bazei de date s-a realizat în mod manual, presupunându-se cunoscute numele coloanelor și numărul lor.

PHP-ul dispune de o funcție, numita **mysqli_fetch_field_direct** cu ajutorul careia se pot obține informații asupra campurilor unei tabele, informații care pot fi utilizate pentru

realizarea ‘automată’ a unui cap de tabel pentru afișare (coloanele lui fiind practic câmpuri ale tablei bazei de date). Aceasta funcție permite oferirea de informații asupra unui set de date obținut în urma unei interogări, returnând ca rezultat un obiect a carui proprietăți furnizează diverse informații, printre care, numele, dimensiunea și tipul campurilor din setul de date la care face referire. Funcția **mysqli_fetch_field_direct** are ca și argumente un identificator de pointare la setul de date obținut în urma interogării (\$query), respectiv un index de coloană (pornind de la zero) reprezentând poziția coloanei referite.

Funcția **mysqli_num_fields** returnează numărul de coloane (câmpuri) rezultat în urma interogării. Dacă **mysqli_query** conține o comandă SQL de forma **SELECT * ...**, practic va returna numărul total de coloane (câmpuri) ale tablei (sau juncțiunii de tabele) referite.

//Conectare la server

```
$mysqli=mysqli_connect('localhost','root','','Admitere');
```

//Interogare tabelă (comandă SQL)

```
$query=mysqli_query($mysqli,"select * from Candidati");
```

//Calculeaza numarul de coloane returnate prin interogare

```
$columns = mysqli_num_fields($query);
```

```
for ($i = 0; $i < $columns; $i++){
$variab = mysqli_fetch_field_direct($query, $i);
echo "Camp".$i."  
";
echo $variab->name."  
";
echo $variab->length."  
";
echo $variab->type."  
  
";}
```

Efectul rulării scriptului anterior, constă în afișarea, pe coloană, a numelui, lungimii și tipului fiecarui camp al tablei Pacienti.

In continuare, se prezintă un exemplu de interogare a unei tabele, în care s-a inserat generarea automată a capului de tabel HTML, cu ajutorul funcțiilor **mysqli_num_fields**, **mysqli_fetch_field_direct** și a proprietății **name** a obiectului returnat de aceasta din urma (*lab5_ex3.php*).

```
<?php
$con=mysqli_connect("localhost","root","","Admitere") or die ("Nu se poate conecta la serverul MySQL");
$query=mysqli_query($con,"select * from Candidati");
$nr=@mysqli_num_rows($query);

if($nr>0){
echo "<center>";
echo "S-au gasit " . $nr . " inregistrari";
echo"</center>";}
```

```
//Genereaza capul de tabel
echo "<table align='center' border='1'>";
echo "<tr bgcolor='silver'>";

$colums = mysqli_num_fields($query);
for ($i = 0; $i < $colums; $i++) {
$variab = mysqli_fetch_field_direct($query, $i);
echo "<th>";
echo $variab->name;
echo "</th>";
}

echo "</tr>";
while ($row = mysqli_fetch_row($query)) {
echo" <tr>";
foreach ($row as $value) {
echo "<td>$value</td>";
}
echo "</tr>";
}
}

else
die ("Nu gasesc nici o inregistrare ...");
mysqli_close($con);
?>
```

O numărare a liniilor tăbelei care îndeplinește o anumită condiție impusă, se poate face și folosind funcția SQL COUNT:

```
$query=mysql_query("select COUNT(*) from Candidati where
nume='Ion Popescu'");
$row = mysqli_fetch_row($query);
echo $row[0];
sau
$query=mysql_query("select COUNT(*) as numar from
Candidati where nume='Ion Popescu'");
$row = mysqli_fetch_array($query);
echo $row['numar'];
```

De remarcat faptul că, deși interogarea returnează o singură valoare (un scalar), funcția de extragere (*fetch*) returnează tot un șir, este adevărat cu un singur element, iar referirea lui se va face în mod adecvat referirii unui element de șir.

Rezumând, principalele funcții folosite pentru lucru cu baze de date MySQL sunt următoarele:

- **mysqli_connect** - permite conectarea la un server MySQL;
- **mysqli_query** - execută o comandă SQL;
- **mysqli_fetch_row** - returnează o linie din tabelă ca șir;
- **mysqli_num_fields** - returnează numărul de coloane(câmpuri) dintr-un set de rezultate creat ca urmare a unei comenzi SQL SELECT;
- **mysqli_num_rows** - returnează numărul de linii obținute în urma executării unei comenzi SQL SELECT;
- **mysqli_fetch_field_direct** - oferă informații asupra coloanelor dintr-un set de date rezultat în urma unei interogări;
- **mysqli_close** - închide o conexiune cu baza de date,

4.4. Interogare parametrizată a unei tabele a bazei de date

Realizarea unei interogări a unei tabele aferente unei baze de date, implică de cele mai multe ori transmiterea unor parametri după care se face căutarea.

a) Interogare cu căutare exactă (cu toți parametrii obligatoriu de introdus)

Acet prim exemplu presupune o interogare a unei tabele a bazei de date, cu afișarea liniilor găsite, filtrarea făcându-se funcție de valorile introduse pentru un *nume* și o *varsta* (câmpuri ale tăbelei), ambele furnizate obligatoriu (ca parametri de intrare și implicit de interogare).

Un prim fișier script HTML (*ex1.html*) necesar, realizează doar o preluare a acestor parametri de la tastatură (fig.4) și o "pasare" a lor (printr-un *form* însoțit de o acțiune "POST") spre pagina propriu-zisă de interogare (scriptul PHP), plasată la adresa locală <http://localhost/ex1.php>:

```
<html>
<head>
<title>Interogare cu parametri</title>
</head>
<body>
<center>
<H1>Interogare cu parametri</H1>
</center>
<form method="POST" action="http://localhost/ex1.php">
<table border="10" align="center" bgcolor="Silver">
<td>
Nume:<input type="text" name="nume_form" size="10"><br>
Varsta:<input type="text" name="varsta_form" size="4"><br>
<input type="SUBMIT" value="Cauta" >
<input type="RESET" value="Reset" >
```

```
</td>
</table>
</form>
</body>
</html>
```

Elementul esențial al fișierului HTML anterior, este obiectul HTML `<form action="POST" ...>`, având rolul de preluare de la tastatură a unor valori pentru cei doi parametrii (`nume_form` și `varsta_form`) și pasarea lor spre scriptul PHP localizat de linia: `action=http://localhost/ex1.php`.

Întreg conținutul formularului `<form...> ...</form>` a fost plasat în celula unei tabele (având culoarea de `background` argintiu), fără a fi obligatorie această afișare tabelată (mai reușită însă din punct de vedere al designului):

```
<table border="10" align="center" bgcolor="Silver">
<td>
  .
  .
</td>
</table>
```

Totuși, nucleul esențialul a acestui fișier HTML este următorul (putându-se reduce doar la secvența următoare):

```
<form method="POST"
action="http://localhost/ex1.php">
Nume:<input type="text" name="nume_form" size="10" ><br>
Varsta:<input type="text" name="varsta_form" size="4" ><br>
<input type="SUBMIT" value="Cauta" >
<input type="RESET" value="Reset" >
</form>
```

Se pot remarcă:

- cele două tag-uri `<input type="text" ...>` care creează două casete pentru preluarea datelor (fig.4) în cele două variabile-parametru, `nume_form` și `varsta_form`;
- tag-ul `<input type="SUBMIT" ...>` care generează un buton, la apăsarea căruia valorile celor doi parametrii sunt trimise către fișierul PHP localizat prin adresa: `action="http://localhost/ex1.php"`;
- tag-ul `<input type="RESET" ...>` care resetează valorile parametrilor și anulează transmiterea lor;

Observație: Referirea într-un script a unei alte pagini WEB (HTML PHP, etc.) se poate face, fie prin adresare directă (caz în care se precizează inclusiv calea completă spre acea pagină, spre exemplu "`http://localhost/ex1.php`"),

fie prin adresare relativă (fără a mai fi necesară calea completă spre pagina referită). În acest ultim caz, dacă scriptul apelant și scriptul apelat/referit sunt în același director/locație, ajunge precizarea doar a numelui paginii apelate ("ex1.php").



Fig. 4 Ecran interogare

Codul PHP al fișierul realizând conectarea la baza de date și interogarea tabelei *Candidati* a bazei de date MySQL *Admitere* este următorul (acest fișier primește de la scriptul HTML anterior prezentat, valorile parametrilor `nume_form=valoare_tastata1`, respectiv `varsta_form= valoare_tastata2`):

```
<?php
//Conectare la server
//(în cazul de față, server MySQL local, user - root, parola - , baza de date -
//Admitere)
$con=mysqli_connect("localhost","root","","Admitere") or
die ("Nu se poate conecta la serverul MySQL");
//Preluarea cu metoda POST a parametrilor transmisi de către fișierul HTML
//spre scriptul PHP
$nume=$_POST['nume_form'];
$varsta=$_POST['varsta_form'];
//Interogare cu parametri tabelă (comandă SQL)
$query=mysqli_query($con,"select * from Candidati where
nume=' $nume ' and varsta=$varsta");
//Calculează nr. înregistrari returnate prin interogare
$nr=@mysqli_num_rows($query);
if($nr>0){
echo "<center>";
echo "S-au gasit " . $nr . " înregistrari";
echo "</center>";}
```

```

//Creează capul de tabel (se face o afișare tabelată)
//Genereaza capul de tabel
echo "<table align='center' border='1'>";
echo "<tr bgcolor='silver'>";

$colums = mysqli_num_fields($query);
for ($i = 0; $i < $colums; $i++){
$variab = mysqli_fetch_field_direct($query, $i);
echo "<th>";
echo $variab->name;
echo "</th>";
}

//Ciclează după nr. înregistrări/linii găsite (realizând o condiție logică și o
// atribuire prin returnarea elementelor unei linii/înregistrări în variabila
// șir (array) $row)
while ($row = mysqli_fetch_row($query)) {
// Variabila $row este un șir (array) conținând succesiv, la un moment dat,
// elementele unei /înregistrări (row[0]) va conține elemente din coloana 1 a liniei
// curente, etc)

//Creează o linie nouă în tabel
echo "<tr>";
//Ciclează după elementele unei înregistrări/linii
foreach ($row as $value) {
//Pune într-o celulă din tabel elementul unei înregistrări (valoarea dintr-un
// câmp al înregistrării)
// Creează o coloană nouă în linia tabelului
echo "<td>$value</td>";
//Închide buclă foreach
echo "</tr>";
//Închide buclă while
//Închide if

else
die ("Nu gasesc nici o înregistrare ...");
//Comanda die închide programul și toate conexiunile (ieșire forțată)
//Comanda următoare se va executa numai în caz de căutare reușită
mysqli_close($con);
?>

```

Față de exemplele prezentate anterior, singura deosebire majoră apare în linia:

```

$query=mysql_query("select * from Candidati where nume='$nume' and
varsta=$varsta");

```

reprezentând o interogare parametrizată (variabilele \$nume și \$varsta, reprezentând valorile preluate de la tastatură și transmise de fișierul HTML spre fișierul PHP apelat).

Rezultatul apelului acestui script cu parametri, și implicit al interogării, este prezentat în fig.5.

S-au gasit 1 înregistrari						
CNP	Nume	Sex	Stare_civila	Varsta	Adresa	Email
123	Ion Popescu	m	Necasatorit	22	Str. Lidia, nr. 5, Timisoara	ion.popescu@yahoo.com

Fig. 5 Rezultatul interogării exacte parametrizate

b) Parolarea accesului

În scriptul anterior, accesul la serverul MySQL se face prin precizarea explicită a parolei direct în cod. În unele situații, din motive de restricționare a accesului, acest lucru nu este însă de dorit. Parolarea accesului, în exemplul de față, se poate face prin transmiterea parolei ca un parametru de intrare, introdus de la tastatură odată cu ceilalți parametri (afejenți interogării - fig. 6).



Fig.6 Ecran de conectare și interogare

În acest sens, în fișierul care implementează formularul de introducere a parametrilor de intrare se adaugă codul următor (practic un *input* suplimentar) pentru preluarea de la tastatură a parolei (fig. 6):

```

Parola: <input type="password" name="parola_form"
size="15">
<br>
<h4>Date de interogare:</h4>
<br>

```

În acest caz, suplimentar în scriptul PHP propriu-zis pot apărea următoarele completări de cod (verificând reușita sau nereușita conectării la serverul MySQL):

```
<?php
//Preluarea cu metoda POST a parametrilor transmisi de către fișierul HTML
//spre scriptul PHP
$parola=$_POST['parola_form'];
$nume=$_POST['nume_form'];
$varsta=$_POST['varsta_form'];

//Conectare la server
$con=mysqli_connect("localhost","root",$parola,"Admitere");
// Returnare într-o variabilă a unui mesaj (string) de eroare,
// dacă apare o eroare la conectare
$var=mysqli_connect_error($con);
if (!$con){          // Verifică reușita conectării
    echo "<br><h3>".$var."</h3><br>";
//afisează mesaj eroare (fig. 7)
    echo "<h1><center> Parola este incorecta!</center>
</h1>";
}
else{    // continuare program

//Interogare cu parametri tabelă (comandă SQL)
$query=mysqli_query($con,"select * from Candidati where
nume='$nume' and varsta=$varsta");

//Calculează nr. înregistrari returnate prin interogare
$nr=@mysqli_num_rows($query);

if($nr>0){
echo "<center>";
    echo "S-au gasit " . $nr . " înregistrari";
echo"</center>";

//Creează capul de tabel (se face o afișare tabelată)
//Genereaza capul de tabel
echo "<table align='center' border='1'>";
echo "<tr bgcolor='silver'>";

$colums = mysqli_num_fields($query);
for ($i = 0; $i < $colums; $i++){
$variab = mysqli_fetch_field_direct($query, $i);
echo "<th>";
```

```
echo $variab->name;
echo "</th>";
}

//Ciclează după nr. înregistrări/linii găsite (realizând o condiție logică și o
//atribuire prin returnarea elementelor unei linii/inregistrări în variabila
//șir (array) $row
while ($row = mysqli_fetch_row($query)) {
// Variabila $row este un șir (array) conținând succesiv, la un moment dat,
elementele unei înregistrări (row[0] va conține elemente din coloana 1 a liniei
curente, etc)

//Creează o linie nouă în tabel
echo" <tr>";
//Ciclează după elementele unei înregistrări/linii
foreach ($row as $value) {
//Pune într-o celulă din tabel elementul unei înregistrări (valoarea dintr-un
//câmp al înregistrării)
// Creează o coloană nouă în linia tabelului
    echo "<td>$value</td>";
} //închide buclă foreach
echo "</tr>";
} //închide buclă while
} //închide if

else
    die ("Nu gasesc nici o înregistrare ...");
//Comanda die închide programul și toate conexiunile (ieșire forțată)
//Comanda următoare se va executa numai în caz de căutare reușită
mysqli_close($con);
} //închidere else
?>
```



Fig. 7. Parolă incorectă

În exemplul anterior, fiind vorba de o conexiune securizată (prinț-o parolă) la un server de baze de date MySQL, parola user-ului utilizat este stocată implicit criptat pe server (în tabela *user* a bazei de date speciale *mysql*). Dacă însă, aplicația Web în sine implică conturi de acces direct la ea (și nu la serverul de baze de date protejat implicit prin useri / parole proprii), este necesară definirea unor useri proprii aplicației, împreună cu parolele lor aferente (stocate într-o tabelă a aplicației creată special pentru acest scop). Fiind vorba de stocarea unor informații vitale pentru securizarea aplicației (parole de acces), în mod normal acestea sunt salvate criptat în baza de date. Deci, la crearea acestor conturi de acces user/parolă, anterior salvării lor în baza de date, parolele trebuie generate criptat. PHP pune la dispoziție mai multe funcții destinate criptării informației (*md5*, *hash*, *sha1*, etc.), iar pentru exemplificare în paragraful de față se prezintă un caz de utilizare al funcției *md5*. În primul rând trebuie generată valoare criptată a parolei dorite:

```
$parola='clar33';
$parola=md5($parola);
echo 'Valoarea criptata -32 caractere: '. $parola;
```

Rezultatul afișat pe ecran este:

Valoarea criptata - 32 caractere: e3e99e29de40ee1c050132e2eb28d4d6

Acest *string* de 32 de caractere reprezintă parola criptată, fiind salvat în baza de date. Presupunând parola introdusă de la tastatură prin intermediul unei casete INPUT, a unui formular POST, și transmisă prin variabila parametru *\$_POST['parola']*, verificarea corectitudinii ei se face criptând conținutul ei (introdus de la tastatură) și comparându-l cu conținutul parolei criptate memorat în baza de date. Altfel spus, nu se face o comparație a unui *string* decriptat cu alt *string* decriptat, neputându-se vorbi de decriptare în cazul algoritmului *md5*. Pașii de parcurs pentru verificarea unei parole ar fi:

- citire parola de la tastatură -> *\$_POST['citita']*
- extragere parolă criptată (din baza de date) -> *\$criptat*
- comparație conținuturi criptate:


```
if ($criptat==md5($_POST['citita'])) {...cod...}
```

c) Interrogare cu căutare aproximativă

Singura modificare din script apare în linia de specificare a interogării SQL efective (de remarcat folosirea în comanda de interogare a ghilimelelor pentru încadrarea variabilelor al căror conținut este interpretat ca un string/șir de caractere, respectiv lipsa acestor ghilimele în cazul variabilelor conținând valori numerice):

```
$query =mysqli_query($con, "SELECT * FROM Candidati where nume>'$nume' and varsta>=$varsta");
```

Rezultatele pot fi următoarele:

- dacă în caseta INPUT corespunzătoare numelui nu se introduce nici o valoare, iar în cea corespunzătoare vârstei se introduce valoarea 0, vor fi afișate toate înregistrările bazei de date;
- dacă se introduce un sir de caractere pentru *nume*, iar dacă în caseta INPUT corespunzătoare varstei se introduce valoarea 0, se vor afișa toate înregistrările pentru care numele sunt superioare alfabetic primului caracter din sirul de caractere introdus;

Un alt exemplu de interogare aproximativa este prezentat mai jos:

```
$query =mysqli_query($con, "SELECT * FROM Candidati where nume like 'I%'");
```

Aceasta interogare va determina afișarea tuturor persoanelor a căror nume începe cu litera 'I'.

d) Interrogare cu ignorarea parametrilor necompletați

În acest caz este necesară următoarea modificare în script-ul PHP, în linia de interogare efectivă (*\$query=mysqli_query(...)*), folosindu-se următoarea secvență:

```
// dacă nu s-a introdus nici un nume și nici o varsta
//(se folosește operatorul == pentru comparație)
if ($nume=='' and $varsta == '')
$query =mysqli_query($con, "SELECT * FROM Candidati");

// dacă s-a introdus doar un nume (operatorul != inseamnă diferit de...)
if ($nume!=='' and $varsta == '')
$query =mysqli_query($con, "SELECT * FROM Candidati where nume='$nume'");

// dacă s-a introdus o varsta
if ($nume=='' and $varsta != '')
$query =mysqli_query($con, "SELECT * FROM Candidati where varsta= $varsta");

/// dacă s-a introdus atât un nume cât și o varsta
if ($nume!=='' and $varsta != '')
```

```
$query =mysqli_query($con, "SELECT * FROM Candidati where
nume= '$nume' and varsta=$varsta");
```

e) Construcția dinamică a string-ului de interogare SQL

Analizând scriptul PHP precedent, se poate constata că, pentru 2 parametri de căutare, numărul de combinații ale *string*-lui implementând comanda SQL de interogare, presupunând o completare parțială a criteriilor de filtrare (unele casete neconținând date), este de $2^2 = 4$, iar prin extrapolare, pentru 3 parametri $2^3 = 8$ și așa mai departe crescând exponențial (pentru n parametri ajungând la 2^n). Structura de cod anterioară nu prezintă o alternativă viabilă pentru cazul unui număr mare de parametri (INPUT), considerați ca date de intrare după care se face interogarea bazei de date, în contextul în care doar o parte dintre ei sunt cunoscuți/folosiți (și deci, aleatoriu, o serie de casete de intrare nu vor conține date).

Soluția în acest caz o constituie construcția dinamică a unui singur *string* generalizat de interogare, valabil pentru n parametri aferenți clauzei WHERE. Tehnica propusă implică o verificare succesivă a conținutului fiecarei casete (parametru) de intrare, respectiv concatenarea sau nu, la un *string* inițial SQL, a cărei condiție suplimentare de filtrare. Astfel, *string*-ul SQL se generează interactiv, dinamic, crescând prin concatenări successive de condiții suplimentare (*substring-uri*), iar numărul de verificări ale completării cu date pentru n casete de intrare se reduce rezonabil față de soluția anterioară, scăzând de la 2^n la n .

Un exemplu concret este prezentat pentru cazul a 4 parametri de intrare și interogare (\$p1, \$p2, \$p3, \$p4), mai mult sau mai puțin cunoscuți, folosiți pentru condiționarea interogării unei baze de date TEST cu 4 coloane (c1,c2,c3,c4 - prima de tip numeric, respectiv celelalte de tip caracter). În faza de testare a scriptului se recomandă o afișare a *string*-ului de interogare construit dinamic, pentru a permite o inspectare vizuală a lui în vedere detectării unor eventuale erori de sintaxă (spații lipsă, spre exemplu). Secvența de cod implementând o posibilă construcție dinamică a *string*-ului comenzi SQL este următoarea:

```
$interogare="select * from test where ";
//verific primul parametru
if (!empty($p1))
$interogare=$interogare."c1=$p1 and ";
//verific al 2-lea parametru
if (!empty($p2))
$interogare=$interogare."c2='$p2' and ";
//verific al 3-lea parametru
if (!empty($p3))
$interogare=$interogare."c3='$p3' and ";
//verific al 4-lea parametru
if (!empty($p4))
$interogare=$interogare."c4='$p4' and ";
```

```
// adaug condiție finală pentru sintaxă validă comandă SQL
$interogare=$interogare."c1>0";
echo $interogare; //afișez string SQL pentru inspecție vizuală
```

Prima linie generează suportul de bază al comenzi SQL, la acest *string* fiind concatenate dinamic construcții suplimentare. Pentru fiecare parametru conținând date, un *substring* suplimentar este concatenat *string*-ului de bază. O problemă de finalizare a construcției *string*-ului de interogare apare în situația în care nici un parametru nu conține date, iar soluția adoptată constă în finalizarea tot timpul a *string*-ului SQL prin concatenarea unei condiții statice, permanent îndeplinite (în cazul de față, c1>0, presupunând coloana c1 cheie primară, spre exemplu). Un asemenea artificiu rezolvă și problema "lipirii" de operatorul logic AND (încheind fiecare *substring* condițional aferent unui parametru non-vid), a unei condiții necesare doar sintactic, dar fără nici o influență asupra rezultatului interogării. Iată și câteva exemple de *string*-uri de interogare generate astfel, în contextul exemplului considerat:

- toți parametri de intrare completează cu date:

```
select * from test where c1=2 and c2='ion' and c3='ing'
and c4='da' and c1>0
```

- doar primul parametru nenul:

```
select * from test where c1=2 and c1>0
```

- toți parametri de intrare nuli:

```
select * from test where c1>0
```

Evident există și alte soluții de construcție dinamică a *string*-ului de interogare, dar din punct de vedere al minimizării numărului de verificări efectuate prin secvențe IF, algoritmul propus este optim. Astfel, o soluție alternativă poate aborda problema printr-o construcție inițială a *string*ului integrând din start condiția permanentă de generare a unei construcții SQL valide sintactice, indiferent de numărul de parametri folosiți pentru interogare), codul PHP fiind următorul:

```
$interogare="select * from test where c1>0 ";
if (!empty($p1))
$interogare=$interogare." and c1=$p1 ";
if (!empty($p2))
$interogare=$interogare." and c2='$p2' ";
if (!empty($p3))
$interogare=$interogare." and c3='$p3' ";
if (!empty($p4))
$interogare=$interogare." and c4='$p4' ";
echo $interogare; //afișez string SQL pentru inspecție vizuală
```

Singurul dezavantaj al soluțiilor de genul celor anterior prezentate constă în faptul că, asupra bazei de date se aplică o cluză suplimentară de filtrare, fără nici o valoare practică (fiecare parametru este permanent îndeplinită) și având drept unic scop rezolvarea cât mai simplă a unei probleme de validitate sintactică a comenzi SQL.

generate dinamic. Dar simplitatea codului PHP necesar face să merite acest compromis.

f) Parametri speciali (ca dimensiune sau conținut)

O situație specială de interogare o constituie și cea în care, interogarea se face după o cheie mai lungă, spre exemplu, un set de cuvinte, grupate toate într-un același câmp/casetă și transmise (toate) ca un singur parametru. Un formular de tip FORM poate conține în acest caz o casetă INPUT (având atributul type="text", implicit de altfel), fie o casetă de tip "TEXTAREA". Pentru exemplificare, în cadrul formularului următor, o casetă de tip TEXTAREA permite scrierea unei informații pe 3 rânduri și 10 coloane (fiind vorba de o formatare a modului de afișare pe ecran a informației, și nu de o limitare a cantității de informație):

```
<form action="unu.php" method="GET">
Detalii: <textarea name="detalii_form" cols="10"
rows="3"></textarea>
<input type="submit" value="GO">
</form>
```

Evident că utilizarea unei casete TEXTAREA este necesară doar în situația unui volum mare de informație (mai multe linii de text, spre exemplu). Dacă nu este vorba de o astfel de situație, se poate folosi o simplă casetă INPUT, permitând scrierea pe o singură linie.

Transferul valorii unui astfel de parametru citit dintr-o casetă TEXTAREA ridică câteva probleme. Acestea se datorează faptului că, conținutul parametrului poate include spații, coduri de sfârșit de linie sau alte caractere speciale, care nu sunt permise într-un string primit ca parametru de către un script. Aceste caractere speciale trebuie convertite în coduri speciale specifice parametrizării apelului unei pagini Web.

Se consideră următorul exemplu de apel (printr-un link) al unui script intitulat "newpage.php", care primește ca parametru de intrare conținutul variabilei \$detalii, al cărei conținut a fost completat printr-un formular de genul celui anterior prezentat (conținând TEXTAREA). Se presupune că variabila \$detalii conține valoarea "Exemplu de folosire a tag-ului Div", inclusiv mai multe spații, care nu sunt permise într-un string de interogare, trimis explicit ca parametru de intrare spre un script, prin intermediul unui link. Convertirea acestor spații într-un cod special acceptat într-un astfel de apel, implică folosirea unei funcții PHP speciale, și anume urlencode(). Iată codul aferent unui mod corect de apel cu transfer al unui astfel de parametru (cod inclus în cadrul script-ului unu.php, care primește parametrul de intrare \$detalii de la formularul anterior):

```
$detalii=$_GET['detalii_form'];
<a href="newpage.php?detalii_url=<?php echo
urlencode($detalii); ?>">
O legatura parametrizata</a>
```

Se poate observa o includere de cod PHP în mijlocul unei etichete HTML.

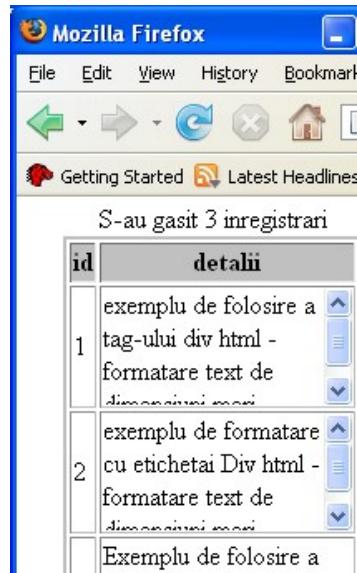
Funcția urlencode() preia caracterele speciale (spre exemplu, spațiile) dintr-un string pasat ca parametru și le convertește în coduri speciale necesare și acceptate în cadrul unui apel parametrizat al unei pagini Web. Pentru cazul unei valori particulare: \$detalii="Exemplu de folosire a tag-ului Div", această funcție convertește acest conținut într-un string de forma "Exemplu+de+folosire+a+tag-ului+Div", acceptat ca parametru valid. După apelul scriptului cu parametrul astfel codat, PHP realizează automat conversia inversă (în cadrul scriptului apelat -newpage.php-), la valoarea inițială, în momentul utilizării variabilei \$detalii.

Observație: Funcția urlencode() se folosește doar în cadrul unei pasări de parametri direct printr-un hyperlink explicit ([, evident folosind metoda GET\), nefiind necesară utilizarea ei explicită dacă transmiterea parametrilor se face prin intermediul unui formular FORM \(codarea făcându-se în acest caz automat\). Cu alte cuvinte, un formular FORM interpretează adevarat un string conținând caractere speciale sau spații și-l transmite mai departe ca parametru, în formatul adevarat \(nefiind nevoie se utilizarea explicită a funcțiilor urlencode sau urldecode\)](#)

În situația în care se dorește afișarea unui volum compact mai mare de informații (preluată, spre exemplu, dintr-un câmp de tip TEXT al unei tabele MySQL), este necesară uneori o formatare a acesteia. Figura 8 prezintă afișarea unei informații neformatate (liniile întinzându-se pe toată lățimea ecranului, iar dacă ar depăși-o, ar apare bara de scroll orizontal pentru fereastră).

id	detalii
1	exemplu de folosire a tag-ului div html - formatare text de dimensiuni
2	exemplu de formatare cu eticheta Div html - formatare text de dimensiuni

Fig. 8 Afișare informație neformatată (nowrap)



A screenshot of a Mozilla Firefox browser window. The title bar says "Mozilla Firefox". The menu bar includes "File", "Edit", "View", "History", and "Bookmark". Below the menu is a toolbar with icons for back, forward, search, and home. The main content area shows a table with the following data:

S-au gasit 3 inregistrari	
id	detalii
1	exemplu de folosire a tag-ului div html - formatare text de dimensiuni mari
2	exemplu de formatare cu eticheta Div html - formatare text de dimensiuni mari

Fig. 9 Afisare informatie formatata cu eticheta DIV

O soluție simplă de rezolvare presupune utilizarea etichetei **DIV** cu parametrii aferenți setați corespunzător. Astfel, dacă în celulele aferente coloanei *detalii* a tabelului din figura 8 se folosește o construcție de genul:

```
<td><div style='height: 100px; width: 330px; overflow: auto'>....continutul coloanei/variabila...</div></td>,
```

rezultatul formatării este prezentat în figura 9, observându-se o îmbunătățire evidentă a design-ului.

PHP

4. Dezvoltare de aplicații PHP cu baze de date MySQL

4.5. Adăugare într-o tabelă a bazei de date

Și în cazul operației de adăugare a unei noi linii într-o tabelă a unei baze de date MySQL este necesară utilizarea unui fișier HTML pentru transmiterea spre scriptul PHP propriu-zis a valorilor corespunzătoare liniei care va fi adăugată în tabelă.

Pentru exemplificare, se consideră baza de date MySQL, numita *Evidenta*, având tabela *Persoane*, cu urmatoarea structură:

```
CREATE DATABASE Evidenta;
USE Evidenta;
CREATE TABLE Persoane(
CNP char(13) PRIMARY KEY,
Nume varchar(30) NOT NULL,
Varsta integer NOT NULL,
Adresa varchar(30) NOT NULL);
```

Fisierul HTML, numit *curs4_ex1.html*, este prezentat mai jos:

```
<html>
<head>
<title>Adaugare</title>
</head>

<body>
<center><H1>Adaugare</H1></center>
<form method="POST"
action="http://localhost/curs4_ex1.php">





```

```
<td><input type="text" name="varsta"></td>
</tr>
<tr>
<td><strong>Adresa:</strong></td>
<td><input type="text" name="adresa"></td>
</tr>
<tr>
<td colspan="2" align="center">
<input type="SUBMIT" value="Adauga">
<input type="RESET" value="Reset">
</td>
</tr>
</table>
</form>
</body>
</html>
```

Formularul de adaugare poate fi vizualizat în Fig. 10.

Adaugare

CNP:	
Nume:	
Varsta:	
Adresa:	
<input type="button" value="Adauga"/> <input type="button" value="Reset"/>	

Fig. 10. Formular de adaugare

Acum acest fișier HTML permite (în cazul de față) transmiterea valorilor a patru parametri (*CNP*, *nume*, *varsta*, *adresa* - cu valori introduse de la tastatură) aferenți datelor necesare construcției unei noi linii care va fi adăugată în tabela bazei de date.

Scriptul PHP (*curs4_ex1.php*) spre care sunt transmise valorile parametrilor și care va realiza adăugarea propriu-zisă în tabela bazei de date are următorul cod:

```

<?php
$con=mysqli_connect("localhost","root","","evidenta")
or die ("Nu se poate conecta la serverul MySQL");

$CNP=$_POST['cnp'];
$nume=$_POST['nume'];
$varsta=$_POST['varsta'];
$adresa=$_POST['adresa'];

// adăugare parametrizată
$query=mysqli_query($con, "insert into persoane
values ('$CNP','$nume','$varsta','$adresa')") or die
("Inserarea nu s-a putut efectua!");

echo "<center>";
echo "Inserare reusita!!!";
echo "</center>";

// selectarea și afișarea doar a înregistrării/liniei nou adăugate
$query=mysqli_query($con, "select * from persoane where
cnp='$CNP'");

// calculează nr. de înregistrări returnate prin interogare
$nr_inreg=@mysqli_num_rows($query);

if ($nr_inreg>0){
    echo "<table border='2' align='center'>";
    // realizare automată a capului de tabel (conținând toate câmpurile)
    echo "<tr bgcolor='silver'>";
    $columns = mysqli_num_fields($query);
    for ($i = 0; $i < $columns; $i++){
        $variab = mysqli_fetch_field_direct($query, $i);
        echo "<th>";
        echo $variab->name;
        echo "</th>";
    }
    echo "</tr>";

    // afișează înregistrările găsite în urma interogării
    while($row=mysqli_fetch_row($query)){
        echo "<tr>";
        foreach ($row as $value){
            echo "<td>$value</td>";
        }
    }
}

```

```

        }
        echo "</tr>";
    }
    echo "</table>";
}
else{
    echo "<center>";
    echo "Nu s-a gasit nici o inregistrare!!!";
    echo "</center>";
}
mysqli_close($con);
?>

```

CNP	Nume	Varsta	Adresa
2830823357797	Deacu Larisa	33	Str. Lidia, nr. 57

Fig. 11. Confirmare adăugare

Rezultatul vizibil pe ecran, produs în urma execuției comenzi SQL de afișare *SELECT...* este prezentat în Fig.11. Comanda *INSERT* nu generează un rezultat vizibil pe ecran, mai precis nu va returna eventuale date care ar putea fi afișate, cel mult apărând un mesaj de eroare dacă operația de adăugare nu poate fi executată.

Observație: În locul unei comenzi *mysqli_query(\$con, "SELECT...")* compacte, se poate scrie echivalent:

```

// string de interogare memorat într-o variabilă
$sql= "select * from persoane where cnp='$CNP'";
$query=mysqli_query($con, $sql);

```

Soluția este recomandată mai ales în cazul construcției dinamice a *string-ului* de interogare, permitând o eventuală afișare și inspectare vizuală a acestuia.

4.6. Ștergere cu confirmare dintr-o tabelă a bazei de date

Ștergerea unor articole/linii dintr-o tabelă a unei baze de date (ștergere presupusă a fi cu o confirmare prealabilă) implică două operații:

- realizarea unei interogări a bazei de date (pe baza unor parametri de intrare pentru interogare) și afișarea înregistrărilor găsite, care vor fi șterse,
- respectiv, confirmarea (sau anularea) ștergerii efective, pe baza informațiilor anterior afișate în urma interogării.

Fișierul HTML (*curs4_ex2.html*) pentru realizarea unei interogări a tabelei în vederea ștergerii (interogare făcută cu un singur parametru *-nume-* introdus de la tastatură) este detaliat în listingul următor (având rezultatul prezentat în fig. 12):

```
<html>
<head>
<title>Cautare pentru stergere</title>
</head>

<body>
<center><H1>Cautare pentru stergere</H1></center>
<form method="POST"
action="http://localhost/curs4_ex2.php">





```

Cautare pentru stergere

Fig. 12. Pagină HTML pentru ștergere selectivă

Fișierul PHP (*curs4_ex2.php*) pentru interogarea prealabilă ștergerii (cu rolul: “găsește și vizualizează ce se dorește a fi șters”), este prezentat în continuare (acest fișier primind ca parametru de intrare valoarea pasată spre el de către scriptul HTML anterior):

```
<?php
$con=mysqli_connect("localhost","root","","evidenta")
or die ("Nu se poate conecta la serverul MySQL");

// transfer valoare parametru într-o variabilă internă
$nume=$_POST['nume'];

// căutare după câmpul nume a înregistrărilor care vor fi șterse
$query=mysqli_query($con, "select * from persoane where
nume='".$nume"'");
// calculează nr. de înregistrări returnate prin interogare
$nr_inreg=@mysqli_num_rows($query);

if ($nr_inreg>0){
    // creare tabel pentru afișare rezultate
    echo "<table border='2' align='center'>";
    // realizare automată a capului de tabel (conținând toate câmpurile)
    echo "<tr bgcolor='silver'>";
    $columns = mysqli_num_fields($query);
    for ($i = 0; $i < $columns; $i++) {
        $variab = mysqli_fetch_field_direct($query, $i);
        echo "<th>";
        echo $variab->name;
        echo "</th>";
    }
    echo "</tr>";
    // extragere informații și afișare
    while (list ($CNP,$nume,$varsta,$adresa) =
mysqli_fetch_row($query)) {
        print ("<tr>".
            "<td>$CNP</td>".
            "<td>$nume</td>".
            "<td>$varsta</td>".
            "<td>$adresa</td>".
            "</tr>");
    }
    echo "</table>";

// Apelarea scriptului de ștergere efectivă/anulare (cu transmitere mai departe
// a parametrilor de intrare, în cazul de față 'nume' după care se face căutarea)
?>
<center>
```

```

<br><br>
<form method="POST"
action="http://localhost/curs4_ex22.php">
<!-- "pasare", transmitere mai departe a parametrului nume ($nume) -->
<!-- sub numele 'numel' -->
<input type="hidden" name="numel"
value=<?php echo $_POST['nume']; ?>>
<input type="SUBMIT" value="Stergere efectiva">
</form>
<!-- link pt. revenire la pagina de start și anulare ștergere -->
<a href="http://localhost/curs4_ex2.html">
Renunt si revin...</a>
</center>
<?php
}
else
die("Nu gasesc nici o inregistrare ...");
mysqli_close($con);
?>

```

Pentru afisarea rezultatelor obtinute in urma interogarii, anterior s-a folosit comanda PHP `list()`, care permite asignarea de valori unei liste de variabile (asemantătoare practic unui sir), printr-o singură operaie, valorile asignate fiind cele extrase cu ajutorul functiei `mysqli_fetch_row()`. După cum se poate vedea și în Fig. 13, fișierul PHP anterior prezentat, realizează, într-o primă etapă, o simplă interogare a tabelei bazei de date, cu afișarea rezultatelor pe ecran.

CNP	Nume	Varsta	Adresa
2830823357797	Deacu Larisa	33	Str. Lidia, nr. 57

[Renunt si revin...](#)

Fig. 13. Confirmare ștergere

Ceea ce conferă posibilitatea luării deciziei de ștergere a articolelor astfel vizualizate se realizează prin partea de cod inclusă în bucla `if` (verificând numărul de linii găsite), constând practic într-un formular HTML `<form...>`. Această parte este următoarea (PHP ‘trimițând’ practic spre ecran un script HTML folosind comanda `echo`):

```

<form method="POST"
action="http://localhost/curs4_ex22.php">
<!-- "pasare", transmitere mai departe a parametrului nume ($nume) -->
<!-- sub numele 'numel' -->
<input type="hidden" name="numel"
value=<?php echo $_POST['nume']; ?>>
<input type="SUBMIT" value="Stergere efectiva">
</form>

```

Secvența de cod prezentată, transmite valoarea parametrului de interogare, memorată în variabila `$nume` (utilizat aici pentru o simplă afișare cu SELECT), spre un alt parametru (`numel`) care va fi pasat spre fișierul PHP de ștergere efectivă. Se remarcă în acest caz, tipul `type="hidden"` utilizat pentru caseta de transfer a parametrului, care face această casetă invizibilă pentru utilizator.

Fișierul pentru ștergere efectivă are în acest caz o structură extrem de simplă:

```

<?php
$con=mysqli_connect("localhost","root","","evidenta")
or die ("Nu se poate conecta la serverul MySQL");
$numel=$_POST['numel'];
//ștergere efectivă
$query =mysqli_query($con, "DELETE FROM persoane where
nume=' $numel '") or die('Comanda esuata! ');
echo "Ok! Am sters!";
mysqli_close($con);
?>

```

4.7. Modificarea unei linii dintr-o tabelă a bazei de date

Modul de modificare a unei înregistrări/linii dintr-o tabelă a unei baze de date (cu vizualizarea liniei care se modifică) presupune următoarele etape:

- o operaie de căutare parametrizată a înregistrării care se dorește a fi modificată și o afișare a ei;
- modificarea efectivă (folosind afișarea realizată).

Fișierul (`curs4_ex3.html`) pentru transmiterea parametrilor de căutare a înregistrării care se dorește a fi modificată (căutarea făcându-se după CNP – vezi Fig. 14) este următorul:

```

<html>
<head>
<title>Cautare pentru modificare</title>
</head>

```

```

<body>
<center><H1>Cautare pentru modificare</H1></center>
<form method="POST"
action="http://localhost/curs4_ex3.php">
<table border=10 align=center BGCOLOR="Silver">
<tr>
<td><strong>CNP:</strong></td>
<td><input type="text" name="cnp"></td>
</tr>
<td colspan="2" align="center">
<input type="SUBMIT" value="Cauta">
<input type="RESET" value="Reset">
</td>
</tr>
</table>
</form>
</body>
</html>

```

Cautare pentru modificare

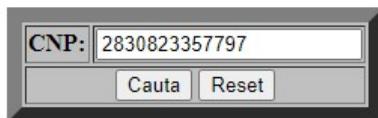


Fig. 14. Căutare pentru modificare (practic o simplă căutare)

Fișierul (*curs4_ex3.php*) pentru interogare în vederea modificării are următorul cod:

```

<?php
$con=mysqli_connect("localhost","root","","evidenta")
or die ("Nu se poate conecta la serverul MySQL");

//transfer valoare parametru într-o variabilă internă
$cNP=$_POST['cnp'];

//căutare după câmpul nume a înregistrărilor care vor fi sterse
$query=mysqli_query($con, "select * from persoane where
cnp='$_CNP'");
//calculează nr. de înregistrări returnate prin interogare
$nr_inreg=@mysqli_num_rows($query);

```

```

if ($nr_inreg>0) {
    //creare tabel pentru afişare rezultate
    echo "<table border='2' align='center'>";
    //realizare automată a capului de tabel (conținând toate câmpurile)
    echo "<tr bgcolor='silver'>";
    $columns = mysqli_num_fields($query);
    for ($i = 0; $i < $columns; $i++) {
        $variab = mysqli_fetch_field_direct($query, $i);
        echo "<th>";
        echo $variab->name;
        echo "</th>";
    }
    echo "</tr>";

    $nr_val=0; //contor indice array
    while ($row = mysqli_fetch_row($query)) {
        echo "<tr>";
        foreach ($row as $value) {
            echo "<td BGCOLOR='Yellow'> $value</td>";
        }
        //memorare într-un sir (array) a datelor din articolul găsit
        $sir[$nr_val]=$value;
        $nr_val++;
    }
    echo "</tr>";
}
echo "</table>";
?>
<!-- trasarea grafică a unei linii --&gt;
&lt;br&gt;&lt;hr&gt;
<!-- apel script pentru modificarea efectivă --&gt;

&lt;form method="POST"
action="http://localhost/curs4_ex33.php"&gt;

<!--transfer (ascuns) spre script a parametrilor de căutare --&gt;
&lt;input type="hidden" name="cnp2" value="&lt;?php echo
$sir[0];?&gt;"&gt;

<!--transfer spre script ca parametrii a datelor care pot fi modificate --&gt;
&lt;input type="text" name="cnp1"
value="&lt;?php echo $sir[0];?&gt;"&gt;
&lt;input type="text" name="nume1" value="&lt;?php echo
$sir[1];?&gt;"&gt;
</pre>

```

```

<input type="text" name="varsta1" value="<?php echo
$sir[2]; ?>">
<input type="text" name="adresa1" value="<?php echo
$sir[3]; ?>">

<input type="SUBMIT" value="Modifica" >
</form>

<!-- link de revenire și renunțare la modificare -->
<a HREF="http://localhost/curs4_ex3.html " > Renunț și
revin...</a>
<?php
}
else
    die ("Nu gasesc nici o înregistrare ...");
mysqli_close($con);
?>

```

Rezultatul fișierului script PHP anterior, este prezentat în Fig. 15. Ceea ce aduce nou acest script PHP, pe lângă o simplă interogare (căutare cu SELECT) și afișare a articolelor găsite pe ecran (tabelat), constă într-o afișare în casete editabile a datelor găsite, permitându-se astfel și o modificare a lor (vezi Fig. 15). Cum se realizează acest lucru?

CNP	Nume	Varsta	Adresa
2830823357797	Ana	3355	Str.

2830823357797	Ana	3355	Str.
---------------	-----	------	------

[Renunt și revin...](#)

Fig. 15. Ecran pentru modificare

În primul rând, în bucla `foreach(...)` încubată în bucla `while(...)`, apare o secvență de genul:

// **memorare într-un sir (array) a datelor din articolul găsit**

 \$sir[\$nr_val]=\$value;

// **incrementare contor număr elemente**

 \$nr_val++;

care permite memorarea tuturor elementelor găsite într-un sir (`array`-ul `$sir`).

Spre exemplu, în cazul de față (vezi Fig. 15) s-a găsit un articol cu 4 elemente, acestea memorându-se într-un sir având 4 elemente (cu indicei 0,1,2 și 3, deoarece contorul `$nr_val` a fost inițializat pe 0, și este incrementat după folosirea lui ca indice al unui element al sirului `$sir`).

În al doilea rând, este utilizată secvența următoare:

```

<form method="POST"
action="http://localhost/curs4_ex33.php">

<!--transfer (ascuns) spre script a parametrilor de căutare -->
<input type="hidden" name="cnp2" value="<?php echo
$sir[0]; ?>">

<!--transfer spre script ca parametrii a datelor care pot fi modificate -->
<input type="text" name="cnp1"
value="<?php echo $sir[0]; ?>">
<input type="text" name="nume1" value="<?php echo
$sir[1]; ?>">
<input type="text" name="varsta1" value="<?php echo
$sir[2]; ?>">
<input type="text" name="adresa1" value="<?php echo
$sir[3]; ?>">

<input type="SUBMIT" value="Modifica" >
</form>

```

Primul tag `<INPUT type="hidden"` permite transferul spre fișierul script PHP care realizează modificarea efectivă a parametrului inițial de interogare (memorat în primul element al `array`-ului `$sir`), acest lucru fiind necesar deoarece acest parametru poate fi ‘alterat’ în urma unei eventuale modificări permise în continuare, după cum se va vedea.

Următoarele 4 tag-uri `<INPUT type="text"` permit afișarea (în casete care permit atât operații de citire cât și de scriere) și alterarea (modificarea) acestor patru parametri, care vor substitui informația existentă în câmpurile articoului localizat (cu parametrii transmiși din primele două `INPUT`-uri invizibile utilizatorului). Cu alte cuvinte, această secvență transferă spre script-ul PHP de modificare efectivă cinci parametri:

- un parametru pentru localizarea articoului de modificat;
- alți patru parametri, pentru substituirea informației din cele patru câmpuri ale articoului deja localizat.

Fișierul PHP (`curs4_ex33.php`) care realizează modificarea propriu-zisă, utilizând comanda SQL `UPDATE`, nu îi mai revine decât o sarcină banală:

```

<?php
$cnn=mysqli_connect("localhost","root","","evidenta");
or die ("Nu se poate conecta la serverul MySQL");

$CNP1=$_POST['cnp1'];
$nume1=$_POST['nume1'];

```

```

$varsta1=$_POST['varsta1'];
$adresa1=$_POST['adresa1'];
$cNP2=$_POST['cnp2'];

// Modificare efectivă
$query =mysqli_query($con,"update persoane set
cnp=' $CNP1' ,nume=' $nume1' ,
varsta=$varsta1,adresa=' $adresa1' where cnp=' $CNP2' ") or
die("Comanda esuata!". mysqli_error($con));
// Afisare mesaj de eroare pentru date incorrect introduse (dacă se dorește)
echo "OK, am modificat!";
mysqli_close ($con);
?>

```

4.8. Aplicație exemplu cu operații SQL multiple

Se consideră baza de date MySQL *angajati*, conținând tabela *salarii* cu urmatoarele câmpuri:

- *id* integer primary key;
- *nume* varchar(14);
- *salar_brut* float;
- *impozit* float;
- *salar_net* float;

Se dorește implementarea unei aplicații care:

- folosește un prim fișier HTML (*exempluFinal.html*) prin intermediul căruia se preiau de la tastatura *id*-ul, *numele* și *salarul_brut* al unui angajat, și sunt transmise valorile acestor parametri catre un script PHP (*exempluFinal.php*). În cadrul scriptului PHP se efectuează urmatoarele operații:

- inserarea (adăugarea) în tabela *salarii* a informațiilor primite (ca linie nouă);
- realizarea unui *update* pe tabelă pentru calculul *impozitului* și a *salariului net* al angajaților cu următoarele formule:
 - *impozit*=*0.16*salar_brut*;
 - *salar_net*=*salar_brut - impozit*;
- afișarea tuturor angajaților existenți în tabelă.

Tabela MySQL (considerată a fi inclusă în baza de date *angajati*) a fost creată folosind comanda:

```

CREATE TABLE salarii(
  id integer primary key,
  nume VARCHAR (14),
  salar_brut FLOAT,
  impozit FLOAT,
  salar_net FLOAT);

```

Id:	2
Nume:	vali
Salar_brut:	300
Insert Reset	

Fig. 16. Formular HTML pentru preluarea parametrilor de la tastatură

Fișierul *exempluFinal.html* are următorul cod posibil (Fig. 16):

```

<html>
<form method="POST"
action="http://localhost/exempluFinal.php">
<table border=10 align=center BGCOLOR="Silver">
<tr>
<td>Id:</td>
<td><input type="text" name="id" size="4"></td>
</tr>
<tr>
<td>Nume:</td>
<td><input type="text" name="nume" size="10"></td>
</tr>
<tr>
<td>Salar_brut:</td>
<td><input type="text" name="salar_brut" size="4"></td>
</tr>
<tr>
<td colspan="2" align="center">
<input type="SUBMIT" value="Insert" >
<input type="RESET" value="Reset" >
</td>
</tr>
</table>
</form>
</html>

```

Fișierul *exempluFinal.php* are codul următor (vezi și Fig. 17):

```

<?php
$con=mysqli_connect("localhost","root","","angajati")
or die ("Nu se poate conecta la serverul MySQL");
echo "<br><br>";

```

```

$id=$_POST['id'];
$nume=$_POST['nume'];
$salar_brut=$_POST['salar_brut'];

//COMENZI SQL SUCCESIVE
$query=mysqli_query($con,"insert into salarii
(id,nume,salar_brut) values($id,'$nume','$salar_brut')");

$query=mysqli_query($con,"update salarii set
impozit=0.16*salar_brut,salar_net=salar_brut-impozit
where id=$id") or die("Comanda esuata!");

$query=mysqli_query($con,"select * from salarii order by
id");

$nr_inreg=@mysqli_num_rows($query);
if ($nr_inreg>0){
    echo "<center>";
    echo "S-au gasit " . $nr_inreg . " inregistrari";
    echo"</center>";
    // creare tabel pentru afisare rezultate
    echo "<table border='2' align='center'>";
    // realizare automată a capului de tabel (conținând toate câmpurile)
    echo"<tr bgcolor='silver'>";
    $columns = mysqli_num_fields($query);
    for ($i = 0; $i < $columns; $i++){
        $variab = mysqli_fetch_field_direct($query, $i);
        echo "<th>";
        echo $variab->name;
        echo "</th>";
    }
    echo "</tr>";
}
// afisare date
while($row=mysqli_fetch_row($query)){
    echo"<tr>";
    foreach ($row as $value){
        echo "<td>$value</td>";
    }
    echo"</tr>";
}
echo"</table>";
}

```

```

else{
    echo "<center>";
    echo "Nu s-a gasit nici o inregistrare!!!";
    echo "</center>";
}
mysqli_close($con);
?>

```

S-au gasit 2 inregistrari

id	nume	salar_brut	impozit	salar_net
1	andreea	500	80	420
2	vali	300	48	252

Fig. 17. Afişarea înregistrărilor găsite

Rolul acestui exemplu este de a arăta că un fișier PHP poate conține oricâte comenzi SQL sunt necesare pentru îndeplinirea funcționalităților dorite. Se poate observa de asemenea că, comenzi precum *UPDATE* (sau *DELETE*) pot fi uneori folosite pentru realizarea unor modificări/actualizări într-o tabelă, fără nici o confirmare prealabilă din partea utilizatorului.

PHP

4.9. Interogare cu extragerea informației dintr-o coloană BLOB

O situație mai deosebită de operare cu baze de date MySQL o constituie interogarea cu extragerea unei imagini dintr-un câmp de tip BLOB (*Binary Large Object*), respectiv afișarea acelei imagini. Se vor prezenta în continuare două soluții care rezolvă această problemă. Un prim exemplu de script are următorul cod PHP:

```
<?php
//functia header trebuie apelata inainte ca orice alta
//iesire sa fie trimisa spre browser
header("Content-type: image/jpg");
$con=mysqli_connect("localhost", "root", "", "bazal") or
die ("Nu gasesc serverul MySQL");
//print ("Connected successfully");
// o linie de genul celei anterioare este INTERZISA
// NU SE REALIZEAZA NICIO ALTĂ IEŞIRE DE ALT TIP PE ECRAN
$query="select imagine from tabel1 where id=1";
$result = mysqli_query($con, $query);
// Extragere rezultat (binar)
$row=mysqli_fetch_row($result);
// Trecere de la tip sir la tip variabila
$var1=$row[0];
print($var1); // sau direct, print ($row[0]);
?>
```

Esențială în acest caz este setarea facuta cu ajutorul functiei header - **header("Content-type: image/jpg")** pentru afișarea pe ecran a unei imagini *JPG*. Aceasta înseamnă că orice ieșire spre browser, va fi interpretată ca și conținut al unei imagini *jpg*, și afișată corespunzător. După o astfel de setare, realizată la începutul scriptului, nu este permisă nicio altă ieșire (de un alt tip) spre afișare in browser. Exemplul prezentat presupune existența unui câmp ‘imagine’ de tip *BLOB* (sau *LONGBLOB*), într-o tabelă referită în program.

Afișarea imaginii extrase din baza de date, se face printr-o simplă directare spre browser a conținutului variabilei (*print*), în care este salvată (în format *binary*) imaginea propriu-zisă.

În cazul (cel mai probabil) al unei interogări cu parametri, linia de interogare se modifică astfel:

```
$query="select imagine from tabel1 where id=$id";
unde, $id este parametrul după care se face interogarea, iar id este un câmp identificator din tabelă.
```

În cazul de față, valoarea parametrului *\$id* se transmite, fie printr-un formular FORM, fie direct din linia de comandă a browser-ului, printr-un apel cu parametru, de forma (spre exemplu, pentru o cale dată și un nume al fișierului script PHP *imag1.php*):

```
http://localhost/imag1.php?id=1;
```

Observație: Pentru a putea extrage date dintr-o coloană de tip BLOB, aceasta trebuie “umplută” anterior cu date binare. În MySQL inserarea de date BLOB într-o linie nouă se poate face fie direct printr-o comandă INSERT implicând utilizarea funcției speciale MySQL *LOAD_FILE*, fie printr-o comandă inițială INSERT care crează linia nouă, eventual cu date pentru celelalte coloane (mai puțin cea de tip BLOB), urmată de o comandă UPDATE setând strict coloana BLOB (folosind de asemenea funcția specială MySQL *LOAD_FILE*). Pentru cazul de față, cele două cazuri rezolvând problema adăugării unei linii noi conținând informație BLOB (o imagine) în coloana *imag* a unei tabele *tabel1* sunt punctate în continuare:

```
INSERT into tabel1 values(1, LOAD_FILE
("c:\\poza.jpg"));
sau
INSERT into tabel1 (id) values(1);
UPDATE tabel1 SET imagine=LOAD_FILE("c:\\poza.jpg")
where id=1;
```

Un al doilea exemplu script PHP, folosește un fișier imagine intermediu (*temp1.jpg*) în care este extras conținutul binar al câmpului BLOB:

```
<?php
$con=mysqli_connect("localhost", "root", "", "bazal") or
die ("Nu găsesc serverul MySQL");
$query="select imagine from tabel1 where id=2";
$result = mysqli_query($con,$query);
// Verifică dacă găsește o astfel de înregistrare
// mysqli_num_rows($result) – returnează nr. de înregistrări găsite
if (mysqli_num_rows($result)>0)
{
$row=mysqli_fetch_row($result);
$var1=$row[0];
```

```

// Deschide un fișier pentru scriere în el
$fp=fopen("temp1.jpg", "wb");
// Scrie în fișier
fwrite($fp,$var1);
fclose($fp);
// Afisează conținutul fișierului ca imagine
echo '<image src="temp1.jpg">';
}
// Dacă nu găsește nimic, șterge fișierul
else{
unlink("TEMP1.JPG"); // sau delete("TEMP1.JPG");
}
?>

```

În acest exemplu, se verifică totodată existența unei înregistrări care corespunde interogării realizate (structura *if*). Dacă se găsește o astfel de înregistrare, se salvează conținutul câmpului *BLOB* într-un fișier (cu extensia *jpg*), și se afișează practic imaginea stocată de acest fișier. Pentru o nouă interogare reușită, acest fișier va fi suprascris. În caz de interogare nereușită, fișierul intermediar va fi șters (pentru a nu se afișa ultima imagine găsită).

Verificarea găsirii unei înregistrări care corespunde condiției de interogare, este asigurată de funcția: *mysqli_num_rows()*. Această funcție returnează numărul de linii găsite de o comandă SQL SELECT.

Observație: Asupra lucrului cu fișiere în PHP se va reveni într-un paragraf ulterior.

Numărul funcțiilor PHP pentru lucrul cu baze de date MySQL este mult mai mare. Prin exemplele prezentate și prin funcțiile specifice exemplificate, s-a dorit doar o simplă introducere în problematica lucrului cu astfel de baze de date. Pentru mai multe detalii, resursele bibliografice în domeniu sunt mai mult decât suficiente, în special cele oferite pe Internet.

Headerele

Headerele sunt elemente prin care browser-ul și serverul web comunică în fundal pentru a afisa o pagina web în bune condiții. Există 2 tipuri de headere: cele emise de browser (header de request) și cele emise de server (header de răspuns).

Request headers

De fiecare dată cand un utilizator accesează o pagina web, browserul trimite către server cantități mici de date, sub forma request headers. Aceste antete cuprind detalii despre pagina care a fost solicitată, modul de transfer a ei, precum și informații despre capabilitățile browser-ului. Headerele sunt trimise de browser în mod implicit, utilizatorul nu trebuie să facă ceva anume pentru asta. De asemenea, headerele nu pot fi (usor) modificate înainte de a fi trimise.

Response headers

In răspuns la request headers primite de la browser, serverele web trimit înapoi 2 tipuri de informație:

- header de răspuns (response headers)
- conținutul efectiv al paginii solicitate (conținut ce poate fi construit/modificat prin intermediul PHP)

Headerele de răspuns contin informații despre pagina solicitată (cum ar fi: dacă există sau nu, dimensiunea ei, tipul de conținut, etc).

Headerele reprezintă prima parte a informației trimise de către server spre client, fiind separate de conținut prin 2 linii goale. Din acest răspuns, browserul "stie" să interpreteze headerele și să afișeze în pagina doar conținutul, astfel că antetele nu vor fi niciodată vizibile utilizatorului final.

La fel ca și în cazul antetelor de cerere, cele de răspuns sunt trimise în mod automat de către server, înainte de a trimite conținut și nu este nevoie ca programatorul să intervina. Există însă situații cand trebuie să anumite antete să fie modificate. Acest lucru este posibil cu ajutorul limbajului PHP.

PHP oferă posibilitatea modificării headerelor de răspuns prin intermediul funcției *header*.

```

// redirectionare - "spune" browserului sa "mearga" la alta adresa
header('Location: http://www.punctsvirgula.ro/');

// specificarea tipului paginii ce se va afisa - în urma acestor apeluri, browser-
// ul va aștepta alte tipuri de conținut decât pagini HTML
// browserul va aștepta să fie trimis un PDF
header('Content-type: application/pdf');
// browserul va aștepta să fie trimis un fisier CSS
header('Content-type: text/css');

```

4.10. Apelul unei proceduri stocate

Toate exemplele de operare cu MySQL din paragrafele anterioare au folosit ca mijloc de comunicare cu serverul de baze de date comenzi SQL standard (SELECT, INSERT, UPDATE, DELETE), pasate succesiv ca *string-uri* de caractere spre acesta și executate în ordinea sosirii lor pe server. Eșecul uneia dintre comenzi (din diverse motive, spre exemplu încălcarea unei constrângeri), nu bloca execuția celorlalte apeluri SQL (care puteau reuși) astfel încât, la finalul execuției scriptului era posibil ca datele actualizate în baza de date să nu-și mai păstreze consistența și valabilitatea logică. Dacă se pune problema unei operații tranzacționale - ori se execută întregul set de comenzi SQL implicate în tranzacție, ori, în caz de nereușită a uneia, efectul și al tuturor celorlalte este anulat – o soluție

elegantă o constituie apelul unei proceduri stocate. O procedură stocată poate conține oricâte comenzi SQL, justificându-se crearea ei în cazul în care multiple operații distincte de actualizare (modificare/adăugare/ștergere) a datelor bazei de date sunt necesare și se dorește fie execuția unitară a tuturor comenziilor, fie a nici uneia. Pentru simple interogări SELECT, care nu modifică date din tabele, nu sunt necesare apeluri de proceduri stocate, utilizarea unor comenzi clasice SELECT (chiar succesive) fiind soluția normală. Codul proceduri stocate (comenzi SQL) este evident salvat pe serverul de baze de date.

În continuare este prezentat modul în care din PHP poate fi realizat un apel de procedură stocată MySQL, aceasta asigurând la finalizarea execuției script-ului PHP caracterul tranzacțional al execuției unor seturi de comenzi SQL de actualizare a datelor din baza de date. Ca și alte sisteme de gestiune a bazelor de date, MySQL oferă suport pentru procedurile stocate, evident cu câteva aspecte specifice lui.

Pentru exemplificare, fie o tabelă *Tabela1*, având 3 coloane de tip *INT* (*c1*, *c2*, *c3*). Se consideră o procedură stocată (numita *Procedural*), care adaugă parametrizat o linie nouă în tabelă (completând cu date doar coloanele *c1* și *c2*), apoi pentru linia nouă introdusă calculează o valoare pentru coloana *c3*. La crearea procedurii stocate, deoarece în MySQL caracterul ; (punct și virgulă) semnifică încheierea unei linii de cod (delimitator de linie), un alt caracter sau secvență de caractere trebuie definită și utilizată pentru marcarea sfârșitului definiției procedurii stocate (urmând ca apoi să se revină la caracterul consacrat ;). Corpul procedurii astfel create (inclusiv comanda de ștergere prealabilă a ei dacă deja există, respectiv definițiile delimitatorilor necesari // și ;) este următorul:

```
Drop procedure Procedural;
Delimiter //
Create procedure Procedural (param1 int, param2 int)
Begin
Insert into Tabela1 (c1,c2) values(param1, param2);
Update Tabela1 set c3=c1+c2 where c1=param1;
End;
//
Delimiter ;
```

Un apel al procedurii stocate (din cadrul clientului textual *MySQL console*), urmat de o afișare a tabelei pentru a verifica efectul ei, este prezentat în continuare:

```
Call Procedural(101,101);
Select * from Tabela1;
```

Pentru apelul din PHP al procedurii stocate (scriptul Web jucând rolul clientul apelant) se poate folosi secvența următoare de cod (presupunând o conexiune validă):

```
$var=100;
mysqli_query($con, "call Procedural($var,101)") or
die("Procedura esuata!");
echo ("Procedura executata!");
```

Observatie: O soluție simplă și comoda pentru a nu mai scrie explicit și repetat pentru fiecare pagină script a unei aplicații Web a setului de comenzi de conectare la serverul de baze de date, respectiv selecție a bazei de date, o constituie crearea unui script distinct de conectare, respectiv utilizarea comenzi PHP *include()* referind acest script de conectare. Fie scriptul de conectare *conectare.php*:

```
<?php
$server='localhost';
$user='root';
$passw='';
$db='baza';
$con=mysqli_connect      ("$server",      "$user",      "$passw",
"$db") or die ("Error connecting to mysql");
?>
```

În fiecare script al aplicației (site-ului) Web care necesită o conectare la serverul MySQL, înainte de realizarea unor operații SQL efective de interogare, se inserează linia următoare (asigurând conectarea la serverul de baze de date):

```
include("conectare.php");
```

4.11. Prevenirea problemei SQL injection

Una dintre problemele de securitate ale aplicațiilor Web cu baze de date o constituie aşa numita problemă a injectării SQL (*SQL Injection*). Tehnica *SQL injection* constă într-o integrare (interactivă, prin datele de intrare) a unor porțiuni suplimentare de cod SQL care permit o atașare la interogările SQL a unor condiții permanente îndeplinite, anulând astfel filtrele condiționale de securizare. [20] Astfel, anumite combinații de condiții atașate string-ului SQL de interogare pot să conducă la o nouă comandă SQL validă, care anulează toate restricțiile vizând accesul la date, sau chiar conduc la crearea și trimiterea spre serverul de baze de date a unor comenzi SQL cu cert potențial distructiv. Ideea de bază a acestei tehnici constă în faptul că, în cadrul codului script, pe baza datelor de intrare (*INPUT*), pot fi create string-uri de comenzi SQL cu rol nedorit, malicios, pasate spre serverul de baze de date și executate necondiționat de acesta. Aceste secvențe atașate codului SQL generat prin concatenări de string-uri, anulează condițiile de filtrare ale comenzi și transformă comanda SQL într-o nouă comandă, având o cu totul altă acțiune decât cea dorită.

Din fericire, la ora actuală interprotoarele PHP de versiuni recente sunt setate (prin intermediul unor valori adecvate ale variabilelor interne de stare din fișierul de configurare *php.ini*) astfel încât să respingă din start aceste atacuri, interpretând anumite forme de *string*-uri ca secvențe malicioase și generând secvențe SQL adecvate, anulându-le astfel potențialul distructiv.

Unul dintre principiile de bază ale unei injectări SQL poate fi descris succint de următorul mod de construcție al *string*-ului de interogare:

String_interrogare='comandă_SQL_cu_conditii_logice'.' OR secvența_conditională_malicioasă', ultima secvență conditională introdusă de la tastatură fiind tot timpul adevărată (iar condiția logică OR face ca întreaga secvență conditională să fie îndeplinită).

Pentru o înțelegere a acestui tip de atac, se consideră o exemplificare concretă. Fie tabelul TEST (MySQL) cu structura și datele descrise în figura 18.

De asemenea, se consideră o aplicație WEB care permite interogarea după coloanele CNP și NUME, permisând afișarea liniei corespunzătoare datelor introduse, interesând afișarea informației aferente coloanei PIN_SECRET. Aplicația necesită un formular HTML pentru introducerea datelor de interogare (CNP și NUME), respectiv un script PHP de interogare efectivă:

- Formular introducere date interogare:

```
<form action="temp2.php" method="POST">
CNP--><input name="param1" size=13> <br>
Nume--><input name="param2" size=13> <br>
<input type="Submit" Value="Cauta!"></form>
```

- Script interogare (PHP):

```
<?php
$con=mysqli_connect('localhost','root','','bd');
$param1=$_POST['param1'];
$param2=$_POST['param2'];
$query = "SELECT * FROM test WHERE cnp='$param1' and
nume='$param2'";
echo $query;
$result=mysqli_query($con,$query);

print('<table align=center BORDER="2">');
print ("<tr>");
echo '<th BGCOLOR="Silver">ID</th>';
echo '<th BGCOLOR="Silver">CNP</th>';
echo '<th BGCOLOR="Silver">PIN_SECRET</th>';
print ("</tr>");
while ($row = mysqli_fetch_row($result))
{
echo" <tr>";
```

```
foreach ($row as $value)
{
echo "<td>$value</td>";
}
echo "</tr>";
?
?>
```

```
c:\wamp\mysql\bin\mysql.exe
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use test;
Database changed
mysql> describe test;
+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| CNP   | char(13) | NO  | PRI |          |
| NUME  | varchar(20) | NO  |      |          |
| PIN_SECRET | int(11) | NO  |      |          |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> select * from test;
+-----+-----+-----+
| CNP  | NUME | PIN_SECRET |
+-----+-----+-----+
| 12345 | ion  | 6655       |
+-----+-----+-----+
```

Fig.18 Structura și date tabela TEST

a) În scenariul considerat, ca primă exemplificare, se realizează o interogare cu parametri “normali” – presupunând introducerea de la tastatură a unor valori concrete pentru CNP, respectiv NUME. Figurile 19.a și 19.b exemplifică o interogare care nu găsește nici o linie adecvată (PIN_SECRET neregăsit), iar figurile 19.c și 19.d- o interogare reușită (PIN_SECRET afișat).



Fig.19.a Ecran interogare – parametri incorecti - protecție activă

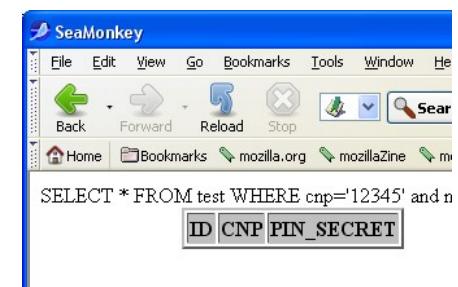


Fig.19.b Căutare nereușită

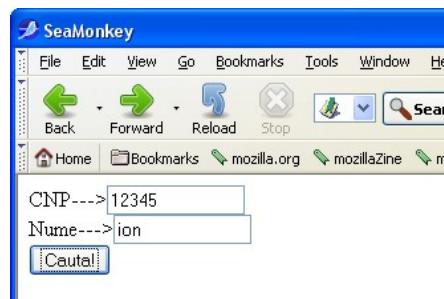


Fig.19.c Ecran interogare – parametri corecti

SELECT * FROM test WHERE cnp='12345' and n		
ID	CNP	PIN_SECRET
12345	ion	6655

Fig.19.d Căutare reușită

b) Următoarea exemplificare presupune o interogare cu parametri “anormali”, deci practic o injectare SQL. În fișierul de configurare *PHP.ini* - **protecția implicită** la instalarea PHP este setată corespunzător prin valoarea adecvată a variabilei parametru **magic_quotes_gpc** (semnificația acesteia fiind *ghilimele magice sau citate magice*):

; Magic quotes for incoming GET/POST/Cookie data.

magic_quotes_gpc = On

În figurile 20.a și 20.b este exemplificat cazul în care, deși printr-o injectare SQL este construit un *string* de interogare malicioasă (care impune o condiție permanentă îndeplinită), protecția implicită a interpretorului PHP nu oferă acces la datele tabeliei.

Secvența de injectare SQL constă în introducerea în a doua casetă (INPUT) a *string-ului* ‘OR’ 1 (neavând importanță valoarea introdusă în prima casetă), astfel încât secvența de interogare pasată spre serverul de baze de date devine:

```
SELECT * FROM test WHERE cnp='orice' and nume='\' OR '1'
```

De asemenea, în figurile 20.c și 20.d este considerată o altă formă de injectare SQL, însă protecția implicită PHP își face din nou datoria, împiedicând accesul la date. Astfel, secvența SQL injectată și deci alterată prin introducere în ultimă casetă a secvenței de cod condițional ‘OR’ ‘=’, devine:

```
SELECT * FROM test WHERE cnp='orice' and nume='\' OR
\'\='\'
```



Fig.20.a Valoare introdusa pentru parametrul al doilea: ‘ OR ‘1

SELECT * FROM test WHERE cnp='orice' and nume		
ID	CNP	PIN_SECRET
12345	ion	6655

Fig.20.b Căutare nereușită Protecția implicită împiedică „injectarea sql”

CNP--> orice Nume--> ' OR ''=		
ID	CNP	PIN_SECRET
12345	ion	6655

Fig.20.c Valoare introdusa pentru parametrul al doilea: ‘ OR ‘ ‘=’

SELECT * FROM test WHERE cnp='orice' and nume='\' C ID CNP PIN_SECRET		
ID	CNP	PIN_SECRET
12345	ion	6655

Fig.20.d Căutare nereușită! Protecția implicită împiedică „injectarea sql”

În ambele cazuri se poate observa caracterul \, precedând de multe ori caracterul ghilimea ‘ (prezența lui fiind un prim semn al unui posibil atac) și generând astfel o interogare SQL al cărei potențial malicioasă este practic anulat (noua condiție atașată nu este îndeplinită).

Același scenariu este reluat considerând dezactivată protecția internă PHP (situație implicită la versiuni mai vechi PHP):

magic_quotes_gpc = Off

Rezultatele sunt prezentate în figurile 21.a și 21.b, respectiv 21.c și 21.d. String-urile de interogare generate în urma injecției SQL sunt comenzi SQL perfect valide, având condiția de filtrare (clauza WHERE) adevărată tot timpul:

```
SELECT * FROM test WHERE cnp='orice' and nume=' ' OR '1'
```

(condiția OR ‘1’ fiind tot timpul îndeplinită)

```
SELECT * FROM test WHERE cnp='orice' and nume=' ' OR
' '=' '
```

(condiția OR ‘ ’=‘ ’ fiind tot timpul îndeplinită)

Mai mult, se observă că în urma injectării SQL, întregul conținut al tabeliei TEST este accesibil, fiind practic depășită bariera de protecție implicând cunoașterea a două informații (CNP și NUME) pentru a avea acces la informația protejată (PIN_SECRET). Practic întreaga informație (inclusiv CNP și NUME) din tabelă este accesibilă, cu consecințe imprevizibile.

CNP--> orice Nume--> ' OR ''=		
ID	CNP	PIN_SECRET
12345	ion	6655

Fig.21.a Valoare introdusa pentru parametrul al doilea: ‘ OR ‘1

SELECT * FROM test WHERE cnp='orice' and nume=' O		
ID	CNP	PIN_SECRET
12345	ion	6655

Fig.21.b Căutare reușită! Protecția dezactivată permite „injectarea sql” și afișarea ÎNTREGULUI conținut al tabeliei



Fig.21.c Valoare introdusa pentru parametrul al doilea: ‘ OR ‘ =‘



Fig.21.d Căutare reușita! Protecția dezactivată permite „injectarea sql” și afișarea ÎNTREGULUI conținut al tabeliei

Și în final, un ultim scenariu care implică o protecție explicită, preventivă, recomandată a fi folosita indiferent de setarea interpretorului PHP (deci indiferent de valoarea variabilei *magic_quotes_gpc*). Soluția constă în completarea scriptului PHP anterior cu liniile de cod următoare:

```
$param1=mysqli_real_escape_string($param1);
$param2=mysqli_real_escape_string($param2);
```

plasate înainte de construcția *string*-ului de interogare *\$query* folosind cei doi parametri.

Evident, dacă sunt mai mulți parametri de interogare, asupra fiecărui se aplică funcția *mysqli_real_escape_string*, anterior interogării (efectul acestei funcții fiind practic similar celui obținut prin setarea adecvată pe *ON* a parametrului *magic_quotes_gpc* din fișierul de configurare *php.ini*).

Paragraful de față a tratat doar problematica unui atac Web prin tehnica injectării SQL, ținta acestuia fiind baza de date aflată în spatele aplicației, codul script propriu-zis nefiind cel direct vizat. Există și atacuri având o astfel de țintă, exploatand în general breșe generate prin configurația inadecvată a serverului de Web. Totuși, în contextul dinamicii aplicațiilor Web asigurată prin preluarea informațiilor din baze de date, o alterare a conținutului tabelelor atacate poate conduce implicit și la o modificare a conținutului dinamic al site-ului.

PHP

5. Transferuri de parametri

5.1. Selectii multiple

În capitolul anterior (HTML) au fost descrise câteva etichete HTML care permit selecții multiple, același parametru putând să fie transmis de către un formular spre un script apelat cu mai multe valori simultan. În acest context, se pune problema modului de preluare a acestor valori multiple de către un script PHP. În principiu este vorba despre modul de interacționare cu PHP a etichetelor INPUT cu atributul *checkbox*, respectiv SELECT.

Eticheta INPUT, cu atributul *checkbox*, permite selectarea (bifarea) sau nu a uneia sau mai multor casete de selecție (cărora li se atașeză valori fixate, predefinite). Problema este simplă în cazul în care o singură casetă este bifată (sau există o singură casetă care poate fi selectată sau nu), variabilei cu numele precizat de parametrul *name* atribuindu-se valoarea singulară, fixată a parametrului *value*. În cazul lipsei unei selecții, variabila cu numele dat de parametrul *name* este creată, dar nu este setată, având practic o valoare vidă. Spre exemplificare, se consideră scriptul următor care utilizează, într-un *form*, o astfel de casetă de selecție INPUT *checkbox* (fig.22).

```
<?php
echo "<form name='myform' method='post'
action='a.php'>";
echo "Selecteaza:<input type='checkbox' name='check'
value='id' >";
echo "<input type='SUBMIT' value='Validez selectiile'
>";
echo "</form>"; ?>
```

Dacă se utilizează și atributul *checked*, linia 3 a scriptului fiind de forma:

```
echo "Selecteaza:<input type='checkbox' name='check'
value='id' checked >";
```

checkbox-ul este implicit selectat (bifat).

Formularul astfel construit, trimite - în caz de selecție (bifare) a casetei *checkbox* – string-ul "*check=id*" (*name=check*, *value=id*) spre un alt script (*a.php*). Scriptul următor arată modul în care poate fi verificat rezultatul acțiunii asupra unei casete INPUT-*checkbox* (selecție sau neselectie). În script apare funcția *empty* care permite o verificare practic a selectării sau nu a casetei. Funcția *empty* testează dacă variabila transmisă ca parametru nu are un conținut (returnând TRUE pentru neselectie, practic conținut vid, respectiv FALSE pentru selecție). Funcția

isset va returna TRUE doar dacă variabila are o valoare setată (selectată), respectiv FALSE în caz contrar.



Fig. 22. Casetă CHECKBOX

```
<?php
echo $check=$_POST['check'];
if (empty($check))
echo "Variabila vida <br>";
if (isset($check))
echo "Variabila este setata <br>";
?>
```

O situație specială este cea în care mai multe casete sunt selectate simultan (cazul INPUT *checkbox*), sau mai multe opțiuni sunt selectate simultan (cazul SELECT cu atributul *MULTIPLE*). Codul următor permite ilustrarea acestei situații, iar problema care se pune este legată de modul în care scriptul PHP apelat știe să preia și să interpreteze acești parametri cu valori multiple.

```
<form METHOD="GET" ACTION="apelat.php">
<SELECT MULTIPLE NAME="meserie[]">
<OPTION selected VALUE="Inginer"> Inginer
<OPTION VALUE="Sofer" > Sofer
<OPTION VALUE="Dentist"> Dentist
</SELECT>
```

```
Auto: <input type="checkbox" name="vehicol[]"
value="Dacia"> Autoturism Dacia
<input type="checkbox" name="vehicol[]" value="Ford">
Ford <br>
<input TYPE="SUBMIT" VALUE="Cauta">
</form>
```

Scriptul PHP necesită, într-un astfel de caz, trimitera parametrilor ca șiruri array (*meserie[]*, *vehicol[]*), iar preluarea și interpretarea valorilor singulare se poate face folosind construcția *foreach*, specifică extragerii

elementelor unui sir. Codul de preluare a acestor valori multiple este prezentat în continuare (construcțile *if* permitând evitarea unor mesaje de avertizare în cazul în care nicio selecție nu este realizată).

```
<?php
$sir=$_GET['meserie'];
if ($sir) {
    foreach ($sir as $element)
    {
        echo $element;
    }
}

$sir1=$_GET['vehicol'];
if ($sir1)
// sau echivalent: if(!empty($sir1)), ! fiind operator de negare logică
{
foreach ($sir1 as $element)
{
echo $element;
} } ?>
```

5.2. Comenzi pentru transferul unor parametri speciali

Comenzile *urlencode* și *urldecode* sunt utilizate pentru pasarea unor parametri *string* al căror conținut include mai multe cuvinte (și deci implicit spații) sau caractere speciale. În cadrul conținutului unui asemenea parametru, utilizând *urlencode*, spațiile vor fi înlocuite cu semnul +, specific construcției unui sir URL (*Uniform Resource Locator*). [1][20]

Spre exemplificare, fie scriptul următor cu numele *a.php* (fig.23):

```
<?php
echo $produs;
$produs1=urlencode($produs);
echo "<form name='myform' method='post'
action='b.php'>";
echo "<input type='text' name='produs' value=$produs>";
echo "<input type='text' name='produs1'
value=$produs1>";
echo "<input type='SUBMIT' value='Validez selectiile'
>";
echo "</form>";
```

Un apel al acestui script (și o apăsare a butonului *SUBMIT*), din linia de comandă a browser-ului, cu un parametru de forma unui *string* format din mai multe cuvinte: *a.php?produs=un sir ceva mai lung*

va avea efectul din figura 23. Se poate observa o trunchiere a valorii parametrului transmisă spre prima casetă de tip *input*, în timp ce a doua casetă conține sirul întreg (cu semnul + între cuvintele distincte).



Fig. 23. Transfer de parametri

O afișare a valorilor pasate și prelucrate de un script țintă *b.php* (apelat în cadrul formularului *form* din cadrul scriptului anterior):

```
<?php
$produs=$_POST[,produs'];
$produs1=$_POST[,produs1'];
echo '1 '.$produs;
echo "<p>";
echo '2 '.$produs1;
echo "<p>";
$produs1=urldecode($produs1);
echo '3 '.$produs1;
?>
```

are următoru l rezultat:

```
1 un
2 un+sir+ceva+mai+lung
3 un sir ceva mai lung
```

Se poate observa ca funcția *urldecode* permite o refacere exactă a sirului (*string*-ului) inițial și preluarea conținutului exact al parametrului astfel transmis.

5.3. Transfer UPLOAD

PHP este capabil să recepționeze fișiere asupra cărora s-a realizat o operație de upload de către un browser client. [1][20] Această caracteristică permite clientilor să realizeze *upload*-uri asupra fișierelor text sau binare.

Upload este operația inversă a download-ului. Dacă prin *download* se pot descărca (transfera) fișiere de pe server pe un host client, prin *upload* se poate realiza un transfer în sens invers: de pe un computer client pe hostul server (cu condiția ca drepturile de scriere pe server să permită acest lucru). Pentru exemplificare, un caz concret vizează stocarea într-o bază de date a unor fișiere imagine (în coloane de tip BLOB), operație necesitând obligatoriu realizarea anterioră a unui transfer *upload*.pe serverul Web al fișierelor vizate.

Un ecran pentru realizarea unui *upload* de fișier, poate fi realizat printr-un FORM special cu structura următoare (spre exemplu):

```
<FORM ENCTYPE="multipart/form-data" ACTION="upload.php"
METHOD="POST">
Send this file: <INPUT NAME="userfile" TYPE="file">
<INPUT TYPE="submit" VALUE="Send File">
</FORM>
```

De remarcat tipul special de FORM folosit, precum și tipul '*file*' folosit pentru prima caseta *INPUT*. Efectul acestui *FORM* pe ecran este prezentat în figura 24.

Următoarele nume de variabile sunt definite și se presupun a fi utilizate în legătură directă cu numele '*userfile*' al fișierului de *upload*-uit din exemplul inițiat:

- *\$userfile* – numele fișierului temporar în care fișierul de transferat este stocat pe server (același cu cel al casetei *INPUT* de tip FILE)
- *\$userfile_name* – numele original, inclusiv calea spre fișierul de pe sistemul transmițător
- *\$userfile_size* – dimensiunea fișierului de transferat în bytes
- *\$userfile_type* - tipul MIME al fișierului, dacă browserul oferă această informație (spre exemplu "image/gif").

În acest caz (legat direct de exemplul dat), conținutul variabilei sistem *\$_HTTP_POST_FILES* este următorul:

- *\$_HTTP_POST_FILES['userfile']['name']* - numele original al fișierului pe mașina client
- *\$_HTTP_POST_FILES['userfile']['type']* - tipul (MIME) al fișierului
- *\$_HTTP_POST_FILES['userfile']['size']* - dimensiunea fișierului upload în bytes
- *\$_HTTP_POST_FILES['userfile']['tmp_name']* - numele temporar al fișierului în care este transferat și stocat pe server

Fișierele sunt stocate implicit în directorul temporar implicit al serverului (dacă nu este precizat un alt director temporar al PHP-ului, prin fișierul de configurație *php.ini*).

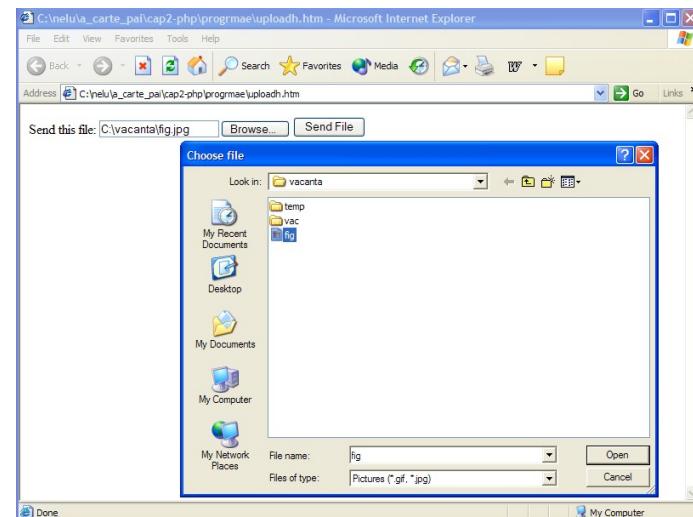


Fig. 24. Formular FORM pentru upload

Iată un exemplu concret (prezentat în două variante):

- prima variantă:

```
<?php
// verificarea operație upload validă
if(is_uploaded_file($_HTTP_POST_FILES['userfile']
['tmp_name']))
{
// transfer efectiv
copy($_HTTP_POST_FILES['userfile']['tmp_name'],
"c:\php\loaded\".$_HTTP_POST_FILES['userfile']['name'])
;
echo "OK - upload valid";
}
else
{
echo "operație eșuata";
}
?>
```

În prima variantă, funcția *is_uploaded_file()* cu sintaxa:
bool is_uploaded_file (string filename)
returnează TRUE dacă fișierul *filename* a fost transferat printr-o operație validă de *upload* via HTTP POST. Funcția COPY realizează un transfer al fișierului temporar *upload*-uit la o locație dorită. Primul parametru al comenzi *COPY* (tabloul special *\$_HTTP_POST_FILES*) localizează fișierul sursă (verificarea unei surse de *upload*

fiind realizată prin funcția `is_uploaded_file`), iar cel de-al doilea este practic destinația.

- a doua variantă:

```
<?php //upload direct
move_uploaded_file($_HTTP_POST_FILES['userfile']
['tmp_name'],
"c:\\\\php\\\\loaded\\\\". $_HTTP_POST_FILES['userfile']['name'])
?>
```

În a doua variantă, funcția `move_uploaded_file()` cu sintaxa:
`bool move_uploaded_file (string filename, string destination)`, înglobează ambele operații executate în varianta unu, făcând atât verificarea anterior menționată, cât și transferul la o locație dorită. Dacă verificarea operației `upload` nu este validă (FALSE), nici o acțiune nu este efectuată. Numele indicilor variabilei tablou (`$_HTTP_POST_FILES`) [`'userfile'`] [`'tmp_name'`] [`'name'`] sunt alese de către programator.

5.4. Variabile cu nume construite dinamic

Variabilele cu nume construite dinamic sunt utile în special în cazul în care este necesară utilizarea unui număr mare de variabile, necunoscut apriori, ca soluție alternativă la utilizarea sirurilor.

Exemplu:

```
$fluctuant=5;
for ($i=1;$i<=$fluctuant;$i++)
{
// generare nume noi variabile (var1, var2, ...,var5)
$var="var".$i;
// atribuire valori ($var1=1, $var2=2,...)
$$var=$i;
// referire variabile și afișare conținut ($$var) atribuit variabilelor
echo $$var."<p>";
}
// 4 variabile construite – afișare valori atribuite
echo $var1."<p>".$var2."<p>".$var3."<p>".$var4;
```

După cum s-a menționat, alternativa o constituie utilizarea construcțiilor de tip sir (`array`).

6. Dezvoltare de aplicații PHP cu baze de date Oracle

6.1. Interrogare fără parametri

Pentru operarea cu baze de date Oracle este necesară utilizarea bibliotecii `php OCI8.dll`, ca extensie PHP. Scriptul următor prezintă o exemplificare a utilizării cîtorva dintre funcțiile oferite de această bibliotecă.

Conecțarea la o bază de date Oracle se face utilizând funcția PHP `OCILogon` (parametri obligatorii ai acesteia fiind numele de user, parola și numele instanței Oracle, implicit **ORCL** pentru versiunile Oracle Enterprise/Professional, respectiv **XE** pentru versiunile Express). Interrogarea unei tabele a unei baze de date Oracle (mai precis, în exemplu de față, tabela *ANGAJATI* având coloanele *ID*, *NUME*, *SALAR*) se face utilizând funcțiile `OCIParse` și `OCIEexecute`, fiind apoi extrase și afișate succesiv pe ecran datele din liniile acesteia (bucla `while` cu `OCIFetch`). Apoi, în exemplul prezentat, înregistrările din tabela *angajati* sunt adăugate într-o altă tabelă *evidenta*, utilizând comanda `SQL INSERT`, împreună cu comenziile PHP `OCIParse` și `OCIEexecute` (practic, se face un transfer al informațiilor din tabela *angajati* în tabela *evidenta*). În final sunt eliberate resursele și este închisă conexiunea cu baza de date Oracle. Se poate observa că procedura de lucru este practic identică cu cea din cazurile anterioare (MySQL), doar numele funcțiilor utilizate și sintaxa lor fiind specifice bibliotecii extensie `OCI8`.

```
<?php
// Conectare la serverul Oracle (ultimul parametru se
// completează doar pentru un server Oracle la distanță)
$connection = OCILogon("student", "student", "ORCL");
// Interogare tabelă
$stmt = OCIParse($connection, "SELECT * FROM angajati");
// Executa comanda SQL
OCIEexecute($stmt);

// Generarea tabelului pentru afișarea rezultatelor obținute
echo "<table border='1' align='center'>";
echo "<tr>";
echo "<th>ID</th>";
echo "<th>NUME</th>";
echo "<th>SALAR</th>";
echo "</tr>";
// Bucla pentru extragerea rezultatelor
while(OCIFetch($stmt))
{
    $id= OCIResult($stmt, "ID");
    $nume= OCIResult($stmt, "NUME");
    $salar= OCIResult($stmt, "SALAR");
```

```

print (" <tr>".
      " <td>$id</td>".
      " <td>$nume</td>".
      " <td>$salar</td>".
      " </tr>");

// Pregătește adăugarea
$stmt1 = OCI_Parse($connection, "INSERT into evidenta
values ($id, '$nume', '$salar')");

// Execută comanda (adăugare)
OCIExecute($stmt1);

// Eliberează resursele și închide conexiunea
OCIFreeStatement($stmt);
OCIFreeStatement($stmt1);
OCILogoff($connection);

?>

```

În situația în care datele adăugate sunt și de tip dată calendaristică (de asemenea, un tip ușual), se va folosi o funcție de conversie la un format Oracle corespunzător (*TO_DATE*). [13] Spre exemplificare:

```
$stmt1 = OCI_Parse($connection, "insert into test values
('$id','$nume',$salar,TO_DATE('$data','YY-MM-DD'))");
```

Pe baza exemplului anterior, analizând funcțiile specifice (OCILogon, OCI_Parse, OCIExecute, OCIFetch, OCIResult, OCIFreeStatement, OCILogoff), se pot remarcă câteva deosebiri față de cazul operării cu MySQL. Totuși, mai există un set de funcții (destinate Oracle) a căror sintaxă, respectiv funcționalitate, este foarte apropiată de cea a setului folosit pentru MySQL, facând extrem de facilă transpunerea unui script operând cu MySQL într-un script operând cu Oracle. Se consideră următorul exemplu în acest sens:

```
<?php
$connection = OCI_connect("student", "student", "orcl");
$stmt = OCI_Parse($connection, "SELECT * FROM
angajati");
OCI_Execute($stmt);
echo "<table border='4' align='center'>";
echo "<tr>";
echo "<th>ID</th>";
echo "<th>NUME</th>";
echo "<th>SALAR</th>";
echo "</tr>";
while($row=OCI_Fetch_row($stmt))
```

```

{
  print (" <tr> ".
        " <td>$row[0]</td>".
        " <td>$row[1]</td>".
        " <td>$row[2]</td>".
        " </tr>");

OCI_Free_Statement($stmt);
OCI_close($connection);
?>
```

Se pot remarca similitudinile sintactice și funcționale ale noilor funcții utilizate, comparativ cu cele destinate MySQL-ului, să cum este prezentat comparativ și în tabelul 1:

Tabel 1

Oracle	MySQL
OCI_Connect	MySQLI_Connect
OCI_Parse + OCI_Execute	MySQLI_Query
OCI_Fetch_Row, OCI_Fetch_Array, OCI_Fetch_Assoc, OCI_Fetch_Object	MySQLI_Fetch_Row, MySQLI_Fetch_Array, MySQLI_Fetch_Assoc, MySQLI_Fetch_Object
Oci_Num_Rows	MySQLI_Affected_Rows
OCI_Close	MySQLI_Close

6.2. Operare tranzacțională

Așa cum s-a precizat pentru MySQL, și în cazul Oracle se poate face un apel dintr-un script PHP la o procedură stocată (utilă în special pentru operarea tranzacțională, cu seturi de multiple comenzi SQL destinate actualizării informației din bazele de date). Pentru o exemplificare pur didactică, se consideră următoarea procedură stocată (cu numele *joc*), operând asupra unei tabele (*testull*):

```
Create or replace procedure joc(par1 int, par2 int)
As
Begin
Insert into testull (c1,c2) values(par1, par2);
Update testull set c3=c1+c2 where c1=par1;
End;
/
```

Un script PHP care face un apel al acestei proceduri stocate poate avea codul următor (remarcându-se apelul procedurii stocate prin intermediul unui bloc PL/SQL, specific Oracle [13]):

```
<?php
$connection = OCILogon("student", "student", "orcl");
// Pregatire execuție bloc PL/SQL integrând apelul procedurii
$stmt = OCIParse($connection, "BEGIN joc(222,222);
end;");
// execuție procedură
OCIExecute($stmt) or die("Esec !")
echo "ok"; ?>
```

În cazul Oracle, operarea tranzacțională (“totul sau nimic”) asigurată de folosirea procedurilor stocate, are și o altă soluție alternativă oferită de existență unui parametru optional (neutilizat în exemplificările anterioare) în cadrul funcției *OCIExecute* [20] :

```
OCIExecute ($identifier_Oci_Parse [, $mod])
```

Parametrul optional **\$mod** poate să ia valori predefinite de genul:

-**OCI_COMMIT_ON_SUCCESS** - după fiecare comandă SQL (trimisă spre sever cu *OCIExecute*) se face COMMIT (acțiune implicită, chiar prin lipsa parametrului)

-**OCI_NO_AUTO_COMMIT** – folosit pentru modul tranzacțional de operare: la ieșirea din script, nu se salvează nici o modificare în baza de date (ROLLBACK implicit), dacă anterior nu s-a executat o comandă **OCI_commit** explicită.

În contextul anterior precizat, validarea sau anularea explicită a unei tranzacții Oracle se poate face folosind funcțiile PHP **OCI_commit** sau **OCI_rollback**.

Pentru o înțelegere a modului de operare cu acest parametru, se consideră o exemplificare concretă vizând un comportament tranzacțional și având codul următor:

```
<?php
$connection = OCILogon("student", "student", "orcl");

$stmt = OCIParse($connection, "insert into angajati
values (2, 'dan', '1000')");
OCIExecute($stmt, OCI_NO_AUTO_COMMIT) or die ('error 1');
$stmt = OCIParse($connection, "insert into angajati
values (3, 'dan', '1000')");
OCIExecute($stmt, OCI_NO_AUTO_COMMIT) or die ('error 2');

$stmt = OCIParse($connection, "insert into angajati
values ($id, '$nume', '$salar')");
OCIExecute($stmt, OCI_NO_AUTO_COMMIT) or die ('error 3');
    OCI_COMMIT($connection );
OCIFreeStatement($stmt);
OCILogoff($connection);      ?>
```

Se poate observa că scriptul încearcă execuția succesivă a trei comenzi INSERT, scopul acestuia fiind fie executarea tuturor, fie a nici uneia. Astfel:

- dacă toate cele trei comenzi se încheie cu succes (deci nu se execută nici o comandă **die** care conduce la ieșirea din script și implicit, prin folosirea parametrului **OCI_NO_AUTO_COMMIT**, la invalidarea oricărei modificări în baza de date), prin comanda de finalizare a tranzacției **OCI_COMMIT** noile date sunt salvate în baza de date (COMMIT pe toate comenzile);

- dacă una dintre comenzi eşuează (chiar dacă altele anterioare au reușit), o comandă **die** de părăsire a scriptului este executată (ne mai ajungându-se la execuția comenzi **OCI_COMMIT**) și toate eventualele modificări ale bazei de date efectuate prin apelul scriptului sunt anulate (ROLLBACK pe toate comenzile).

Utilizarea uneia sau alteia dintre soluțiile propuse rămâne la latitudinea programatorului.

6.3. Interrogare cu parametri pasăți prin auto-apelare

Exemplul următor, conținând o interogare parametrizată, se caracterizează prin aceea că, atât formularul de furnizare a parametrilor, cât și partea de preluare a acestora și realizare a interogării, sunt incluse într-un același script (auto-apelare). Practic, la un prim apel al scriptului se execută doar o parte a acestuia, constând într-un formular (*FORM* – fig. 2.25) pentru preluarea/furnizarea unor valori de parametri. Prin apăsarea butonului din formular, se realizează o auto-apelare a aceluiași script spre care este pasat și un parametru, și va fi executată o altă parte a scriptului (interrogarea efectivă). Aceste decizii sunt luate printr-o structură PHP *if-else*, ale cărei secțiuni pot include atât cod HTML ‘în clar’, cât și cod PHP încubat (fișierul având în mod obligatoriu extensia *.php*). O astfel de structură are forma următoare:

```
<?php
if ( condiție )  {
?>
```

cod HTML și/sau PHP (încubat, încadrat în delimitatori)

```
<?php
}
else   {
?>
```

cod HTML și/sau PHP (încubat, încadrat în delimitatori)

```
<?php
} ?>
```

Codul sursă complet al unui script exemplificativ (cu auto-apelare) este următorul:

```
<?php
```

// Secvența care se execută la primul apel al scriptului,

```
// situație în care NU a fost încă transmis încă nici un parametru
// de la formular

if (!isset($_GET['nume'])) { // setată și nenulă
?>
<center>
<form action=<?php $PHP_SELF ?>" method="get">
Nume: <input type="text" name="nume">
<input type="submit" value="GO" />
</form>
</center>
<?php
}

// Secvența care se execută după furnizarea unui parametru de interogare
// (autoapelare)
else{
?>

<table border="4" align="center">
<tr>
<th>ID</th>
<th>NUME</th>
<th>SALAR</th>
</tr>
<?php
$numePreluat=$_GET['nume'];
$connection = OCILogon("system", "manager100", "orcl");
print ("Connected successfully");

$stmt = OCIParse($connection, "SELECT * FROM angajati
where NUME='$numePreluat'");
OCIExecute($stmt);

while(OCIFetch($stmt)) {
    $id= OCIResult($stmt, "ID");
    $nume= OCIResult($stmt, "NUME");
    $salar= OCIResult($stmt, "SALAR");
    print ("<tr>".
        "<td>$id</td>".
        "<td>$nume</td>".
        "<td>$salar</td>".
    "</tr> ");
}
}
}
```

```
}
?>

</table>

<?php
    OCIFreeStatement($stmt);
    OCILogoff($connection);
}
?>
```

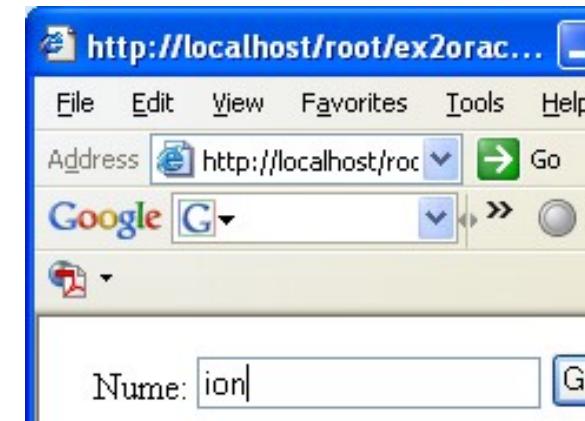


Fig. 25 Formular interogare (apel fără parametrii)

În cadrul script-ului de mai sus se putea utiliza, ca și variantă alternativă, o structură *if-else-end if* de forma următoare, oarecum similară cu cea anterioară:

```
<?php if ( condiție ): ?>
cod HTML și/sau PHP (încuibat, încadrat în delimitatori)
<?php else: ?>
cod HTML și/sau PHP (încuibat, încadrat în delimitatori)
<?php endif; ?>
```

De remarcat utilizarea în cadrul FORM-ului a unei variabile PHP speciale (*\$PHP_SELF*), care furnizează o hyper-legatură (*link*) spre scriptul care conține acest cod (practic un auto-apel cu transferul unui parametru, lucru observabil și în linia de comandă a browser-ului - fig. 25). Figura 26 prezintă rezultatul aplicației (implementată în maniera cu auto-apelare).

Observație: Sintaxa *action="<?php \$PHP_SELF ?>"* din cadrul scriptului anterior poate fi înlocuită cu sintaxa *action=""* sau *action="nume_script.php"*, unde *nume_script.php* reprezintă numele scriptului curent, care se auto-apelează.

De remarcat de asemenea, funcția `isset()` care determină dacă unei variabile i s-a atribuit o valoare (a fost setată, deci nenulă). În caz afirmativ, funcția returnează valoarea logică TRUE, altfel returnează valoarea FALSE.



Fig. 26 Rezultate interogare (auto-apel cu parametri)

Exemplul anterior arată că, atât partea de cod specifică introducerii de la tastatură a parametrilor, cât și cea de preluare și utilizare a lor în interogarea SQL efectivă, pot coexista în cadrul unui același script PHP, acesta practic putându-se auto-apela în mod repetat, reacționând de fiecare dată în mod diferit funcție de contextul auto-apelului.

Observatie: Spre deosebire de funcția `isset` (care returnează TRUE pentru o variabilă cu o valoare setată și nenulă), funcția `empty` returnează FALSE dacă variabila testată are o valoare și aceasta nu este 0 (zero). Din acest motiv, în exemplul anterior, `isset` nu putea fi substituit de `empty`. Funcția `empty` returnează TRUE pentru string-uri vide ("", "0"), 0 (zero numeric), NULL, FALSE, și vid, respectiv variabilă declarată și neinitializată (caz specific programării pe obiecte).

PHP

7. Funcții pentru operarea cu fișiere

PHP-ul dispune de un puternic set de funcții dedicat operațiilor de intrare/ieșire cu fișiere: creare/ștergere de fișiere, citire/scriere/adăugare în fișiere (âtât pentru fișiere în format text, cât și pentru format binar). Exemplul următor, implementând practic un contor al acceselor realizate de către clienți la scriptul în cauză, încearcă o exemplificare a doar câteva dintre aceste funcții, considerate ca fiind mai importante.

```
<?php
// Declarare nume pentru fișier
$filename = "counter.nr";

// Deschidere fișier doar pentru citire (avertisment, dacă nu există)
// @fopen- pentru inhibare avertisment
$fp = fopen($filename, "r") or die("Eroare");

// Determinare dimensiune fișier –doar pentru exemplificare!
// Probleme cu fișiere mai mari de 2 GB
echo filesize($fp);

// Citire din fișier (se citesc 26 bytes)- citire contor existent
$hits = fread($fp, 26);

// Incrementare contor
// cu forțare la tip întreg, necesară fără o valoare 0 inițială în fișier
$hits = (int)$hits+1;

// Închidere fișier
fclose($fp);

// Deschidere fișier doar pentru scriere
$fp = fopen($filename, "w");

// Scrie în fișier (suprascriere) sau fputs($theFile, $hits);
$a=fwrite($fp,$hits,26);
echo "<p> Sunteti vizitatorul cu numarul ";
echo $hits;
```

?>

Aplicația începe prin precizarea unui nume pentru fișierul (de tip text) care va memora contorul de accese (practic un număr întreg, incrementat după fiecare acces). În exemplul de față, numele ales pentru acest fișier este “*counter.nr*” (\$filename=“*counter.nr*;”).

În cazul în care localizarea fișierului nu este implicită în directorul curent, este necesară precizarea căii complete de localizare a fișierului. Spre exemplu:

```
$filename="http://localhost/scripts/test/counter.nr;
//sau
$filename ="c:\\text\\counter.nr";
```

Urmează apoi deschiderea pentru citire a acestui fișier, utilizând funcția *fopen()*. Fișierul trebuie să fie creat anterior, în caz contrar (adică fișierul nu există) operația terminându-se printr-un avertisment (*warning*). Deja s-a făcut apel la o primă funcție de lucru cu fișiere, cu sintaxa uzuală:

```
int fopen (string filename, string mode);
```

Funcția returnează un identificator de tip întreg. Primul parametru al funcției este un *string* reprezentând numele fișierului care va fi deschis (putând include și calea spre acesta), iar al doilea parametru reprezintă modul în care se face deschiderea fișierului, putând lua următoarele valori prezentate în tabelul următor:

Tabel 1

“r”	deschidere numai pentru citire, pointerul de fișier fiind plasat la începutul fișierului
“r+”	deschidere pentru citire și scriere, pointerul de fișier fiind plasat la începutul fișierului
“w”	deschidere numai pentru scriere, pointerul de fișier fiind plasat la începutul fișierului, iar conținutul inițial al fișierului este șters; dacă fișierul nu există –este creat ca fișier nou
“w+”	deschidere pentru citire și scriere, pointerul de fișier fiind plasat la începutul fișierului, iar conținutul inițial al fișierului este șters; dacă fișierul nu există, se încearcă crearea lui
“a”	deschidere numai pentru scriere, pointerul de fișier fiind plasat la sfârșitul fișierului (practic o operație de adăugare); dacă fișierul nu există, se încearcă crearea lui
“a+”	deschidere pentru citire și scriere, pointerul de fișier fiind plasat la sfârșitul fișierului (practic o operație de adăugare); dacă fișierul nu există, se încearcă crearea lui

Acest al doilea parametru poate include și valoarea “b”, utilă în cazul în care se realizează operații cu fișiere binare. Această valoare se utilizează în

combinăție cu cele precedente. Spre exemplu, "wb" semnifică scriere într-un fișier în format binar.

În continuarea programului se citește contorul existent într-o variabilă. Se utilizează în acest sens funcția `fread()` cu sintaxa:

```
string fread (int fp, int length)
```

unde primul parametru reprezintă identificatorul fișierului din care se citește, iar al doilea parametru -numărul de *bytes* citiți. În cazul de față, numărul 26 s-a ales pur aleator. Se observă că funcția returnează chiar *string*-ul pe care îl citește.

Se realizează apoi o incrementare a contorului (mai precis a conținutului variabilei în care s-a memorat contorul citit din fișier). De remarcat modul în care se forțează la tip *intreg* conținutul variabilei ((int)\$hits), operație care permite startarea procesului de contorizare fără ca în fișierul creat să existe o valoare inițială (zero). Dacă se dorește o forțare la tipul *float*, se utilizează o construcție de forma *(float)\$variabila*.

Urmează închiderea fișierului: `fclose($fp)` (acesta fiind deschis inițial doar pentru citire). În acest moment se dispune (într-o variabilă) de numărul de accesări ale paginii curente. Ceea ce mai rămâne de făcut este o actualizare a contorului și în fișierul în care acesta este păstrat și eventual, o afișare a acestuia pe ecran. Pentru aceasta, este necesară o redeschidere a fișierului, dar de data asta, pentru o operație de scriere (mai precis, o operație de suprascriere):

```
$fp = fopen($filename, "w");
```

Se scrie variabila contor actualizată în fișier, utilizând funcția `fwrite()` cu sintaxa următoare:

```
int fwrite (int fp, string string [,int length])
```

Primul parametru al funcției reprezintă identificatorul fișierului în care se va face scrierea, al doilea parametru – sirul de caractere scris, iar ultimul parametru (optional) numărul de *bytes* scriși (dacă acest număr este mai mare decât lungimea *string*-ului de scris, scrierea se oprește când se ajunge la finalul *string*-ului, caz implicit în lipsa parametrului optional). În locul funcției `fwrite()` se putea folosi funcția `fputs()`, cu aceeași parametrii și cu o acțiune identică. În finalul scriptului s-a făcut și o afișare pe ecran a numărului de accese la pagina curentă.

În exemplul prezentat s-a realizat o suprascriere a conținutului fișierului la fiecare accesare a scriptului. Secvența următoare prezintă o operație de adăugare a unui text într-un fișier, pe un rând nou (simbolul \n), rândurile fiind separate între ele printr-o linie liberă:

// deschidere pentru scriere (adăugare)

```
$fp = fopen($filename1, "a+");
```

// adăugare cu câte un rând gol

```
fwrite($fp, "\Nadaugat\n", 20);
```

O situație des întâlnită este cea în care sunt generate fișiere în format PDF, ca alternativă pentru stocarea și/sau tipărirea în formate predefinite a unor date.

Crearea unui fișier PDF implică utilizarea bibliotecii extensie `php_pdf.dll`, acesta trebuind a fi activată (prin setarea corespunzătoare în fișierul `php.ini`). [20] Există și alte soluții în acest sens, implicând utilizarea altor biblioteci externe, chiar mai performante, dar cea bazată pe extensia implicit oferită de PHP, fiind și cea mai la îndemână, este considerată în continuare pentru o scurtă exemplificare. Codul următor (cu comentariile corespunzătoare explicării comenziilor folosite și acțiunilor efectuate) prezintă modul în care se creează un document PDF având două pagini (soluția putând fi generalizată pentru oricâte pagini). Sunt punctate, prin comentariile aferente, câteva elemente de bază vizând setarea paginii, poziționarea informației în pagină, formatarea textului, grafică, inserarea de imagini:

```
<?php
    // creare identificator fișier
$mypdf = PDF_new();
    // creare fișier PDF și deschidere cu precizare locație
    // dacă fișierul există, este suprascris
PDF_open_file($mypdf, "c:\wamp\www\pdf\gen01.pdf");
    // creare pagina 1 cu setare dimensiuni
PDF_begin_page($mypdf, 595, 842);

    // căutare font text
$myfont= PDF_findfont($mypdf, "Times-Roman", "host", 0);
    // setare font text
PDF_setfont($mypdf, $myfont, 22);
    //scriere text cu fontul setat și la locația precizată
PDF_show_xy($mypdf, "Text exemplu în fisier.", 50, 750);
PDF_show_xy($mypdf, "Creat cu PHP.", 50, 730);

    // încărcare imagine (precizând sursa și tipul imaginii)
$myimage = PDF_load_image($mypdf, "gif",
"c:\wamp\www\pdf\logo_i.gif", "");
    // inserare imagine (precizând locația în pagină și scara dimensională)
PDF_place_image($mypdf, $myimage, 150, 600, 1.0);

    // trasare linie (setare grosime, punct start, punct final)
pdf_setlinewidth($mypdf, 10);
PDF_moveto($mypdf, 150, 550);
PDF_lineto($mypdf, 450, 550);

pdf_stroke($mypdf);           // trasare linie anterior definită
PDF_end_page($mypdf);         // sfârșit pagina 1

// generarea celei de-a doua pagini
```

```

PDF_begin_page($mypdf, 595, 842);
$myfont= PDF_findfont($mypdf, "Times-Roman", "host", 0);
PDF_setfont($mypdf, $myfont, 16);
PDF_show_xy($mypdf, "Made with the PDF libraries for
PHP.", 50, 730);
PDF_end_page($mypdf);

PDF_close($mypdf);           // închidere fișier (obligatoriu)
PDF_delete($mypdf);         // ștergere identificator fișier
echo "Fisier PDF creat!"; // mesaj confirmare în pagina Web
?>

```

Fișierul PDF creat are două pagini, fiind prezentat în fig.27.

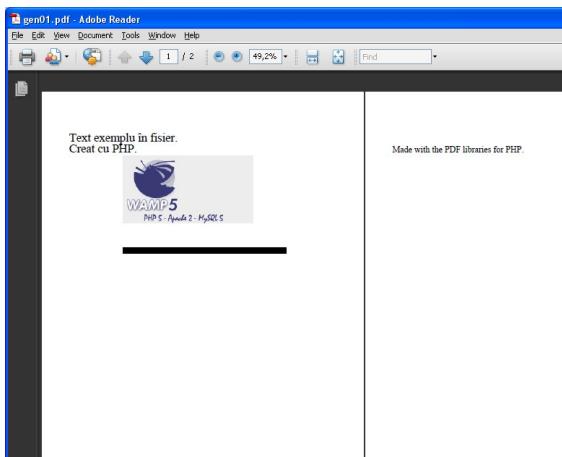


Fig.27 Conținut fișier PDF generat (cu două pagini)

8. Operare cu sesiuni (session)

Crearea unei sesiuni (numită și ‘sesiune client’) este asigurată de către limbajul PHP prin utilizarea în cadrul unui script a aşa numitului **mecanism SESSION**, startat (spre exemplu) printr-o linie de program de forma:

```
session_register("global");
```

Ce înseamnă **modul de lucru sesiune** și care este utilitatea lui? Două întrebări esențiale, la care se va încerca un răspuns în continuare. În momentul executării liniei de cod `session_register("global")`, interpreterul PHP startea o sesiune de lucru valabilă, doar pentru clientul Web curent, atât timp cât acesta nu închide browser-ul (din care a apelat respectivul script conținând această linie de cod) sau având o durată de viață în conformitate cu setările interpretorului PHP privind valabilitatea unei sesiuni. [15] Fizic, startarea unei sesiuni însemnă

crearea unui fișier sesiune salvat pe server (implicit în directorul `PHP\SESSIONDATA` sau `WAMP\TMP`), cu un nume aleator, unic, câte unul pentru fiecare sesiune pornită. Spre exemplificare, numele fișierului sesiune poate fi de forma:

```
sess_1b308801f20323d7713880f389f2489e
```

În cadrul comenzi `session_register`, parametrul cu numele global (ales pentru exemplificare) este practic o variabilă (sau sir, tablou etc.) ale cărei/cărui valori (initializate prin comenzi PHP uzuale) sunt vizibile și deci apelabile / utilizabile în orice alt script referit de browser pe parcursul sesiunii de lucru. Cu alte cuvinte, conținutul acestei variabile (de regulă sir sau tablou monodimensional sau bidimensional) devine public pentru orice script apelat pe parcursul dureatei de viață a sesiunii, cu condiția ca fiecare un astfel de script al site-ului să conțină, obligatoriu la începutul lui, linia de cod `session_register("global")`.

O sesiune de lucru se încheie în momentul închiderii browser-ului sau explicit prin cod program (comanda `session_destroy`). La oprirea browser-ului, datele din fișierul sesiune rămân stocate în acesta, iar o încheiere prin cod program a sesiunii conduce la ștergerea fișierului sesiune. De asemenea, închiderea sesiunii se poate face și automat, la expirarea durei ei de viață prevăzută prin setările interne PHP.

Fișierul `php.ini` conține variabila `session.gc_maxlifetime` având o valoare predefinită exprimând, în secunde, durata de viață a unei sesiuni (implicit `session.gc_maxlifetime=1440`).

Utilizarea sesiunilor este utilă (uneori chiar obligatorie) în mai multe situații, enumerându-se aici doar câteva:

- transferul/ pasarea unor valori de parametri prin/spre toate paginile unei aplicații Web (conținând o multitudine de pagini), aceste valori fiind vizibile și accesibile din orice pagină a site-ului (cazul implementării unui “coș virtual de cumpărături” specific aplicațiilor Web gen magazin on-line);
- asigurarea accesul securizat (parolat) la paginile unui site Web, printr-un transfer și “purtare” a parolei de acces prin fiecare pagină securizată, accesată de clientul vizitator;
- diverse situații impunând expirarea valabilității unor date (folosind mecanismul de închidere temporizată a unei sesiuni).

Mecanismul de operare cu sesiuni permite transferarea sigură, comodă și aproape implicită (necessitând doar o linie de cod, repetată în fiecare script al site-ului) a unui număr practic nelimitat de parametri între o mulțime de pagini Web (care constituie un site unitar).

Exemplu: Fie următoarele două fișiere script PHP, prezentate în tabelul 2. Primul script startea o sesiune, creându-se un fișier sesiune cu numele specificat, în care se va memora conținutul unei variabile `$var` (atribuit explicit ulterior startarii

sesiunii în cadrul scriptului). În acest moment, fișierul sesiune va conține (vezi tabelul 2.4): numele variabilei (var), tipul și lungimea conținutului (s : 4) – string 4 -, precum și conținutul efectiv: string-ul "ceva".

În scriptul *a.php* este prevăzut un hyperlink către un al doilea script *b.php*, în care, sesiunea curentă fiind în continuare activă (comanda `session_register("var")`), se asigură disponibilitatea conținutului variabilei \$var (care este și afișată pe ecran).

Tabel 2

Script <i>a.php</i>	Script <i>b.php</i>
<pre><?php session_register("var"); \$var="ceva"; echo "mai departe"; ?></pre>	<pre><?php session_register("var"); echo \$var; echo "back"; session_destroy(); ?></pre>
Nume fișier sesiune: <code>sess_1b308801f20323d7713880f389f2489e</code>	
Conținutul fișierului sesiune: <code>var s:4:"ceva";</code>	

La finalul scriptului *b.php* sesiunea este distrusă, fișierul sesiune fiind șters prin utilizarea comenții `session_destroy()`. De asemenea, în al doilea script este prevăzut un *hyperlink* de revenire spre scriptul apelant (*a.php*), care restartează aceeași sesiune (creând un fișier sesiune cu același nume) și întreg procesul se reia încă o dată. Cu alte cuvinte, cele două scripturi implementează o buclă repetitivă de (re)creare/distrugere succesivă a unei sesiuni.

Menținerea disponibilității și accesabilității unor informații (date utile) pe toata durata navigării prin mai multe pagini Web constituind o aplicație unitară (în diverse scopuri particularizate la specificul aplicației) constituie principalul beneficiu conferit de lucrul cu sesiuni.

Observație: Transferul de parametrii de la un script spre altul, în vederea asigurării disponibilității acestor parametrii în mai multe pagini Web, se poate realiza și utilizând casete de tip INPUT cu atributul `type='hidden'`. Metoda este eficientă atunci când trebuie transferat un număr fix, redus, de parametri, dar devine greoaiă atunci când numărul parametrilor de transferat crește.

O altă tehnică de lucru cu sesiuni, care se poate folosi în versiunile mai noi de PHP (ulterioare versiunii 4.0), implică utilizarea unor variabile de tip `$_SESSION[]`. Tehnica oferă un nivel mai ridicat de siguranță al aplicației (similar

cu cazul folosirii pentru transferul parametrilor a construcțiilor de genul `$_GET[]`, respectiv `$_POST[]`), fiind cea mai recomandată.

Un posibil scenariu de folosire a acestei tehnici presupune următoarele:

- Pentru deschiderea unei noi sesiuni PHP, scriptul trebuie să înceapă cu apelarea funcției `session_start()`. Aceasta funcție verifică, mai întâi, dacă există un identificator (ID) de sesiune. Dacă nu există, se va crea unul, și astfel se va deschide o nouă sesiune. Dacă există, atunci funcția încarcă variabilele sesiune sesiunii client curente stocate pe server, ele fiind astfel pregătite pentru utilizare. Atunci când se lucrează cu sesiuni, aceasta funcție (`session_start()`) trebuie apelată la începutul tuturor scripturilor care folosesc sesiunea în cauză.
- Variabilele de sesiune sunt stocate în tabloul superglobal numit `$_SESSION[]`, (în versiunile mai vechi de PHP fiind numit `$HTTP_SESSION_VARS[]`).
- Variabilele sesiune sunt disponibile și pot fi folosite până la ștergerea lor voluntară sau până la încheierea sesiunii.
- Pentru a crea o variabilă de sesiune trebuie să se introducă un element în tabloul `$_SESSION[]`: `$_SESSION['variabila_noua']=10;`
- Pentru a verifica dacă o anumită variabilă este înregistrată ca variabilă de sesiune în acest tablou, se folosește funcția `isset()` returnând o valoare booleană `true` sau `false`, ca în exemplul următor: `if (isset($_SESSION['variabila_noua']))` ... În funcție de valoarea logică a condiției impuse cu funcția `isset()`, execuția programului urmează o cale sau altă.

Pentru exemplificare, se prezintă o rescriere a scripturilor anterioare, utilizând această a doua tehnică de operare cu sesiuni (tabel 3):

Tabel 3

Script <i>a.php</i>	Script <i>b.php</i>
<pre><?php session_start(); \$_SESSION['var']="ceva"; echo "mai departe"; ?></pre>	<pre><?php session_start(); \$_SESSION['var']; echo \$_SESSION['var']; session_destroy(); ?></pre>

Ștergerea selectivă a unei variabile sesiune (nu a tuturor variabilelor, deci nu distrugerea sesiunii) se poate face folosind comanda `unset` (spre exemplu, `unset($_SESSION['var']);`).

Construcția `$_SESSION[]` poate folosi ca parametru de intrare nu doar nume de variabile (vezi exemplele anterioare), ci și nume de siruri (`array`), situație frecventă în cazul necunoașterii apriorice a numărului de variabile sesiune. Spre exemplificare:

Tabel 4

Script a.php	Script b.php
<pre><?php session_start(); \$_SESSION['var[1]']="ceva"; \$_SESSION['var[2]']="ceval"; echo "mai departe"; ?></pre>	<pre><?php session_start(); echo \$_SESSION['var[1]']; echo \$_SESSION['var[2]']; session_destroy(); ?></pre>

Ca o concluzie, o variabilă sesiune este salvată într-o resursă de tip fișier sesiune (de tip text) stocat pe partea de server Web, fiind vizibilă în întreaga aplicație Web (în toate scripturile PHP ale aplicației site). Pentru o variabilă sesiune denumită VAR, referirea în scriptul PHP se face sub forma `$_SESSION['VAR']`, respectiv `$_SESSION['VAR[indice]']` -dacă este vorba despre o variabilă sir (array).

Iată și o exemplificare privind transferul printr-o sesiune a unui sir de variabile sesiune între două scripturi, referite ca elemente ale unui sir:

Tabel 5

Script a.php	Script b.php
<pre><?php session_start(); for(\$i=0;\$i<5;\$i++) { \$_SESSION['var[\$i]']="ceva"; } echo "Go!"; ?>\$</pre>	<pre><?php session_start(); for(\$i=0;\$i<5;\$i++) { echo \$_SESSION['var[\$i]']; } ?></pre>

9. Operare cu cookies

Spre deosebire de o sesiune, un *cookie* permite salvarea unor informații posibil necesare și reaccesate repetat ori de cate ori se reapelează pagina respectivă, salvarea lor făcându-se însă pe partea de client (browser-ul trebuind setat să accepte *cookie*). [15][19] Un exemplu des întâlnit este cel în care, un utilizator dorește să salveze numele de user, respectiv parola de acces pe anumit site Web astfel încât, la o reaccesare a acelui site, acestea să fie disponibile implicit, fără a mai fi necesară reintroducerea lor explicită de la tastatură.

Comanda PHP pentru crearea unui *cookie* are următoarea sintaxă uzuală (doar trei dintre principali parametri fiind referiți aici):

```
setcookie (string $nume_cookie, string $valoare, int
temp_expirare)
```

Câteva exemple de definire a unui *cookie* folosind acești parametri:

- setare *cookie* cu nume USER, având o singură valoare ION și expirând după 60 de secunde:

```
SetCookie ("USER", "Ion", time () + 60);
```

- setare *cookie* cu numele securizare1, având valorile ION, respectiv PAROLA123, doar a două valoare expirând după 30 de secunde (prima valoare neavând timp de expirare):

```
setcookie ("securizare1[user]", "ION");
```

```
setcookie ("securizare1[parola]", "PAROLA123", time () + 30);
```

- ștergere *cookie* prin setarea unui moment de timp anterior (ștergerea explicită a fiecărei variabile *cookie* are loc după 1 secundă):

```
setcookie ("securizare1[user]", "ION", , time () - 1);
```

```
setcookie ("securizare1[parola]", "PAROLA123", time () - 1);
```

Accesul la o variabilă *cookie* se realizează (oarecum similar ca la sesiuni) printr-o construcție de genul:

`$_COOKIE[nume]`, parametrul fiind numele *cookie* interpretabil ca variabilă sau *array*, după caz.

Pornind de la secvențele anterioare, se prezintă două exemple de cod:

- primul, realizând o setare a unui *cookie* cu două variabile (definit ca sir cu două elemente), respectiv afișarea valorilor lor dacă *cookie*-ul există deja;
- al doilea, realizând o ștergere a *cookie*-ului (a ambelor valori, putându-se face și doar o ștergere selectivă după cum este cazul).

```
<?php
if (!isset($_COOKIE[securizare1]))
{
//echo "setez cookies: ";
setcookie ("securizare1[user]", "ION" );
setcookie ("securizare1[parola]", "PAROLA123", time () + 30);
}
else
{
echo "cookies setate:: ";
foreach ($_COOKIE[securizare1] as $value) {
    $value=htmlspecialchars($value);
    echo "$value <br>";
}
} ?>
```

Prima secvență a codului anterior verifică dacă există un *cookie* cu numele securizare1, în caz contrar creând unul, conținând două variabile cu valorile lor

explicite (eventual cu timp de expirare). A doua secvență detectează existența *cookie*-ului, afișând valorile variabilelor acesteia (tratarea făcându-se ca în cazul unui sir). Funcția `htmlspecialchars` este recomandată a fi folosită pentru tratarea cazului folosirii unor caractere speciale (<,>, &, ', etc.)

Stergerea *cookie*-ului se face prin setarea unui moment de timp anterior de expirare, (practic expirare într-o secundă pentru exemplul următor). Exemplu:

```
<?php
setcookie("securizare1[user]", "ION", time()-1);
setcookie("securizare1[parola]", "PAROLA123",time()-1);
?>
```

Observație: Dacă se sterge doar o variabilă a *cookie*-ului, acesta rămâne în continuare definit/setat cu cealaltă valoare, iar funcția `isSet` va returna TRUE.

10. Informații sistem

10.1. Informații asupra datei și timpului curent

Sunt prezentate câteva funcții utile pentru obținerea datei și timpului curent al sistemului (`time()`, `strftime()`, `getdate()`). Este importantă uneori obținerea exactă a datei (inclusiv a momentului de timp) aferente execuției unei anumite operații, astfel încât doar timpul de pe serverul Web (același pentru toți clienții apelańii) poate fi luat în considerare. [1] Codul sursă al unui exemplu de utilizare este următorul:

```
<?php
//data curentă ca string
echo "Timpul ca string: ".time()."<br>";
// funcția strftime() convertește timpul curent într-o dată formatată
$data=strftime("%d.%m.%Y",time());
echo "Data ca string an, luna, zi: ".$data."<br>";
// tot funcția strftime() convertește timpul curent într-un timp formatat
$time=strftime("%H%M",time());
$time1=$time/100;
echo "Timpul (ora, minut): ".$time1."\n"."<br>";

//extragere zi, luna, an din time() ca elemente ale unui sir ( data ca array)
$data1=getdate(time());
// construcție data și timp afișat formatat
$var="Data: ".$data1["mday"].".".{$data1["mon"]}.".
    {$data1["year"]}. " Timp: ".$time1."";
print ($var."<br>"); // afișare

//strtoupper - conversie la caractere mari
```

```
echo "<strong>".strtoupper($var); // afișare
?>
```

Figura 28 este elocventă asupra efectului fiecărei funcții utilizate.

Funcția `time()` returnează timpul în secunde (ca *string*), scurs de la începutul "Epoch Unix" (1 Ian. 1970). Funcția este utilă, prin utilizarea împreună cu alte funcții, pentru determinarea datei sau timpului curent (folosit de regulă ca referință). Funcția `strftime()` formatează timpul/datea, conform unui şablon dat ca argument, returnând un *string*, pe baza informației furnizate de `time()`.



Fig. 28. Rezultate privind timpul și data calendaristică

Funcția `getdate()` returnează un sir cu data calendaristică, permitând referirea și afișare element cu element (zi, lună, an). Funcțiile `strtoupper()`, `strtolower()` realizează o conversie a unui sir de caractere (furnizat ca argument) la caractere mari, respectiv mici.

10.2. Informații privind accesul client

În exemplul din paragraful anterior s-a realizat o contorizare a numărului de accese la o pagină Web. O altă informație utilă, referitoare la accesarea unei pagini Web, rezidă din întrebarea "Cine a accesat pagina Web?" Acest "cine" însemnă mai precis "Care este adresa IP a clientului care s-a conectat la pagina Web curentă?".

Pentru a răspunde la această întrebare și nu numai, se consideră scriptul prezentat în continuare [1]:

```
<?php
//obtinere informații de mediu
$I=getenv ("REMOTE_ADDR");
$j=getenv ("SERVER_NAME");
//afișare informații de mediu
printf(" Your IP number is : %s\n", $I);
```

```
printf(" The server name is : %s\n", $j);
?>
```

Funcția getenv cu sintaxă:

```
string getenv (string nume_variabilă)
```

are ca parametru de intrare o variabilă predefinită de mediu. În cazul de față, cele două variabile au următoarea semnificație:

REMOTE_ADDR – adresa IP a clientului

SERVER_NAME – numele serverului

Evident, utilizarea funcției getenv () având ca parametru o variabilă predefinită de mediu, are ca rezultat returnarea unui string conținând însăși informația asociată semnificației acelei variabile.

Afișarea rezultatelor pentru exemplul considerat s-a realizat folosind o funcție de afișare formatată: printf().

În cazul de față %s însemnă că, conținutul variabilei de afișat va fi tratat și formatat ca un string (\n însemnă deja cunoscutul ‘linie nouă’). Alte argumente de formatare posibile pentru printf(): %d –întreg cu semn în reprezentare zecimală, %f – număr în virgulă flotantă, %b – întreg în reprezentare binară etc.

10.3. Generare e-mail

Realizarea operației de trimitere a unui e-mail utilizând un script PHP implică în primul rând existența unei conexiuni la un server de mail SMTP, urmată de o configurare corespunzătoare a câtorva opțiuni din fișierul *php.ini*.

Presupunând că, numele serverului SMTP este aut.upt.ro, iar adresa de e-mail a emitentului este adm@domeniul.ro, configurarea fișierului *php.ini* implică următoarele (pentru o platformă Win32):

```
[mail function]
; For Win32 only.
SMTP= aut.upt.ro ; for Win32 only
; For Win32 only.
Sendmail_from= adm@domeniul.ro; for Win32 only
```

În acest moment, totul se rezumă la utilizarea funcției mail(), aşa cum se poate observa și în scriptul următor:

```
<?php
$a=mail("dan@aut.upt.ro", "Acesta e un subject!!!",
"Asta este continutul");
echo $a;
// 1-pt. Reușită, 0-pt. Nereușită
?>
```

În exemplul prezentat, s-a utilizat o formă sintactică redusă a funcției mail(), având doar 3 argumente de intrare: adresa destinatarului, *subjectul mail-*

ului, conținutul propriu-zis. Funcția returnează ‘1’ în caz de reușită, respectiv ‘0’ în caz de nereușită.

11. Grafică

PHP-ul dispune un set de funcții extrem de puternice pentru crearea și manipularea elementelor de grafică și a imaginilor, oferind astfel utilizatorului facilități deosebite și în domeniul graficii (2d).

Realizarea unor aplicații grafice cu PHP implică utilizarea unor biblioteci suplimentare (fișiere .dll). [1][20] Astfel, în cazul de față, s-a utilizat biblioteca *PHP_GD2.DLL* configuriind corespunzător fișierul *php.ini* (*extension=php_gd2.dll*). De regulă, formatul imaginilor manipulate (*jpg*, *gif*, *png* etc) depinde de versiunea bibliotecii grafice suplimentar utilizate

În cele ce urmează, fără a detalia toate cele câteva zeci de funcții dedicate creării și manipulării imaginilor, respectiv a elementelor specifice graficii, se va realiza o trecere în revistă a doar câtorva dintre ele, prin exemplificarea a două aplicații.

O primă aplicație face apel la câteva funcții privind crearea /manipularea unei imagini *jpg*. Codul complet este prezentat în continuare, iar efectul rulării acestui script este descris în figura 29 [1]:

```
<?php
// Anterior: setare ca fișier extensie activ (în php.ini): php_gd.dll
// Creare imagine nouă
$im = ImageCreate (500, 100) or die ("Cannot Initialize new GD image stream");

// deschiderea unei imagini existente
// $im = ImageCreateFromJPEG("leopard.jpg")
// or die ("Cannot Initialize new GD image stream");
// linia următoare nu-și are sensul pentru o imagine sursă dată

$background_color=ImageColorAllocate($im,1,255,25);
$text_color = ImageColorAllocate ($im, 230, 14, 191);

//ImageString – dimensiunea scrisului,- poziția pe orizontală (pixeli), poziția
//pe verticală (pixeli)
ImageString($im,10,100,35, "A Simple Text String",
$text_color);

// Salvare imagine prelucrată sau creată și afișare
ImageJpeg ($im,"tinta.jpg");
echo '';
```

```
// Creare unei FUNCTII care va fi apelată
function LoadJpeg($imagine, $text)
{
$im1 = ImageCreateFromJPEG($imagine)
or die ("Cannot Initialize new GD image stream");
$text_color = ImageColorAllocate ($im1, 23, 14, 191);
ImageString ($im1, 10, 10, 35, $text, $text_color);
ImageJpeg ($im1,"a.jpg");
echo '';
}
// Apel funcție
LoadJpeg("leopard.jpg","Ceva de afisat");
?>
```

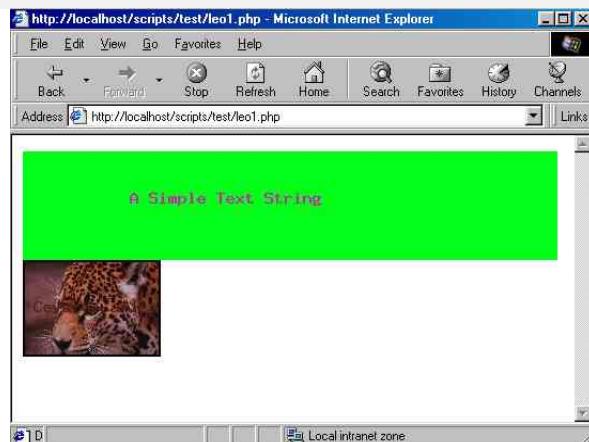


Fig. 29 Crearea și afișarea a două imagini

Practic, în cadrul exemplului prezentat sunt create și afișate două imagini, peste fiecare fiind suprascris un text (șir de caractere).

Prima imagine este creată cu un fundal verde, fără a porni de la o sursă inițială, iar a două imagine este creată pornind de la un fișier imagine *jpeg* ca sursă. Ambele imagini astfel create sunt salvate, iar afișarea lor se face utilizând eticheta HTML *<img...>*. Cea de-a două imagine a fost creată pe baza apelului unei **funcții**, necesitând ca parametrii inițiali fișierul sursă *jpeg*, respectiv textul de afișat.

O a doua aplicație, puțin mai complexă, permite realizarea graficului scalat al unei funcții matematice oarecare. Codul sursă comentat este redat în continuare, iar efectul script este prezentat în figura 30.

```
<?php
//definire constante (dimensiuni axe)
```

```
define("xmax", 500);
define("ymax", 200);
//creare imagine (lățime, înălțime)
$im=ImageCreate (xmax, ymax) or die ("Cannot Initialize
new GD image stream");
//fundal imagine (rgb-galben)
$background_color = ImageColorAllocate ($im, 255, 255, 0);
//culoare axe (verde)
$xaxe=ImageColorAllocate ($im, 0, 255, 0);
//culoare text (negru)
$text_color = ImageColorAllocate ($im, 0, 0, 0);
//titlu grafic (font, x, y)
ImageString ($im, 5, 200, 10, "Grafic test",
$text_color);
//trasare axe ($xaxe -culoare)
imageline ($im, 0,100,500,100,$xaxe);
imageline ($im, 0,0,0,200,$xaxe);
//afișare text pe axe (orizontal, vertical)
ImageString($im, 4, 425, 104, "timp [s]", $text_color);
ImageStringup($im, 4, 4, 25, "val", $text_color);
//culoare histograme (albastru)
$abc=imagecolorallocate($im,255,0,0);
//culoare grafic (albastru)
$grafic=imagecolorallocate($im,0,0,255);

//histograme (dreptunghiuri)
imagefilledrectangle ($im,200,100,220,140,$abc);
imagefilledrectangle ($im,300,50,320,100,$abc);

// coordonate inițiale
$x0=0;
$y0=0;

// calcul coordonate grafic și afișare grafic
for ($x=0;$x<500;$x+=1)
{
//generare număr aleator întreg într-un interval prestabilit
$a=rand(-30, 30);
//calcularea succesivă a coordonatei 'y' a funcției
$y=(2*sin($x)+log($x*10+10))*15-100+$a;
//trasare grafic linie cu linie
if ($x0!=0)
    imageline ($im,$x0,ymax/2-$y0,$x,ymax/2-$y,$grafic);
```

```
$x0=$x;
$y0=$y;
}
```

//salvare imagine într-un fișier și afișare

```
ImageJpeg ($im,"tinta.jpg");
echo '';
?>
```

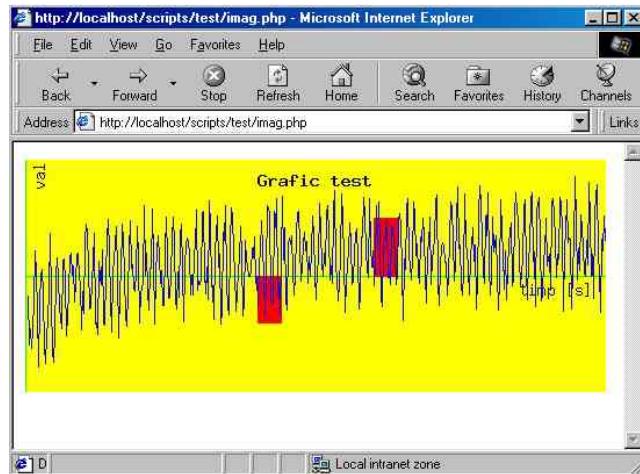


Fig. 30. Graficul unei funcții matematice

Analizând aplicația, se poate observa parcurgerea următoarelor etape:

- crearea imaginilor și stabilirea culorii fundalului (ImageCreate, ImageColorAllocate);
- stabilirea culorii axelor și a textului (ImageColorAllocate);
- afișarea titlului, trasare axe, scriere text pe axe (ImageString, ImageLine);
- realizarea unor histograme (exemplificative, fără legătură cu graficul funcției: ImageFilledRectangle);
- stabilirea culorii graficului (ImageColorAllocate);
- calculul funcție și afișarea prin linii succesive (For, ImageLine);
- salvare și afișare imagine grafic (ImageJpeg, <img...>).

Observatie: Pentru afișarea imaginii graficului, fără o salvare prealabilă într-un fișier a acesteia, se poate utiliza în finalul script-ului o secvență de genul:

```
// setarea tipului conținutului (MIME)
header ("Content-type: image/jpeg");
// afișare imagine
@imagejpeg ($im);
```

//caracterul @ care prefixează o funcție, face ca mesajele de eroare aferente execuției acelei funcții să nu fie afișate de browser; eventualele mesaje de eroare pot fi consultate prin analiza conținutului variabilei globale \$php_errormsg.

Exemplul prezentat arată că PHP-ul dispune de puternice facilități grafice, permitând o manipulare și prelucrare facilă a imaginilor, generarea dinamică a acestora, precum și salvarea lor în câteva formate standard.

Rezumând, o scurtă selecție a celor mai uzuale funcții utilizabile pentru dezvoltarea unor aplicații de grafică utilizând PHP este prezentă în continuare (fiecare funcție fiind precedată de tipul returnat):

- int ImageCreate() — creează o imagine cu dimensiuni precizate, returnând un identificator de imagine de tip întreg;
- int ImageCreateFromJpeg() — creează o imagine pornind de la un fișier de tip JPEG (returnează un identificator întreg în caz de succes sau 0 în caz de eșec). Pentru alte tipuri de fișiere imagine: ImageCreateFromGif(), ImageCreateFromPng().
- int ImageColorAllocate() — creează un identificator de culoare pentru o imagine, pe baza paletei RGB, culoare care poate fi alocată fundalului, unui text etc.;
- int ImageString() — inserează un sir de caractere de o anumită dimensiune în cadrul imaginii, pe orizontală, la coordonatele precizate. Identic, ImageStringUp() pentru scriere pe verticală;
- int ImageJPEG() — afișează imaginea desemnată printr-un identificator, sau o salvează într-un fișier specificat în format JPEG (idem pentru formatele GIF, PNG).
- int ImageLine() — afișează un segment de dreaptă identificat de coordonatele a două puncte, cu o anumită culoare (ImageDashedLine() pentru linii întrerupte);
- int ImageRectangle() — desenează o formă rectangulară;
- int ImagePolygon() — desenează un poligon (coordonatele precizate printr-un sir);
- int ImageArc() — desenează un arc de cerc;
- int ImageSetPixel() — desenează un pixel de o anumită culoare, la anumite coordonate;
- int ImageFilledRectangle() — umple cu o anumită culoare o zonă rectangulară (identic pentru o zonă poligonală: ImageFilledPolygon());
- int ImageDestroy() — distrugă un identificator de imagine și eliberează memoria alocată acestuia.

PHP

12. Programare PHP orientată pe obiecte

Programarea orientată pe *obiecte* (**OOP** –*Object Oriented Programming*) a apărut ca o necesitate în contextul **creșterii complexității codului** aplicațiilor software. Pentru aplicațiile de mari dimensiuni, o **dezvoltare structurată** a codului (orientată pe funcții/proceduri) implicând existența unui număr foarte mare de linii de cod (uneori puternic **redundant**), prin modul de organizare a codului conduce la o **lizibilitate scăzuta** a acestuia și implicit, la mari **dificultăți** privind realizarea unor **modificări** ulterioare în cadrul aplicației.

OOP oferă o modalitate diferită de organizare a codului și a datelor în cadrul unui program. [29]

Din acest punct de vedere, elementele constructive de cod specifice OOP sunt **clasa**, respectiv **obiectul**. Clasa reprezintă definiția unui obiect (“planul obiectului”). Prin **instantierea** unei clase este creat un obiect (evident se pot face instantieri multiple, construindu-se mai multe obiecte ale aceleiași clase). În cadrul clasei (definiția obiectului) sunt precizate

- atribute sau **proprietăți** – (practic partea de date a obiectului) reprezentate prin declarații de variabile, inclusiv posibile inițializări ale acestora;
- **metode** – (partea de cod a obiectului) reprezentate prin funcții (sau proceduri) constituind totodată și interfața obiectului destinată manipulării datelor acestuia.

OOP este fundamentată pe 3 principii de bază:

- **încapsulare** – fiecare obiect este de sine stătător și complet autonom (conținând atât date –proprietăți-, cât și cod –metode). Un obiect are o interfață clară, bine definită, folosită pentru a manipula obiectul;
- **moștenire**;
- **polimorfism**.

Asupra ultimelor două se va reveni în contextul programării în PHP (existând câteva particularități specifice acestuia). Particularizat la PHP, în continuare sunt tratate următoarele aspecte [20]:

- Crearea unei clase (proprietăți, metode)
- Crearea unui obiect (instantierea clasei)
 - Utilizarea proprietăților obiectului
 - Apelul metodelor obiectului
- Moștenirea
- Polimorfismul

În limbajul PHP, crearea unei clase se face utilizând instrucțunea **class**. O definire minimală a unei clase, este prezentata mai jos:

```
class denumire_clasa {  
}
```

În vederea atribuirii unei funcționalități clasei, este necesară definirea unor proprietăți și metode. Proprietățile se creează prin declararea unor variabile la începutul definiției unei clase folosind instrucțunea **var**. Următoarea secvență PHP creează o clasă denumită *Clasa1* având două atribute (proprietăți): *\$message*, *\$data* (ultimul fiind și inițializat).

```
class Clasa1 {  
    var $message;  
    var $data="initializat";  
}
```

Metodele se creează prin declararea unor funcții PHP în definiția clasei. Codul următor va crea (pentru clasa *Clasa1*) două metode:

- metoda *setMessage*, având un parametru de intrare (*\$param*) și permitând o setare a valorii proprietății *\$message*. De remarcat că, în cadrul definiției clasei, referirea unei proprietăți a acesteia se face folosind operatorul *\$this* care precede numele proprietății (nume utilizat fără *\$* în față). În cazul de fata: *\$this->message*.
- metoda *getMessage*, fără parametrii de intrare, care afișează un mesaj, respectiv returnează valoarea proprietății *\$message*.

```
<?php  
    class Clasa1  
{  
        var $message;  
        var $data="initializat";  
  
        function setMessage($param)  
        {  
            $this->message = $param;  
        }  
  
        function getMessage()  
        {  
            echo "Mesajul pentru obiectul 1 este:<br>";  
            return $this->message;  
        }  
    }  
//...cod PHP instantiere clasă  
?>
```

Evident s-a inclus codul în tag-urile de delimitare specifice PHP. După ce a fost creată clasa, în continuare se prezintă modul de **instantiere** a clasei în vederea creării unui obiect, precum și modul de setare a proprietăților acestuia și de apel al metodelor (prin completarea secvenței anterioare, după încheierea definiției clasei):

```
$Obiect1 =new Clasa1();

$Obiect1->setMessage("Setare proprietate folosind o
metoda");
echo $Obiect1->getMessage();
$var=$Obiect1->getMessage();
echo $var;

$Obiect1->message="Setare directa a proprietatii prinr-
o atribuire ";
echo $Obiect1->getMessage();

echo $Obiect1->data;
```

Rezultatul rulării scriptului este următorul:

```
Mesajul pentru obiectul 1 este:
Setare proprietate folosind o metoda
Mesajul pentru obiectul 1 este:
Setarea proprietate folosind o metoda:
Mesajul pentru obiectul 1 este:
Setare directa a proprietatii prinr-o atribuire
initializat
```

Ulterior declarării unei clase, este necesara crearea un obiect cu care aceasta să opereze. Operația este denumita instanțierea unei clase sau crearea unei instanțe. În limbajul PHP pentru aceasta operație, se utilizează instrucțiunea, cuvântul cheie ***new***.

- Deci prima linie din secvența de cod anterioară creează un nou obiect *Obiect1* prin instanțierea clasei *Clasa1*.

- În continuarea se face un apel al metodei *setMessage* (\$Obiect1->setMessage) pasându-i un parametru, metoda care permite setarea proprietății *message*. Linia următoare apelează metoda *getMessage* (fără parametrii), care afișează un mesaj și returnează o valoare (a proprietății *message*, afișată pe ecran folosind *echo*).

- Valoarea returnată poate fi evident memorată într-o variabilă (\$var în cazul de față).

- În acest caz, setarea proprietății *message* a fost realizată prin apelul unei metode. Setarea unei proprietăți a unui obiect se poate face și prin atribuirea directă a unei valori către proprietatea referită (\$Obiect1->message="Setare directa a proprietatii") sau chiar prinr-o inițializare în definiția clasei (vezi proprietatea *\$data*). Toate comenziile de afișare (*echo*) au fost utilizate doar pentru a evidenția modul de referire a atributelor/metodelor obiectului.

Observație: În cazul limbajul PHP nu se limitează accesul la proprietăți. Implicit toate proprietățile sunt de tip *public*, neputând fi declarate *private* sau *protected*.

Orice proprietate declarată în definiția clasei poate fi referită în exteriorul ei printr-o construcție de tipul: \$Obiect->nume_proprietate;

O metodă specială a unei clase este aşa numita metodă ***constructor***. O metoda constructor are același nume cu al clasei, fiind apelata automat la crearea unui obiect (realizând operații de inițializare, crearea de alte obiecte necesare obiectului în cauza etc.). Un constructor se declara similar cu celelalte metode, singura deosebire fiind ca are același nume ca și clasa. În PHP definirea unei metode constructor nu este obligatorie. Clasa anterior creată nu avea definit un constructor (unele proprietăți fiind însă inițializare direct odată cu declararea lor).

Pentru exemplificare, se va crea un constructor pentru *Clasa1*, care va afișa un mesaj și va inițializa proprietatea *\$message*. Definiția anterioară a clasei se va completa cu încă o metodă (având același nume cu al clasei):

```
function Clasa1()
{
    echo "Obiect creat<br>";
    $this->message = "initializare_obiect1";
}
```

O simplă secvență de creare a obiectului va apela imediat metoda constructor:

```
$Obiect1 =new Clasa1();
echo $Obiect1->message;
```

iar rezultatul va fi:

```
Obiect creat
initializare_obiect1
```

Pentru ca o clasa deja creată să poată fi instantiată în mai multe scripturi PHP, fără a se face o replicare (copiere) a codului clasei în fiecare script, de regulă definițiile claselor sunt păstrate în fișiere distincte de cele în care sunt create obiectele. Pentru cazul de fata, spre exemplu, într-un fișier *clase.php* este salvată definiția clasei, în timp ce într-un alt fișier PHP (*obiect.php* spre exemplu) se face instanțierea clasei (și utilizarea obiectului):

```
<?php
// fișier obiect.php
include("clase.php");
$Obiect1 =new Clasa1();
echo $Obiect1->data;
?>
```

Evident fișierul *clase.php* poate conține definițiile mai multor clase.

O caracteristică importantă a OOP o reprezintă ***moștenirea*** care permite crearea unei relații ierarhice între clase, folosind subclase. O subclasa, practic moștenește proprietățile și metodele unei superclase. Prin intermediul moștenirii, pe lângă elementele moștenite (proprietăți și metode), în cadrul noii clase se pot

construi și adăuga noi elemente (proprietăți sau metode). Astfel, pornind de la clase de baza simple se pot **deriva** clase mai complexe și mai specializate pentru o anumită aplicație. Utilizarea moștenirii crește gradul de reutilizare și lizibilitate al codului sursă, avantage important al OOP (reducându-se substanțial munca programatorului în cazul în care aceleași metode pot fi scrise doar o singură dată într-o superclasă, în loc să fie scrise de mai multe ori în subclase separate). Pe baza superclasei *Clasa1* (în același fișier script), se construiește prin derivarea acesteia o nouă subclasa *Clasa2* (utilizând cuvântul cheie **extends**), care moștenește toate proprietățile și metodele superclasei, dar în același timp are și alte noi proprietăți și metode:

```
class Clasa2 extends Clasa1
{
    var $message2="gama";
    function getMessage()
    {
        echo "mesajul nou pentru obiectul 2 este<br>";
        return $this->message2;
    }

    function plus()
    {
        echo "<br>ceva nou<br>";
    }
}
```

De remarcat că, în cadrul subclasei *Clasa2* se definește o metodă având un nume similar cu al unei metode din superclasa (*getMessage*), realizându-se în cadrul subclasei o suprascriere a metodei moștenite, precum și o nouă metodă (*plus*). De asemenea, noua clasă are și o proprietate în plus (*\$message2*).

O secvență de instanțiere și utilizare a obiectelor subclasei se poate face prin liniile de cod următoare:

```
$Obiect2 =new Clasa2();
$Obiect2->plus();
$Obiect2->setMessage("beta");
echo $Obiect2->message."<br>";
echo $Obiect2->getMessage();
are rezultatul următor:
```

```
Obiect creat
ceva nou
beta
mesajul nou pentru obiectul 2 este
gama
```

Crearea obiectului prin instanțierea subclasei *Clasa2* conduce și la moștenirea constructorului corespunzător (afișându-se *Obiect creat*). Apelul metodei *plus* conduce la afișarea mesajului *ceva nou*. Apelul metodei moștenite *setMessage* permite setarea proprietății moștenite *message*. Apelul metodei suprascrise *getMessage* (plasat într-un *echo*), conduce la afișarea ultimelor două rânduri.

Observație: Mecanismul de moștenire funcționează într-un singur sens, subclasa (copil) moștenește de la superclasa (părinte). Orice modificare a clasei părinte este automat preluată și de clasa copil.

O altă caracteristică importantă a OOP o reprezintă **polimorfismul**, prin intermediul căruia clase diferite pot conține metode cu același nume, dar cu comportamente diferite. Chiar în exemplul cu moștenire, superclasa respectiv subclasa conțin metode cu același nume (*getMessage*), dar funcționalități diferite. Se consideră încă un exemplu. În același script cu definiția clasei *Clasa1*, fie definiția unei clase *Clasa3*:

```
class Clasa3
{
    var $message;
    function setMessage ($param)
    {
        $this->message = $param;
        echo "Clasa 3";
        return $this->message;
    }
}
```

La o instanțiere a claselor și folosirea obiectelor create:

```
$Obiect1 =new Clasa1();
$Obiect3 =new Clasa3();

$Obiect1->setMessage ("Setare proprietate folosind o
metoda:");
echo $Obiect1->getMessage();

$Obiect3->setMessage ("Setare proprietate clasa 3");
```

rezultatele sunt:

```
Obiect creat
Mesajul pentru obiectul 1 este:
Setarea proprietate folosind o metoda:
Clasa 3
```

Un exemplu concret de utilizare a programării orientate pe obiecte pentru realizarea conexiunii la un server MySQL, respectiv la baza de date, este prezentat în continuare, în contextul în care aceste operații implică o refolosire repetată a același sevențe de cod pentru fiecare script operând cu baza de date.

Ca punct de start, se definește o **clasă** (*Clasa1*) implementând o metodă (*setServer*) executând operațiile dorite (clasa salvată într-un fișier script distinct-*Clase.php*-, care poate fi referit și inclus în orice alt script este necesar utilizând comanda *include*):

- Fișier *Clase.php*:

```
<?php
    class Clasa1
{
    function setServer($var1, $var2,$var3,$var4)
    {
        $con = @mysqli_connect ("$var1", "$var2", "$var3",
"$var4") or die ("Eroare SERVER");
        return $con;
    }
?>
```

Metoda clasei are patru parametri de intrare (nume server, user, parolă, numele bazei de date), returnând în caz de reușită un identificator de conectare (util pentru cazul conectării la mai multe servere SQL, pentru identificarea unei conexiuni).

Utilizarea clasei, în orice script fiind necesară o conexiune la MySQL, presupune o **instantiere** a ei (creând un nou obiect), urmată de un **apel al metodei** ei, setată cu parametri adecvați:

```
<?php
include("clase.php");
$Obiect1 =new Clasa1();
$db=$Obiect1->setServer("localhost","root",
"parola","baza");
echo 'Conectare OK!';
$query = "SELECT * FROM test";
$interoghez=mysqli_query($db, $query);
... 
```

O soluție și mai compactă ca și cod, presupune folosirea unui **constructor** și astfel, la o instantiere a clasei se face și un apel automat al metodei realizând conectarea la server, respectiv la baza de date. Fișierul conținând clasa are codul următor:

- Fișier *Clase.php*:

```
<?php //folosind constructor
    class Clasa2
```

```
{
    var $con;
    function Clasa2($var1, $var2,$var3,$var4)
    {
        $this->con = mysqli_connect("$var1", "$var2", "$var3",
"$var4") or die ("Error connecting to mysql");
        return $this->con;
    }
?>
```

Scriptul în care se face un apel la această clasă pentru realizarea unei conexiuni la o bază de date MySQL poate conține următorul cod:

```
<?php
include("clase.php");
$Obiect2=new Clasa2("localhost","root","parola",
"baza");
$db=$Obiect2->con; // referire identificator conectare
//cod interogare
. . .
?>
```

13. MYSQLI vs PDO

Dupa cum se cunoaste deja, Mysql este un sistem de gestiune a bazelor de date relationale, foarte des folosit împreună cu [limbajul de programare PHP](#) pentru dezvoltarea de aplicatii WEB cu baze de date.

PHP are trei moduri diferite prin care se poate conecta și interacționa cu o bază de date MySQL :

- extensia originală MYSQL, extensie care a fost depreciată în 2012 și care nu mai este recomandată pentru proiecte PHP-MYSQL noi;
- extensia MySQLi (MySQL Improved);
- extensia PDO (PHP Data Objects).

PHP 5 și versiunile ulterioare pot opera cu o bază de date MySQL folosind fie MySQLi, fie PDO.

Principalul avantaj al extensiei PDO, fata de Mysqli constă în abilitatea să de a opera cu 12 sisteme diferite de baze de date (inclusiv IBM, Oracle), în timp ce Mysqli permite operarea doar cu baze de date MySQL.

Deci, dacă trebuie să vă schimbați proiectul pentru a utiliza o altă bază de date, PDO ușurează procesul. Trebuie doar să schimbați sirul de conexiune și câteva interogări. Cu MySQLi, va trebui să rescrieți întregul cod.

Ambele extensii ofera metode de programare orientata pe obiecte, insa Mysqli ofera si un set de functii pentru programarea procedurala.

Ambele extensii ofera suport pentru prevenirea problemei SQL injection.

```
// PDO
$name = PDO::quote( $_POST['code'] );
$pdo->query( "SELECT id, name FROM products WHERE code = '$code' ");

//MySQLi
$name = mysqli_real_escape_string( $_POST['code'] );
$mysqli->query( "SELECT id, name FROM products WHERE name = '$code'" );
```

Un principal avantaj al MySQLi consta in posibilitatea utilizarii de noi caracteristici disponibile in versiunile mai noi ale serverelor MySQL. PDO nu dispune de un suport extins pentru a profita din plin de noile capabilitati MySQL.

In continuare se prezinta cateva exemple de operare a limbajului PHP cu o baza de date MySQL, atat prin intermediul extensiei MySQLi, cat si prin intermediul extensiei PDO.

Deschiderea unei conexiuni la o baza de date MySQL, utilizand MySQLi si PDO

Pentru a putea accesa date dintr-o baza de date MySQL, mai intai trebuie sa ne conectam la serverul de baze de date.

- Conectarea la serverul de baze de date MySQL, utilizand MySQLi – metoda orientata pe obiecte

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password,
$dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
```

```
}
echo "Connected successfully";
?>
```

- Conectarea la serverul de baze de date MySQL, utilizand MySQLi – metoda procedurala

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username,
$password, $dbname);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";
?>
```

- Conectarea la serverul de baze de date MySQL, utilizand PDO

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

try {
    $conn = new
        PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
} catch(PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
}
?>
```

Inchiderea unei conexiuni, utilizand MySQLi si PDO

Inchiderea conexiunii cu serverul de baze de date se poate face:

- **utilizand MySQLi – metoda orientata pe obiecte**
\$conn->close();

- **utilizand MySQLi – metoda procedurala**
mysqli_close(\$conn);

- **Inchiderea conexiunii cu serverul de baze de date, utilizand PDO**
\$conn = null;

Inserarea de date intr-o tabela MySQL, utilizand MySQLi si PDO

Dupa ce o baza de date si o tabela sunt create, putem incepe popularea bazei de date cu inregistrari.

Comanda INSERT utilizata pentru inserarea de noi inregistrari in tabela MySQL are urmatoarea forma:

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

- **Inserarea unei noi inregistrari cu MySQLi – metoda orientata pe obiecte**

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username,
$password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname,
email)
```

```
VALUES ('John', 'Doe', 'john@example.com');

if ($conn->query($sql) === TRUE) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```

- **Inserarea unei noi inregistrari cu MySQLi – metoda procedurala**

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username,
$password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname,
email) VALUES ('John', 'Doe', 'john@example.com')";

if (mysqli_query($conn, $sql)) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

- **Inserarea unei inregistrari cu PDO**

```
<?php
$servername = "localhost";
```

```
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
$conn = new PDO("mysql:host=$servername;dbname=$dbname",
$username, $password);
// set the PDO error mode to exception
$conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
$sql = "INSERT INTO MyGuests (firstname, lastname,
email) VALUES ('John', 'Doe', 'john@example.com')";
// use exec() because no results are returned
$conn->exec($sql);
echo "New record created successfully";
} catch(PDOException $e) {
echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

3. Elemente de JavaScript

3.1. JavaScript

3.1.1. Elemente introductive JavaScript

JavaScript este la ora actuală cel mai popular limbaj de script, rulând pe partea de client (browser Web), fiind destinat, în principal, sporirii interactivității paginilor Web. [8][17] Limbajul JavaScript, lucrând împreună cu elementele (etichetele) limbajului HTML, asigură o reacție imediată la anumite evenimente atașate acestor elemente, furnizând o dinamică sporită conținutului paginii Web. [18] De asemenea, limbajul asigură posibilitatea unei validări rapide a datelor de intrarea preluate prin formulare HTML, precum și diverse prelucrări de informații pe partea de client (inclusiv salvarea/accesarea unor date/resurse ale acestuia). Fiind practic un limbaj interpretat de browser-ul Web, dependența acestuia de tipul de browser este evidentă. [30] Astfel, este posibil ca o secvență de cod JavaScript să aibă un comportament pe un anumit browser Web, respectiv alt comportament pe un alt browser. Primul browser care a înglobat un interpreter JavaScript a fost Netscape Navigator 2.0, la ora actuală toate browser-ele Web integrând JavaScript într-o formă care se dorește cât mai standardizată. În procesul de dezvoltare al unei aplicații Web, având drept fundament HTML-ul, codul JavaScript interacționează cu etichetele statice HTML, oferind o comportare dinamică a acestora și implicit o apropiere a comportamentului unei aplicații Web de cel al unei aplicații desktop. Comparativ cu limbajele operând pe partea de server, codul JavaScript, pe lângă dezavantajul dependenței de browser, nu poate fi ascuns, fiind total accesibil oricărui apelant al paginii respective. Din acest motiv, exemplele de utilizare fiind la îndemâna oricui fără nici o restricție, în cadrul acestui capitol nu se va face o prezentare în extenso a limbajului, orice programator familiarizat cu limbajul C putând să preia de pe Internet secvențe de cod JavaScript (integrate în diverse site-uri) dedicate celor mai diverse funcționalități dorite.

Observație: A nu se confunda JavaScript cu Java, ambele făcând parte de familia de limbaje C/C++, dar fiind complet diferite ca și concept și structură.

Un simplu exemplu introductiv este prezentat în continuare pentru a evidenția modul de integrare a HTML-JavaScript într-un script rulând *client-side*:

```
<html>
<head>
<title>Exemplu JavaScript</title>
</head>
```

```
<body>
<a href="javascript:buna()">Rulare JavaScript</a>
</body>
</html>

<script language="JavaScript">
function buna()
{
    alert("Salut JavaScript!");
}
</script>
```

Codul precedent implementează o simplă funcție JavaScript generând un mesaj într-o fereastră *popup* la accesarea unei referințe. Evenimentul declanșator îl constituie un *click* pe referința respectivă. Se poate remarcă sintaxa de descriere a unei secvențe de cod JavaScript (marcajele de început/sfârșit cod), respectiv modul ei de apelare (referire integrată în codul HTML).

3.1.2. Validare date numerice de intrare

Exemplul din paragraful curent prezintă o funcție JavaScript declanșată la apăsarea unei taste în interiorul unei casete INPUT dintr-un formular. Funcția permite tastarea doar a cifrelor în interiorul casetei (respectiv a unor caractere de control, spre exemplu *backspace*), evitând astfel prelucrarea unor date eronate și asigurând o validare în timp real a conținutului casetei. De remarcat, în corpul funcției, structura bifurcată care tratează două situații de rulare a codului pe un browser din familia Internet Explorer, respectiv familia Mozilla (sau altele). Evenimentul atașat etichetei INPUT este *onkeypress*. Funcția returnează o valoare *true* sau *false* (funcție de codul ASCII al tastei apăsate), valoarea *false* invalidând prelucrarea caracterului tastat (*onkeypress="return false"* → invalidează prelucrarea de caracter de la tastatură). Codul anterior explicitat este următorul:

```
<FORM action='tinta.php' method='GET'>
Salariu: <input type="text" name="sal"
onkeypress="return onlyNo(event)">
<input type='SUBMIT' value='Start'>
</FORM>

<script language="JavaScript" type="text/javascript">
    function onlyNo(evenim)
    {
        var e = evenim;
        if(window.event)
        { // Internet Explorer
            var charCode = e.keyCode;
```

```

        }
    else if (e.which)
    { //Mozilla, Firefox
        var charCode = e.which;
    }
    //verificare plaja valori ASCII pentru cifre și caractere de control
if (charCode > 31 && (charCode < 48 || charCode > 57))
    return false;           // &&, || -> ȘI, SAU LOGIC
    return true;
}
</script>

```

3.1.3. Validare conținut casete INPUT

Tot ca o aplicație de validare a unor date de intrare, codul script următor integrează JavaScript pentru a forța completarea cu informații a unor casete din cadrul unui formular. [30] Comparativ cu exemplul din paragraful precedent, în aplicația de față însă nu se verifică conținutul informației, ci doar prezența unei informații (date) în aceste casete. Scopul acestei aplicații este și de a puncta alte câteva evenimente declanșatoare atașate unor casete INPUT sau chiar unui întreg formular FORM, respectiv exemplificarea modului de referire a unor etichete HTML în cadrul codului JavaScript. Codul este următorul:

```

<!--Cod formular HTML cu atașare evenimente JavaScript-->

<form method="GET" action="a.php" onsubmit="return
checkform(this);">


|                                      |  |
|--------------------------------------|--|
| <input type="submit" value="Adauga"> |  |
|--------------------------------------|--|


```

```

</tr>
</table>
</form>

```

<!--Descriere funcții JavaScript atașate codului HTML-->

```

<script language="JavaScript" type="text/javascript">

function checkform(form) // verifică conținut pentru toate casetele
{
    if (form.nume.value == "") {
        alert( "Nume student." );
//var bool=confirm("Alege"); ->Dialog Box
        form.nume.focus();
        return false ;
    }

    if (form.nota1.value == "") {
        alert( "Nota Parcurs student." );
        form.nota1.focus();
        return false ;
    }

    if (form.nota2.value == "") {
        alert( "Nota Examen student." );
        form.nota2.focus();
        return false ;
    }
    return true ;
}


```

```

function checkinput(form) // verificare casetă precedentă nota1
{
    if (form.nota1.value == "")
    {
        alert( "Scrie ceva în campul 2 - Nota parcurs..." );
        form.nota1.focus();
        return false ;
    }
    return true ;
}

function checkinput_alt(form) // verificare casetă precedentă nume
{

```

```

if (form.nume.value == "") {
    alert( "Scrie ceva in campul 1 - Nume..." );
    form.nume.focus();
    return false ;
}
return true ;
} </script>

```

Funcția `checkform` atașată formularului FORM și apelată cu parametrul `this` (semnificând eticheta curentă HTML în cadrul căreia este referit evenimentul declanșator al funcției - în cazul de față, formularul FORM), verifică succesiv (la apăsarea butonului de tip SUBMIT- fig.3.1.a.) conținutul celor trei casete din formular. Pe fiecare din cele trei ramuri `if`, dacă conținutul casetei referite este vid, focusul rămâne în acea casetă până la o completare cu date a acelei casete (spre exemplu, pentru prima casetă cu numele NUME din formularul FORM: `form.nume.focus()`). Se poate remarcă modul în care poate fi referit în JavaScript un element al unui formular: `form.nume.value` – precizând succesiv numele formularului (`form` – ca parametru sau `this` – ca formular curent), numele etichetei INPUT, respectiv proprietatea `value`. Această verificare realizează o validare/invalidare a conținutului casetelor abia în momentul unui eventual transfer al acestor date spre o altă pagină, prin apăsarea unui buton SUBMIT (deci după eventuala completare a tuturor câmpurilor formularului).

O variantă mai rapidă de validare constă într-o verificare a conținutului casetelor în momentul părăsirii lor – evenimentele declanșatoare fiind `onclick` pe caseta următoare (deci s-a părăsit o casetă al cărei conținut este verificat, făcând-se `click_mouse` pe caseta următoare), respectiv `onblur` (pe caseta curent verificată - fig.3.1.b.) – semnificând părăsirea casetei *focusate* (tastă TAB, `click mouse` pe altă casetă), respectiv pierderea focusului acesteia.

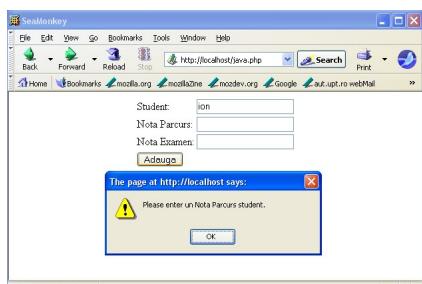


Fig. 3.1.a Validare la SUBMIT
(verificare la final)

Construcția JavaScript de genul “`return funcție()`” (integrată în formularul HTML) permite, în cazul în care funcția referată returnează valoarea

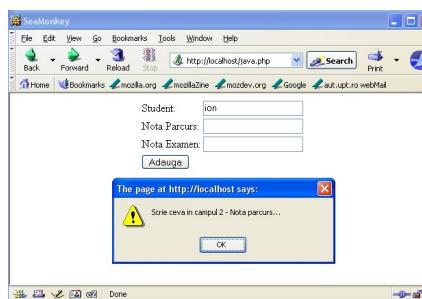


Fig.3.1b. Validare caseta la *OnClick* pe caseta următoare (verificare on-line)

booleană `FALSE`, anularea efectului evenimentului declanșator (spre exemplu, pentru funcția `checkform`, anulează efectul SUBMIT). Toate funcțiile referite în aplicația de față returnează `FALSE` în cazul neîndeplinirii condiției verificate.

Funcțiile `checkinput`, respectiv `checkinput_alt` verifică conținutul unei anumite casete precizate prin cod (`nota1`, respectiv `nume`). Important este evenimentul care le declanșează, respectiv eticheta HTML de care este atașat acel eveniment. Astfel:

- Evenimentul `onclick` atașat casetei `nota1`, apelează funcția `checkinput_alt` (având ca parametru chiar FORM-ul curent), verificând conținutul casetei INPUT precedente - `nume`.
- Evenimentul `onclick` definit pe caseta `nota2`, apelează funcția `checkinput` (având ca parametru chiar FORM-ul curent), verificând conținutul casetei INPUT precedente- `nota1`.
- Evenimentul `onblur`, definit suplimentar pe caseta `nota2`, apelează tot funcția `checkinput`, deosebind față de cazul precedent constând în faptul că `onblur` semnifică o părăsire a casetei curente prin tastă TAB sau prin `click mouse` pe orice altă casetă, declanșându-se la o pierdere a *focusului* de către caseta curentă (*prompter*-ul părăsește caseta). Altfel spus, `onclick` interceptează evenimentul corespunzător generat de `click mouse` pe casetă, iar `onblur` pe cel generat de o părăsire a casetei (echivalent cu `onLeave` în alte limbi).

Prin strategia astfel adoptată, orice tentativă de părăsire fără completare a unei casete este anulată (prin returnare valoare `FALSE` de către funcția apelată), respectiv focusul este repoziționat pe caseta în cauză (printr-o comandă de genul: `form.caseta.focus()`). Acest mod de verificare “încrucisată” a conținutului casetelor INPUT, dublată de o verificare suplimentară la *submitarea* finală, face practic imposibilă pasarea de către formular, spre scriptul apelat, a unor parametri nuli.

3.1.4. Aplicație de mapare pe o imagine

Aplicația din acest paragraf are drept obiectiv funcțional crearea unei “hărți” pe o imagine (operație numită și mapare pe o imagine), care capturează evenimentele legate de deplasarea `mouse`-ului (`onMouseMove`), respectiv `click`-uri ale `mouse`-ului pe aceasta (interceptate de o etichetă ancoră `<A>`), reacționând în consecință prin afișarea unor mesaje. Un *map imagine* definește anumite locații pe o imagine (în genul unei hărți), browser-ul răspunzând (prin apeluri JavaScript) la anumite interacțiuni ale utilizatorului cu acestea.

În cazul de față, sunt definite trei zone dreptunghiulare pe imagine: zona titlul cărții (partea de sus a imaginii), numele autorilor (immediat sub zona titlu), respectiv zona editură (în partea de jos a imaginii). Interacțiunea cu tot restul zonei global definită prin atributele `WIDTH` și `HEIGHT` ale etichetei `IMG`, este tratată de o ramură separată a secvențelor de cod (ramura `else`). Pe tot restul ecranului aplicația nu asigură nici o interacțiune. Figura 3.2 prezintă pagina aşa cum apare în

browser-ul Microsoft *Internet Explorer* (scriptul, în forma descrisă în continuare, funcționând doar pe acest tip de browsere).

Codul complet al aplicației este următorul:

```
<HTML>
<FORM NAME="form1">
<A href="javascript:apasare_mouse()" >
<IMG SRC="poza.gif" WIDTH=162 HEIGHT=200
onMouseMove="javascript:miscare()" ></A><BR>
<INPUT TYPE="text" NAME="descriere" SIZE=50 value=" " >
</FORM>
```

```
<SCRIPT LANGUAGE="JavaScript">
<!--
```

```
    var x_curent, y_curent;

function apasare_mouse()
{
    if (in_zona(x_curent,y_curent,0,0,161,50))
        alert("Programare in JavaScript");
    else if (in_zona(x_curent,y_curent,100,60,161,90))
        alert("Autorii cartii");
    else if (in_zona(x_curent,y_curent,0,120,161,199))
        alert("Publicata in editura Sams");
    else
        alert("Alege o zona de imagine!"+ " x: "+x_curent+" y:
"+y_curent);
}
```

```
function in_zona(x,y,Rect_x1, Rect_y1, Rect_x2, Rect_y2)
{
    return( x>Rect_x1 &&
           x<Rect_x2 &&
           y>Rect_y1 &&
           y<Rect_y2);
}
```

```
function afiseaza_text(text)
{
    form1.descriere.value=text; //formular lipsă: descriere.value=text;
}
```

```
function miscare()
{
```

```
    x_curent=event.x;
    y_curent=event.y;

    if (in_zona(x_curent,y_curent,0,0,161,50))
        afiseaza_text("Programare in JavaScript");
    else if (in_zona(x_curent,y_curent,100,60,161,90))
        afiseaza_text("Autorii cartii");
    else if (in_zona(x_curent,y_curent,0,120,161,199))
        afiseaza_text("Publicata in Editura Sams");
    else
        afiseaza_text("Alege o zona de imagine!"+ " x:
"+x_curent+" y: "+y_curent);
}

//-->
</SCRIPT>
</HTML>
```

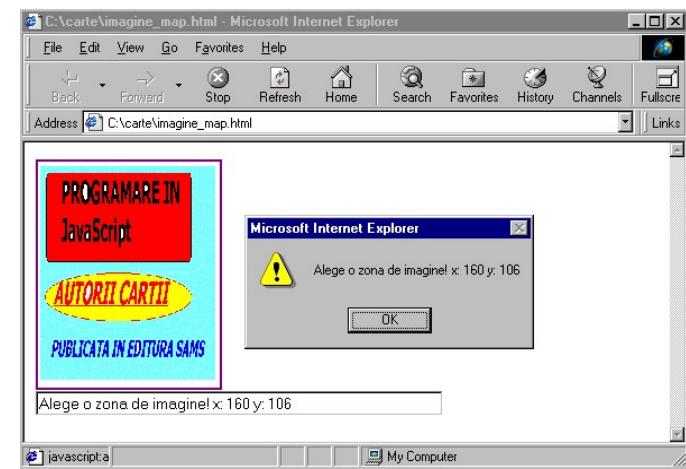


Fig. 3.2 Mapare mesaje pe o imagine-hartă folosind JavaScript

Din punct de vedere funcțional, următoarele aspecte sunt relevante:

- la apelul scriptului (integrând cod HTML și JavaScript), o pagină cu o imagine (*poza.gif*), respectiv o casetă de tip INPUT (înglobate într-un formular HTML având numele *form1*) este generată și afișată pe ecran;
- La o deplasare a mouse-ului pe ecran, în caseta INPUT este afișat un mesaj textual precizând zona deasupra căreia se deplasează mouse-ului. Dacă pointerul mouse-ului nu se găsește într-o zonă explicit mapată a imaginii, un mesaj solicitând alegerea unei zone de pe imagine, respectiv coordonatele curente ale mouse-ului sunt de asemenea afișate în caseta INPUT.

- Dacă utilizatorul face un *click mouse* pe o zonă mapată a imaginii, un mesaj corespunzător, descriind numele zonei, este afişat într-o casetă popup *ALERT*. În caz contrar (*click* pe o zonă din afara hărții), un mesaj de avertizare, însotit de coordonatele *mouse*-ului, este de asemenea afişat într-o casetă JavaScript *ALERT*.

Din punct de vedere al codului, două secvențe distincte pot fi remarcate:

a) Secvența de cod HTML generând fundalul "hartă" al imaginii, respectiv zona de afișare a unor mesaje într-o caseta HTML INPUT. De remarcat folosirea unei etichete ancoră *<A>*, al cărei atribut *href* (referință hipertext - setat în mod normal spre un alt fișier script apelat), referă în cazul de față o funcție JavaScript (integrată în scriptul curent), interceptând orice apăsare de *mouse* (eveniment implicit specific unei referințe ancoră *<A>*). Evident, în această secvență, elementelor HTML le sunt atașate evenimente, respectiv apele ale unor funcții JavaScript adecvate funcționalităților dorite.

b) Secvența de cod JavaScript, conținând o declarație de două variabile (vizibile în toate funcțiile din cadrul scriptului), respectiv patru funcții JavaScript, fiecare cu o anumită funcționalitate.

În continuare este descris rolul fiecăreia dintre cele patru funcții JavaScript integrate în cadrul aplicației:

Funcția *miscare()* este prima apelată în momentul detectării unei mișcări a *mouse*-ului, citind și stocând în permanență coordonatele curente ale *mouse*-ului (*x_current=event.x;* *y_current=event.y*) în două variabile publice, accesibile în tot codul JavaScript. Integrat în cadrul codului acestei funcții, se realizează un apel al unei a doua funcții numită *in_zona*.

Funcția *in_zona*, având drept parametri de intrare coordonatele curente ale *mouse*-ului, respectiv coordonatele a două puncte care definesc o anumită zonă vizată pe imagine (deci în total, 6 parametri), returnează o valoare booleană TRUE sau FALSE, după cum coordonatele curente al *mouse*-ului definesc un punct din interiorul, respectiv din afara zonei definite prin două puncte (corespunzând celorlalți patru parametri ai ei). Operatorul *&&* are semnificația unui *ȘI* logic, primii doi parametri ai funcției fiind coordonatele *x* și *y* ale *mouse*-ului, iar următorii parametri fiind coordonatele *x* și *y* ale celor două colțuri opuse (stângasus și dreapta-jos) definind o zonă dreptunghiulară de mapare pe imagine.

Funcția *miscare()* verifică, printr-o structură, *if-else*, încadrarea pointerului *mouse*-ului în una din cele trei zone distinct mapate pe imagine. În caz afirmativ, se apelează o funcție *afiseaza_text* care va afișa în caseta INPUT un mesaj corespunzător. În caz contrar (ramura *else*), se va afișa un mesaj adecvat, incluzând și coordonatele curente ale *mouse*-ului.

Și în fine, ultima funcție *apasare_mouse* face și ea apel la funcția care verifică încadrarea coordonatelor curente ale *mouse*-ului în zonele definite/mapate pe imagine (apelând funcția *in_zona* într-o structură condițională *if-else*), iar funcție de rezultatul boolean returnat, va afișa mesajele adecvate, folosind de data aceasta o casetă JavaScript *ALERT*.

După cum s-a mai menționat, aplicația anterioară funcționează corect doar pe browsere din familia Internet Explorer, fiind o dovadă a problemelor de compatibilitate cu browserul pe care le ridică o operare cu JavaScript.

Dacă se dorește o rulare și pe browsere din familia Mozilla, codul anterior trebuie modificat corespunzător, astfel [33]:

- Etichetei *IMG* i se atribuie un nume pentru a putea fi referită în JavaScript:
**

- Scriptul JavaScript se modifică /completează astfel:

Direct în secvență publică executată la încărcarea aplicației se adaugă linia:
document.ceva.onmousemove = miscare1;
 unde *miscare1* este o nouă funcție (construită pe scheletul deja existentei funcții *miscare*), fiind apelată doar de browsere Mozilla la o deplasare a *mouse*-ului în interiorul paginii, doar pe obiectul cu numele *ceva* (practic doar pe imaginea referită prin eticheta *IMG*). Astfel, scriptul JavaScript va debuta cu secvența următoare:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
    var x_curent, y_curent;
document.ceva.onmousemove = miscare1;
// document.onmousemove = miscare;
```

Dacă se folosește linia comentată, funcția *miscare1* interceptează coordonatele *mouse*-ului de pe tot ecranul (și nu doar de pe zona limitată, impusă de coordonatele etichetei *IMG* cu numele *ceva*).

Funcția *miscare1* se construiește pornind de la funcția *miscare* existentă, modificându-se liniile citind coordonatele curente ale *mouse*-ului (parametru de intrare *e* având semnificația evenimentului *onmousemove* interceptat și având o acțiune definită prin secvența de cod JavaScript anterior precizată):

```
function miscare1(e)
{
    x_curent=e.pageX;
    y_curent=e.pageY;
    . . .
```

Restul codului aplicației rămâne nemodificat. Problema majoră rezolvată prin modificările anterioare a fost cea de a citi coordonatele curente ale *mouse*-ului utilizând o secvență de cod JavaScript compatibilă cu browsere Mozilla, completând totodată scriptul astfel încât aplicația să poată rula adecvat pe ambele tipuri de browser Web.

Observație: Deseori utilă pentru rezolvarea unor probleme de compatibilitate ale limbajului JavaScript cu diverse tipuri de browsere, următoarea secvență de cod permite o identificare și afișare a numelui unui browser (utilizând funcția

JavaScript **navigator.appName** [36]):

```
<html> <body>
<div id="afis"></div>
<script type="text/javascript">
nume = "Nume Browser: " + navigator.appName + "</p>";
document.getElementById("afis").innerHTML=nume;
</script>
</body></html>
```

Codul anterior afișează **Netscape** ca nume pentru browser-ele familiei Mozilla , respectiv Microsoft Internet Explorer pentru cealaltă familie.

Există o mulțime de evenimente JavaScript, fiecare putând fi atașate diverselor elemente-eticheți HTML, conducând la o interacțiune dinamică a acestora cu utilizatorul. Tabelul 3.1 prezintă doar o foarte scurtă selecție a unor astfel de evenimente, numite și evenimente HTML DOM (*Document Object Model*), precizându-se acțiunea la care sunt declanșate, respectiv eticheta HTML de care pot fi atașate (multe dintre aceste evenimente putând fi comune mai multor etichete HTML).[34][35]:

Tabel 3.1

Etichetă HTML	Atribut - Eveniment	Descriere
INPUT	<i>onclick</i>	Click mouse pe casetă
	<i>onblur</i>	Pierdere focus casetă (<i>onLeave</i>)
	<i>onmousemove</i>	Deplasare mouse pe caseta INPUT
	<i>onkeypress</i>	Apăsare și lăsare tastă în casetă
	<i>ondblclick</i>	Dublu click în caseta
FORM	<i>onclick</i>	Click mouse pe formular (pe orice element al acestuia)
	<i>onkeydown</i>	Apăsare o tastă în formular (pe orice element al acestuia)
	<i>onsubmit</i>	Apăsare buton SUBMIT
	<i>onmouseover</i>	Pointer mouse se mișcă deasupra formularului (deasupra oricărui element din formular)
SELECT	<i>onchange</i>	Schimbare opțiune (selecție)
	<i>onmousemove</i>	Deplasare mouse pe caseta SELECT
	<i>onfocus</i>	Focus pe o anumită opțiune
TEXTAREA	<i>onclick</i>	Click mouse pe casetă
	<i>onmouseout</i>	Pointer mouse părăsește casetă

3.1.5. Transfer de informație între HTML și JavaScript

3.1.5.1. Citire valori etichete INPUT în variabile JavaScript

Una dintre problemele care apar la operarea cu JavaScript este modul în care conținutul/valorile unor elemente/etichete HTML se leagă (comunică) cu variabilele JavaScript. Prezentul paragraf tratează, cazul în care se dorește o preluare a conținutului unor casete INPUT (procedura putând fi aplicată și altor etichete HTML) și transferarea lui spre variabile JavaScript. În acest sens, cinci exemplificări privind referirea în JavaScript a conținutului unor etichete INPUT sunt prezentate și comentate, astfel:

- primele trei corecte și funcționale - evidențierind mai multe forme de referire în JavaScript a etichetelor HTML;
- una parțial funcțională - evidențierind dependența de browser;
- una incorectă - evidențierind caracteristica *case-sensitive* a limbajului JavaScript.

Codul exemplificativ este următorul (comentariile însoritoare fiind relevante pentru acțiunea lui):

```
<form name="myForm" method="GET" action="a.php"
onsubmit="functie()">
Name: <input type="text" id="nume" />
Time: <input type="text" name="timp"/>
<input type="SUBMIT" value="Query" />

<script language="javascript" type="text/javascript">
function functie()
{
//Corect (folosire ID în loc de NAME ca proprietate)
var var1=myForm.nume.value;
alert("ceval "+ var1);

//Corect:
var var1=document.getElementById("nume").value;
alert("ceva2 "+ var1);

//Corect:
var var1=myForm.timp.value;
alert("ceva3 "+ var1);

//Eroare pe Mozilla (proprietatea NAME nu substituie proprietatea ID),
//Funcțional doar pe Internet Explorer!
var var1=document.getElementById("timp").value;
alert("ceva4 "+ var1);

//Gresit (case-sensitive!)
}
```

```

var var1=myForm.Timp.value;
// sau tot greșit: var var1=myForm.timp.VALUE;
alert("ceva5 "+ var1);

//Dupa prima eroare, restul de cod nu se mai execută!
}
</script>

```

Pentru depanarea codului JavaScript, browser-ele dispun de facilități care semnalează eventuale erori și linia de cod în care apar (fie prin afișarea unor mesaje de eroare – cazul Internet Explorer, fie prin afișarea unei console de erori – cazul Mozilla). Figura 3.3 arată modul cum se poate afișa consola erorilor (pentru Mozilla), iar figura 3.4 detaliază, pentru scriptul anterior comentat, eventuale mesaje de avertizare, precum și prima eroare apărută (funcționarea scriptului întrerupându-se după aceasta).

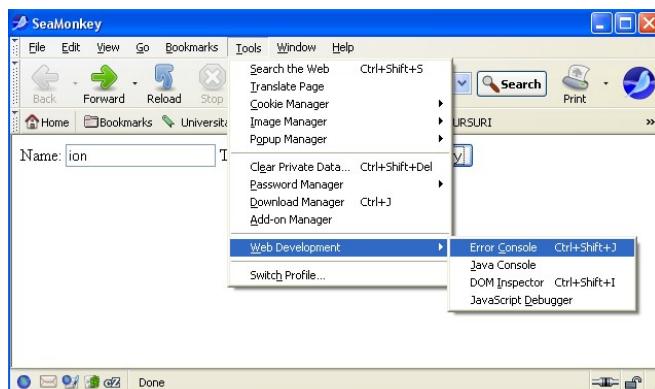


Fig. 3.3 Error Console (Mozilla)

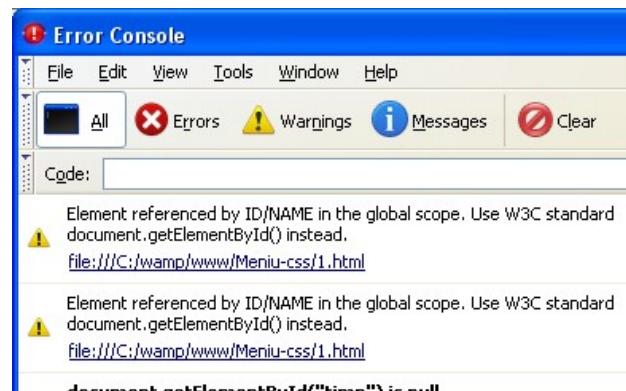


Fig. 3.4 Eroarea semnalată

3.1.5.2. Setare atribute etichete HTML cu conținut variabile JavaScript

Operație inversă celei tratate în paragraful anterior, constă în setarea unui atribut corespunzător (atribut de conținut) al unei etichete HTML (*INPUT*, *IMG*) cu valoarea unei variabile JavaScript.

Pentru exemplificare se consideră o aplicație Web care la apăsarea unui buton permite afișarea aleatoare a unei imagini dintr-un set predefinit de patru imagini (folosind eticheta *IMG*), respectiv afișarea în cadrul unor casete text *INPUT* a conținutului unei variabile indice, respectiv a conținutului elementelor unui sir (*array*) JavaScript (fig. 3.5).

În momentul startării aplicației, evenimentul ***onload*** (atașat etichetei ***body***), apelează funcția JavaScript ***initializare()***. De asemenea, în codul JavaScript sunt declarate două variabile sir folosind constructorul ***Array()*** (vizibile în toate funcțiile, unul dintre siruri fiind și inițializat). Funcția ***initializare()*** creează patru elemente de tip imagine folosind constructorul ***Image()***, atribuie elementelor sirului neinițializat (anterior declarat în secțiunea publică), setând atributul ***src*** al fiecărei imagini spre un anume fișier *jpg*.

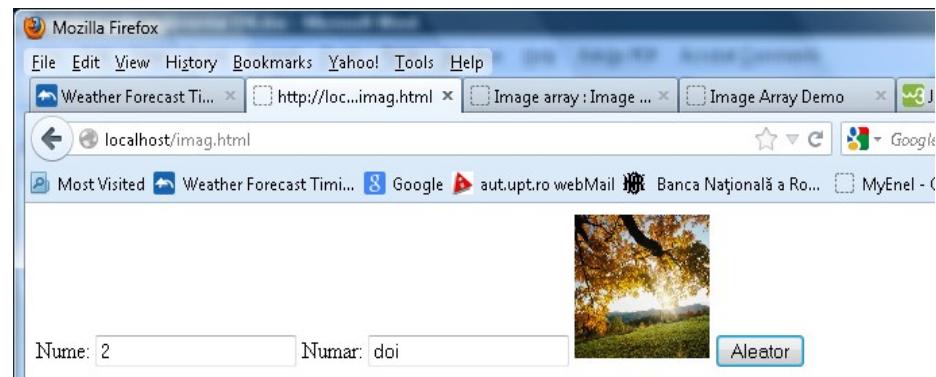


Fig. 3.5 Setare atribute etichete HTML cu valori variabile JavaScript

La apăsarea unui buton este apelată funcția ***afisez()*** care, generând aleator un număr întreg (în intervalul 0...3) folosit ca indice pentru referirea unor elemente din cele două siruri, permite afișarea unei imagini (folosind eticheta *IMG*), respectiv setarea unui conținut pentru cele două casete *INPUT* (vezi fig. 3.5).

Codul complet al aplicației este următorul (comentările suplimentare integrate în el fiind elocvente, fără a mai fi necesare alte explicații):

```

<body onload="initializare()">
<form name="myForm">
Nume: <input type="text" id="nume" >
Numar: <input type="text" name="numar" >
<img src = "1.jpg"

```

```

name = "imagine"
height = 100
width = 100>

<input type = "button" value = "Aleator" onClick =
"afisez()">
</form>

<script language="javascript" type="text/javascript">

    //declarații, inițializări vizibile în tot codul JavaScript
var descriere= new Array("zero", "unu", "doi", "trei");
var imagini = new Array();

function initializare()
{
    imagini[0] = new Image(50, 50);
    imagini[0].src = "1.jpg";
    imagini[1] = new Image(50, 50);
    imagini[1].src = "2.jpg";
    imagini[2] = new Image(50, 50);
    imagini[2].src = "3.jpg";
    imagini[3] = new Image(50, 50);
    imagini[3].src = "4.jpg";
}

function afisez()
{
    //random - numere intre 0 si 1, floor - rotunjire la cel mai apropiat intreg
    var contor=Math.floor(Math.random() * 4 );

    document.imagine.src = imagini[contor].src;
    document.myForm.numar.value = descriere[contor];
    myForm.nume.value = contor; //ambele forme (cu și fără document)
}
</script>

```

Relevant pentru obiectivul paragrafului – setarea atributelor unor etichete HTML cu valori ale unor variabile JavaScript – sunt liniile de cod următoare:

```

document.imagine.src = imagini[contor].src;
document.myForm.numar.value = descriere[contor];
myForm.nume.value = contor;

```

Ca o concluzie privind modalitățile de transfer bidirectional de informație între HTML și JavaScript (mai precis între atrbute ale unor etichete HTML și variabile JavaScript), tabelul 3.2 prezintă o sinteză relevantă în acest sens:

Tabel 3.2

<u>Citire valori etichete HTML (INPUT) în variabile JavaScript</u>									
- etichete HTML referite prin id sau name	...id="nume"								
- etichete plasate într-un FORM myForm	...name="nume"								
<table border="1"> <thead> <tr> <th>...id="nume"</th><th>...name="nume"</th></tr> </thead> <tbody> <tr> <td>var1=myForm.nume.value;</td><td>var1=myForm.nume.value;</td></tr> <tr> <td>var1= document.getElementById("nume").value;</td><td>var1= document.getElementById("nume").value;</td></tr> <tr> <td colspan="2" style="text-align: center;">* Doar pe Internet Explorer (Nu și pe Mozilla !!!)</td></tr> </tbody> </table>		...id="nume"	...name="nume"	var1= myForm.nume.value;	var1= myForm.nume.value;	var1= document.getElementById("nume").value;	var1= document.getElementById("nume").value;	* Doar pe Internet Explorer (Nu și pe Mozilla !!!)	
...id="nume"	...name="nume"								
var1= myForm.nume.value;	var1= myForm.nume.value;								
var1= document.getElementById("nume").value;	var1= document.getElementById("nume").value;								
* Doar pe Internet Explorer (Nu și pe Mozilla !!!)									
<u>Setare (scriere) atribute elemente HTML (INPUT, IMG, DIV) cu conținut variabile JavaScript</u>									
...id="nume" sau ...name="nume" (unde e posibil) sau ...name="imagine" document.myForm.nume.value = variabila; myForm.nume.value = variabila;									
document.imagine.src = figura.src; // (figura = new Image(50, 50); figura.src="ceva.jpg");									
var elem_div = document.getElementById('nume_DIV'); elem_div.innerHTML = variabila;									
<i>sau direct:</i>									
document.getElementById('nume_DIV').innerHTML = variabila;									

Observație: JavaScript este CASE-SENSITIVE (inclusiv la referirea numelui atributelor unor elemente HTML!). Exemplu: atributul *value* – corect, *VALUE* – incorrect.