Tartalomjegyzék

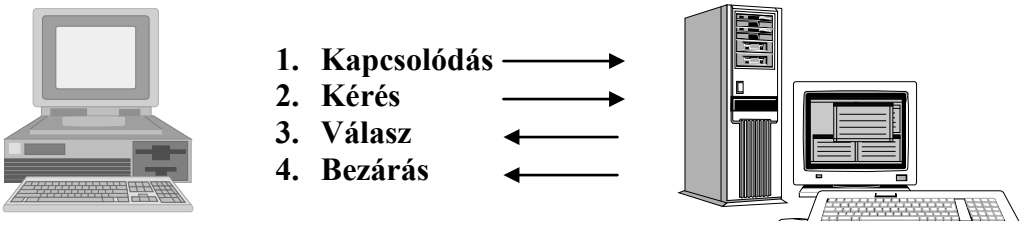
CGI - Elméleti alapok

A dinamikus adattartalom szükségessége

A WEB immár nem statikus adattároló közeg, dinamikussá vált. Ahogyan a WEB információtartalma növekedni kezdett, vált inkább szükségessé a dinamikus tartalomszolgáltatás. Gondoljunk csak egy e-shop-ra, amely 1000 terméket tart nyilván. Hogy a felhasználók mindegyik terméket megtekinthessék (megrendelhessék, esetleg módosíthassák), minimum 1000 különböző weblapot kell készítenünk – még belegondolni is szörnyű! Nem egyszerűbb egy dinamikus weblap létrehozása, amely „röptében" állítja elő a kliens által kért információs oldalt egy adatbázis rekordjaiból? Ebben nyújtanak kiváló segítséget a szerver-oldali programok, scriptek, a CGI-k. A **CGI (Common Gateway Interface)** az a szabvány, amely meghatározza azokat az adatformátumokat, amelyeket a böngészők, kiszolgálók és programok az ilyen információcserékhez használnak. Egy CGI parancsállomány (script fájl) olyan – majdnem bármilyen programozási nyelven (C/C++, Perl, PHP, TCL, VBScript és mások) megírt – program, amely feldolgozza a felhasználó által bevitt adatokat, és – szükség esetén – ezek alapján elkészíti a választ (HTML dokumentum formájában).

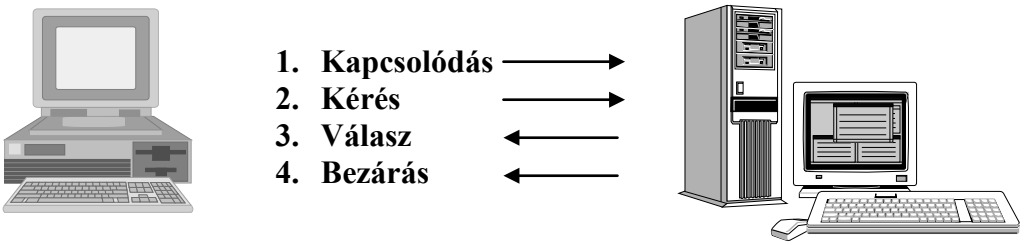
A CGI helye a kommunikációs modellben

Mint tudjuk, amikor a böngésző egy kiszolgálóval kommunikál, négylépéses HTTP tranzakciót hajt végre az alábbi módon:



Kliens böngésző *HTTP kiszolgáló*

Hogyan illeszkedik ebbe a struktúrába a CGI? Lássuk:



Kliens böngésző
HTML

HTTP kiszolgáló
CGI script file-ok

Mint láthatjuk, a CGI fájlok a HTTP kiszolgálón találhatók, így a kiszolgáló és a programok közvetlenül kommunikálhatnak egymással. Ez a kommunikáció teszi lehetővé, hogy egy CGI script fájl dinamikusan kapjon adatokat, és továbbítsa azokat a kiszolgáló felé. Az igazsághoz tartozik, hogy nem kell feltétlenül CGI script fájlokat írnunk ahhoz, hogy dinamikus HTML állományokat hozzunk létre, hiszen erre képesek a kiszolgálók is. A problémák csak akkor jelentkeznek, amikor egy új, dinamikus weblap létrehozására van szükség, hiszen akkor át kellene írni a kiszolgáló programját, amely kaotikussá tenné, és hatalmas méretűvé duzzasztaná a kiszolgáló programját.

Az előző ábrából látszik, hogy közvetlenül a böngészőből nem hajthatjuk végre a CGI script fájlt. Ehhez el kell helyezni a scriptet egy kiszolgálóra, majd a böngészőnkből meghívva azt, láthatjuk a script kimenetét. FONTOS! Nem a scriptet látjuk, csupán a kimenetét, amelyet a HTML protokoll szerint generál a script. Ez általában text/html formátumú kimenetet jelent, de akár grafikus eredményt is produkálhat, mint például egy grafikon, vagy egy grafikus számjegy, amelyet például számlálónál használhatunk.

Standard input, standard output

Amikor a böngésző elküld valamilyen információt a CGI scriptnek, akkor voltaképpen egy POST kérést küld a kiszolgálónak. Ezek az információk a script fájl ***stdin* (standard input)** fájlleírójából érkeznek. A kiszolgáló beállítja a script fájlnak a CONTENT_LENGTH környezeti változót, amely a bemenő adatok bájtokban mért hosszát tartalmazza. A script ez alapján tudja eldönteni, hogy mennyi adatot kell feldolgoznia a *stdin* fájlleíróból. Ugyanakkor a kiszolgáló a CONTENT_TYPE környezeti változót is definiálja, amely alapján a script eldöntheti, hogy miként kezelje az érkező adatokat. Ezen adatfolyam végén a kiszolgáló elküldhet egy fájlvége jelet is.

Ha például egy űrlap a HTTP POST metódust használja és a kiszolgálóra küldött adat *nev=t123&server=torpilla* formában van kódolva, akkor a kiszolgáló a CONTENT_LENGTH és a CONTENT TYPE környezeti változókhoz az alábbi értéket rendeli:

```
CONTENT_LENGTH      =       24
CONTENT_TYPE        =       application/x-www-form-urlencoded
```

Miután a CGI script végzett a kiszolgálóról a bemenetére küldött adatok feldolgozásával, vissza kell küldenie a kimenetét a kiszolgálónak. Ez egyszerűen úgy történik, hogy a kimenő (kiküldendő) adatot elküldik a ***stdout* (standard output)** fájlleíróba. A CGI script által a kiszolgálóra küldött adatok formátuma rendszerint HTTP válasz, amely fejlécből, az ezt követő üres sorból (!), majd a script fájl többi adataiból áll, tehát egy szabványos HTML dokumentum, amelyet maga a script fájl generál. Normál esetben ezek olyan kimenetek, amelyeket – miután megkapta a kiszolgáló – értelmez, majd visszaküld a böngészőnek. Ennek az az előnye, hogy a script fájlnak nem kell minden egyes kéréskor a teljes HTTP fejlécet elküldenie. Vannak viszont olyan scriptek, amelyek a kiszolgáló megkerü-

lésével, azonnal a böngészőnek küldik vissza a kimenetet. Ilyen esetekben a script fájl felelős azért, hogy érvényes HTTP választ küldjön a böngészőnek. A CGI protokoll szabályai szerint az ilyen fájlok neve ***nph-*** betűkkel kell kezdődjön (**Non-Parsing Header** – nem elemzendő fejléc).

A CGI és az adatbázisok

A CGI script fájlok egyik legelterjedtebb alkalmazási területe az adatbáziskezelés. Ehhez természetesen szükség van egy adatbázis-szerverre, hogy a kéréseket teljesíteni tudja. Ebből következik, hogy a scriptek csak voltaképpen az adatbázis és a kiszolgáló közötti kapcsolatot valósítják meg. Kéréseket továbbítanak, megfelelő struktúrába rendezett adatokat kapnak és dolgoznak fel, majd küldenek vissza a kiszolgálónak. Ezzel később behatóbban is fogunk foglalkozni, hiszen ennek látjuk majd legnagyobb hasznát a dinamikus adattartalom létrehozásában.

Fejlesztői környezet

Szükséges hozzávalók egy személyre

Ha hozzá akarunk kezdeni a kódoláshoz, mindenképpen el kell döntenünk pár dolgot: milyen webkiszolgálót fogunk használni, szükségünk lesz-e adatbáziskezelésre, ha igen, milyen adatszervert fogunk használni, továbbá ehhez nem árt eldönteni (bármennyire platformfüggetlen a PHP), hogy milyen platform alatt fogjuk mindezt megvalósítani. Talán a legutolsó a legegyszerűbb: igazán csupán kétféle lehetőség közül tudunk választani: vagy valamelyik Windows alapút (Windows NT, Windows 2000, Windows XP, Windows Server 2003 és a jövendő, szintén legalább 32 bites rendszerek), vagy valamelyik Linux disztribúciót választjuk. Mint láthattuk, a Windows platformú operációs rendszerek közül csak az NT alapúakat soroltam fel. Ez nem jelenti azt, hogy akár a Windows 95/98/Me trió bármelyike nem lenne alkalmas dinamikus weblapok szerkesztésére. Tökéletesen alkalmas, hiszen ha otthon fejlesztünk, nem biztos, hogy szerver-alapú operációs rendszert fogunk használni. A szerveralapú rendszerek viszont lehetővé teszik a *service*-k futtatását, ezen szolgáltatások a rendszer hátterében elfoglalják helyüket és szolgáltatnak – feladatuktól függően.

Ha a Linux mellett döntünk, mindegy melyiket választjuk. Néhány változatukban már eleve be van építve valamelyik webkiszolgáló, esetleg minimális konfigurációval tökéletesen használható fejlesztőkörnyezethez juthatunk.

Részemről, mivel Windows XP-t használok fejlesztéshez, egyszerűbbnek tűnik a Windows alatti megoldás, ami valószínűleg inkább a másik platformról szerzett csökevényes tudásomnak tudható be. De lássuk be: otthon többen használnak Windows rendszert, akinek pedig Linux rendszere van, már meg tud birkózni a telepítés, konfigurálás „mágikus” problémáival. ☺

A következő fontos összetevő a webszerver. Megszámlálhatatlan webszervert készítettek már szerte a világban, mindegyik letölthető a netről, legtöbbjük kereskedelmi termék, de számos freeware változatot is találunk közöttük. Egy részük támogatja a PHP-t, míg mások nem. Természetes, hogy a PHP támogatás fogja eldönteni, melyiket választjuk. Szépen szűkült a kör, de még mindig nem eléggé. Mindegyik sokat tud a maga módján, én mégis egy kereszt-platformos webkiszolgálónál maradnék, amelyik ráadásul még ingyenes is… Ez az Apache. Könnyen telepíthető, rugalmas, kicsi, gyors. Jelen pillanatban a 2-es verziónál tartanak a fejlesztésben, amit érdemes letölteni, bár sokan esküsznek az 1.3.x-es verzióira. Tapasztalatain szerint érdemes használni az Apache2-t, de akkor mindenképpen a PHP-ből is minimum 4.3-as verzióra van szükség. Az Apache a http://www.apache.org címről tölthető le mind Windows, mind Linux platformra. Mivel a másik nagy rivális a webszerverek piacán a Microsoft-féle Internet Information Server (IIS), a PHP teljes vállszélességgel támogatja azt is, használatakor semmi fennakadás nem tapasztalható.

Hol is tartunk? Van már operációs rendszerünk, egy működő webkiszolgálónk, illene beszerezni hozzá a PHP, no meg egy adatbázisszerver sem ártana. A

PHP-vel csak annyi dolgunk van, hogy meglátogatjuk a PHP hivatalos oldalát: http://www.php.net. Innen letölthetjük a bináris állományokat, de ha a forráskódra van szükségünk, azt is megtaláljuk. Más kérdés az adatbázisszerver. Szinte magától adódik a megoldás, hogy valamilyen SQL alapú szervert használjunk. A termékskála itt is széles: ingyenestől egészen a több milliós szerverig, minden van a piacon. Vigyázat! Nem mindig a drága a jobb! A PHP-be sokféle adatbázisszerver támogatása be van építve: Adabas D, dBase, Informix, Interbase, mSQL, MySQL, Oracle, PostgreSQL, Sybase, Unix dbm, hogy csak a legfontosabbakat említsük. Ha azt halljuk: PHP, szinte biztosan felmerül mellette egy másik név is: MySQL. Talán a MySQL-hez van legerőteljesebb támogatása a PHP-nek, bár a most megjelenő ötös bétaverzióból éppen kihagyni készülnek bizonyos jogi problémák miatt…

Most foglaljuk össze, mivel fogunk az elkövetkező jónéhány oldalon foglalkozni, milyen eszközökre lesz szükségünk:

operációs rendszer	Bármelyik NT alapú operációs rendszer (p. Windows XP)
webkiszolgáló	Apache2
PHP	PHP 4.3.0
Adatbázisszerver	MySQL 4.x

Ha megtekintjük a fenti táblázatot, láthatjuk, hogy az operációs rendszert leszámítva, teljesen ingyenes fejlesztőkörnyezetet alakítottunk ki magunknak. S ha operációs rendszerként mégis a Linux mellett döntünk, teljesen ingyenesen fejleszthetünk.

Telepítés, konfigurálás

Ha sikeresen letöltöttük a PHP-t egy bináris csomagot kapunk. Ezt kell feltelepítenünk az operációs rendszer alá. Attól függően, hogy mit töltöttünk le, kétféle utat választhatunk.

FONTOS! Ha Windows 95 alatt akarjuk használni a PHP-t, akkor győződjünk meg arról, hogy letöltöttük a DCOM frissítést a Microsoft DCOM oldaláról. Ennek címe:
http://download.microsoft.com/msdownload/dcom/95/x86/en/dcom95.exe

Automatikus telepítés

Az InstallShield telepítőprogrammal nagyon könnyű életre kelteni a PHP-t, de ez nagyon sok szempontból korlátozott változat, például a kiterjesztések automatikus telepítését nem végzi el.

Futtassuk a telepítő EXE fájlt, követve a varázsló által adott utasításokat. Kétféle telepítés közül választhatunk: a *standard* telepítés jól használható alapbeállításokat ad, az *advanced* kérdéseket tesz fel (amelyekre tudni kell válaszolni ☺).

A telepítés varázslója elég információt gyűjt ahhoz, hogy elvégezhesse a **php.ini** fájl beállítását és konfigurálja a szervert a PHP számára. Mikor a telepítés befejeződött, a varázsló informál arról, hogy szükséges-e a rendszer, ill. a szerver újraindítása, vagy rögtön elkezdhetjük a munkát a PHP-vel.

Legyünk tekintettel arra, hogy a PHP ezen telepítési módja nem biztonságos. Ha biztonságos PHP után vágyódunk, jobban tesszük, ha a kézikönyv erről szóló fejezeteit is elolvassuk, és minden beállítást körültekintően elvégzünk. Ez az automatikus telepítő egy azonnal használható PHP-t varázsol a gépünkre, de nem online szerverekre szánták.

Kézi telepítés

A Windows-os PHP4 disztribúciók kétféle formában kaphatók - CGI futtató (php.exe) és számos SAPI modulként (például php4isapi.dll). Az utóbbi elég új a PHP 4-ben, és jelentős teljesítményjavulást és néhány új lehetőséget biztosít az előzőhöz képest. A következő lépéseket minden telepítés esetén a szerver specifikus beállítások előtt kell végrehajtani.

Tömörítsük ki a disztribúciós fájlt egy tetszőleges mappába. Ez általában a C:\PHP\ mappa, a következőkben is erre fogok hivatkozni.

Biztosítani kell, hogy a PHP az általa használandó DLL-eket meg is találja. Hogy milyen DLL-ekre vonatkozik ez, az függ attól, hogy a milyen webszervert használsz és, hogy azon a PHP modulként vagy CGI-ként fog futni. A *php4ts.dll* minden esetben szükséges. Ha szerver modulként (pl. Apache vagy ISAPI) hasz-

nálod, akkor a sapi alkönyvtárból a megfelelő DLL kell. Hogy ezek a DLL-ek elérhetőek legyenek, bemásoljuk őket a rendszerkönyvtárba (pl. winnt/system32 vagy windows/system), vagy rakhatjuk egy könyvtárba a fő PHP állománnyal (pl. php.exe, php4apache.dll). Ez a művelet legtöbbször annyiban merül ki, hogy a *php4ts.dll* nevű állományt a rendszerkönyvtárba kell másolni.

Két ini fájl került bele a zip-be: *php.ini-dist* és *php.ini-optimized*. Azt javaslom, hogy a *php.ini-optimized*-t használjuk, mert azokban az alapértékek úgy kerültek meghatározásra, hogy minél jobb teljesítménnyel és nagyobb védelemben fussanak programjaink. Szóval ez az állományt másoljuk át a Windows mappánkba (alapértelmezésben: c:\windows, vagy c:\winnt), majd ott nevezzük át ***php.ini***-re.

Most jöhetnek a konkrét beállítások:

- Az 'extension_dir' bejegyzést át kell írni, hogy az arra a könyvtárra mutasson, ahol a 'php_*.dll' fájlok vannak, pl.: 'c:\php\extensions'.
- Állítsuk be a 'doc_root'-ot, hogy az a webszerverünk document_root könyvtárára mutasson, például c:\apache\htdocs vagy *c:\webroot.* Ezt a b eállítást (hivatalosan) már az Apache feltelepítésekor meghatároztuk.
- Válasszuk ki, mely kiterjesztéseket töltse be a PHP induláskor. Ehhez érdemes a hivatalos weblapon tájékozódni a kiterjesztések mibenlétéről, általában az alapbeállításokkal tökéletesen meg leszünk elégedve, csak speciális esetekben lesz szükségünk ezen kiterjesztések használatára. Hogy mégis (alapjaiban) milyen kiterjesztések közül választhatunk, s hogy melyik mire használható, íme egy táblázat (ijesztésképpen):

Kiterjesztés	Leírás	Megjegyzés
php_bz2.dll	bzip2 tömörítő függvények	nincs
php_calendar.dll	Naptár átváltási függvények	PHP 4.0.3 óta beépített
php_cpdf.dll	ClibPDF függvények	nincs
php3_crypt.dll	Crypt függvények	-
php_ctype.dll	ctype féle függvények	nincs
php_curl.dll	CURL, Client URL library függvények	szükséges: libeay32.dll, ssleay32.dll (része a disztribúciónak)
php_cybercash.dll	Cybercash payment függvények	nincs
php_db.dll	DBM függvények	ellenjavallt, DBA-t használd helyette (php_dba.dll)
php_dba.dll	DBA: (dbm jellegű) adatbázis absztrakciós réteg függvényei	nincs
php_dbase.dll	dBase függvények	nincs
php3_dbm.dll	Berkeley DB2 eljáráskönyvtár	-
php_domxml.dll	DOM XML függvények	szükséges: libxml2.dll (része a disztribúciónak)
php_dotnet.dll	.NET függvények	nincs
php_exif.dll	EXIF fejlécek olvasása JPEG-ből	nincs
php_fbsql.dll	FrontBase függvények	nincs
php_fdf.dll	FDF: Forms Data Format függvények.	szükséges: fdftk.dll (része a disztribúciónak)
php_filepro.dll	filePro függvények	csak olvasható hozzáférés-sel
php_ftp.dll	FTP függvények	PHP 4.0.3 óta beépített
php_gd.dll	GD eljáráskönyvtár képkezelő függvényei	nincs
php_gettext.dll	Gettext függvények	szükséges: gnu_gettext.dll (része a disztribúciónak)
php_hyperwave.dll	HyperWave függvények	nincs

php_iconv.dll	ICONV characterset conversion	szükséges: iconv-1.3.dll (része a disztribúciónak)
php_ifx.dll	Informix függvények	szükséges: Informix eljáráskönyvtárak
php_iisfunc.dll	IIS kezelési függvények	nincs
php_imap.dll	IMAP POP3 és NNTP függvények	PHP 3: php3_imap4r1.dll
php_ingres.dll	Ingres II függvények	szükséges: Ingres II libraries
php_interbase.dll	InterBase függvények	szükséges: gds32.dll (része a disztribúciónak)
php_java.dll	Java extension	szükséges: jvm.dll (része a disztribúciónak)
php_ldap.dll	LDAP függvények	szükséges: libsasl.dll (része a disztribúciónak)
php_mhash.dll	Mhash Functions	nincs
php_ming.dll	Ming függvények for Flash	nincs
php_msql.dll	mSQL függvények	szükséges: msql.dll (része a disztribúciónak)
php3_msql1.dll	mSQL 1 client	-
php3_msql2.dll	mSQL 2 client	-
php_mssql.dll	MSSQL függvények	szükséges: ntwdblib.dll (része a disztribúciónak)
php3_mysql.dll	MySQL függvények	PHP 4 óta beépített
php3_nsmail.dll	Netscape levelező függvények	-
php3_oci73.dll	Oracle függvények	-
php_oci8.dll	Oracle 8 függvények	szükséges: Oracle 8 kliens eljáráskönyvtárak
php_openssl.dll	OpenSSL függvények	szükséges: libeay32.dll (része a disztribúciónak)
php_oracle.dll	Oracle függvények	szükséges: Oracle 7 client kliens eljáráskönyvtárak
php_pdf.dll	PDF függvények	nincs
php_pgsql.dll	PostgreSQL függvények	nincs
php_printer.dll	Printer függvények	nincs
php_sablot.dll	XSLT függvények	szükséges: sablot.dll (része a disztribúciónak)
php_snmp.dll	SNMP get and walk függvények	csak NT -n!
php_sybase_ct.dll	Sybase függvények	szükséges: Sybase kliens eljáráskönyvtárak
php_yaz.dll	YAZ függvények	nincs
php_zlib.dll	ZLib tömörítő függvények	nincs

Beillesztés

Ezzel még nem vagyunk ám készen! Szükséges lesz annak meghatározása, hogy a kiszolgáló (Apache) felismerje a PHP-ban írt sorokat, s végre is hajtsa azokat. Ha nem tennénk, a kiszolgáló megpróbálná egyszerű HTML forrásként értelmezni a PHP programjainkat, annak pedig kevés (használható) eredménye lenne. Ennek egyetlen módja van: meg kell mondanunk az Apache-nak, hogy a PHP alá van telepítve, s hogy tessék figyelni rá. Ehhez az Apache *http.conf* nevű állományát kell egy picit átírnunk.

Mielőtt továbbmennénk, álljunk meg egy pillanatra! Két módja van a PHP Windows-on futó Apache alá telepítésének. Az egyik a CGI kezelőként futtatható php.exe, a másik az Apache modulként használandó DLL. Ha az elsőt választjuk, minden egyes munkafolyamat esetén be fog töltődni a PHP.EXE a memóriába, értelmezi a szükséges scriptet, majd kitörli magát onnan. Ennek előnye, hogy nem marad az értelmező a memóriában. Hátránya, hogy minden egyes alkalommal be kell tölteni. A profik (ahogy mi is) a modulos módszert használják. Itt a php modulként beül a memóriába, ha szükség van rá, akkor végrehajtja a feladatát. Hátránya, hogy foglalja a memóriát, de előnye, hogy csak egyszer kell betölteni a memóriába. Lássuk, mit is kell tegyünk:

Írjunk be pár sort a fent említett *http.conf* állományba:

```
LoadModule php4_module c:/php/sapi/php4apache.dll
AddType application/x-httpd-php .php
```

Készen is vagyunk, az Apache újraindítása után (kötelező!) a kiszolgáló tudni fog a PHP létezéséről, elkezdhetjük a munkát. Vagy mégsem?

SQL támogatás

Nem bizony! Addig nem fogjuk elkezdeni a munkát, amíg nincs feltelepítve egy SQL szerver. Mint említettük, ehhez a mysql nevű, ingyenes kiszolgálót fogjuk használni. Töltsük le a http://www.mysql.com -ról a bináris állományt (jelen pillanatban a 4.x-es verziót érdemes használni, bár már az 5.0-s verzió tesztelésénél tartunk), majd telepítsük föl. Általában a C:\MySQL nevű mappába érdemes telepíteni, ezt használja a program alapértelmezésképp. Ha a telepítős állományt töltöttük le, akkor az vezet minket, ha a csomagolt (ZIP) állományt, akkor csak bontsuk ki a C:\ főkönyvtárába az egészet, hagyva, hogy automatikusan létrehozza a szükséges mappákat. Ha készen vagyunk, a C:\MySLQ\bin mappában találunk néhány futtatható állományt. A mysqld*.exe-k a szerverek, ezek közül bármelyiket installálhatjuk, mint Windows szolgáltatást.

A mysqld service-ként történő beállítása legegyszerűbben a ***myslqd.exe -install*** paraméteres futtatással történik. Ezzel beillesztettük a mysql-t a service-k közé, de még nem indítottuk el. Az elindítás a Felügyeleti eszközökben található Szolgáltatások párbeszédablakban történik. Keressük meg a mysql sort, majd

indítsuk a szolgáltatást. Ettől kezdve a gép bekapcsolásakor automatikusa elindul a mysql szolgáltatás is.

Telepítés – összefoglalás

Lássuk, mi van feltelepítve a gépünkre, mire van szükségünk a fejlesztőkörnyezet kialakításához:

- Apache, lehetőleg 2.x-es verzió
- PHP, minimum 4.3-as verzió. Modulként installáltuk az Apache alá, ehhez a *httpd.conf*-ban végeztünk módosítást. Nem felejtettük el átírni a *php.ini* állományt, amely a windows mappában található, továbbá bemásoltuk a *php4ts.dll*-t a rendszerkönyvtárba. Ez általában a \windows\system32 mappa – operációs rendszertől függő.
- MySQL 4.x. Service-ként installáltuk, feltehetőleg a C:\MySQL mappába.

Tesztelés

Teszteljük le a beállításainkat! Ehhez hozzunk létre a *php.ini*-ben meghatározott document root mappában egy *index.php* nevű állományt az alábbi tartalommal:

```
<? phpinfo(); ?>
```

Töltsük be kedvenc böngészőnket, majd gépeljük be a címsorba webszerverünk kezdőoldalát: http://localhost. Ha minden jól ment, egy hatalmas, több oldalas táblázatot láthatunk, amely a PHP szinte minden paraméteréről részletekbe menő jelentést ad. Ha mégsem ez történne, akkor valami elrontottunk, érdemes visszatérni a telepítési fejezet elejére.

Ha sikeresen vettük az első akadályt, ellenőrizzük le, hogy az SQL szerverünk készen áll-e a munkára. Keressük meg a C:\MySQL\bin mappában a *mysql.exe* nevű állományt és indítsuk el. Ha hibaüzenettel tér vissza, miszerint nem tudott csatlakozni a szerverhez, akkor elbaltáztunk valamit, térjünk vissza a kezdetekhez. Ha sikeres volt a bejelentkezés, egy promptot kapunk, amivel most még nem foglalkozunk, egy „quit”, vagy „exit” parancs segítségével ki tudunk lépni a felkínált szerkesztőből.

Eljutottunk idáig? Akkor ez azt jelenti, hogy mindent sikeresen feltelepített, s valószínűleg megértette az eddig leírtakat. Csak akkor haladhatunk tovább a következő fejezetre, ha válaszolni tud a következő kérdésekre:

Összefoglaló kérdések

- Mi a CGI, mire használják?
- Ismertesse a HTTP tranzakció négylépéses modelljét (CGI-vel és anélkül)
- Mit jelentenek: *stdin, stdout*?
- Milyen összetevők szükségesek a PHP-s fejlesztőkörnyezet kialakításához?
- Mire jó az SQL szerver?

PHP alapozás

Ismerkedés

Napjainkban és az elmúlt néhány évben újabb és újabb programozási nyelvek születésének lehetünk (lehettünk) tanúi. Ezeknek az új nyelveknek nagy része a tradicionális, széles körben alkalmazható társaik (C, C++, Pascal) szintaxisát, logikáját veszi alapul, és úgy egészíti ki azokat, hogy valamilyen a nyelv által kitűzött célnak jobban megfeleljen, tehát jobban specializálódjon.

Hogy egy kicsit konkrétabb legyek és a címhez is tartsam magam, ezek közül a programozási nyelvek közül ebben a sorozatban a dinamikus oldalak készítésére használhatóakkal foglakozunk, azon belül is elsősorban az egyre szélesebb körben elterjedő PHP-vel.

Azt már elárultam, hogy mire specializálódott a PHP lássuk, hogy honnan indult, hogyan tett szert ekkora népszerűségre a fejlesztők körében és egyáltalán mekkora is ez a népszerűség.

A PHP születése 1994-re tehető, amikor Rasmus Lerdorf elkészítette első, a nyilvánosság számára nem elérhető verzióját, melyet csupán azért írt, hogy megkönnyítse olyan, egyszerű script-ek írását, mint például egy vendégkönyv vagy számláló. A program 1995 elején Personal Home Page Tools néven kezdett egyre ismertebbé válni. Még ebben az évben újraírta a szerző a program script-feldolgozó (parser) részét és kiadta a 2-es verziót PHP/FI néven. Az FI rész a névben az újonnan beépített form-feldolgozó részből adódott. De nem csak ez az egy extra került bele az új kiadásba: a nyelv immár támogatta az mSQL adatbázisok használatát, amely nagyban hozzájárult ahhoz, hogy újabb emberek kapcsolódjanak a fejlesztésbe, bővítve ezzel a nyelv palettáját.

1997-re már több mint 50 000 szerveren futott a PHP. Ekkor új irány vett a fejlesztés azzal, hogy társultak Rasmus-hoz más fejlesztők is, akik teljes egészében újraírták az interpretert, mely az 1998-ban megjelenő 3.0-s verzió alapja lett.

A nyelv sikerességét azt hiszem legjobban a statisztikai adatok bizonyítják: A NetCraft (http://www.netcraft.com/) felmérése szerint 1998-ban 150 000 domain név alatt futott PHP, ami már önmagában nem kis szám és mondhatni azóta se csökkent, mivel az 1999. novemberi felmérés szerint ez 1 114 021, t ehát jóval meghaladja az egymilliót.

A történetnek azonban még nincs vége, folytatódik a jelenben a PHP 4-es verziójával, melynek a script-értelmezője szintén teljesen újra lett írva, ami minimum 2-3-szoros, de előfordulhat, hogy 200-szoros sebességkülönbséget jelent a PHP3-hoz képest. Kapható lesz hozzá (sajnos nem ingyen) fordítóprogram is, mely a Java-éhoz hasonló hordozható, félig lefordított állapotba hozza a programunkat, ezáltal az még gyorsabban fog futni (3-4-szeres a különbség a PHP4-hez képest!!) és meg is védi a szerzőt attól, hogy mások az engedélye nélkül felhasználják a kódját.

A PHP (hivatalos nevén PHP: Hypertext Preprocesszor) egy szerver oldali HTML-be ágyazott scriptnyelv. Ugye, milyen bonyolultan hangzik? De máris tisztább lesz, ha azt mondom, hasonló a javascripthez. No nem mindenben, sőt! Hogy összehasonlíthassuk őket, ismételjük át – gyorsított tempóban – a javascript néhány, számunkra fontos momentumát.

Ha egy HTML oldalon a megszokott, statikus elemek helyett némi dinamizmust is szeretnék látni, erre nagyszerű lehetőséget biztosít a javascript. Bizonyos megszorításokkal látványos, felhasználóbarát lapokat készíthetünk vele. Nézzünk egy egyszerű példát:

```
<HTML>

<SCRIPT>
	var ora = (new Date()).getHours();

	if (ora<=6 || ora>=20) {
		document.writeln ("Jó estét, ");
	} else if (ora < 10) {
		document.writeln("Jó reggelt, ");
	} else {
		document.writeln("Jó napot, ");
	}
</SCRIPT>

kedves látogatóm!<HR>

</HTML>
```

Ha megnézzük a fentieket, láthatjuk, hogy az aktuális órától (napszaktól) függően az oldal üdvözli látogatóját. Figyeljük meg, hogy a javascript betét a <SCRIPT>…</SCRIPT> tagek közé van ékelve. Ezt hívjuk voltaképpen HTML-be ágyazásnak, hiszen a HTML forráson belül bárhol nyithatunk javascript betétet, legyen az akár egy sor közepe – természetesen a megfelelő szintaktikai szabályok betartásával. A PHP-vel ellentétben a javascriptet az aktuális böngésző hajtja végre, azaz a kliens (felhasználó) gépén fut, tehát függ annak teljesítményétől. A PHP részeket a PHP értelmező a szerveren futtatja le, s csak az eredményt adja vissza a böngészőnek. Ebből származik a PHP egyik nagy előnye: *nem látható a forráskód*! Egy javascriptet használó oldal forráskódját lekérve láthatjuk teljes forrását. Némi tudással bárki átírhatja a rajta található kódot saját képére, s használhatja azt. Ugye senki sem örülne, ha kemény munkával kidolgozott javascript rutinjait valaki kéretlenül használná? A PHP-s oldalak forrását lekérve csupán a generált HTML részt fogják látni a kíváncsi szemek, tehát a PHP rutinok sohasem kerülnek forrásukban képernyőre. (Természetesen, hacsak nem akarom…) Hogy tisztább legyen, erre is lássunk egy nagyon egyszerű példát. A fenti javascriptes oldalrészletet valósítsuk meg PHP-ben. Magyarázatok a megoldás után:

```
<HTML>

<?php
	$ora = date ("g");

	if ($ora<=6 or $ora>=20) {
		echo "Jó estét, ";
	} else if ($ora < 10) {
		echo "Jó reggelt, ";
```

```
        } else {
                echo "Jó napot, ";
        }
?>

kedves látogató!<HR>

</HTML>
```

Láthatjuk, hogy minimális különbségek vannak a javascriptes változattal szemben. Az első, ami feltűnhet, hogy nem <SCRIPT>…</SCRIPT> **tag**eket használunk. A PHP-s részek kezdéséhez a "**<?php**" előtagot használjuk, míg zárásához "?>" zárótagot kell írni. Ami e kettő között van, azt a PHP értelmezője fogja végrehajtani. A második, ami különböző, hogy az "ora" nevű változó előtt mindig szerepel egy "$" jel. Ez bizony kötelező, minden változó dollárjellel kezdődik. Hogy a date("g") mit jelent, egyelőre elégedjünk meg annyival, hogy az aktuális dátum órarészét adja vissza. A többi ugyanaz. Vagyis… Ha lekérnénk mindkét oldal forrását, beigazolódna, amit előzőleg említettem. Első esetben a teljes forrást megkapnánk, míg a PHP-s változat esetében a PHP rész helyett csak az aktuális napszakhoz tartozó köszöntés szövege jelenne meg.

Nos, nem véletlenül írtam az előző mondatot feltételes módban. Ha a javascriptes változatot valaki kimásolja és beilleszti egy HTML oldalba, bármilyen javascriptet ismerő böngészővel hehívva azt, megjelenik az eredmény: a napszaknak megfelelő köszöntés. Próbáljuk ki ugyanezt a PHP változattal. Nem lesz sok köszönet benne, ráadásul, ha lekérjük az oldal forrását, visszakapjuk szóról-szóra a begépelt PHP részletet is. Persze, hiszen a PHP-t nem értelmezi a böngésző! De akkor mivel foglalkoztunk az előző lapokon, amikor a fejlesztőkörnyezetet próbáltuk meg felépíteni? Semmi probléma, nem sokat kell már ügyködnünk, hogy PHP-s betéteket írhassunk. Lássuk! Az előző példákban ún. HTML fájlokat hoztunk létre. Ez független attól, hogy beillesztettünk esetlegesen javascriptet, vagy PHP-t. Vagyis nem egészen. Ma már minden böngészőbe be van építve a javascript értelmezése, elképzelhetetlen e nélkül. A PHP viszont a szerveren hajtódik végre, tehát csak az eredményt (HTML formátumú) kapja vissza böngésző, amit értelmezni fog. Honnan tudja a szerver, hogy most PHP fájlt kapott? Nagyon egyszerűen: megmondjuk neki. Eddig csak htm, vagy html kiterjesztésű állományokkal dolgoztunk. Ezek vagy HTML, vagy javascript anyagot tartalmaztak. Ha olyan oldalt írunk, amely PHP is tartalmaz, ne feledjük el, hogy a fájl kiterjesztése **PHP** kell legyen. Tehát ha nem tartalmaz a fájl PHP-s betétet, nyugodtan lehet a neve *valami.html.* De amint egyetlen sor PHP-t tartalmaz, azonnal nevezzünk át *valami.php*-re.

Első programjaink

Beszéltünk róla, hogy a /root/ könyvtárba kell tenni az elkészített lapjainkat. Kezdjük valami egyszerűvel. Hozzunk létre egy *index.html* nevű állományt a /root/ mappában, s írjunk bele valami izgalmasat:

```
<HTML>

        Hurrá, ez az első programunk!
        <br>
        Még nincs benne semmi PHP, de tesztnek jó!

</HTML>
```

Böngészőprogram betölt, cím http://localhost, s láthatjuk első, saját webszerverünkön futó oldalunkat. Nem sok, csupán két sor, de működik. Kérjük le a forráskódot, s láthatjuk, mit tettünk. Most próbáljunk valamit alkotni PHP-ben. *Töröljük ki az előzőleg beírt sorokat*, próbálkozzunk a következővel:

```
<HTML>

        Ez már a második programunk!
        <br>
        <?php phpinfo(); ?>

</HTML>
```

Frissítsük az oldalt. Az "Ez már a második programunk!" szöveg szépen meg is jelenik, de más nem. Ha lekérjük a forráskódot, meglepetés, de a PHP forrás is benn lesz, méghozzá szó szerint. Pedig a phpinfo() parancs a telepített PHP környezetről adna néhány oldalnyi információt. Mi a probléma? Ugye még emlékszünk? Csak akkor használhatunk PHP betéteket, ha .PHP kiterjesztést adunk a file-unknak. Nevezzük tehát át az index.html-t, *index.php*-re. Vigyázzunk, az index.html ne legyen a könyvtárban, mert ezt fogja keresni először a böngésző. Frissítsünk! Az oldal tartalmát nem fogom leírni, mindenki láthatja maga: egy szép, nagy táblázat, tele mindenféle információval. Nézzük meg újra a forrást! Ez már nem PHP, hanem HTML! Gyakorlásképp gépeljük be első és második programrészletünket, amely a napszaknak megfelelően köszönti az oldal nézőjét. Az elsőt mentsük el elso.html néven, majd hívjuk be a böngészőbe: http://localhost/elso.html. Ugye működik? A második neve masodik.php legyen (vigyázzunk, ide már PHP kiterjesztés kötelező!), ezt a http://localhost/masodik.php cím beírásával jeleníthetjük meg. Mi történne, ha az elsőt nem html, hanem php kiterjesztéssel mentettük volna le? Semmi probléma, csak a cím változott volna egy picit. A legfontosabb tehát, hogy nyugodtan adhatunk php kiterjesztést állományainknak akkor is, ha az nem tartalmaz PHP betéteket. Visszafelé – tapasztalatunk szerint – nem működik, tehát html kiterjesztést ne adjunk PHP rutinokat tartalmazó állománynak.

PHP – a nyelv

A PHP igen engedékeny nyelv. Nem ragaszkodik sok olyan dologhoz, amely miatt más programozási nyelv már sikoltozna. Teljesen mindegy, hogy a PHP részt hol kezdem, illetve, hogy hol fejezem be. Kezdhetem közvetlenül a sor elején, de használhatok írás közben tabulátorokat a könnyebb olvashatóság kedvéért. Egy példa:

```
<?php echo "Ez egy egysoros PHP script"; ?>
```

Ez az egy sor kiírja a megadott szöveget. Ugyanezt megoldhattam volna az alábbi módon is:

```
<?php
        echo "Ez egy többsoros PHP script";
?>
```

Voltaképpen addig, amíg az értelmező pontosvesszővel nem találkozik, egy sorként értelmezi a beírt anyagot. Hogy tovább fokozzam pozitív értelemben vett "igénytelenségét", újból példákkal illusztrálok:

```
<?php echo "1. példa"; ?>
<? echo "2. példa"; ?>
<script language="php">echo "3. példa";</script>
<% echo "4. példa"; %>
```

Nézzük meg ezt a négy példát. A különbség tulajdonképpen csak a PHP értelmező hívásában rejlik. Az első esetben a megszokott "<?php" kezdőtaggal indítottunk. Ez a legelterjedtebb, mondhatni a hivatalos verzió. A második példa könnyedebb, de ugyanúgy elfogadott. A harmadik azok kedvéért van, akik megszokták a Javascript nyelvezetét. Ezzel illik vigyázni, mert bizonyos HTML editorokba betöltve nemkívánatos eredményeket kaphatunk. A negyedik pedig azok számára lehet ismerős, akik ASP-ben otthon vannak. Az ASP a Microsoft szerver oldali scriptnyelve - sajnos, alacsony szintű támogatottsággal.

Hasonlóan engedékeny megjegyzések elhelyezése terén is:

```
<?php
        echo "Ez egy C++ stílusú megjegyzés"; // megjegyzés
        /* Bár ha akarom, több sorba is tördelhetem
        a megjegyzéseimet. */
        echo "vagy esetleg shell típusú?"; # ekkor így kell
?>
```

Lehetőségek tárháza, csak győzzük őket kihasználni. Érdemes általában egy stílus mellett megmaradni, hiszen mint tudjuk, madarat tolláról, programozót programjáról...

Változók

Annyit már tudunk a változókról, hogy "$" jellel kezdődnek. Fontos tudnunk róluk, hogy a változónevekben **a kis- és nagybetűk különbözőek**. Nem egyenértékű a "var" és a "Var" nevű változó! Ez elég nagy hibaforrást eredményezhet, főleg kezdők számára.

A PHP ötféle típust kezel, ami a PHP4 kibocsátásával 6-ra növekedett. Lássuk őket:

Integer (egész számok)
Floating-point (lebegőpontos számok - törtek)
Strings (karakterfüzérek - szövegek)
Arrays (tömbök)
Objects (objektumok)
Bool (logikai - csak a PHP4 verziójától kezdődően)

Egész

Nézzük először az egész típusú változókat:

```
$a = 1234;      # decimális szám
$a = -123;      # negatív szám
$a = 0123;      # 8-as számrendszerbeli szám (ez 83 tizesben)
$a = 0x12;      # hexadecimális szám (ez 18 tizesben)
```

Itt nincs magyaráznivaló, ilyen egyszerű. Nem kell meghatározni, a maximális és minimális értékeket, a PHP mindig annyi byte-ot foglal le, amennyi feltétlenül szükséges.

Lebegőpontos

Lebegőpontos számok esetén még egyszerűbb, csak a tizedespontot kell a megfelelő helyre kitennünk:

```
$a = 1.234;
$a = 1.2e3;
```

Az első eset a szokásos forma, a második pedig egy úgynevezett tudományos forma (exponenciális alak). Jelentése: 1.2 * 10^3, azaz 1.2 * 100 0, még pontosabban 1200.

Szöveg

Amilyen egyszerűek a számok, annyira sokoldalúak a stringek. Ha valaki ismerős egyéb nyelvekben, tudja, hogy a karakteres értékeket vagy idézőjelek ("), vagy aposztrófok (') közé kell tenni. Nos, a PHP mindkettőt használja.

A legfontosabb különbség a kettő között, hogy az idézőjelek közé tett kifejezés kiértékelődik, míg aposztrófok között nem. Ahhoz, hogy világosabb legyen, újból példákhoz folyamodunk:

```
<?php
	$a = "Ez egy string";

	$a = $a . ", meg még egy kicsi...";
	// a PHP-ben hozzáfűzésre nem "+" jelet, hanem pontot
	// használunk

	$a .= " a végére.";
	// egy másik módszer az összefűzésre,
	// talán ismerős más nyelvekből

	$num = 5;		// vezessünk be egy számot is

	$str = "A szám értéke: $num";
	echo $str;
	// Az eredmény:
	//			A szám értéke: 5

	$str = 'A szám értéke: $num';
	echo $str;
	// Az eredmény:
	//			A szám értéke: $num

?>
```

Itt már több ismeretlen is szerepel. Egyrészt, mint láttuk, szöveg összefűzésre nem "+" jelet, hanem "."-ot használunk. A másik módszer kicsit ravaszabb, nem kell kétszer kiírni a módosítandó változót, ettől eltekintve ugyanaz, mint az előző. Alatta láthatjuk a különbséget az idézőjelek és az aposztrófok között. Ha idézőjelek között szerepel egy változó, az értelmező kiszámítja azt, és behelyettesíti. Ha aposztrófok közé tesszük, minden értelmezés nélkül kiírja. Ez a kétféle lehetőség egy kicsit megzavarhatja a műveleteket:

```
<?php
	$a = 1 + "10.5";		// $a = 11.5
	$a = 1 + "-1.3e3";		// $a = -1299
	$a = 1 + "bob-1.3e3";		// $a = 1
	$a = 1 + "bob3";		// $a = 1
	$a = 1 + "10 kicsi indián";	// $a = 11
	$a = "10.0 indián " + 1;	// $a = 11
?>
```

Láthatjuk, hogy számnak értelmez mindent a PHP, amíg egy nem-szám karakterrel találkozik. Jó példa erre a negyedik sor (1+"bob3"), hiszen a végeredmény nem négy, csupán egy lesz, míg az alatta levő sorban 1+"10 kicsi indián"-ból a 10 még számként értelmezhető.

Sok más egyéb nyelvhez hasonlóan (C, Perl) lehetőség van "escape" karakterek használatára. Ennek akkor vehetjük hasznát, amikor vezérlőjelet szeretnénk elhelyezni a karaktersorozatban, vagy olyan jelet, aminek beírásával szintaktikailag helytelen kifejezéshez jutnánk. Erre egy jó példa, ha idézőjelet szeretnénk elhelyezni a szövegben. Egy táblázatban összefoglaltam a lehetőségeket:

Megnevezés	Jelentés
\n	Új sor
\r	Carriage Return (kocsivissza)
\t	Tabulátor
\\	Backslash (\)
\$	Dollárjel (amit ugye a változók jelölése miatt nem lehetne…)
\"	Idézőjel

```
<?php
        echo "Így kell idézőjelet kiírni: \"";
        echo "Soremelés következik\n";
        echo "- ez már új sorban van.";
?>
```

A szöveges típusra szintén jellemző a dinamizmus, azaz csak annyi helyet foglal el, amennyire mindenképpen szüksége van. Ez lehet egy szónyi információ, egy mondat, de egy többboldalas szöveget is érthetünk alatta.

Tömb

A következő típus a tömb. Minden nyelvben van egyfajta tömb, amely ugyanúgy használandó, mint itt:

```
<?php
        $a[1] = 100;
        $a[2] = "példa";
        $a[3] = 3.1415926;
?>
```

A példa alapján látható az első nagy különbség. A tömb egy összetett adattípus, minden eleme egy egyszerű típus (integer, floating-point, string), és ezt úgy keverhetjük, ahogy tetszik, nem kötelező egy tömbön belül ugyanazt a típust használnunk. A második különbség a megadási módban rejlik:

```
<?php
        $a[1] = 100;
        $a[] = "példa";
        $a[] = 3.1415926;
?>
```

Ha nem adunk indexet, akkor automatikusan a tömb végéhez fűződik az elem, azaz az előző két példa teljesen megegyezik.

Természetesen használhatunk többdimenziós tömböket is.

```
<?php
	$a[1][2] = "első sor második eleme";
	$a[1][2][3][4] = "ez egy négydimenziós tömb";
?>
```

A tömb elemekkel való feltöltésére két mód is kínálkozik. Az egyik a szokásos, megszokott változat:

```
<?php
	$a[1] = "alma";
	$a[2] = "körte";
	$a[3] = "barack";
	$a[4] = "szilva";

	echo $a[3];
?>
```

Míg a másik sokkal egyszerűbb, amolyan PHP-s módszer:

```
<?php
	$a = array ("alma", "körte", "barack", "szilva");
	echo $a[3];
?>
```

Vigyázat, a két módszer mégsem egyenértékű! A végeredmény bizony más. Első esetben a barack lesz a megjelenő gyümölcs, míg a második példában a szilva. Ennek oka, hogy ha nem adok meg indexet, a PHP a nulladik (0.) elemtől kezdi el a tömb feltöltését. Vigyázzunk vele, érdemesebb, ha mi is a nulladik elemtől kezdjük a számozást. S nem utolsósorban takarékosabb!

Most pedig egy olyan pozitívumát ismerhetjük meg a PHP tömbkezelésének, amely nem található meg csak nagyon kevés nyelvben. Ez az úgynevezett asszociatív tömbök használata. Ez annyit jelent, hogy a tömb indexe helyén nem szám, hanem egy karakteres azonosító szerepe. Példával talán egyszerűbb lesz:

```
<?php
	$a[1] = "piros ";
	$a["gyümölcs"] = "alma";

	echo $a[1];
	echo $a["gyümölcs"];
?>
```

Mondhatnánk a "gyümölcsödik" elem az alma. Meglátjuk, milyen hasznos lesz a későbbiekben, le sem tudunk majd szokni róla. Ha asszociatív tömböt használunk, elemeinek megadása egy kicsit változik:

```
<?php
	$a["szín"]		= "piros";
	$a["íz"]		= "édes";
	$a["forma"]		= "gömbölyű";
	// Ez volt a régi forma

	$a = array(
		"szín"	=> "piros",
		"íz"	=> "édes",
		"forma"	=> "gömbölyű"
		);
	// Ez pedig az új módszer
```

```
?>
```

Látjuk, hogy a hozzárendelés a "=>" jelsorozattal történik. Most pedig lássunk egy példát, amely ötletesen bemutatja a módszer előnyét:

```
<?php
  $a = array(
      "alma"       => array(
          "szín"  => "piros",
          "íz"    => "édes",
          "forma" => "gömbölyű"
      ),
      "narancs"    => array(
          "szín"   => "narancssárga",
          "íz"     => "fanyar",
          "forma"  => "gömbölyű"
      ),
      "citrom"     => array(
          "szín"   => "sárga",
          "íz"     => "savanyú",
          "forma"  => "gömbölyded"
      )
  );

  echo $a["narancs"]["íz"];

?>
```

Mit is csinál a fenti példa? Próbáljuk megfejteni működését, próbáljuk begépelni, letesztelni, megváltoztatni, s újra tesztelni. Vigyázzunk, hova teszünk pontosvesszőt (ugye emlékszünk, csak a kifejezés végére...), illetve vesszőt. Sok sikert!

Típuskonverziók

A PHP ún.: "gyengén típusos" nyelv, ami azt jelenti, hogy nem kötöttek a változó-típusok és a **változók** egyeztetése (pl. értékadásnál) automatikus konverzióval történik. Egy változó típusát csupán tartalma határozza meg:

```
$a = "0";                          // $a típusa karakteres, értéke "0"
$a++;                              // $a típusa karakteres, értéke "1"
$a += 1;                           // $a típusa egész, értéke 2
$a = $a + 1.3;                     // $a típusa lebegőpontos, értéke 3.3,
                                   // mivel egyik összetevője szintén
                                   // lebegőpontos
$a = 5 + "10 kicsi indián";        // $a típusa egész, értéke 15
```

Természetesen lehetőség van egyértelmű konverzióra is. Ez nagyon hasonlít a C-re, azaz a konvertálandó típus nevét zárójelbe írjuk a változó elé:

```
$a = 10;                    // $a egész típusú
$b = (double) $a;           // $b lebegőpontos
```

Lehetőségeink:

- (int), (integer) - egész konverzió
- (real), (double), (float) - lebegőpontos (double) konverzió
- (string) - string konverzió

- (array) - tömbbé konvertál
- (object) - objektum típusúra konvertál

Gyakorlati alapproblémák

A HTML, javascripttel keverve, továbbá egy is CSS-sel (Cascading Style Sheets – stíluslapok) megfűszerezve, csodákra képes. A piacon egyre inkább magára maradó Internet Explorer új 5.5-ös, 6.0-s verziói már eléggé stabilak, gyorsak ahhoz, hogy ne a böngésző, az operációs rendszer, hanem a vonalsebesség legyen a gyenge pontja a rendszernek. Ha valóra válnak azok a tervek, amelyek a jelenlegi internetes vonalak sebességét a többszörösére emelik, ez sem fog útban állni. S miért ne válhatna valóra? Az egyedüli, amiért aggódom, az Interneten megtalálható információk minősége. Egy felmérés szerint az internetes oldalak 70%-a "szemét", azaz információértéke majdnem nullával egyenlő. Nagy része valóban az alábbi sémára épül: "X.Y. vagyok, tizenx éves, itt-meg-itt lakom, imádom az alábbi énekeseket: blablablabla…, nem szeretem őket: blablabla. Itt láthatod tavaly nyáron készült képeimet a Balatonról (Görögországból, sítáborból). Ha teszik, ide írj.". Ehhez tökéletesen elegendő a HTML.

Mikor kell mégis olyan fegyverhez nyúlnunk, mint például a PHP? Miért több, mint a javascript? Tömböket az is tud kezelni, a változók és a meglevő funkciók szintén elégségesek interaktív weblapok elkészítéséhez. Azért egy óriási különbség mégis van: a javascript forráskódja letöltődik a felhasználó gépére, tehát bárki számára hozzáférhetővé válik. Éppen ebből következik, hogy fájlműveletekre nem is gondolhatunk. Képzeljük el, hogyan adnánk hozzáférési jogokat, ha a jelszavak listájához bárki hozzáférhetne. Az adatkezelés algoritmusa végtelenül egyszerű. A felhasználó megad néhány bemenő adatot (input), végrehajtunk a bevitt adatokon valamilyen műveletet, vagy egy meglevő adatbázisban keresést végzünk az input alapján (query), végül valamilyen kimenetet (output) produkálunk. Az input beviteléhez és az output megjelenítéséhez a HTML lehetőségeit fogjuk használni, viszont az adatfeldolgozást nem bízhatjuk rá. Ezért a PHP a felelős. Egyelőre elégedjünk meg annyival, hogy a PHP ugyanúgy képes szöveges állományok kezelésére, mint régi dBase adatállományaink manipulálására, de erejét az SQL adatbázisok használatai során mutatja meg. Nem feladatom méltatni a régi adatkezelő rendszerekkel szemben az SQL előnyeit, csak annyit mondhatok, hogy nem hiába használják a legnagyobb rendszerek kiszolgálásához az SQL-t. Ezekhez a kiszolgálókhoz való csatlakozást a PHP-hez letölthető DLL állományok segítségével tudjuk megvalósítani. Szerencsénkre a legelterjedtebb SQL kiterjesztést készen kapjuk a PHP-hez (a 4-es verzióba már készen beépítették, a PHP3-hoz a php3_mysql.dll állomány szükségeltetik). Ez a MySQL, amiről már beszéltünk, s ha miden igaz, szolgáltatásként már a gépünkön figyel. Sokak szerint ez az az SQL implementáció, amely leginkább illeszkedik a PHP-hez. Ez lesz az SQL szerver, ehhez csatlakozunk PHP-ból, amikor adatbáziskezeléssel foglalkozunk majd. Nem is oly' sokára, ez is bekövetkezik.

Az adatkezelésről általában

Mondottam, hogy az adatok bevitelét és a kimenetet HTML-ben fogjuk produkálni, s csak a feldolgozást végezzük PHP-ben. Ez bizony azt is jelenti, hogy vége a kényelmes, WYSIWYG honlapszerkesztéseknek, mondhatnám, sutba dobhatjuk FrontPage, DreamWeaver, Adobe PageMill és ehhez hasonló, rendkívül hatékony HTML editorainkat. Egyelőre ugyanis még egyikük sem tudja kezelni a PHP betéteket tökéletes módon. Marad a régi bevált módszer, a kézzel történő szerkesztgetés. Tudom, most sokan felhördülnek, hogy micsoda mazochista technikákat alkalmazunk céljaink eléréséhez, de sajnos (dehogy sajnos!) más lehetőségünk nincs. Azok kedvéért, akik nincsenek tökéletesen képben a HTML-t illetően, megengedjük, hogy használják editoraikat olyan részek elkészítésében, amelyek még bizonytalanul mennek. Hogy beilleszthessék eme részeket a PHP betétek közé, ki kell metszeni a szükséges részletet a szerkesztőprogram forrásablakából, s nem árt, ha tudjuk mit, és miért teszünk. Egyben biztos vagyok: egy jól működő, jól használható, komplex PHP-s feldolgozóprogram elkészítése után mindenki profi HTML kóder lesz!

Ismételjük át, amit a HTML adatbeviteli lehetőségeiről tudnunk kell. HTML-ben úgynevezett űrlapokat hozhatunk létre a **<FORM>** objektum beszúrásával. Ez fogja egybe a különböző beviteli formákat, mint például a szöveges mező, vagy a checkbox. Ezekről később szólunk. A <FORM> objektumnak különböző attribútumai (jellemzői, paraméterei) lehetnek:

```
<FORM
                target = "ablaknév"
                action = "végrehajtó script neve"
                method = GET | POST
                name = "űrlap neve"
                onReset = "reset_rutin"
                onSubmit = "submit_rutin ">
```

Nem kell megijedni, megmagyarázom őket:

- target: megadhatom, hogy az eredmény melyik ablakban/frame-ben jelenjen meg.
- action: a Rögzítés (Submit) gomb megnyomása után ennek a scriptnek fogja elküldeni az űrlap adatait a böngésző. Ha nem adjuk meg, akkor a HTML állomány **önmaga hívódik meg**! Ez fontos!
- method: ez határozza meg, hogy az adatok hogy kerülnek a scripthez. Ha GET, akkor az űrlap adatai az URL után csapódnak egy kérdőjellel (pl. http://www.sajatdomain.com/?mezo1=ertek1&mezo2=ertek2). Láthatjuk, hogy a különböző űrlapmezők az "&" jellel vannak elválasztva. Ha POST, akkor a mezők értékeit a standard inputon keresztül kapják meg a scriptek. Szerencsére PHP-ben ezzel nem kell törődnünk, leginkább a POST metódust fogjuk használni.
- name: megadja az űrlap nevét. Ez csak esetleges javascript betéteinknél jelent majd hivatkozási alapot.
- onReset: ha megnyomjuk a FORM-hoz tartozó Törlés gombot, akkor az itt megadott javascript függvény kerül végrehajtásra, mielőtt törlődnek az adatok az űrlapról.
- onSubmit: Mint az előző, csak a Rögzítés gombra vonatkozólag. Ez nagyon hasznos lehet, ha nem akarjuk addig átküldeni az adatokat a scriptnek (action), amíg egy megadott feltétel nem teljesül.

Az űrlapot a </FORM> zárótaggel kell zárni. Közé tehetjük az űrlapmezőket. Hogy milyen lehetőségeink vannak, tekintsük át őket.

Beszúrhatunk egyszerű **TEXT** (szöveges) mezőt:

```
<INPUT TYPE = "TEXT"
                    name = "név"
                    value = "kezdőérték"
                    size = "méret"
                    maxlength = "hossz">
```

- name: a szöveges mező neve. Rendkívül fontos szerepet kap a következőkben.
- value: ha kezdőértéket adunk a mezőnek, itt megtehetjük. Ha nem adjuk meg, a szöveges mező üres lesz.
- size: a mező szélessége karakterekben mérve.
- maxlength: a mezőbe írható karakterek maximális száma.

Lehetőségünk van többsoros szövegablak (**TEXTAREA**) létrehozására:

```
<TEXTAREA
                    name = "név"
                    rows = "sorok száma"
                    cols = "oszlopok száma">

KEZDŐSZÖVEG

</TEXTAREA>
```

- name: a többsoros beviteli mező neve. Később ezzel hivatkozunk rá.
- rows: megjelenítendő sorok száma. Ha nem fér ki a beírt szöveg, görgetősávok használatával lehet a kilógó részeket megtekinteni.
- cols: Ugyanaz, mint rows, csak oszlopokra.

Ugye látjuk, hogy itt zárótag is kötelező?

A következő elem a **PASSWORD**, azaz jelszó objektum. Nem ragozom, hiszen csak deklarációjában különbözik a TEXT beviteli mezőtől, no meg persze abban, hogy a beírt szöveg minden karaktere helyett "*" jelenik meg. Jelszavak beolvasására jól alkalmazható:

```
<INPUT TYPE = "PASSWORD"
                    name = "név"
                    value = "kezdőérték"
                    size = "méret"
                    maxlength = "hossz">
```

Legyen a következő a **SELECT** objektum. Ez egy olyan választhatóan többsoros, szükség esetén gördíthető listát képez, amelyből a felhasználó egy vagy több elemet kiválaszthat. Formátuma a következő:

```
<SELECT
                    name = "név"
                    size = "méret"
                    MULTIPLE>

                    <OPTION VALUE = "érték" SELECTED> szöveg
                    <OPTION VALUE = "érték" SELECTED> szöveg
                    …
</SELECT>
```

- name: a lista hivatkozási nevét adhatjuk meg.
- size: a látható sorok száma
- MULTIPLE: ha egyszerre több választás is megengedett, akkor ezt a MULTIPLE kulcsszóval jelezhetjük. Ilyenkor a szokásos módon, a CTRL, SHIFT billentyűk segítségével tudunk többszörös választást eszközölni.
- <OPTION…: láthatjuk, hogy az előbb lezártuk a SELECT taget. Jöhetnek a listából választható elemek leírásai. Egy elemnek három tulajdonsága lehet. Első az értéke, ez megy át a scripthez. A második a SELECTED opció, amely ha létezik, akkor az aktuális elem kiválasztott elem, míg ha nem írjuk be, akkor nem lesz kiválasztva indításkor. A harmadik az aktuális elemhez tartozó, képernyőn (listában) megjelenő szöveg. Egy példa:
  <OPTION VALUE=1>Első sor
  <OPTION VALUE=2 SELECTED>Második sor

  A példa alapján két értéke lesz a listának, a második érték lesz alapértelmezettként kiválasztva. Ha változtatás nélkül megnyomjuk a Rögzítés gombot, akkor a scriptnek a második érték megy át, pontosabban ennek a sornak a VALUE tagja (2).

A végén ezt se feledjük lezárni </SELECT> taggel.

Alapértelmezésben a SELECT egy legördülő lista, amelynek tetején az éppen kiválasztott elem található. Gondoljunk csak például a drive (meghajtó) választó listára a Windows bármelyik töltés/mentés dialógusablakában. Ha viszont a fent említett paraméterek közül a SIZE vagy a MULTIPLE bármelyikét megadjuk, akkor kapjuk az egyszerű listát, amelyből többet is ki lehet választani, és ez már több soros is lehet.

Következő elemünk a lefordíthatatlan **RADIOBUTTON** nevet viseli. Talán rádiógombnak lehetne fordítani? Ezek azok a kör alakú gombok, amelyek közül mindig egy lehet kiválasztva. Formátuma a következő:

```
<INPUT TYPE = "RADIO"
                    name = "név"
                    value = "érték"
                    CHECKED>
```

- name: a már megszokott hivatkozási név. Ha több rádiógomb csoportot is létrehozunk, akkor a következő csoportnak más nevet kell adnunk. Minden csoportban egy gomb lehet bekapcsolva.
- value: ha be van kapcsolva a rádiógomb, akkor ezt az értéket küldi el a scriptnek a Rögzít gomb megnyomása után.
- CHECKED: szintén egy olyan kulcsszó, amely elhagyható. Ha viszont meg van adva, akkor alapértelmezésben a rádiógomb be lesz kapcsolva. Természetesen mindig csak EGY gomb lehet bekapcsolva. Ha többet is meghatározunk, az utolsó CHECKED lesz az érvényes.

A CHECKBOX objektum (Ellenőrző doboz? Maradjunk a checkboxnál…) egy kétállású kapcsolót reprezentál. Lehetőségei megegyeznek a rádiógombnál leírtakkal, azt leszámítva, hogy itt több kiválasztott gomb is lehet.

Végül az utolsó objektumtípus, amit űrlapon elhelyezhetünk a nyomógomb, azaz a **BUTTON**. Ennek három fajtáját különbözteti meg a HTML. Első, amit megemlítünk a valódi BUTTON. Definíciója a következőképpen történik:

```
<INPUT TYPE = "BUTTON"
                    name = "név"
                    value = "felirat"
                    onClick = "click_rutin">
```

- name: a gomb hivatkozási neve.
- value: a gomb felirata
- onClick: a gombra történő kattintáskor az itt megadott javascript rutint hajtja végre.

A következő gombtípus a **SUBMIT** gomb, azaz magyarul "Rögzít"-nek fordíthatjuk. Ezen gomb megnyomásakor a böngésző elküldi az űrlap tartalmát az ACTION részben megadott scriptnek:

```
        <INPUT TYPE = "SUBMIT"
                        name = "név"
                        value = "felirat">
```

Hivatalosan itt is van onClick eseménykezelő, de ritkán van rá szükség, ezért itt, most nem tárgyaljuk.

Az utolsó, **RESET** gomb az a bizonyos "Törlés" gomb, amely alaphelyzetbe állítja az űrlap minden mezőjét, azaz mindegyik a módosítások előtti (kezdeti) állapotba kerül. Definíciója:

```
<INPUT TYPE = "RESET"
                    name = "név"
                    value = "felirat">
```

Nos, ennyi lenne az űrlapok kezelésére vonatkozó ismétlésünk. Ha valaki többre kíváncsi, érdemes átlapozni a legutolsó HTML specifikációt, amelyet a http://www.w3.org címen mindig megtalálunk. Másik lehetőség, hogy előveszünk egy magyar nyelvű kézikönyvet, kikeressük a megfelelő fejezeteket, kipróbáljuk a példákat és addig nem állunk fel a gép mellől, amíg tisztán nem látjuk a témakör összes aspektusát. Ne feledjük: befektetés nélkül nincs eredmény! Nos, ennyi bölcsesség után térjünk rá következő fejezetünkre, amelyből megtudhatjuk, hogy hogyan mennek át űrlapunk mezői PHP scriptjeinkbe.

Adatátvitel a HTML és a PHP között

A cím egy kicsit megtévesztő. A HTML egy lapleíró nyelv, nincs szüksége semmilyen változóra, hiszen statikus adatokkal dolgozik. Az előző mondatom igaz is, meg nem is. A HTML valóban nem használ a szó szoros értelmében vett változókat, viszont minden, a weblapon található objektum sok-sok tulajdonsággal rendelkezik, amelyet nagyrészt magam szabhatok meg. Természetesen főként az űrlapok objektumaira gondolok. Hogy világosabbá váljék gondolatmenetem, készítsünk egy egyszerű űrlapot, amely bekéri a felhasználó nevét és születési évét. Valami ilyesmit fogunk írni:

```
<HTML>
        <FORM name=adatok>
                Neved:<br>
                        <input type=text name=nev><br>
                Születési éved:<br>
                        <input type=text name=szev><br>

                <br>
                <input type=submit name=submit value="Mehet">
        </FORM>
</HTML>
```

Az űrlap neve "adatok", rajta két mező ("nev" és "szev" névvel). Szükségünk lehet egy "Mehet" gombra is, amellyel átadjuk az űrlap tartalmát a <FORM> tag action részében megadott scriptnek. Aki figyelt, észrevette, hogy én ezt a részt kihagytam. Beszéltünk róla, hogy ha üresen hagyom, akkor a HTML oldal önmagát fogja visszahívni a Mehet (Submit) gomb lenyomása után. Ezt ki is próbálhatjuk. Ha begépeljük a fenti forrást, töltsük ki a két mezőt, majd nyomjuk le a "Mehet" gombot. A lap újrahívódik, ami azt jelenti, hogy törli a két mező tartalmát. Viszont ezzel még nincs vége! Nézzük csak a címsort! Egy kicsit megváltozott:

http://localhost/?nev=Joc&szev=1971&submit=Mehet

Ugye, nem kell mondanom, hogy mindenkinek más lesz a neve és a születési éve? Ha visszalapozunk a FORM leírásához, nézzük meg újból a "method" paraméterét. Ott azt mondtuk, ha GET, akkor egy kérdőjellel fűzi hozzá az URL-hez az űrlap mezőinek értékeit. Ez az! A method attribútum kihagyása esetén ugyanis az alapértelmezett metódus lép életbe, ez pedig a GET. Mielőtt tovább mennénk, nézzünk rá erre a sorra, s elemezzük ki, mit látunk.

Odáig biztosan mindenkinek tiszta, hogy http://localhost/. Ez ugye a szerverünk neve (illetve URL címe). Ezek után a fent említett kérdőjel jön, majd ez egyes űrlapmezők neve és értéke:

nev = Joc
szev = 1971
submit = Mehet

Tényleg három objektumot helyeztünk el az űrlapra, két szövegmezőt, s egy nyomógombot. Fontos, hogy a nyomógomb értékét (value) mi adtuk meg, ezt a felhasználó nem tudja megváltoztatni, a két szövegmező viszont szabad préda, azt

ír bele, amit akar. Ha akarja, üresen hagyja, de olyat is elkövethet, hogy a születési év mezőbe betűket ír. Sebaj, megoldjuk ezen problémákat is.

Folytassuk tovább eszmefuttatásunkat, amelyet a metódusoknál hagytunk abba. Ha ezt POST-ra módosítom, nem fog megjelenni a címsorban semmi, de sebaj, a P HP egyszerű megoldást kínál, ennek lekezelésére. Az egyszerűség kedvéért megadom a megoldást, s később jön a magyarázat:

```
<?php

        if (getenv(REQUEST_METHOD)=="POST")
        {

                echo $nev;

        } else

        {

?>

<HTML>
        <FORM name=adatok method=post>
                Neved:<br>
                        <input type=text name=nev><br>
                Születési éved:<br>
                        <input type=text name=szev><br>

                <br>
                <input type=submit name=submit value="Mehet">
        </FORM>
</HTML>


<?php } ?>
```

Mentsük el a fenti forrást index.php néven (ugye mindenki tudja, hogy a C:/web/root/ könyvtárba mentünk minden anyagot?), majd hívjuk be a böngészőt, s kérjük le a http://localhost URL-t. Ha minden tökéletes (a webszerver is el van indítva), akkor egy üres űrlapnak kell megjelennie a weblapon. Kövessük végig a folyamatot, értelmezzük az egyes sorokat.

```
<?php

        if (getenv(REQUEST_METHOD)=="POST")

        {

                echo $nev;

        } else
```

Nézzük meg egyelőre idáig. Az első sor egy szokásos PHP nyitás, azt jelenti, hogy most PHP nyelvű rész következik. Utána egy vizsgálatot végzünk: ha a REQUEST_METHOD nevű környezeti változó értéket POST, akkor… De ne siessünk. Szóval az "if" a többi nyelvhez hasonlóan itt is a logikai vizsgálatot végzi el. Maga a kiértékelendő feltétel zárójelben következik. Használunk benne egy getenv() függvényt, amely környezeti változók tartalmát olvassa be. Ezek közül egy a REQUEST_METHOD. Ez a FORM method nevű attribútumát adja át a PHP-nek. Az egyenlőség-vizsgálat néhány nyelvtől eltérően nem "=" hanem

"==", azaz dupla egyenlőségjel. Kezdők egyik kedvenc hibája, hogy feltételek megfogalmazása estén szimpla egyenlőségjelet használnak. Nem helyes, ugyanis az "=" az értékadás jele!

Bezártam a feltételhez tartozó zárójelet, amely után a logikai "IGAZ" ág következik. A PHP ezt kapcsos-zárójelek közé teszi. Jelen pillanatban egy sor van közöttük, ilyen esetekben nem kötelező használni őket, de inkább szokjuk meg, strukturáltabbá és áttekinthetőbbé teszi programunkat. Szóval a kapcsos-zárójelek között mit művelünk? Kiírunk valamit, hiszen az "echo" parancs a kiírást jelenti PHP-ben. Kiírunk egy $nev nevű változót. Mi ez a változó, honnan jött? Remélem, sokan kitalálták. Ez bizony az űrlapunk első szövegmezője, amit "nev" névvel illetünk. Ezért jegyeztem meg a FORM objektumainak leírásakor, hogy adjunk nevet az egyes mezőknek. Mivel a PHP-ben minden változó "$" jellel kezdődik, itt sincs kivétel. Tehát kiírjuk az első szövegmező tartalmát, már ha egyáltalán kedves felhasználónk kitöltötte. Ezek után egy "else" kulcsszó következik, ami a logikai "HAMIS" ág bevezetője. Lássuk, mit tartalmaz ez az ág.

```
        {

?>

<HTML>
        <FORM name=adatok method=post>
                Neved:<br>
                        <input type=text name=nev><br>
                Születési éved:<br>
                        <input type=text name=szev><br>

                <br>
                <input type=submit name=submit value="Mehet">
        </FORM>
</HTML>


<?php } ?>
```

Figyeljük meg, hogy a "HAMIS" ág is ugyanúgy kapcsos-zárójelek közé teendő, mint az "IGAZ" ág. Csakhogy egy érdekes konstrukció következik. Amint megnyitom a "HAMIS" ágat, máris kilépek a PHP környezetből, visszalépek HTML-be. Sebaj, a PHP követi ezt, s tudja, hogy a most következő HTML rész is a "HAMIS"ághoz tartozik, egészen addig, amíg egy záró kapcsos-zárójelre nem talál. Mivel a zárótag szintén PHP nyelvű, a legutolsó sorban nyitnom kellett egy PHP részt, s ebben helyeztem el a záró kapcsos-zárójelet. Közötte pedig az előzőleg már megtárgyalt űrlapot írtam le, vigyázva arra, hogy a küldési metódus POST legyen. (Egy megjegyzés: a HTML részben mindegy, hogy nagybetűvel, vagy kisbetűvel írom a paramétereket. Ugyanez viszont nem érvényes a PHP-ben. Már egyszer említettem, de nem árt újra feleleveníteni: a PHP megkülönbözteti a kis/nagybetűvel írt változókat egymástól.)

Kövessük csak a lap betöltődésének és értelmezésének folyamatát! A vezérlés először a REQUEST_METHOD vizsgálatára fut rá. Mivel a lap első betöltésekor ennek nem POST az értéke (csak akkor lesz POST, ha megnyomom a "Mehet" gombot), továbbfut a "HAMIS" ágra. Ez kiírja az űrlapot, s vége. Amint a felhasználó megnyomja a "Mehet" gombot, az űrlap újrahívja az oldalt. Immár

van értéke a REQUEST_METHOD-nak, méghozzá pont POST, tehát az "IGAZ" ág fog végrehajtódni, a "HAMIS" ág kimarad. Az oldal letöltése a "HAMIS" ág utáni részen folytatódik, de mivel jelen helyzetben itt már nincs semmi, kész az oldal.

Gyakorlásképpen próbáljuk megoldani, hogy a gomb megnyomása után a lap köszöntse nevén a felhasználót, és írja ki, hány éves. Valami ilyenre gondoltam:

Szia, kedves Tamás! Ebben az évben töltöd be 20. életéved.

Egy megoldási lehetőség:

```
if (getenv("REQUEST_METHOD")=="POST")

        {

                $kor = 2003 - $szev;            // ha most 2003 van!
                echo "Szia, kedves $nev! Ebben az évben töltöd $kor. életéved.";

        }
```

Csak az "IGAZ" ágat írtam le, hiszen módosítani csak itt kellett. Azért nézzük át: bevezettem egy $kor nevű változót. Ha a jelen évből kivonom a születési évet, akkor az aktuális évben betöltött évek számát kapom. Ez egyértelmű. A kiírás viszont sokak számára érdekes lehet, kihasználtam, hogy az idézőjelek közé változókat a PHP kiértékeli, azaz a $nev és a $kor változók helyett azok értékeit fogja behelyettesíteni.

Tudom, elkészíthettük volna szebben is. Például kiemelhetnénk a felhasználó nevét félkövérrel:

```
echo "Szia, kedves <b>$nev</b>! Ebben az évben töltöd $kor. életéved.";
```

Nos, jó példa, milyen hatékonyan lehet keverni a PHP-n belül a HTML forrást. Ha arra támad kedvünk, a teljes HTML kódot kiírhatjuk a PHP segítségével. Lássuk az előző programot, csak PHP nyelven:

```
<?
        if (getenv("REQUEST_METHOD")=="POST")

        {

                $kor = 2003 - $szev;
                echo "Szia, kedves <b>$nev</b>! Ebben az évben töltöd $kor. életéved.";

        } else

        {

        echo "<HTML>";
        echo "    <FORM name=adatok method=post>";
        echo "              Neved:<br>";
        echo "                      <input type=text name=nev><br>";
        echo "              Születési éved:<br>";
        echo "                      <input type=text name=szev><br>";
        echo "              <br>";
        echo "              <input type=submit name=submit value=\"Mehet\">";
        echo "    </FORM>";
```

```
        echo "</HTML>";

        }
?>
```

Hogy miért nem szoktuk ezt a megoldást alkalmazni? Kérjük csak a forráskódot, s rögtön megértjük. Persze, van megoldás: ha minden echo sort egy "\n" (soremelés) jelsorozattal fejezünk be, máris szebb lesz a látvány. De vajon megéri-e?

Ha valaki figyelmesen átolvasta az előző kódot, észrevehette, hogy a "Mehet" gomb deklarálásánál az idézőjeleket kicseréltem. Ennek magyarázata, hogy az "echo" parancs paraméterét szintén idézőjelek közé kellett tennem. Ilyenkor több megoldás is kínálkozik. Az egyik, hogy valamelyik idézőjelpárt aposztrófpárra cserélem. A másik, hogy **Escape-szekvenciát** alkalmazok (\"), amit már összefoglaltam egy táblázatban:

```
echo "          <input type=submit name=submit value=\"Mehet\">";
echo "          <input type=submit name=submit value='Mehet'>";
echo '                  <input type=submit name=submit value="Mehet">';
```

A fenti példák mindegyike helyes, a lényeg, hogy kerüljük ki az idézőjelen belüli idézőjel problémáját, mert egy kövér "Parse error…" hibaüzeneten kívül mást nem fogunk kapni.

A fenti egyszerű feladatra egy nagyon egyszerű megoldást adtunk. Bonyolultabb weblapokon sokszor előfordul, hogy nem egy, de több űrlap található egy lapon. Ha továbbra is a REQUEST_METHOD lekérdezése alapján döntjük el, hogy volt-e űrlapküldés, akkor gondjaink támadhatnak. Honnan tudjuk, melyik űrlap gombját nyomták meg? Erre is van megoldás. Térjünk vissza oda, amikor megnéztük, milyen értékeket ad át az űrlap a scriptnek. Átadja a szövegmezők tartalmát, és átadja a nyomógomb értékét is. A szövegmezőket a felhasználó adja meg, a gomb értékét viszont én írhatom be a forrásba. Ez fontos szempont, hiszen úgymond felhasználó-független. Ezt kihasználva, a következőképpen módosíthatom az előző program forrását:

```
if (isset($submit))
        {
                $kor = 2000 - $szev;
                echo "Szia, kedves <b>$nev</b>! Ebben az évben töltöd $kor. életéved.";
        }
```

Az új információ az első sor. A $submit változó a "Mehet" gomb neve. Vigyázzunk, hogy ugyanúgy írjuk le, ahogy a HTML forrásban használtuk, jelen helyzetben végig kisbetűvel. Az isset() függvény azt vizsgálja le, hogy a zárójelben szereplő változónak van-e tartalma, tehát létezik-e egyáltalán a változó. Ha egy következő űrlapon az "Elküld" gombot máshogy nevezzük el (name=…),

akkor lehetőségünk nyílik arra, hogy megkülönböztessük a küldő űrlapokat egymástól. Lássunk erre egy példát:

```
if (isset($gomb1))
        {
                echo "Az első űrlapról jött a hívás.";
        } else
if (isset($gomb2))
        {
                echo "Második űrlapról jött a hívás.";
        } else
// jöhet az űrlapok leírása, mivel egyik gombot sem nyomták meg.
```

Mondanom sem kell, hogy az első űrlapon létre kell hoznunk egy *gomb1* nevű submit gombot, míg a második űrlapon egy *gomb2*-t. Az, hogy mi lesz e gombok felirata, teljesen lényegtelen (a program futása szempontjából), csak legyen meg a *name* paraméterük a fentiek szerint.

Foglaljuk össze, mit tudunk eddig! Tudunk már létrehozni PHP betétes HTML állományt, ismerjük a PHP változótípusait, megtanulhattunk pár utasítást, továbbá megbeszéltük, hogyan kommunikálnak egymással a PHP és az űrlapok. Válaszoljunk először az alábbi kérdésekre, s csak utána menjünk tovább!

Összefoglaló kérdések

- Mit jelent az, hogy szerver oldali scriptnyelv?
- Sorolja fel a PHP előnyeit a javascripttel és HTML-lel szemben!
- Mi a különbség a *html* és a *php* kiterjesztés között?
- Hogyan jelezheti a kiszolgáló számára, hogy PHP-betét következik?
- Hogyan készíthet megjegyzést a PHP-n belül?
- Soroljon fel a hatféle PHP változótípust!
- Milyen űrlap objektumokat ismer?
- Hogyan azonosítjuk az űrlap objektumait?
- Mi a különbség a GET és a POST metódus között?
- Hogyan oldhatjuk meg az azonosítást, ha több űrlapot alkalmazunk egy oldalon?

Adatbázis előkészületek

Mielőtt hozzákezdenénk a valódi dinamikus weboldalak létrehozásához, tisztáznunk kell egy-két dolgot! Kétféle módon tudunk adatbázist létrehozni MySQL-ben: az első, amikor a PHP-t használjuk fel erre a célra, azaz készítünk egy weblapot, amelyet távolról meghívva, elkészíthetjük az adatbázist. Ez kényelmes, hiszen nem kell a webszerver előtt ülnünk. Ezt a módszert nem szeretik annyira. No nem a weboldal készítőiről beszélek, hanem a szerverek fenntartóiról. Hiszen ehhez olyan jogra van szükség, amely voltaképpen bárkinek írási lehetőséget biztosít egy megadott könyvtárban a szerveren, ahol senki sem garantálhatja, hogy ténylegesen csak arra használja a felhasználó a tárterületét, amihez a fenntartók hozzájárulásukat adták. Ezért inkább egy másik megoldást szoktak az üzemeltetők javasolni: a szerver előtt ülve valaki létrehozza az adatbázist, a jogokat, a többit bízzák csak a weboldal készítőire. Ez a biztonságosabb vonal, mi is ezt a változatot fogjuk követni. Aki mégis a másik megoldás mellett szeretne maradni, ajánlom megtekintésre az alábbi programrészletet:

```
@mysql_query("drop database $mysql_db_name");
mysql_query("create database $mysql_db_name") or die("Hiba történt az adatbázis létrehozásokor!");
mysql_query("use $mysql_db_name") or die("Hiba történt az adatbázis kijelölésekor!");
```

Csak pár szóban: az első sor megszünteti a már létező adatbázist. Az adatbázis neve a *$mysql_db_name* nevű változóban van tárolva. A sor előtti „@” jel a felmerülő hibaüzenetet hivatott elnyomni. (Magyarul, ha még nem volt ilyen nevű adatbázis, akkor sem kapunk hibaüzenetet!) A második sor hozza létre az adatbázist, vagy hiba esetén értesít, a harmadik sor pedig használatba veszi azt. Ezek után jöhet a táblák létrehozása, miegymás… De ne rohanjunk ennyire előre, nézzük meg a másik módszert is, sokat tanulhatunk belőle.

Kezdjük azzal, hogy keressük meg az SQL szerver /bin/ mappáját. Ez normál esetben a C :\MySQL\bin mappát jelenti. Itt találjuk a szerverprogramokat (mysqld*.exe), továbbá egy igen hasznos, bár annál „fapadosabb” eszközt, *mysql.exe* néven. Ha tisztában vagyunk az SQL parancsokkal, nem lesz nehéz dolgunk, de ez előfeltétele is a további munkánknak – ha valakinek hiányosságai vannak e téren, pótolja!

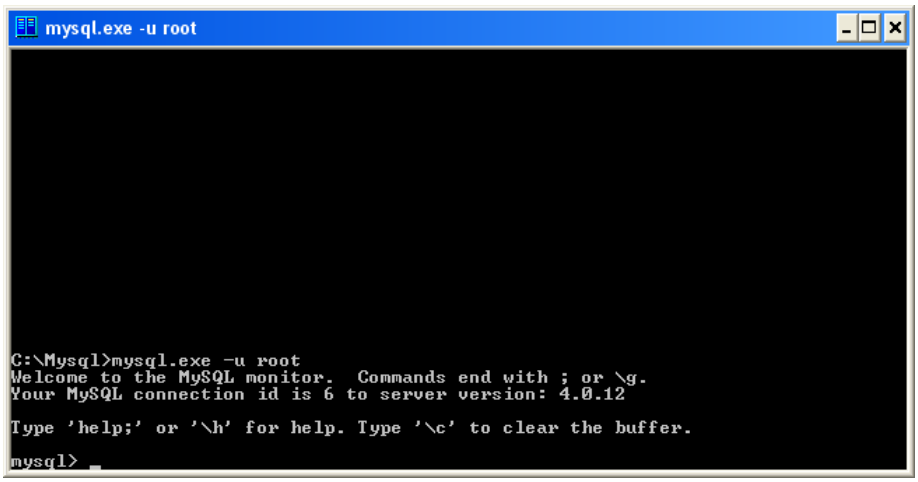
Adatbázis és adattábla

Eleddig csak adatbázisokról beszéltem, pedig nincs adatbázis adattáblák nélkül! Vágjunk bele, hozzunk létre egy adatbázist! Indítsunk egy konzolt (parancssort), majd a *mysql.exe*-t indítsuk el az alábbi paraméterekkel:

```
mysql –u root
```

Ezt már csináltuk egyszer, csak akkor paraméterek nélkül, tesztelésre használtuk ezt a sor. Most csak annyival egészítettük ki, hogy **root** felhasználóként (-u

= user) léptünk be az SQL szerverre, tehát vigyázzunk, mert csúnya dolgokat lehet művelni! Megint kaptunk egy promptot, elkezdhetjük gépelni parancsainkat.



Ide szabványos SQL parancsokat gépelhetünk be, amivel manipulálhatjuk az adatbázist, a táblákat, szelektálhatunk, törölhetünk, módosíthatunk. Mivel a Windows-hoz szokott felhasználók számára egy csöppet ijesztő lehet a felület (jó öreg DOS! ☺), nem sokat fogunk vele dolgozni, csak ami feltétlenül szükségeltetik.

Első parancsunk az adatbázis létrehozása:

```
create database jegyzet;
```

Ugye, nem feledkeztünk meg a sor végén a pontosvesszőről? Ha mindent pontosan sikerült begépelnünk, egy „Query OK” üzenet tájékoztat a művelet sikeréről, sőt, még a művelet végrehajtásának idejét is láthatjuk…

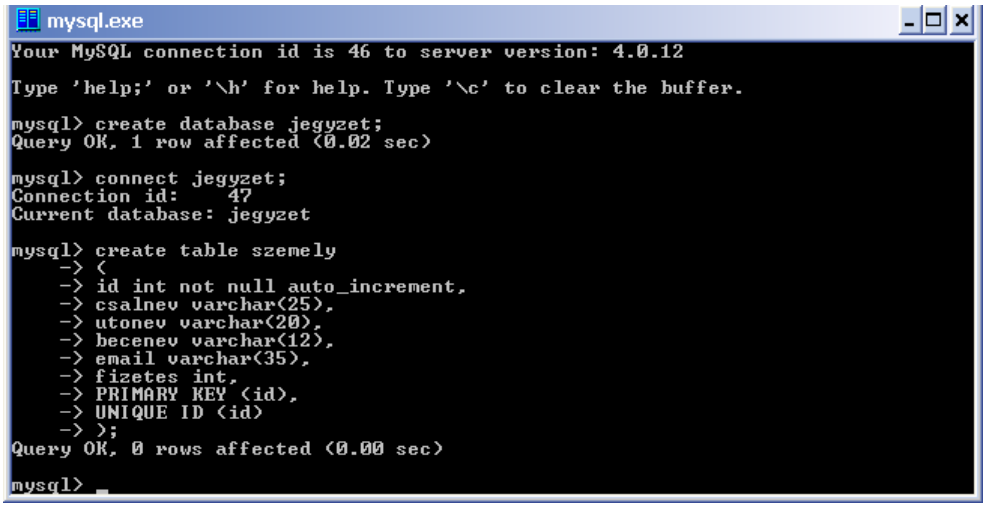
Kapcsolódjunk az adatbázishoz. Nem elég, ha létrehozzuk az adatbázist, kapcsolódnunk is kell hozzá. Ez az alábbi módon történik:

```
connect jegyzet;
```

Meg is volnánk! Megkapjuk az adatbázis azonosítóját (connection id), amire egyenlőre nem lesz szükségünk, majd a PHP írásánál lesz jelentősége. Hozzunk létre egy táblát, amiben majd dolgozgatunk!

```
CREATE TABLE szemely
(
id int NOT NULL AUTO_INCREMENT,
csalnev varchar(25),
utonev varchar(20),
becenev varchar(12),
email varchar(35),
fizetes int,
PRIMARY KEY (id),
UNIQUE id (id)
);
```

Láthatjuk, hogy a tábla létrehozása is szabványos SQL paranccsal történik. Én több sorba írtam az egészet, ami semmit sem változtat a kifejezés értelmén. Annyi előnye viszont van, nem kell az egész kifejezést áttekintenem, logikailag sorokra bonthatom, egyszerűbbé téve a beírást.



Készen is vagyunk. Készítettünk egy adatbázist (jegyzet), majd konnektáltunk (hozzákapcsolódtunk), s elkészítettük az első táblánkat (szemely). Azért nézzük meg, mit hoztunk létre:

Tehát létrehoztam egy *id*-t (azonosítót), egy nevet (*utonev* és *csalnev*), egy becenevet (*becenev*), e-mail címet, fizetést (*fizetes*, hogy legyen benne szám is). Ezek voltak a mezők. A maradék két sor az elsődleges azonosítót határozza meg, s hogy ez egyedi kell legyen.

Menjünk is tovább! Én még hozzáadnék legalább két rekordot, hogy majd tudjunk vele dolgozni. Még mindig parancssorból:

```
INSERT INTO szemely VALUES (1,'Lajos','Kiss','Lali', 'lali@freemail.hu,75000);
INSERT INTO szemely VALUES (2,'Ilona','Helyes','Ili', 'hili@hotmail.com',66000);
```

Figyeljük meg, hogy csak a karakteres mezőket raktam aposztrófok közé, a numerikus típusúakat nem szükséges. Ha mégis megtesszük, hát akkor sincs semmi probléma. A valódi típust a tábla definíciója határozza meg. Visszafelé nem működik: a szöveges adatot **idézőjel**, vagy **aposztróf** közé kell tenni! Azt is érdemes megfigyelni, hogy nem írtam ékezetes karaktersorozatokat. Mivel csak így mysql-ból nem igazán szoktunk beszúrni rekordokat (ezt kerüljük is!), érhetnek kellemetlen meglepetések. Ugyanis a mysql DOS-os kódkészletet használ, míg ha Windows alól, a weblapon keresztül nézzük meg az adatokat, akkor mindenképpen a Windows-os karakterkészletet fogjuk látni. (Közép-európait általában.)

Adatkezelés

Listázás

Már van egy adattáblánk, amit igaz, még parancssorból hoztunk létre. Ennyi elég is volt ebből a fantasztikus parancssoros megoldásból, térjünk át valami kellemesebbre. Vegyük elő kedvenc ASCII szerkesztőnket, majd a /root/ mappában (document root) hozzunk létre egy *listazas.php* nevű állományt!

```
<HTML>
<?
        $db = mysql_connect("localhost", "root", "");
        mysql_select_db("jegyzet",$db);
        $result = mysql_query("SELECT * FROM szemely",$db);
        echo "<TABLE border=1>";
        echo"<TR><TD><B>Teljes név</B></TD> <TD><B>Becenév</B></TD> <TD><B>Fizetés</B></TD>
</TR>";
        while ($myrow = mysql_fetch_array($result))
        {
                echo "<TR><TD>";
                echo $myrow["csalnev"];
                echo " ";
                echo $myrow["utonev"]."</TD>";
                echo "<TD>";
                echo $myrow["becenev"]."</TD>";
                echo "<TD>";
                echo $myrow["fizetes"]." Ft</TD>";
        }
        echo "</TABLE>";
?>
</HTML>
```

Ha sikeresen begépeltük, töltsük be a böngészőnket, majd az alábbi cím beírásával teszteljük: http://localhost/listazas.php. Valami ilyesmit kell látnunk:

Teljes név	Becenév	Fizetés
Kiss Lajos	Lali	75000 Ft
Helyes Ilona	Ili	66000 Ft

Természetesen nem maradunk magyarázatok nélkül… A PHP-s kezdés után (<?) megpróbálunk konnektálni a **localhost**-on működő SQL szerverre. Ha nem a saját gépünkre van feltelepítve a kiszolgáló, akkor ide természetesen a megfelelő címet kell ide írnunk. Ez lehet egy IP cím, vagy akár egy URL (http://www.szerverneve.hu). A második paraméter a felhasználó neve, ami (még az előzőekből okulva) root lesz, a harmadik paraméter pedig a felhasználó jelszava. Mivel nem használunk jelen pillanatban jelszót, ezt üresen hagyjuk. Az előbb megtárgyalt **mysql_connect** egy függvény, a v isszaadott érték pedig szám, az a bizonyos *connection id*, amivel már találkoztunk a mysql tárgyalásakor. Ezzel azonosítjuk a kapcsolatot, hivatkozunk rá, ha több, egyszerre megnyitott állománnyal dolgozunk. Tehát a $db változóban megkaptuk az adatbázis kapcsolódásához szükséges azonosítót.

A következő sorban kiválasztjuk az adatbázist. Ehhez a **mysql_select_db** utasítást használjuk, amelynek két paramétere van: az adatbázis neve, és az előbbi, az adatbázis-kapcsolódást azonosító változó.

Mindkét esetben előfordulhatnak hibák. Az adatbázishoz való kapcsolódáskor például lehet, hogy nem fut az adott címen mysql kiszolgáló. A második esetben lehet, hogy nincs ilyen nevű adatbázis. Ezek úgynevezett kritikus, vagy fatális hibák. Az első esetben mindenképpen le kell állítani a weboldal működését, a második esetben is ajánlott, hiszen nincs mivel dolgozzunk. Ennek kiküszöbölésére találták ki a **die** parancsot. Ez – utolsó leheletként – még visszaküld egy megadott üzenetet a böngészőre, aztán befejezi az oldal futását. Akkor így kellene módosítani az előbbi két sorunkat:

```
$db = mysql_connect("localhost", "root", "") or die ("Nem sikerült kapcsolódni a szerverhez!");
mysql_select_db("jegyzet",$db) or die ("Nincs jegyzet nevű adatbázis!");
```

Lám, mennyire egyszerű a megoldás! Érdemes használni ezt a lehetőséget, ha másért nem azért, hogy hiba esetén nyomonkövethessük a weblapunkat. Sok esetben csak annyit érzékelünk, hogy weblapunk nem azt csinálja, amit szeretnénk, s nem tudjuk, hol követtük el a hibát. Elég, ha egyetlen karaktert elírunk pl. a kapcsolódáskor… - ezzel legalább az ilyen jellegű hibákat ki tudjuk küszöbölni.

A következő sor egy picit komplikáltabb. A **mysql_query** egy parancsot küld az aktív adatbázisnak a szerveren. Paraméterezése nagyon egyszerű: az első az SQL string, a második paraméter pedig az adatbázis azonosítója. Ez opcionális paraméter, ha nincs megadva, akkor a legutolsó megnyitott fájl azonosítóját fogja használni. A biztonság kedvéért érdemes használni. Figyeljük meg, hogy az SQL stringet nem kell lezárni pontosvesszővel.

A **mysql_query** által visszaadott érték kétféle lehet: FALSE, amikor nem sikerült valami miatt végrehajtani a parancsot, vagy TRUE, ebben az esetben sikeres volt a végrehajtás. Ezt a visszaadott értéket elraktározzuk egy változóban ($result), mivel később még szükségünk lesz rá.

A következő sorok egyértelműek: Egy HTML táblát készítünk, három oszloppal definiáljuk, ahová kiírjuk a táblázat fejsorait is. (Teljes név, Becenév, Fizetés) Egy ciklus következik. Egy egyszerű elöltesztelős **while** ciklus. Akinek már vannak efelől előző ismeretei, biztosan emlékszik, hogy a while ciklus után egy feltételt kell megadnunk, amely feltétel teljesülése esetén hajtódik végre a ciklusmag. Álljunk meg, s bontsuk szét a mi while ciklusunkat:

```
while ($myrow = mysql_fetch_array($result))
{
        // CIKLUSMAG
}
```

Ezzel valami probléma van! Ha feltételről beszélünk, akkor általában egy egyenlőségvizsgálatot hajtunk végre. Mielőtt bárki azt hinné, hogy itt is ez van (a *$myrow* változót hasonlítjuk a *mysql_fetch_array($result)* függvény eredményével, jelzem, hogy téved! Az egyenlőségvizsgálat nem egy darab egyenlőségjellel történik, hanem kettővel! Akkor elrontottunk valamit? Természetesen nem, hiszen

működik. A zárójelben kifejezés azt jelenti, hogy a $mysql változó értéke legyen egyenlő a **mysql_fetch_array**($result) kifejezés értékével. Ez addig TRUE (igaz) értéket ad vissza, amíg a *$myrow* változó kap értéket. Azonnal tiszta lesz, ha megtudjuk, mit csinál a mysql_fetch_array. A paramétere az előbb említett $result nevű változó, amely megadja nekünk, hogy melyik SQL parancs után érkezett eredménnyel fogunk dolgozni. Az adott függvény tehát visszatér egy asszociatív tömbbel, amely az SQL parancs által visszaadott sorok közül az aktuális lesz. Tehát első helyzetben a „select * from szemely" parancs által generált sorok (rekordok) közül az elsőt adja vissza. Lesz egy asszociatív tömbünk, amely az összes kiválasztott mezőt tartalmazza. A rekordmutató ilyenkor *automatikusan lép egyet*, tehát a következő olvasásnál már a második sort olvassa ki, aztán a harmadikat, stb. Ha elfogyott az összes sor, akkor FALSE értéket ad vissza, amely a while ciklust fogja módosítani, ekkor fog a ciklus kilépni.

Tehát a *while* ciklus addig fogja a *$myrow* változóba beolvasni a rekordokat egyesével, amíg az el nem fogyott, azaz a tábla végéhez értünk. Fontos, hogy nem a valódi táblát olvassuk, csupán az SQL parancs által generált táblában kutakodhatunk. Az előbbi példában e kettő megegyezik, hiszen a select sql sorban minden rekordot beolvastunk, de ha pl. „SELECT * FROM szemely where id>10" szűrőfeltételt adjuk ki, akkor csak a 10. rekord utáni rekordokat kapjuk meg.

Jöhet a ciklusmag! Nem olyan bonyolult, hiszen nagyrészt HTML kódot írunk. Ami nem HTML:

```
echo $myrow["csalnev"];
```

Az előbb beolvasott *$myrow* tömbről azt mondtam, hogy *asszociatív tömb*. Ezek szerint az egyes mezőkre lehet névvel is hivatkozni. De milyen névvel? Bizony, a SQL tábla mezőinek nevével. Tehát lesz $myrow["id"] változónk is, amit jelen pillanatban nem használunk. Próbáljuk meg kibővíteni még egy oszloppal a HTML táblánkat, amely a kiírt személy sorszámát is tartalmazza.

A program végén lezárja a táblát, s ezzel be is fejeztük az SQL tábla listázását. Ugye, nem is olyan bonyolult?

Mielőtt továbbhaladnánk, foglaljuk össze azokat a PHP parancsokat, amelyek segítenek számunkra megvalósítani programozói interfészünk nagy részét. Jegyezzük meg őket, hiszen mint majd látjuk, nem túloztam, amikor azt mondtam, nagy részét. Ezzel a pár paranccsal szinte mindent meg lehet majd oldani:

mysql_connect – kapcsolódás a megadott kiszolgálóhoz a megadott néven és jelszóval.

myslq_select_db – kiválasztja a megadott adatbázist.

mysql_query – végrehajt egy SQL parancsot a megadott adatbázisban.

mysql_fetch_array – beolvas egy rekordot az előzőleg végrehajtott SQL parancs eredményéből, és automatikusan lépteti a rekordmutatót.

Ezek az utasítások (függvények) szinte mindig ilyen sorrendben követik egymást: csatlakozunk a szerverhez, kijelöljük a munka-adatbázist, végrehajtunk egy SQL stringet az egyik (vagy több) táblán, majd az eredménnyel műveletet végzünk úgy, hogy soronként beolvassuk a rekordokat. Érdemes még megjegyezni, hogy az asszociatív tömbök nagy segítségünkre lehetnek. Természetesen, ha nem vagyunk tisztában a mezőnevekkel, akkor használhatunk a mezőnevek helyett számokat is, viszont ilyenkor – szokás szerint – *nullától* kezdődik a számozás!

A lekérdezést lehet bonyolítani. Ehhez csupán SQL és HTML ismeretekre van szükség. SQL-re azért, hogy a táblából kedvünk szerint válogathassunk (where, order by, group by, …), HTML-re pedig azért, hogy a visszakapott adatokat megfelelő módon tudjuk megjeleníteni a böngészőnkben.

Megtekintés

Az előző példában azt oldottuk meg, hogy a teljes listát kiírtuk a böngészőben. Sokszor szükség van arra, hogy egyedi rekordokat írjunk ki. Ennek egyik megoldása, amikor a SELECT utasításban korlátozzuk a kiírandó rekordok számát:

```
SELECT * FROM szemely WHERE id=$id
```

Az $id változónknak a hívás előtt adunk egy értéket, s vagy egy rekordot kapunk vissza, vagy egyet sem, ha nincs ilyen azonosítójú rekord. Ez az egyszerűbb megoldás. Mi most egy picit bonyolultabb, ám annál gyakorlatközelibb verziót fogunk megtekinteni.

Írjuk ki a teljes listát a képernyőre (böngészőablakba), de most módosítsunk a kiíráson egy csöppet. Hogy is volt?

```
        while ($myrow = mysql_fetch_array($result))
```

```
{
        echo "<TR><TD>";
        echo $myrow["csalnev"];
        echo " ";
        echo $myrow["utonev"]."</TD>";
        echo "<TD>";
        echo $myrow["becenev"]."</TD>";
        echo "<TD>";
        echo $myrow["fizetes"]." Ft</TD>";
}
```

Ez volt az előző kiírás kódja. Annyit módosítsunk, hogy ha egy linkre rákattint a felhasználó, akkor írja ki a felhasználó adatait külön képernyőre. Ehhez egy anchort kell létrehoznunk (nem kell aggódni, ez az „<a href…”-et jelenti). Pl. így:

```
while ($myrow = mysql_fetch_array($result))
{
        echo "<TR><TD>";
        echo $myrow["csalnev"];
        echo " ";
        echo $myrow["utonev"]."</TD>";
        echo "<TD>";
        echo $myrow["becenev"]."</TD>";
        echo "<TD>";
        echo $myrow["fizetes"]." Ft</TD>";
        echo "<TD><a href=\"egyed.php?id=".$myrow[id]."\">Megtekintés</a>/TD>";
}
```

Az utolsó sor rejti a lényeget: bevezetünk még egy oszlopot, amely a „Megtekintés” linket hozza létre. Használunk egy escape-szekvenciát (\”), amely kiírja az idézőjelet. Ha visszafejtjük a sor, az alábbi eredményt kapjuk:

```
<TD><a href="egyed.php?id=1">Megtekintés</a></TD>
```

Természetesen az id értéke nem mindig egy lesz, valószínűleg növekvő sorrendben fogja tartalmazni a számokat, a felvitt rekordok számától függően, így:

```
<TD><a href="egyed.php?id=1">Megtekintés</a></TD>
<TD><a href="egyed.php?id=2">Megtekintés</a></TD>
<TD><a href="egyed.php?id=3">Megtekintés</a></TD>
<TD><a href="egyed.php?id=4">Megtekintés</a></TD>
```

Ha több rekord van, több sor lesz, a táblázatunk egyre növekszik. A link hová mutat? Egy *egyed.php* nevű állományra mutat, paraméterként átadunk egy *id* nevű változót, természetesen értékkel együtt.

Akkor próbáljuk meg megírni az *egyed.php*-t! Mit kell tudnia? Az átadott azonosító alapján írja ki a megadott rekord többi mezőjét is, azaz adott felhasználó paramétereit. Legegyszerűbben a SELECT módosításával érhetjük el úgy, hogy bevezetünk egy WHERE feltételt:

```
SELECT * FROM szemely WHERE id=???
```

De mit írjunk a kérdőjelek helyére? (Ugye senki nem próbálta meg beírni a kérdőjeleket…?)

Én a következőképen oldanám meg:

```
<HTML>
<?
$db = mysql_connect("localhost", "root", "");
mysql_select_db("jegyzet",$db);
$result = mysql_query("SELECT * FROM szemely WHERE id=$id", $db);
$myrow = mysql_fetch_array($result);

echo "<br>Családi név: ".$myrow["csalnev"];
echo "<br>Utónév: ".$myrow["utonev"];
echo "<br>Becenév: ".$myrow["becenev"];
echo "<br>Email cím: ".$myrow["email"];
echo "<br>Fizetés: ".$myrow["fizetes"];
?>
</HTML>
```

Elmentettem *egyed.php* néven, ezzel egyedenként ki tudom listázni a rekord minden mezőjét. A jelen példában nem sok új információt nyújtunk a felhasználó számára, de ha a tábla nem csak ennyi mezőt tartalmaz, akkor már lehet jelentősége. Valószínűleg szerény példánk jó alapul szolgál egy nagyobb, bonyolultabb adattábla kezeléséhez is.

Felvitel

Ilyet már csináltunk! Bár eddig csak konzolról vittünk fel adatokat, ez is a felvitel egyik formája. Említettem, hogy nem minden esetben tudunk parancssorból felvinni, főként az ékezetes karakterek miatt. Jó lenne, ha weblapról tudnánk adatokat felvinni. Ehhez szükségünk van egy űrlap létrehozására:

```
<HTML>

<form method="post">
Családi név: <input type="Text" name="csalnev"><br>
Utónév: <input type="Text" name="utonev "><br>
Becenév: <input type="Text" name="becenev"><br>
E-mail: <input type="Text" name="email"><br>
Fizetés: <input type="Text" name="fizetes"><br>
<input type="submit" name="submit" value="Adatfelvitel ">
</form>

</HTML>
```

Mentsük el ezt a HTML forrást *felvitel.php* néven! Tudom, nem valami szép az eredmény, de céljainknak tökéletesen megfelel!

Tudtuk, hogyan kell behozni ezt az oldalt? Ha nem, akkor még érdemes átnézni az előző fejezeteket. Így: http://localhost/felvitel.php.

Jöhet a magyarázat? Sok nem lesz, hiszen a HTML alapokkal már mindenki tisztában van. Egyetlen sor igényel magyarázatot, ez pedig a *form* deklarációs sor. Már beszéltünk róla, hogy ha megnyomjuk a felvitel (*submit*) gombot, akkor az *action* által meghatározott lap hívódik meg. Ha viszont kihagyjuk az action-t, akkor a lap önmagát fogja visszahívni, de már a POST-ol értékekkel.

Emlékszünk még az „Adatátvitel a HTML és a PHP között” című fejezetünkre? Csupán ennyit kell megvalósítani, csak most majd kiegészítjük egy kis SQL-lel is.

```
<HTML>

<?
if (isset($submit))
{
$db = mysql_connect("localhost", "root","");
mysql_select_db("jegyzet",$db);
$sql = "INSERT INTO szemely (csalnev, utonev, becenev, email, fizetes)
VALUES ('$csalnev', '$utonev', '$becenev', '$email', $fizetes)";

$result = mysql_query($sql, $db) or die("Nem sikerült az adatok felvitele: ".mysql_error());

echo "Felvitel megtörtént!\n";
}
?>

<form method="post">
Családi név: <input type="Text" name="csalnev"><br>
Utónév: <input type="Text" name="utonev"><br>
Becenév: <input type="Text" name="becenev"><br>
E-mail: <input type="Text" name="email"><br>
Fizetés: <input type="Text" name="fizetes"><br>
<input type="submit" name="submit" value="Adatfelvitel ">
</form>

</HTML>
```

Megint láthatunk némi finomságot: ha újrahívtuk a lapot, azaz van értéke a $submit változónak, akkor végrehajtja a „{}”-ek közötti részt:

Kapcsolódunk az adatszerverhez, kiválasztjuk az adatbázist. Ezek után (hibakeresést most nem írtam, mindenki elvégezheti önmaga) egy *$sql* változóba írtam be végrehajtandó SQL parancsot. Mit is teszünk? Beszúrunk az adattáblába (szemely) egy rekordot, amelynek öt mezőjét határozzuk meg: utonev, csalnev, becenev, email, fizetes, a VALUES utáni öt értéket adjuk ezeknek eredményül. Figyeljük meg, hogy a numerikus adat esetén nem kötelező az aposztróf. Miért nem idézőjelet használunk? Egyszerű: az idézőjelet már ellőttük, hiszen a változó

(string) kezdődik ezzel. Ez a megszokott forma, a szöveges változó idézőjellel kezdődik, azon belül pedig aposztrófot használunk.

Most jöhet az SQL parancs végrehajtása, ami a mysql_query utasítás segítségével történik. Ezen sor végén egy hibakezelést is végrehajtunk, ám egy picit másképp: „*or die("Nem sikerült az adatok felvitele: ".mysql_error());"*, azaz hiba esetén írja ki a megadott szöveget, plusz az aktuális hibát! A **mysql_error** paraméterek nélkül alkalmazandó, a legutolsó művelet eredményét fogja visszaadni szöveges formában. Ez általában angol nyelvű, amit kis buherálással magyarítani is lehet. (A legegyszerűbb módszer, ha a /mysql/share mappában levő hungarian mappa tartalmát áttesszük az english mappába.)

A végén kiírunk egy üzenetet, amely csak akkor hajtódik végre, ha sikeres volt a felvitel. Ha probléma adódott, a lap futása megszakad, erre már nem kerül át a vezérlés.

Figyeljük meg, hogy nem adtunk meg „id" mezőt. A tábladeklaráció során meghatároztuk, hogy ez egy egyedi érték legyen, ráadásul automatikusan kerül meghatározásra (auto_increment), azaz automatikusan növekszik az értéke. Amint egy rekordot fűzünk a táblához, a rekordszámláló (id) automatikusan megkapja az értékét, s mindig az előző rekord számlálójánál eggyel nagyobbat.

A felvitel az SQL táblába PHP-n keresztül nem igényel igazán több magyarázatot. Megnyitás, INSERT, ennyi. Természetesen a MySQL még ad jónéhány apró lehetőséget kiszélesíteni lehetőségeinket, amelyeket az SQL implementáció leírásában megtalálunk.

Törlés

Úgy tűnik, immár tudunk felvinni és megtekinteni a felvitt adatokat. Azonban a szükségtelenné vált adatok törlését még nem oldottuk meg. Szükségünk lehet rá, hiszen adattáblánk nem végtelen, ráadásul a keresés során sem mindegy, mennyi rekordot kell átnéznie az SQL-motornak. Elsőként azt kell tisztázni, hogy törölni csak *rekordokat* tudunk! Teljes rekordokat, minden mezőjével együtt!

Hogy felhasználóbarát programozóként tűnjünk fel, bővítsük ki a listázás, megtekintés témakörben írt modulunkat (listazas.php). Írjunk bele még egy oszlopot, amely a „Megtekintés" mellett „Törlésre" is lehetőséget ad:

```
while ($myrow = mysql_fetch_array($result))
{
        echo "<TR><TD>";
        echo $myrow["csalnev"];
        echo " ";
        echo $myrow["utonev"]."</TD>";
        echo "<TD>";
        echo $myrow["becenev"]."</TD>";
        echo "<TD>";
        echo $myrow["fizetes"]." Ft</TD>";
        echo "<TD><a href=\"egyed.php?id=".$myrow[id]."\">Megtekintés</a>/TD>";
        echo "<TD><a href=\"torles.php?id=".$myrow[id]."\">Törlés</a>/TD>";
}
```

Nem is ez volt a nehéz, hanem a következő lépés: hozzuk létre a *torles.php* állományt, ami végre is hajtja a fizikai törlést a táblából. Figyeljük meg, hogy az előbbiekben a törlendő rekord id-je átadásra került!

```
<HTML>
<?
$db = mysql_connect("localhost", "root", "");
mysql_select_db("jegyzet",$db);
mysql_query("DELETE FROM szemely WHERE id=$id", $db);

echo "A megadott rekord törölve!";
?>
</HTML>
```

Megint hibavizsgálat nélkül készítettem el a scriptet, mindenki kibővítheti maga. Ez esetben azért arra illik figyelni, hogy több hiba is lehetséges: nincs kapcsolat a s zerverrel, nincs meg az adatbázis, esetleg a törlés nem sikeredett valamilyen ok miatt. Külön kell viszont választani azt az esetet, ha a törlés szerverhiba, esetleg jogosultsági problémák miatt nem történt meg, vagy esetleg, mert nem létezik a megadott rekordsorszám. V álasszuk külön a problémát, hiszen tudjuk: rendkívül felhasználóbarát programozók vagyunk! (bolondbiztos programozók…)

Azért, hogy ne legyen olyan gyorsan vége a fejezetnek, pár mondatban beszéljük meg a **DELETE** SQL parancs hatását. Szintaxisa rendkívül egyszerű: egy táblanévre van szükség, továbbá egy feltételre, ami alapján a feltételnek megfelelő rekordok törlésre kerülnek. A jobb olvashatóság kedvéért mindig nagybetűvel írom az SQL parancsokat. Láthatjuk, hogy az általános szintaxis valahogy így néz ki: DELETE FROM tábla WHERE feltétel. A törlés tehát – mint látjuk – rendkívül egyszerű. Csak óvatosan vele, a törölt rekordokat már nem tudjuk visszahozni!

Editálás

Meg tudjuk tekinteni a rekordjainkat, ki tudjuk törölni a feleslegessé váltakat, itt az ideje, hogy megtanuljuk, hogyan lehet módosítani az egyes mezők tartalmát. Az egyik lehetséges módszer, hogy beolvasok egy teljes sort (rekordot) a táblából, a memóriában módosítom, a táblából törlöm, majd az INSERT paranccsal felviszem a módosított rekordot. Persze ez is egy megoldás, de mi nem ezt fogjuk használni. Már csak azért sem, mert az *id* nevű mező automatikusan kerül meghatározásra, nem érdemes befolyásolni tartalmát. Ráadásul a felvitel sorrendje is megváltozik (lyukak keletkeznek a táblában), ami persze lényegtelen, hiszen egy ORDER BY klauzával megoldható a táblából nyert adatok rendezése. Aki járatos a bűvös SQL parancsokban, találkozhatott az UPDATE paranccsal is. A megadott rekord kijelölt mezőit módosíthatjuk vele:

```
UPDATE tábla SET mező1=mező1_érték, mező2=mező2_érték, …., WHERE feltétel
```

Máris meg tudnánk írni azt a myslq_query sort, amely ezt megoldja, ugye?

Jobb ötletem van! Keressük meg azt az oldalunkat, amelyet *felvitel.php* néven mentettünk le. Mit is csinált ez a kód? Első hívásnál kitett egy formot, amelyet kitöltve sor kerülhetett a második hívásra, ahol egy INSERT paranccsal felvittük a rekordot a táblába. Most ezt fogom egy „picit" kibővíteni, teszek bele pár egyedi funkciót. Lássuk!

```
<HTML>
<?
If (isset($submit))
{
$db = mysql_connect("localhost", "root","");
mysql_select_db("jegyzet",$db);
$sql = "INSERT INTO szemely (csalnev, utonev, becenev, email, fizetes)
VALUES ('$csalnev', '$utonev', '$becenev', '$email', '$fizetes')";

$result = mysql_query($sql, $db) or die("Nem sikerült az adatok felvitele: ".mysql_error());

echo "Felvitel megtörtént!\n";
}
else if (isset($update))
{
$db = mysql_connect("localhost", "root","");
mysql_select_db("jegyzet",$db);
$sql = "UPDATE szemely SET csalnev='$csalnev', utonev='$utonev', becenev='$becenev', email='$email',
fizetes='$fizetes' WHERE id=$id";

$result = mysql_query($sql) or die("Nem sikerült az adatok módosítása: ".mysql_error());
echo "Módosítás megtörtént! \n";
}
else if (isset($id))
{
$db = mysql_connect("localhost", "root", "");
mysql_select_db("jegyzet",$db);
$result = mysql_query("SELECT * FROM szemely WHERE id=$id",$db);
$myrow = mysql_fetch_array($result);
?>
<form method="post" action="<? echo $PHP_SELF; ?>">
<input type="hidden" name="id" value="<? echo $myrow["id"]?>">
Családi név:<input type="Text" name="csalnev" value="<? echo $myrow["csalnev"]?>"><br>
Utónév:<input type="Text" name="utonev" value="<? echo $myrow["utonev"]?>"><br>
Becenév:<input type="Text" name="becenev" value="<? echo $myrow["becenev"]?>"><br>
E-mail:<input type="Text" name="email" value="<? echo $myrow["email"]?>"><br>
Fizetés:<input type="Text" name="fizetes" value="<? echo $myrow["fizetes"]?>"><br>
<input type="submit" name="update" value="Módosítás ">
</form>
<?
}
else
{
?>
<form method="post" action="<? echo $PHP_SELF; ?>">
Családi név: <input type="Text" name="csalnev"><br>
Utónév: <input type="Text" name="utonev"><br>
Becenév: <input type="Text" name="becenev"><br>
E-mail: <input type="Text" name="email"><br>
Fizetés: <input type="Text" name="fizetes"><br>
<input type="submit" name="submit" value="Adatfelvitel ">
</form>
<?
}
?>
</HTML>
```

Hmmm… elég komplexnek tűnik a kód, pedig nem az. Vannak benne ismerős részek. A legalsó, HTML kód a FORM-ot írja ki, amivel már találkoztunk. A legelső *(isset($submit))* ág szintén előfordult a *felvitel .php*-ben. Ez volt az a részlet, amely az oldal újrahívása után rögzíti az űrlapon megadott értékeket.

Tehát csak két új funkcióval bővítettünk. Ha megfigyeljük s struktúrát, egy *if... else if... else* struktúrával állunk szemben. Az első rutin a *$submit* változótól függ, a második az *$update*, a harmadik pedig az *$id* függvényében következik be. Mindhárom esetben a változó „létét" vizsgáljuk, azaz hogy a változónak van-e egyáltalán értéke. Sok kezdő beleütközik abba a problémába, hogy megpróbálja az alábbi módon levizsgálni a változó értékét:

```
If ($valtozo=="") { üres változó esetén }
```

Miért nem helyes ez a sor? Mi van abban az esetben, ha a változó még nem fordult elő a kódban, azaz még sosem kapott értéket. Ekkor a változó még nem létezik, a PHP hibát fog jelezni (feltétele, hogy a hibaellenőrzés be legyen kapcsolva). Tehát ragaszkodjunk az *isset* használatához, sosem tudhatjuk a szerveren levő PHP beállításait. Ugye, milyen csúnya az, amikor egy lap PHP/SQL hibaüzenetekkel tarkított?

Szóval eme kis kitérő után mit is csinál ez a szerkezet? Ha a *$submit* változónak van értéke, akkor megnyomták a lenti űrlapon található submit gombot. Ez csak ebben az egy esetben fordulhat elő. A második eset, az *$update* változó értékétől függ. Ennek akkor lesz értéke, amikor a scriptben szereplő második űrlapot POST-olta valaki. Erre az űrlapra csak akkor kerül sor, ha az *$id* változónk értéke meghatározott. Ezt elérhetem úgy is, hogy a *felvitel.php* mögé bigygyesztek egy értékadást a böngészőben:

```
http://localhost/felvitel.php?id=1
```

Természetesen lehet más számot is írni, attól függ, mennyi rekordot vittünk már fel. Mi is történik, ha így hívom meg a lapot? Az *$id* változó értéket kap, tehát a harmadik elágazásra adódik a vezérlés. Ez konnektál az adatbázishoz, a táblából lehívja az *$id*-edik rekordot a *$myrow* nevű tömbbe. A következő részt kiemelem, hiszen az egyik legtöbbet használt formációt láthatjuk:

```
?>
<form method="post" action="<? echo $PHP_SELF; ?>">
<input type="hidden" name="id" value="<? echo $myrow["id"]?>">
Családi név:<input type="Text" name="csalnev" value="<? echo $myrow["csalnev"]?>"><br>
Utónév:<input type="Text" name="utonev" value="<? echo $myrow["utonev"]?>"><br>
Becenév:<input type="Text" name="becenev" value="<? echo $myrow["becenev"]?>"><br>
E-mail:<input type="Text" name="email" value="<? echo $myrow["email"]?>"><br>
Fizetés:<input type="Text" name="fizetes" value="<? echo $myrow["fizetes"]?>"><br>
<input type="submit" name="update" value="Módosítás ">
</form>
<?
```

Rögtön az elején kilépünk a PHP értelmezőből. Ez nem kötelező akkor, ha tovább is PHP-s utasításokat szeretnénk írni. Ugye megtehetnénk, hogy *echo* parancsokkal kiírjuk a HTML tag-eket, de ez nem kifizetődő. Ehelyett visszalépünk HTML-be. Elindítunk egy űrlapot. Az *action* részben egy eddig ismeretlen kifejezés áll. Az idézőjel után visszaváltunk PHP-be, majd kiíratunk egy változót, aztán azonnal le is zárjuk a PHP blokkot. Idézőjel bezárva, *FORM* tag lezárva. Mi ez a *$PHP_SELF* változó? Nagyon egyszerű, és azonnal érthetővé is válik a sor: az aktuális script nevét adja vissza ez a változó. Tehát írhattam volna ezt is:

```
<form method="post" action="felvitel.php">
```

Funkcióját tekintve tőkeletesen megegyezik az előzővel, de emlékszünk ugye, a régi megoldásra is, ami szintén ugyanazt a hatást eredményezi:

```
<form method="post">
```

Itt kihagytam az *action* részt. Ha kihagyom, a PHP önmagát fogja visszahívni. A három lehetőség ugyanazt jelenti, talán a második, amikor a script nevét beágyazom az oldalba nem olyan elegáns. Gondoljunk csak arra, hogy ha megváltoztatom a script nevét, akkor vele együtt kell változtatnom a scriptben alkalmazott fájlnevet is. Kerüljük az ilyen helyzeteket!

A következő sorok kiírják az űrlapot. Figyeljük meg, hogy az *$id* mezőt nem jelenítjük meg a képernyőn, ezt egy *hidden* típusú mezőbe helyezzük el. Emlékszünk talán, hogy a *hidden* mezők nem kerülnek megjelenítésre, viszont a tartalmuk POST-oláskor szintén átadódik. A többi mező megjeleníthető, ugyanazt a technikát alkalmazzuk, mint az előbb a *FORM* definíciójakor. PHP bekapcsol, változó kiír, PHP kikapcsol. **Láthatjuk, hogy a PHP-t bárhol ki és be tudom kapcsolni, a végrehajtott parancs eredménye azonnal beszúródik a HTML kódba!**

Ebben az űrlapban hozzuk létre az *$update* változót egy submit gomb formájában. Ha ezt az űrlapot küldjük el, akkor a *$update* változónak lesz értéke (Módosítás), ami az újrahívott script futását fogja más irányba terelni, hiszen a második elágazás immár gond nélkül lefut. Ami mit is csinál? Konnektál, majd végrehajt egy UPDATE SQL parancsot. Ezért kellett a hidden mező, hiszen innen tudjuk, hogy melyik rekordot kell módosítani. Kétféle eredménnyel végződhet a frissítés. Ha nem sikerült, akkor hibaüzenetet küldünk, ha igen, akkor pedig egy egyszerű üzenettel nyugtázzuk a sikert.

Megjegyzendő! Figyeljük meg, hogy a régebben mondottakkal ellentétben, most a numerikus adatnál is használtam aposztrófot. ($fizetes) Miért? Immár tisztában vagyunk azzal, hogy a PHP a művelet eredményét beilleszti a kódba. Lássunk mindkét módra egy-egy példát! Tegyük fel, hogy egy INSERT parancsot szeretnék végrehajtani egy táblában, ahol csak egy numerikus mezőt határozok meg. Így oldhatjuk meg PHP-ben:

```
$sql = "INSERT INTO tabla (numertek) VALUES ($php_valtozo)";
$result = mysql_query($sql, $db) or die("Nem sikerült az adatok felvitele: ".mysql_error());
```

Tehát beszúrunk a táblába egy rekordot, amelynek a *numertek* mezője fel fogja venni a *$php_valtozo* értékét. Egyszerű… Mivel numerikus értékről van szó, nem is tettem aposztrófok közé a PHP változóját.

Most próbáljuk meg elképzelni, mi van akkor, amikor a PHP változó értéke behelyettesítődik! Tegyük fel, hogy a *$php_valtoz* értéke 5:

```
INSERT INTO tabla (numertek) VALUES (5)
```

Eddig nincs is gond! Akkor van gond, ha a kezdő felhasználó véletlenül elfelejti kitölteni a mezőt, s úgy küldi el. Mi is lesz az eredmény?

```
INSERT INTO tabla (numertek) VALUES ()
```

Nos innen már keletkezhetnek problémák! Nézzük ezt meg egy UPDATE paranccsal is:

```
$sql = "UPDATE tabla SET numertek=  WHERE id=$id";
$result = mysql_query($sql) or die("Nem sikerült az adatok módosítása: ".mysql_error());
```

Láthatjuk, hogy hiányzik valami a kifejezésből. Ott van az, csak mivel a *numertek* változónak nem volt értéke, a kifejezésbe is ezt a „semmit” illesztette be a kiértékelő rutin. Bezzeg, ha használunk aposztrófot:

```
$sql = "UPDATE tabla SET numertek='  WHERE id=$id";
$result = mysql_query($sql) or die("Nem sikerült az adatok módosítása: ".mysql_error());
```

Máris szebb! Csak azt ne feledjük el, hogy a *numertek=’’* kifejezésben ez nem egy idézőjel, hanem két darab aposztróf, amelyek között nincs semmi… Nem ugyanazt jelenti!

Térjünk vissza egy előzőleg meghatározott kapcsolódási pontra. Arról volt szó, hogy hogyan kerülhet át a vezérlés az UPDATE űrlap kiírására. Akkor azt a megoldást választottuk, hogy a *felvitel.php* URL-jét kiegészítettük az *$id* változó közvetlen meghatározásával. Természetesen ennél egy sokkal elegánsabb (felhasználóbarátabb) megoldással kell előrukkolnunk.

Ha eddig becsületesen elkészítettünk minden modult, léteznie kell a *root* mappánkban egy *listazas.php* állománynak. Csak egy részletét mutatom meg, hogy könnyebb legyen:

```
while ($myrow = mysql_fetch_array($result))
{
        echo "<TR><TD>";
        echo $myrow["csalnev"];
        echo " ";
        echo $myrow["utonev"]."</TD>";
        echo "<TD>";
        echo $myrow["becenev"]."</TD>";
        echo "<TD>";
        echo $myrow["fizetes"]." Ft</TD>";
        echo "<TD><a href=\"egyed.php?id=".$myrow[id]."\">Megtekintés</a>/TD>";
        echo "<TD><a href=\"torles.php?id=".$myrow[id]."\">Törlés</a>/TD>";
}
```

Láthatjuk, hogy kilistázza a rekordokat, s létrehoz két további oszlopot, amelyek mindegyike egy-egy linket tartalmaz. Az első a *Megtekintés* a második pedig a *Törlés* funkciót fogja végrehajtani. Ehhez természetesen létrehoztunk egy *egyed.php* és egy *torles.php* nevű állományt. Mi a további teendő? Hozzunk létre még egy oszlopot, amely a *listazas.php*-t fogja meghívni úgy, hogy paraméterként

átadja neki az aktuális rekord azonosítóját. Nem lesz nehéz a sort hozzáilleszteni, egyik megoldása lehet ez:

```
echo "<TD><a href=\"felvitel.php?id=".$myrow[id]."\">Módosítás</a>/TD>";
```

Tehát a *felvitel.php*-nek kétféle funkciója is lesz. Az egyik - amikor paraméterek nélkül hívjuk meg -, egyszerű felviteli funkciót valósít meg. Kiír egy űrlapot, amelyiket elküldve egy INSERT utasítással beszúrja a következő rekordot a táblába. A másik, amikor paraméterrel hívjuk: ebben az esetben szintén megjelenít egy űrlapot, de az űrlap mezői nem üresek lesznek, hanem a módosítandó rekord mezőit fogják tartalmazni. Ebben az esetben egy UPDATE parancs lesz az elküldés végeredménye. Sikerült elkészíteni pár sorban a rekordok módosítását.

Keresés

Mit ér egy hatalmas adatbázis, ha nem tudunk benne keresni? Az nem elég, ha listázni tudunk benne (browse – tallózás), bizonyos helyzetekben szükség lehet arra, hogy a megadott feltételeknek megfelelő rekordokat láthassuk, esetleg rajtuk végezhessünk műveleteket. PHP-ben megoldani a keresést igen egyszerű. Voltaképpen nem is a PHP végzi a keresést, hanem az SQL szerver. A PHP csak elindítja a keresést, majd a visszaérkezett rekordokat kezelhetjük le vele. (pl. kilistázhatjuk őket)

Mivel jelen pillanatban nem törekszünk (nem törekedhetünk) tökéletességre, az alábbi megoldást javaslom: készítsünk egy űrlapot, amelyben megadhatunk egy rész-stringet, amelynek szerepelnie kell a rekordban, továbbá hozzunk létre egy legördülő listát, amelyből kiválaszthatjuk a keresés helyét, amiben keresünk (családi név, utónév, becenév, email). Nincs más dolgunk ezek után, hogy újrahívjuk a scriptet az adatokkal, s leválogatást végezzünk ezen adatok alapján. Mivel a leválogatáson kívül mindent ismerünk, lássuk az ehhez szükséges tudnivalót:
A SELECT parancsnak van egy operátora, amely arra készült, hogy a megadott változót összehasonlítsa a megadott mezővel. Ez a LIKE operátor. Működése:

```
SELECT * FROM szemely WHERE $mezo LIKE '%$kereses%'
```

Azaz: válogassuk ki a „szemely” nevű adattáblából azokat a rekordokat, ahol a mező értéke hasonlít a *$kereses* nevű, megadott stringre. Ha megfigyeljük, százalékjeleket tettem a változó elé és mögé. Ezzel azt érjük el, hogy a keresett adat elég, ha részstringként szerepel a változóban. Pár példa:

```
$mezo = 'Nagy'          $kereses = 'ag'      eredmény: egyezés!
$mezo = 'Balaton'       $kereses = 'alatt'   eredmény: nem egyezik!
$mezo = 'Nagymező'      $kereses = 'nagy'    eredmény: egyezik!
```

```
<HTML>
<?
if (isset($mit))
{
        $sql="SELECT * FROM szemely WHERE $miben LIKE '%$mit%' ORDER BY csalnev ASC";
        $db = mysql_connect("localhost", "root", "");
        mysql_select_db("jegyzet",$db) or die ("Nem tudom megnyitni az adatbázist!");
```

```
            $result = mysql_query($sql,$db) or die ("Hiba a tábla olvasása közben: ".mysql_error());
            if (mysql_num_rows($result)==0) { echo "Nincs találat!"; }
            else
            {
            echo "<TABLE BORDER=1>";
            echo"<TR><TD><B>Teljes név</B></TD><TD><B>Becenév</B></TD><TD><B>Teendő</B></TD>
</TR>";


            while ($myrow = mysql_fetch_array($result))
            {
                        echo "<TR><TD>".$myrow["csalnev"]." ".$myrow["utonev"]."</TD>";
                        echo "<TD>".$myrow["becenev"]."</TD>";
                        echo "<TD><a href=\"egyed.php?id=".$myrow["id"]."\">Megtekint</a></TD></TR>";
            }
            echo "</TABLE>";
            }
}
else
{
            ?>
            <form method="POST" action="<?php $PHP_SELF ?>">
            <table border="1">
              <tr><td>Ide írhatod a keresendő szöveget</td>
              <td>Keresés helye</td></tr>
              <tr>
              <td><input type="text" name="mit" size="28"></td>
              <td><select size="1" name="miben">
              <option selected value="csalnev">Családi név</option>
              <option value="utonev">Utónév</option>
              <option value="becenev">Becenév</option>
              <option value="email">E-mail</option>
              </select></td>
              </tr>
            </table>

            <p><input type="submit" value="Submit" name="keres"><input type="reset" value="Reset"></p>
            </form>
<?
}
?>

</HTML>
```

Ilyen egyszerű a keresés megoldása. Annyi bővítést azért tettem bele, hogy a megtalált rekordokat névsorrendbe tettem (ORDER BY). Most elég beírni a keresendő részsorozatot a szövegdobozba, kiválasztani a megfelelő mezőt, s indulhat a k eresés. Eredményképpen vagy a „ Nincs találat” szöveget, vagy a megtalált rekordokat kapjuk. Még egyetlen megjegyzés: ha csak azokat a rekordokat keressük, ahol „K” betűvel kezdődik a családi név, elég a „K%” karaktersorozat beírása. Valahogy úgy működik a „%” jel, mint a „*” karakter fájlok keresésekor. (Vigyázat, a régi DOS-os rendszerekben a „*keres*” sorozat még nem úgy működik, mint a WINDOWS-os rendszerekben.)

Összefoglalás

Az adatkezelés az egyik legjobban kidolgozott része a PHP-nek. Rendkívül gyorsan, egyszerűen lehet megoldani vele a felmerülő problémákat. Amit mindenképpen tudnunk kell, az egy alapszintű SQL tudás. E nélkül sehová sem jutunk. Ha meglehetősen járatosak vagyunk pl. az Access adatkezelő rendszerben, nem lesz semmi problémánk. A feladatok nagy része megoldható a SELECT, UPDATE, INSERT, DELETE parancsokkal, amihez már csak a WHERE, ORDER BY klauzák ismerete szükségeltetik. Természetesen, előfordulnak komplex problémák is, amelyek speciális SQL ismeretet igényelnek. Ehhez mindenképpen érdemes elolvasni a m ySQL leírását, amelyet megtalálunk a http://www.mysql.com weblapon.

Létrehozás

Ne feledjük, hogyan néz ki az adatbázis struktúra: kell egy adatbázis, amely adattáblákból épül fel, minden adattábla rekordokból áll, a rekordok pedig mezőkre vannak osztva. A létrehozáshoz meg kell tervezzük a mezőket, esetenként még optimalizálásra is szükségünk lehet. Miután a rekordstruktúra készen van, ezt táblába kell tenni, amit egy adatbázis fog magába fogadni. A nagyobbtól kell a kisebb felé haladni: adatbázis létrehozása, majd az adattábla meghatározása a mezők definiálásával. Ezt vagy PHP-ből, vagy parancssorból végezhetjük el. Készültek már rá speciális programok, amelyek segítségével vizuális módon tudjuk megtervezni a szükséges adatbázisokat. Egyik legkellemesebb kezelőfelületű program az EMS MySQL manager.

Weblapja: http://www.mysqlmanager.com/, bár csak harminc napig működik regisztráció nélkül, érdemes kipróbálni. Ahogy a nevéből is látszik, nem csak létrehozásra, de minden, SQL-lel kapcsolatos feladat elvégzésére alkalmas. Még szabvány SQL parancsokat is kiadhatunk, a létrehozott listát pedig menthetjük, törölhetjük, frissíthetjük: ahogy kedvünk tartja.

Listázás

A listázás a legegyszerűbb feladatok egyike. Csupán egy SELECT-et kell kiadnunk, majd a kapott eredményt feldolgozni. Figyelni kell arra, hogy talán azért nincs visszaadott rekord, mert esetleg hibás a parancsunk. Megismerkedtünk a „DIE” paranccsal, amely segíthet a h ibák lekezelésében. A régi, DOS-os, egyszálú programok ismerőjének ez egy kicsit érthetetlen lehet, hiszen ha egy DOS-os programban kiadtunk egy DIE (HALT, EXIT) parancsot, a program futása megszakadt, újra kellett indítani, ha szerettük volna újra megpróbálni az előzőleg hibás parancs hatását. Itt viszont erről nincs szó. Ha a DIE parancs végrehajtódott, a vezérlés visszaadódik a böngészőnek, amely kezeli az eseményeket. Tartsuk mindig észben, hogy az operációs rendszerünk multitasking felépítésű, ehhez kell igazítanunk programjainkat, az eseményvezérlés az alapvető nézőpont.

Egyéb: nem maradhatnak el a listázást befolyásoló klauzák: WHERE, ORDER BY. Ezek a legfontosabbak. Az adattáblában a rekordok a felvitel sorrendjében szerepelnek, listázáskor, ha nem használunk rendezési kitételt, ezt a sorrendet fogjuk visszakapni. Ha mégis rendezünk, megadhatjuk a rendezés mezőjét (mezőit), mely mezők alapján végezze el a rendezést, illetve növekvő, vagy csökkenő rendezés közül választhatunk (ASC, DESC). A visszaadott rekordlista csak logikailag rendez, az eredeti tábla változatlan marad! A WHERE-rel feltételeket határozhatunk meg, szűkítve ezzel a találatok számát. Használhatjuk benne a LIKE operátort, de természetesen rendelkezésre állnak a szabvány operátorok: <, >, =, <=, >=, <>.

Megtekintés

Különválasztottam a listázás funkcióját a megtekintés funkciótól. Ez a valóságban is nagyon sokszor külön kezelendő. Gondoljunk csak egy olyan lapra, ahol regisztrációra van szükség bizonyos funkciók eléréséhez. A regisztrált tagok módosíthatják saját adataikat, tehát csupán egyedi megtekintésre van szükség, valószínűleg nem láthatja senki (a rendszergazdát leszámítva) a teljes taglistát…

Technikáját tekintve nem különbözik a listázástól, hacsaknem annyiban, hogy a WHERE feltételben mindenképpen egy kulcsot kell megadnom, hogy csak egy rekord legyen a végeredmény. Ehhez érdemes átnézni előzetes tanulmányainkat a kulcsról, mint egyedi azonosítóról.

Felvitel

Logikailag talán ezt a funkciót kellene először megbeszélnünk. De mivel már létezett pár rekord a táblánkban, ezt kihagyhattuk. De lássuk, mit is kell tudni erről a feladatról. Elsősorban azt, hogy itt van először igazán szükségünk egy FORM használatára. Nagyon hasznos, ha ismerjük a FORM lehetőségeit, objektumait. Nem utolsó szempont, hogy szert tegyünk némi javascript ismeretre, meglátjuk, szükségünk lesz rá, pl. a hibakeresést könnyebben el lehet vele intézni. Fontosnak tartom a hibakezelést: nem vihetünk fel azonos azonosítójú rekordokat, vigyáznunk kell az üres mezők felvitelére. Csak akkor hajtsuk végre a felvitelt, amikor teljesen biztosak vagyunk abban, hogy az adatok megfelelőek és megfelelő formában vannak. Egy rosszul felvitt rekord olyan hibaforrás lehet, amelyet nehéz kideríteni, ráadásul a weblapra is kikerülhet egy-egy hibaüzenet, amely az egész weblapot és a készítőjét is minősíti.

Sokak által alkalmazott technika az, amit bemutattam a FORM-ok használatával kapcsolatban. A SUBMIT gomb megnyomásával újrahívódik a lap, ha az ACTION tulajdonságot nem határoztuk meg, vagy a PHP_SELF változót alkalmaztuk. Természetesen lehetőségünk van arra is, hogy egy létező PHP script nevét adjuk az ACTION értékeként. Mindenki azt választja, amelyik szimpatikus. Talán egy érv az önmagát hívó mellett: ugyanazon funkciót egy scripten belül valósítjuk meg, könnyebb transzportálni a programrészletet más weblap elkészítésekor. Olyan, mintha egy objektum lenne a script, hiszen benne van minden, ami a funkció megvalósításáért felelős. Nem kétséges viszont, hogy ha több scriptre

szétszedjük, akkor áttekinthetőbb lesz a program. (Megjegyzés: a PHP jelentős lépéseket tesz az objektumorientált PHP megteremtéséért. Az 5-ös verzió már tartalmaz beépített objektumokat, használja a s zabályokat. Egyéni véleményem, hogy ezzel elveszítjük a PHP egyszerűségét, könnyedségét. Majd kiderül, menynyivel lesz hatékonyabb a nyelv a hétköznapi felhasználói számára.)

Törlés

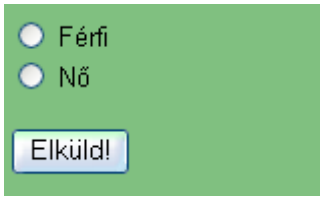
Legalább olyan egyszerű művelet, mint a listázás. Az SQL parancs végrehajtása után a rekord visszavonhatatlanul törlődik a táblából, tehát figyeljünk oda! Ugyanolyan szűkítési lehetőségeink vannak, mint listázáskor, használhatjuk a WHERE klauzát. Hol lehet mégis a törlés buktatója? (Mert van!)

Képzeljük el a következő weblapot: regisztrált tagok különböző anyagokat tölthetnek fel a szerverre. Az egyszerűség kedvéért elég lesz, ha szöveget tudnak írni, s az megjelenik egy oldalon. (Mint egy vendégkönyv…) Legyen olyan lehetőségünk, hogy szerző alapján listázhatjuk ki a szövegek címeit. Továbbá nem árt, ha az éppen nézett szöveg szerzőjéről kaphatunk némi információt, esetleg levelet tudunk neki küldeni a saját megjegyzéseinkkel. Ez eddig könnyen elintézhető. De mi van akkor, ha egy tag meggondolja magát, s kijelentkezik a rendszerből? Megszűnik az azonosítója a táblában, azaz minden általa készített szöveg tulajdonos nélkül marad. Erre gondolnunk kell! Mit tehetünk ebben az esetben? Kitörölhetjük az általa készített cikkeket, de megtehetjük azt is, hogy az elkészített szövegek tulajdonosaként a „Kijelentkezett!” jelzőt illesztjük, ilyenkor viszont szüntessük meg a lehetőséget arra is, hogy e-mailt küldhessen a felhasználó egy nem létező, vagy üres e-mail címre. Mindig gondoljunk arra, hogy törlésünkkel előre nem látható problémákat okozhatunk: mindig gondoljuk át, s varrjuk el a szálakat a program (az oldal) szerkesztése közben.

Editálás

Az egyik legnehezebb funkció. A legegyszerűbb megoldás (amit nem engedhetünk meg magunknak), hogy kiírjuk az aktuális rekordot a weblapra, majd megjelenítünk egy üres FORM-ot, amelyben felvihetjük az új adatokat. Ennél sokkal jobb megoldás, ha a FORM-ba írjuk be az előző (módosítandó) adatokat. Persze itt is vannak szabályok: az SQL-ből visszaolvasott jelszót soha nem írjuk ki sehová, még editálási céllal sem! Írjunk ki egy üres password mezőt, majd vizsgáljuk meg, hogy a felhasználó kitöltötte-e, s ha nem, akkor hagyjuk meg a régi jelszavát.

Addig nincs is nagy probléma, amíg szöveges (text) mezőket használunk. Egy példa: a személynyilvántartásunkba felveszünk ilyen mezőket: neme (férfi/nő), iskolai végzettsége (8 általános, középfokú, felsőfokú), érdeklődési köre (ide felsorolhatunk pár témakört…). A neme mezőt érdemes rádiógombokkal megoldani, hiszen egymást kizáró választási lehetőségek:




```
<input type="radio" name="neme" value="1"> Férfi<br>
<input type="radio" name="neme" value="2"> Nő<br><br>
```

Ha az egyik opciót alapértelmezettnek akarjuk beállítani (itt nem!), akkor a CHECKED paramétert kell alkalmazzuk. A jelen példában a $neme változó értéke 0, 1, vagy 2 lesz, annak függvényében, mit választott a felhasználó. Ezt kell majd letárolnunk. De nem ez a probléma, hanem, hogy hogyan járjunk el editálás esetén. Itt még nem olyan nehéz! A rádiógombok egy tömbben vannak tárolva, amelyek számozása 0-tól kezdődik. (Miért is ne? ☺) Ha bármelyik tömbelem CHECKED értékét true-ra (igazra) állítjuk, az bekapcsol, a többi automatikusan kikapcsolt állapotot vesz fel. Például a

```
document.formneve.neme[1].checked=true;
```

sor a FORMNEVE űrlapunk neme nevű rádiógombjának első (nullától) elemét bekapcsolja. A fenti példa alapján a Nő opció lesz kiválasztva. Editáláskor csak annyit kell tennünk, hogy a fenti példa „1” értékét átírjuk a beolvasott változó értékére. Mivel ez egy javascript betét, a következőképpen oldanám meg:

```
<SCRIPT>document.formneve.neme[<? echo $neme; ?>].checked=true;</SCRIPT>
```

Ezt a sort a FORM kiírása közben, a rádiógombok létrehozása után kell végrehajtani. Egyetlen hátulütője van: a $neme változónak legyen értéke. Ha nincs, akkor tömbindex hiányában hibaüzenetet fogunk kapni.

Ugyanígy járhatunk el CHECKBOX-ok létrehozásakor is, csak abban az esetben a TRUE-ra váltás nem jelenti a többi opció kikapcsolását. A kikapcsolást a FALSE érték átadásával tehetjük meg:

```
<SCRIPT>document.formneve.neme[<? echo $neme; ?>].checked=false;</SCRIPT>
```

Az iskolai végzettség esetén legördülő listát választanék. Hasonló a helyzet, csak az adatokat tartalmazó tömb neve változott, továbbá nem CHECKED, hanem SELECTED a kiválasztást jelző opció.

```
<SCRIPT>document.formneve.vegzettseg.options[<? echo $vegzettseg; ?>].selected=true;</SCRIPT>
```

Láthatjuk, hogy egy *options* tömb szolgál a választási lehetőségek tárolására. Ez magára a SELECT objektumra mutat, hiszen az is egy tömb. Minden tömbelemnek van egy *selected* tulajdonsága, amit állítani tudunk TRUE, vagy FALSE értékre.

Keresés

A keresésre csak nagyon szűk határok között lehet ötleteket adni, hiszen minden feladat egyedi. Ez a funkció a **szintézis**. A felvitel, a törlés, a listázás, az editálás lehet bármilyen spártai, a keresés az, amelyik konkrét eredményeket ad vissza, s ezen eredményeket fogjuk használni további munkáinkhoz. Mondhatnám azt is, hogy a leglényegesebb része az adatfeldolgozásnak. Egyben azt hiszem, a legnehezebb is. Tudom, példánkban kimerült a keresés a SELECT alkalmazásával és egy WHERE feltétel megfogalmazásával. A valós életben viszont lesznek bonyolultabb feladatok is, amelyekhez esetleg segédtáblákat kell létrehoznunk, csoportokba kell szerveznünk adatainkat. A lehetőségek adottak, a mySQL pedig kiváló segéd ehhez.

Összefoglaló kérdések

- Milyen lépései vannak egy adatbázis létrehozásának PHP-ben?
- Milyen mezőtípusokat alkalmazhatunk egy tábla létrehozásakor?
- Hogyan nyithatjuk meg az adatbázist?
- Mikor kell lezárni egy adatbázist?
- Mire szolgálnak az alábbi SQL parancsok:
    - *SELECT*
    - *INSERT*
    - *DELETE*
    - *UPDATE?*
- Mire használhatjuk a *WHERE* és *ORDER BY* klauzákat?
- Felvitel során milyen változók esetén használunk aposztrófot?
- Melyik mezőket kell kötelezően kitöltenünk felvitel esetén?
- Melyik PHP függvény adja vissza a *SELECT* rekordjainak (sorainak) számát?
- Milyen hibák léphetnek fel törlés esetén, és hogyan orvosolhatjuk őket?
- Mi a különbség az *INSERT* és az *UPDATE* felviteli módszere között?
- Editáláskor hogyan jelenítjük meg az előzőleg beolvasott jelszót szerkesztés céljából?
- Mire szolgál a *LIKE* operátor, s mi a helyettesítő karakter SQL-ben?

Eljárások és függvények

Mint minden programozás nyelvben, a PHP-ben is vannak eljárások és függvények. Pascalban jobban megkülönböztetik őket: *Procedure*-nek hívják az eljárásokat, ezek nem adnak vissza értéket, csupán végrehajtanak egy rutint, majd visszatérnek. Ezzel szemben a függvények (*Function*) végrehajtván a rutint, visszaadnak egy kiszámolt értéket, amellyel tovább számolhatunk. Ezzel szemben a PHP-ben nincs ilyen megkülönböztetés, csak *Function*-ök vannak, amelyek vagy visszaadnak értéket, vagy sem. Hívásuk is attól függ, mit szeretnénk vele kezdeni. Lássunk egy egyszerű példát:

```
$mysql_host = "localhost";
$mysql_username = "root";
$mysql_password = "";
$mysql_db_name = "jegyzet";


function db_connect()
{
global $mysql_username, $mysql_password, $mysql_host, $mysql_db_name;

$db = mysql_connect("$mysql_host", "$mysql_username", "$mysql_password") or die("Hiba történt az SQL
szerverhez történő csatlakozáskor!");
            mysql_select_db("$mysql_db_name", $db) or die("Hiba történt az adatbázis kiválasztásakor!");

            return $db;
}
```

Nevezzük el a fájlt *unit.inc*-nek, mentsük le. Menjünk sorban, mi, micsoda! Az első négy sorban változókat deklaráltunk. Ezek a változók, úgynevezett globális változók lesznek, a program minden pontján elérhetőek lesznek, értékük megváltoztatható. Ezek után jön a *Function*. A kulcsszó után megadjuk a függvény nevét, majd zárójelben a bemenő paraméterek listáját határozhatjuk meg. Jelen függvénynél ez hiányzik, tehát nincs bemenő paramétere. Kapcsoszárójellel ({) indítjuk és zárjuk (}) a függvényünket. Közöttük található a függvény törzse. Az első sorban máris egy rendkívül fontos parancsot fedezhetünk el: a *GLOBAL* parancs ez. Hatása, hogy az előzőleg globálisan deklarált változókat a függvényen belül is elérhetővé teszi. Ha ezt a sort kihagynánk, az összes globális változó lokálisként jönne létre, azaz hatásuk csak a függvényen belül érvényesülne. Mivel globálisan már meghatároztuk a változók értékét, érdemes beimportálnunk őket a függvénybe.

A következő sorok ismerősek lehetnek: csatlakozunk az adatbázis szerverhez, majd az adatbázishoz! Megvalósítjuk a hibakezelést is, majd az utolsó sorban visszaadunk egy értéket ($db).

Miért jó ez nekünk, ha készítünk egy ilyen függvényt? Csak egyszer kell megírnunk, s minden egyes alkalommal elég lesz csak meghívni. Hogyan hívjuk meg? A PHP erre kétféle megoldást is kínál: az egyik az *INCLUDE*, a másik pedig a *REQUIRE* függvény. Mindkettő egy megadott fájlt illeszt be a forrásba. A leglényegesebb különbség közöttük, hogy az *include* minden egyes előforduláskor beilleszti a fájlt, míg a *require* csak egyszer. Lássunk egy példát:

```
<?
```

```
include("unit.inc");
?>
```

Ha ezt az egy sort beszúrjuk valamelyik scriptünkbe, a „unit.inc” mintegy fizikailag beillesztődik a megadott helyre. Ettől kezdve elég lesz csak a *db_connect()* függvényt meghívnunk:

```
$valtozo = db_connect();
```

Ha nem akarunk a visszatért értékkel semmit sem kezdeni, meghívhatjuk eljárásként is e függvényt:

```
db_connect;
```

Ilyen egyszerű függvényből eljárást készíteni! Itt is látszik, mennyire szabad nyelv a PHP, mennyire nem törekszik a fölösleges konvenciók betartására.

Milyen függvényeket érdemes még elkészítenünk. Ez teljes mértékben a feladattól függ. Személy szerint az űrlapok változóinak nullázására, default értékadásra, űrlapok eredményeinek vizsgálatára, hibaüzenetek kiírására használom leginkább. Egyre inkább terjed az a programozási módszer, hogy a program által kiírt összes szöveg egy fájlban található (pl. egy tömbben), majd azokat használja fel a programozó. Gondoljunk bele, milyen egyszerű ebben az esetben a teljes weblapot átírni egy másik nyelvre, hiszen csak egy fájl tartalmát kell átböngészni, s lefordítani. Hasznos ötlet, ezzel magunkat is megkímélhetjük a megrendelők folyton változó ízlésétől… Próbáljunk modulárisak lenni, ebben nagy hasznunkra lesz az *include* és a *require* függvény.

Nyomkövetés

Többfelhasználós rendszerek egyik alapvető feladata a felhasználóik azonosítása és a jogosultságaik betartatása. Egy ilyen rendszertől többek között azt is elvárjuk, hogy a beléptetett felhasználók adatai, az általuk kiadott parancsok nem keverednek össze, egyértelműen megállapítható legyen minden egyes műveletről, hogy ki volt a kezdeményezője. A következő részben erre keresünk lehetséges megoldásokat.

Érdemes először azt tisztáznunk, hogy mit jelent a nyomon követés Web-es környezetben, s hogy miben tér el a h agyományos, ún. asztali alkalmazások hasonló indíttatású feladataitól.

Ennek megvalósítása a mai RAD (Rapid Application Development) eszközökkel általában viszonylag egyszerű megvalósítani, mivel azokkal együtt készen kapjuk a magas-szintű könyvtárakat, így legtöbbször nem kell törődnünk a szerverhez való kapcsolat-felépítéssel, és - nem állandó kapcsolat esetén - az egyes kérések közötti azonosítással. Sok esetben nem is kell tudnunk, hogy hogyan kommunikáljunk hálózaton, vagy hogy egyáltalán hálózaton keresztül kommunikálunk. Sajnos a HTTP protokollra épülő fejlesztői környezetek (ha egyáltalán beszélhetünk ilyenekről) még gyerekcipőben járnak e témában, ezért a munka nagy részét nekünk kell elvégeznünk. A probléma forrása a HTTP protokoll, mivel az egy ún. á llapot-mentes (stateless) protokoll, ami annyit tesz, hogy két oldal-lekérés között semmilyen kapcsolat nincsen. Tehát lefordítva PHP-re: egy program a két futása között olyannyira nem tud kapcsolatot teremteni, hogy azt se tudja megmondani, ugyanaz a felhasználó futtatta-e kétszer. Ezt a HTTP protokoll okozza, melynek megoldása az lenne, hogy amíg fut az egyik program addig valami köztes tárolóhelyre elmentjük az átadandó adatokat, majd a másik program meghívásakor kiolvassuk és feldolgozzuk ezeket. A következő részben erre az egyszerűnek tűnő tárolás-olvasásra keresünk lehetséges megoldásokat.

Módszerek

Olyan eljárást kell keresnünk tehát, hogy az egyik lapon használt PHP változóinkat át tudjuk adni egy másikra, méghozzá úgy, hogy az nem az első program hívja a másodikat, sőt az elsőnek arról sincs tudomása, hogy mi lesz a második (lehet akár önmaga is), mivel ez a felhasználó cselekvésén múlik. Ebből kiindulva kézenfekvőnek tűnik, hogy a PHP-ből generált oldalba rejtsük el az átadandó információkat, mivel legtöbbször az az egyetlen produktum, ami létrejön. Ezt indokolja az is, hogy követnünk kell a felhasználót, amit így könnyen elérhetünk mivel a neki előállított oldalon elrejtett adatokat biztos, hogy ő fogja felhasználni (még ha tudtán kívül is). A gyakorlatban ez például annyit jelenthet, hogy beléptetünk egy felhasználót és azt szeretnénk, hogy a belépés ténye megmaradjon oldalról oldalra (nem mutat túl jól, ha minden lap elején be kell ütni a jelszavát), ezért miden lapra valamilyen módon elhelyezzük mondjuk a nevét és a jelszavát (nem túl biztonságos tudom, csak a példa kedvéért legyen most így), amikor pedig meghív egy új lapot (PHP programot) erről a lapról "átadódik" ez a két adata. Már csak ez utóbbit kell kitalálnunk és kész is vagyunk.

Mivel már járatosak vagyunk egy picit a HTTP protokollban, és az űrlapok feldolgozásában, már találkozhattunk a GET metódussal. De még azok is, akiknek így nem ismerős a neve láttak már hasonlót, mert a keresők például ezt az eljárást használják adatátadásra. Ez csak annyiból áll, hogy az URL-t úgy alakítjuk, hogy az általunk kívánt adatok is bele kerüljenek. Például legyen a felhasználónév *jancsi*, a jelszava pedig *asdf*. Ezt a két adatot a következőkképpen tudjuk átadni URL-ben a helyi környezetet figyelembe véve:

```
http://localhost/levelek.php?user=jancsi&password=asdf.
```

A levelek.php-ben ekkor két változónk lesz: $user=jancsi és $password=asdf (gondolom nehéz volt kitalálni). Amelyik oldalról tovább szeretnénk vinni Jancsi adatait, azon az URL-eket úgy kell létrehoznunk, hogy tartalmazzák a fenti két változót. Például egy levelező rendszer menüjében (mondjuk menu.php) a következő linkek lehetnének generálva:

```
ujlevel.php?user=jancsi&password=asdf
olvas.php?user=jancsi&password=asdf
```

Ha jobban elgondolkozunk ezen a megoldáson, hamar felötlik bennünk, hogy ez nem mondható túl biztonságosnak, mivel az átadott URL-en keresztül bárki olvashatja a felhasználónevünket és a jelszavunkat. Nyilvánvaló tehát, hogy az URL-ben csak olyan adatokat érdemes átadni, melyeknél nem kritikus a mások - vagy akár a felhasználó - általi láthatóság.

Másik oldalba rejtett adatátadási mód az űrlapok mezői közé rejtés. Ehhez a már többször is alkalmazott HIDDEN űrlapmezőt fogjuk alkalmazni, ami pont megfelel a mi céljainknak feltételezve persze, hogy űrlapokat szeretnénk használni. Lássuk, hogy néz ki a levelezős példánk:

```
<INPUT type="hidden" name="user" value="jancsi">
<INPUT type="hidden" name="password" value="asdf">
```

és az ezt generáló PHP kód:

```
<?
echo "<INPUT type='hidden' name='user' value='$user'>";
echo "<INPUT type='hidden' name='password' value='$password'>";
?>
```

Egyszerűen behelyettesítjük az aktuális lapon érvényes $user és $password változónkat az űrlap két megfelelő mezőjébe, így a program futása után is megmaradnak adataink és átadódhatnak a következő PHP programnak.

A probléma ezzel az eljárással hasonló, mint az előzővel: az adatok könnyedén leolvashatók. Igaz itt már a HTML forrást kell megnéznie a felhasználónak. Ez komoly gondot okozhat például akkor, ha egy virtuális boltot szeretnénk készíteni. Ekkor el kell tárolnunk, hogy mit tett a "kosárba" a látogató, mennyit kell fizetnie, és ehhez hasonló adatokat a vásárlásról, amiket nem szeretnénk, hogy a felhasználó "kézzel" saját maga tudjon manipulálni. A HTML űrlapban való tároláskor viszont sajnos a látogató letöltheti az űrlapot (a HTML forrást) a

saját gépére és a saját kedvére alakíthatja úgy, hogy ebből a mi programjaink semmit nem észlelnek.

Sütik

De sebaj, keresünk egy jobb megoldást. A ma elterjedt összes böngésző közül nagy részük támogatja az ún. cookie-k (sütik) tárolását a látogató számítógépén. Ezek a kis sütik általában egyszerű szöveges állományok, amelyeket a böngésző a saját könyvtárában vagy egyszerűen csak a memóriájában szokott tárolni (az utóbbi természetesen a böngésző bezárása után elveszik). Ezzel még jobban megnehezíthetjük a kritikus adatok meghamisítását vagy ellopását. Viszont sajnos még mindig megmarad a lehetőség az állományrendszerben tárolt sütik megváltoztatására, illetve egy "felspécizett" böngészőprogramon keresztüli adathamisításra. Azért lássuk a szokásos levelezős példánkat sütikkel:

```
<?
// az első lapon (menu.php) beállítjuk a sütiket:
setcookie("user", "$user");
setcookie("password", "$password");
?>
```

```
<?
// majd a következőn (mondjuk ujuzenet.php) kiolvassuk az értékét:
echo "a belépett felhasználó neve:$user<br>";
echo "jelszava: $password<br>";
?>
```

Látható, hogy a sütik kiolvasása nagyon egyszerűen működik PHP-ben: az előzőleg tárolt sütik nevével megegyező változóban kapjuk meg az értékeket.

Session-kezelés

Fejlesszük egy kicsit tovább a sütis megoldást. A gondunk ugye az, hogy valamiféleképpen azonosítanunk kell a felhasználót ahhoz, hogy kövessük, viszont mégsem szeretnénk, hogy az ő gépére átkerüljenek a hozzá tartozó bizalmas adatok. A probléma megoldása nagyon egyszerű (nem is tudom, miért nem jutott eddig az eszünkbe): állítsunk elő egy egyedi azonosítót a felhasználó aktuális belépéséhez és ezt az azonosítót tároljuk a kliens gépen (a tárolás módja az előzőleg felsorolt körül bármelyik lehet, de legkényelmesebb a cookie). Majd ezt az azonosítót minden PHP programból kiolvassuk és egy a szerveren tárolt adathalmazt azonosítunk vele. Ez az adathalmaz lehet például egy sima szöveges állomány, vagy egy adatbázisban tárolt rekord a felhasználó adott belépéshez tartozó adataival (neve, jelszava, mit vásárol, mennyiért stb.). A lényeg, hogy ezek az adatok soha nem kerülnek ki a szerverről, csak egy véletlen-generált hosszú azonosító, amit hiába próbál a felhasználó változtatni, mert az egyetlen lehetősége az lenne, hogy eltalálja egy másik látogatónak generált azonosítót, ami viszont a nagy véletlen számok miatt hiúsul meg (próbáljon valaki kitalálni egy 30 karakteres, betűkből és számokból álló (hexa) azonosítót). Ezen kívül az azonosítóba bele "rejthetjük" például a belépés dátumát és, ha egy nappal később akarják használni

ugyanazt az id-t, akkor kiírjuk, hogy az idő-korláton kívül van, lépjen be újból. Fontos megjegyezni, hogy ennek az azonosítónak semmi köze nincs a felhasználó nevéhez, jelszavához vagy egyéb adatához. Ez minden belépéskor generálódik és csak a szerveren tárolt adathalmazt azonosítja, amiből persze ki tudjuk olvasni a felhasználó egyéb adatait. Amikor a felhasználó elhagyja a site-ot, vagyis kilép, akkor ezek az azonosító alatt eltárolt ideiglenes adatok törlődnek, és ha legközelebb visszajön a látogató, akkor megint új rekord készül az aktuális látogatásáról. A címben említettem egy ronda szót: session, illetve session-kezelés. Az előbbiekben taglalt módszert hívják session-kezelésnek. Maga a session pedig a felhasználó belépésétől a távozásáig tartó folyamat (magyarul talán munkafolyamatnak lehetne fordítani), melynek célja a felhasználó tevékenységéből adódó információk összegyűjtése későbbi felhasználás céljából (ilyen cél lehet például a vásárlás, az oldal testreszabása, stb.).

Gyakorló példák

Egy egyszerű számlálóprogramot nézünk meg három megvalósításban (URL, Cookie, PHP Session). Mindegyikben a lényeg az, hogy a növelt számláló értékét továbbadjuk a program következő példányának. A programok újbóli meghívásakor megkapjuk a számláló előző értékét, majd megnöveljük egyel. Az újbóli meghívás a különböző megoldásokban különbözőképpen működik. URL-beli átadás esetén csak az URL-re való kattintással oldható meg a számláló értékének továbbadása, sütik és a PHP-es session-ök esetében egy sima "reload" is megteszi (kivéve, ha ki vannak kapcsolva a sütik a böngészőben).

URL-beli átadás példa:

```
<html>
<body>
<?
if (!isset($counter)) $counter = 0;
else
        $counter += 1;
        echo("a számláló értéke: $counter<br>");
        echo("<a href=\"$PHP_SELF?counter=$counter\">növel</a>");
?>
</body>
</html>
```

Ez a legegyszerűbb módszer, ugyanakkor láttuk, hogy ennek van a legtöbb hátránya is. Akkor érdemes alkalmaznunk, ha egyszerű, nem bizalmas adatokat kell átadnunk (pl. keresőknél a kulcsszavak és hogy hányadik találattól kell kezdeni a megjelenítést).

A következő programban sütiket használunk a számláló értékének köztes tárolására:

```
<?
 if (!isset($cookiecounter))
 {
   $cookiecounter = 0;
 }
 else
   $cookiecounter += 1;
```

```
   setcookie("cookiecounter", $cookiecounter);
?>


<html>
<body>
<?
   echo("a számláló értéke: $cookiecounter<br>");
   echo("<a href=\"$PHP_SELF\">növel</a><br>");
?>
</body>
</html>
```

A *setcookie* függvénnyel tároljuk a $counter változó aktuális értékét, a program többi részében viszont nem kell vele törődnünk, mert a PHP automatikusan konvertálja a cookie-kat változókká, ezért tudjuk használni egyszerűen $counter-ként. **Fontos megjegyezni, hogy a *setcookie* függvény HTTP fejlécek használatával működik, ezért - fejlécről lévén szó- a legelső HTML kiíratás elé kell raknunk a függvényhívást.**

Lássuk a megoldást PHP-beli session-ökkel:

```
<?
   session_start();
?>

<html>
<body>
<?php
  if (!isset($counter)) {
     $counter = 0;
     session_register("counter");
 }
 else
     $counter += 1;
     echo("a számláló értéke: $counter<br>");
     echo("<a href=\"$PHP_SELF?" . SID . "\">növel</a><br>");
?>
</body>
</html>
```

A *session_start* függvényhívás állítja elő a session-ben tárolt globális változókat, vagy létrehoz egy új session-t, ha eddig még nem volt az adott látogatónak. Ez után már nyugodtan használhatjuk a $counter változót, melynek értékét a PHP a program befejeztével újra eltárolja a szerveren. Az session első használatakor tudatnunk kell a PHP-vel, hogy milyen globális változókat szeretnénk maradandóvá tenni. Erre szolgál a *session_register* függvény. A PHP dönti el (illetve beállíthatjuk), hogy a session azonosítóját sütiben vagy az url-ben adja át. Ezért, ha sütiket használunk, itt is figyelnünk kell rá, hogy a *session_start* függvény a legelső kiíratás előtt szerepeljen. Ha azonban nem használunk sütiket (mert mondjuk a felhasználó nem engedélyezi őket), akkor a session azonosítóját át az url-en keresztül kell átadnunk, így már nem elég egy egyszerű frissítés gombot nyomni a böngészőben, csak a link-re kattintva tudjuk fenntartani a session-t. A session azonosítót a SID konstansban tudjuk elérni, ezt kell az oldalon szereplő url-ek mögé beszúrnunk:

```
echo("<a href=\"$PHP_SELF?" . SID . "\">növel</a><br>");
```

Eddig láttuk a három módszert. Egyik-másik már egész használhatónak tűnik, de mindegyikben vannak problémák. Talán a harmadik megoldással lehetne kezdeni valamit.

Autentikáció

Biztosan sok weblapon találkoztunk már az előző fejezetben tárgyalt nyomkövetéssel. A nyomkövetés azonban önmagában nem elég, legtöbbször kombinálják egy azonosítással, ami egy beléptetésen (login) keresztül valósul meg. Ilyen esetekben bekérik a felhasználó nevét, majd a jelszavát. Ha sikeres volt a bejelentkezés, a felhasználó hozzáférhet a saját adataihoz. Erre a legegyértelműbb példák az ingyenes levelezőoldalak. Hogyan is kell egy ilyen oldalt elkészíteni? No nem a levelezőrendszerre gondoltam, csupán a felhasználó beléptetésére, illetve a jogosultságok meghatározására. Egy komplett beléptetőrendszert fogunk a fejezet végére összeállítani, minden buktatójával egyetemben.

Adatok

Első feladatunk az lesz, hogy megtervezzük az adattáblát, amelyben a beléptetéshez és a nyomon követéshez szükséges adatokat fogjuk tárolni. Voltaképpen a tag (felhasználó) törzsadatait kell tárolnunk ebben a táblában. Mire lehet szükségünk? Egy azonosító mindenképpen szükséges lesz, ez alapján fogjuk keresni, listázni, törölni a rekordokat. Milyen személyes adatokat fog tárolni a rekord? Egy nevet biztosan. Hasznos, ha két nevet tárolunk: egyik a belépési név (felhasználói név), a másik pedig a tag teljes neve. Ki tudja, mikor lesz rá szükségünk? Aztán persze a jelszó! Erről még beszélünk! Kell ezeken kívül az e-mail cím, ha esetleg levelet akarunk neki küldeni. De akkor is jó szolgálatot tesz, ha a tagunk elfelejtette a jelszavát. Aztán egy számláló! Ebben fogjuk tárolni, hogy hányadik alkalommal jár már a felhasználó a lapunkon. Vigyázat, ez nem látogatószámláló, hanem egyéni számláló. Ha már számlálót hozunk létre, kell egy dátummező is, amelyben az utolsó látogatás dátumát és időpontját tárolhatjuk. S ha már dátum, tegyük el azt is, hogy tagunk mikor regisztráltatta magát rendszerünkbe.

Ezek eddig mind viszonylag egyértelmű mezők. Egy utolsót azért még vegyünk számításba: ez pedig a munkamenet azonosító lesz (session_id), amely az azonosításhoz szükségeltetik.

Egy tanács: programozásban jártas szakemberek is elérhetik a program olyan szintjét, amikor az eljárások, a változók száma annyira felduzzad, hogy nehéz lesz őket nyilvántartani. Ezért – ilyen esetekben – trükkhöz szoktunk folyamodni: a változók nevét úgy választjuk meg, hogy egyértelmű legyen, honnan származik. Nálam ez úgy történik, hogy a több helyen is használt változók neve elé azonosító jelet teszek. Pl. ha a tábla mezőinek nevét határozom meg, nem elégszem meg az *id* változóval, mint azonosítóval. Lehet, hogy egy másik táblában szintén használni fogom ezt a mezőnevet, vagy össze fogom keverni a sok hasonló azonosítóval. Ezért ha a tábla neve *members*, akkor egy „m_” tagot helyezek minden mező elé, amely ehhez a táblához tartozik. Így lesz m_id, m_name, m_password, stb. mezőnevünk. E a trükköt használva elkerülhetők a kavarodások, meglátjuk, ebben lesz is részünk.

Mivel meghatároztuk a tárolandó mezőket, itt az ideje meghatározni a mezők típusait, illetve hosszát. Leírom a saját megoldásomat, aztán egy csöppnyi magyarázatot is adok hozzá:

```
a_id smallint unsigned,
a_name varchar (16) default '' not null,
a_password varchar (32) default '' not null,
a_fullname varchar (50),
a_email varchar (50),
a_counter mediumint unsigned,
a_lastvisit datetime,
a_regdate datetime,
a_session_id varchar(32)
```

Mint az előzőekben megbeszéltük, a mezőnevek elé egy azonosító karaktersorozatot teszek, jelen helyzetben a tábla nevének kezdőbetűjét. A tábla neve *auth* lesz. Lássuk a mezőket:

a_id: ez lesz a tag azonosítója. A smallint változó –32768-tól 32767-ig terjed, de unsigned (előjel nélküli) változata 0-65535 közötti számokat tud tárolni. Egyelőre talán elég lesz, nem szándékszunk ettől több felhasználót fogadni.

a_name: a felhasználó belépési neve. Talán 16 karakter elég lesz, érdemes figyelni rá, hogy legalább három, legfeljebb 16 karakter engedjünk beírni.

a_password: a bejelentkezéshez szükséges jelszó. Hogy miért 32 karakter hosszú? Fogadjuk el, aztán később elmagyarázom.

a_fullname: teljes név. Maximum 50 karakter.

a_email: e-mail cím, szintén elég lesz 50 karakter.

a_counter: a személyes számláló. A mediumint típus –8388608-től 8388607-ig vehet fel értéket. Mivel ez is előjel nélküli, 0-tól 16777215-ig írhatunk bele. Használhattunk volna smallint-et is, így legalább megismerhettük ezt a típust is.

a_lastvisit, a_regdate: az utolsó látogatás dátuma, és a regisztráció dátuma. Mindkettő tárol egy év-hó-nap típusú dátumot és egy óra:perc:másodperc formátumú időpontot.

a_session_id: munkamenet azonosító. Használatáról később.

Hozzunk létre egy PHP állományt (create.php néven), amely létrehozza az adatbázist (ezt nyugodtan megtehetjük kézzel is), majd létrehozza a táblát is (*auth* néven). Még nincs vége! Kérje be a rendszergazdai jelszót és e-mail címet, majd tárolja le mint 0. rekordot a táblába. Mivel ez még elsőre nehéznek tűnhet, próbáljuk meg együtt!

unit.inc

Mielőtt fejest ugranánk a témába, egy picit gondolkodjunk el. Előző fejezetben már beszéltünk az eljárások és függvények létrehozásának lehetőségéről, illetve az *include* és a *require* függvényekről. Használjuk fel az ott tanultakat, s hozzunk létre egy *unit.inc* nevű include (beszúrandó) állományt. Határozzuk meg az SQL szerver és az adatbázis eléréséhez szükséges globális változókat és írjuk meg a belépést és hibakezelést biztosító függvényünket:

```
<?

$mysql_host = "localhost";
$mysql_username = "root";
$mysql_password = "";
$mysql_db_name = "auth_test";


function db_connect()
{
global $mysql_username, $mysql_password, $mysql_host, $mysql_db_name;

$db = mysql_connect("$mysql_host", "$mysql_username", "$mysql_password") or die("Hiba történt az SQL
szerverhez történő csatlakozáskor!");
    mysql_select_db("$mysql_db_name", $db) or die("Hiba történt az adatbázis kiválasztásakor!");

    return $db;
}

?>
```

Piciny magyarázat, ami – remélhetőleg – már nem is szükséges: A *localhost*-ra csatlakozunk *root* névvel, nincs jelszó, az adatbázis neve auth_test. Aztán létrehoztunk egy *db_connect* nevű függvényt, amely megpróbál az SQL szerverhez csatlakozni, hiba esetén hibát jelez, majd kiválasztja a munkaadatbázisunkat, kiegészítve a hibakezeléssel itt is. Visszatérésként az adatbázis azonosítót adja meg, amire még szükségünk lehet…

Figyeljük meg a függvény első sorában a *global* használatát. Az előzőleg deklarált globális változókat elérhetővé tettük a függvényen belül is. Ha ez a sor véletlenül kimaradna, a változók nem lennének deklarálva, nem lenne értékük, hibás eredményt kapnánk. Ez elég kellemetlen hibaforrás, kezdők jellegzetes hibája, de még a gyakorlottabbak is belefutnak néha.

Ne tegyük el messzire ezt a *unit.inc*-et, majd még bővítjük…

create.php

Egy előző fejezetben (Adatbázis és adattábla) már hoztunk létre adatbázist és adattáblát. Elindítva a mysql.exe-t, a szükséges SQL parancsok segítségével létrehozhattunk mindent. Próbáljuk meg másként! Ha már helyi gépen dolgozunk, használjuk ki, hogy nincsenek jogosultsági korlátjaink. Ez a változat Linuxon nem működne, ott a régi módszer szerint tudunk csak dolgozni. Lássuk megint a teljes programot, aztán a szükséges tudnivalókat:

```
<?
include("unit.inc");

if(!isset($admin_password))
{
$db = mysql_connect($mysql_host, $mysql_username, $mysql_password) or
die("Hiba történt az SQL szerverhez történő csatlakozáskor!");

@mysql_query("drop database $mysql_db_name");
mysql_query("create database $mysql_db_name") or die("Hiba történt az adatbázis létrehozáskor:
<b>".mysql_error()."</b>!");
mysql_query("use $mysql_db_name") or die("Hiba történt az adatbázis kijelölésekor!");


mysql_query("
```

```
CREATE TABLE auth (
  a_id smallint unsigned,
  a_name varchar (16) default '' not null,
  a_password varchar (32) default '' not null,
  a_fullname varchar (50),
  a_email varchar (50),
  a_counter mediumint unsigned,
  a_lastvisit datetime,
  a_regdate datetime,
  a_session_id varchar(32),
  PRIMARY KEY (a_name)
)", $db) or die ("Hiba az <b>auth</b> tábla létrehozásakor: <b>".mysql_error()."</b>!");

}

if (isset($admin_password) AND isset($password_set))
{
        $admin_password = md5(trim(addslashes($admin_password)));
        $date = date("Y-m-d H:i:s", time());

        $db = db_connect();
        mysql_query("INSERT INTO auth
(a_id,a_name,a_password,a_fullname,a_email,a_counter,a_lastvisit,a_regdate,a_session_id) VALUES
(0,'admin','$admin_password','Ignácz József','$email',1,'$date','$date','')", $db);


        header("Location: index.php");

} else {
print "<FORM METHOD=POST>\n ";
print "Adminisztrátori jelszó: <BR>\n<INPUT TYPE=TEXT NAME=admin_password><BR>\n";
print "E-mail cím: <BR>\n<INPUT TYPE=TEXT NAME=email><BR>\n";
print "<INPUT TYPE=SUBMIT NAME='password_set' VALUE='Mehet'>\n";
print "</FORM>";

}

?>
```

A program megint három részből áll. Az első rész az általános rész, a második a program újrahívása után végrehajtandó műveletek, a harmadik pedig az első futásnál generálódó űrlap, ami a második hívást lehetővé teszi.

Az első blokk azzal indít, hogy behívja az előzőleg megírt *include.php*-t. Emlékezzünk: egyelőre csak a globális változók és a *db_connect* függvényünk van benne. Ez után, mivel az *$admin_password* nevű változónknak még biztosan nem lesz értéke, végrehajtódik a feltétel igaz ága: megpróbálunk csatlakozni az SQL szerverhez, figyelve a hibaüzenetekre is. A következő sorok is ismerősek lehetnek: töröljük a már esetleg létező adattáblát, újra létrehozzuk, kiválasztjuk, természetesen mindezt hibakezeléssel megspékelve. A legszebb része most következik: tábla struktúrájának létrehozása. Elsődleges kulcsként a nevet adtam meg, pedig szokványosan az azonosítót szokták. Ennek egyetlen oka van: az azonosítót generálni fogjuk, tehát nem fordulhat belőle elő két azonos. A név viszont a felhasználón múlik, ő adja meg, tehát előfordulhat, hogy ugyanazt a nevet kétszer is szeretnék regisztrálni. Ezt megakadályozandó, elsődleges kulcsnak tesszük a nevet, ami innentől kezdve biztosan egyedi lesz, ugyanazon név kétszer történő felvitele esetén hibát fogunk kapni, amit már le tudunk kezelni.

Ugorjunk a harmadik blokkra, hiszen időrendben ez következik. Ez, ha végigkövetjük a kapcsoszárójelek szövevényét, az utolsó *else* után következik. PHP-ből hozunk létre egy FORM-ot, amely egy jelszómezőt és egy egyszerű szöveg-

mezőt tartalmaz. A végeredményt egy *password_set* nevű, „Mehet” feliratú gombbal küldjük tovább a második fázisnak. Figyelemreméltó a sorok végén a „\n” szekvencia, amely a HTML forrás csinosítása végett került bele. Ha nem lennének, a teljes űrlap HTML forrása egy sorba rendezve jelenne meg.

A maradék programrész fog végrehajtódni, miután újrahívtuk a programot az űrlap elküldésével. Hadd idézzem, s lássuk sorról sorra, mert itt lesz a kutya elásva!

```
If (isset($admin_password) AND isset($password_set))
{
	$admin_password = md5(trim(addslashes($admin_password)));
	$date = date("Y-m-d H:i:s", time());

	$db = db_connect();
	mysql_query("INSERT INTO auth
(a_id,a_name,a_password,a_fullname,a_email,a_counter,a_lastvisit,a_regdate,a_session_id) VALUES
(0,'admin','$admin_password','Ignácz József','$email',1,'$date','$date','')", $db);

	header("Location: index.php");
}
```

Mivel az *$admin_password* változónak már lesz értéke, s valószínűleg a *submit* gombot is megnyomtuk, a kapcsoszárójel közötti rész végre fog hajtódni. A legelső teendőnk, hogy a jelszót azonnal kódoljuk! Érvényes jelszót lehetőleg ne tároljunk adattáblában, mert elég nagy veszélyforrás. Gondoljunk csak bele: elég, ha valaki megszerzi a t áblát, máris tulajdonában van az összes belépési kódnak és jelszónak. Hogy ezt elkerüljük, máris hármas tortúrának vetjük alá a *$admin_password* nevű változónkat. Elsőként egy picit átalakítjuk: az *addslashes* függvény az adatbázis lekérdezéseknél használt kitüntetett karakterek elé fordított ferde vonalat (backslash) tesz. Ezek a k itüntetett karakterek az aposztróf ('), az idézőjel ("), a backslash (\) és a NUL érték. Ezzel kiküszöböltük, hogy a felhasználó által esetlegesen beírt hibás karaktere hibát okozzon majd a későbbi lekérdezéseknél. A második manipuláció a *trim* függvény. Ez csak annyit tesz, hogy eltávolítja az üres karaktereket a k araktersorozat elejéről és végéről. Ilyen üres karakternek számít a "\n", "\t", "\r", "\v", "\0" bármelyike és a szóköz. Kikerültük, hogy a felhasználó által a jelszó előtt véletlenül nyomott szóközt is a jelszó részének tekintsen a rendszer. Ha pedig direkt ütötte be a szóközt, akkor vessen magára!

Eddig csak olyan manipulációt végeztünk, amely után a jelszó még tökéletesen azonosítható. De bezzeg a következő függvény! Ez az érdekesség az md5. Ez a függvény a megadott karaktersorozat MD5 hash (hasító) értékét számítja ki, s adja vissza. Ehhez az *RSA Data Security, Inc. MD5 Message-Digest* algoritmusát használja fel. További információt kaphatunk a http://www.faqs.org/rfcs/rfc1321.html lapon kaphatunk. Elégedjünk meg egyelőre annyival, hogy egy nehezen visszakódolható, 32 ka rakteres hexadecimális kódsorozatot kapunk eredményül. De hogyan fogjuk ezt a felismerhetetlen kódsorozatot használni arra, hogy leellenőrizzük, helyes jelszót kaptunk-e a felhasználótól. Egy példa alapján megmutatom!

Tegyük fel, hogy a jelszó, amit elküldött a felhasználónk, „titok” lesz. Ha ezt lekódoljuk, a „201016e8206a5f42aa527090511504d5” kódsorozatot kapjuk. Szép, ugye? Bejelentkezéskor kedvenc felhasználónk beírja a jelszavát. Ha helye-

sen írta, az általa beírt jelszó MD5-ös kódolása szintén a fenti sorozat lesz. Tehát valami olyasmit kell vizsgálnunk, hogy a táblából beolvasott jelszó megegyezik-e a felhasználó által beírt jelszó MD5-ös átalakításával. Ha igen, akkor jó a jelszó, ha nem, akkor lehet újra próbálkozni.

Ennek a módszernek hátránya lehet, hogy nem tudjuk elküldeni az elfelejtett jelszót a felhasználónak. Sőt, ha telefonál, akkor sem tudjuk megmondani neki, mi volt az elfelejtett jelszó. Mindkét problémára van megoldás: az első esetben, ha Interneten kéri az elfelejtett jelszót, akkor az által megadott e-mail címre kipostázhatunk egy általunk generált véletlenszerű jelszót (pl. WkjdeZ), amit első bejelentkezéskor kell megadnia, aztán meg tudja változtatni valami emberbarátibb változatra. A második megoldása probléma még egyszerűbb: ha telefonál, kérünk tőle egy új jelszót, amit beírunk a tábla megfelelő mezőjébe. (Persze ekkor sem feledkezünk meg a kódolásról.) Ennek a módszernek a legkisebb munkával járó megoldása, ha az *admin* bárkinek a nevében be tud jelentkezni jelszó nélkül (ha már mint *admin* bejelentkezett a saját jelszavával), s meg tudja változtatni a bejelentkezett felhasználó adatait.

Ott fejeztük be, hogy van egy háromszorosan átalakított jelszavunk. A következő sorban bevezetünk egy új változót, amely a pontos dátumot és időt fogja tartalmazni „Y-m-d H:i:s” formátumban. (év-hónap-nap óra:perc:másodperc) Megnyitjuk az adatbázist, felhasználva a *unit.inc* egyetlen függvényét. Egy hosszú sor következik: végrehajtunk egy SQL parancsot, amely beszúr egy rekordot az *auth* nevű táblába. Láthatjuk, hogy az *admin* azonosítója a nulla lesz, jelszava az a csúnya, kódolt hexa sorozat, a teljes neve egy kis egészséges egoizmustól vezérelve a saját nevünk, az e-mail cím az űrlapon megadott, a számlálót beállítjuk egyesre, az utolsó látogatás és a regisztráció dátumát pedig az előbb generált pillanatnyi dátumra. Nem említettem az *a_session_id* nevű mezőt, amelynek üres (") értéket adunk először. (Nem 1 idézőjel, hanem 2 aposztróf!)

Az utolsó sor a *header* függvényt tartalmazza. Egy szemantikailag helyes HTML fejlécet generál, amelyet a böngésző értelmezni fog. Itt éppen egy új oldalt adunk meg, azaz amint ide ér a script végrehajtása, meghívódik a paraméterként megadott *index.php* nevű állomány.

Készen is vagyunk az adattábla elkészítését végző programmal! Ha lefuttatjuk, vigyázzunk rá, mert kérdezés nélkül törli az adatbázisunk minden táblájával együtt. Ha már előtte voltak regisztrált felhasználóink, búcsúzzunk el tőlük. Legszívesebben azt mondanám, hogy egyszer futtassuk le, majd vagy töröljük, vagy nevezzük át, hogy semmiképpen ne lehessen újra elindítani.

Eredményül van egy *auth_test* nevű adatbázisunk, benne egy *auth* nevű táblával, amely tartalmaz valami kilenc mezőt, ezzel megalkotva a felhasználó törzsadatait. Természetesen ez nem szentírás, ha valaki többet szeretne tárolni ebben a táblában, csak ki kell bővítenie azt.

Feldolgozás

login.php

Jól jönne egy program, amely segítségével be tudnánk lépni a rendszerbe. Amelyik ellenőrzi a jelszavunkat, jelez hiba esetén. Megint háromrészes scriptet írunk, hiszen egyszer be kell kérni a felhasználói nevet és a jelszót, majd újrahívásko r le kell ellenőrizni azt. Az űrlap elkészítése nem is olyan bonyolult.

```
<script>

function SetFocus()
{
      document.login.l_username.focus();
}

</script>


<body bgcolor="White" onLoad="SetFocus()">

<form name="login" method="post">
<table border=0 align=center width=200>
  <tr>
          <td>Neved:</td><td><input type="text" name="l_username" size="12"></td></tr>
          <td>Jelszó:</td><td><input type="password" name="l_password" size="12"></td>
  </tr>
  <tr><td></td><td><input type="submit" name="submit" value="Belépés"></td></tr>
</table></form>
```

Készítettem egy *login* nevű űrlapot. Hogy szebb legyen, tettem bele egy táblázatot, amely szélessége 200 pixel lesz. Ez elég is. A képernyő közepére igazítva jelenik meg, két szövegdoboz lesz rajta, és természetesen egy nyomógomb, amely újrahívja a scriptet. Egy javascript betétet is írtam az egész elé. Láthatjuk, hogy a neve *SetFocus*. Ezt a scriptet hívjuk meg, amikor az oldal betöltődött, azaz amikor az *onLoad* esemény bekövetkezett. Mire jó ez? Amikor betöltődött az oldal, a kurzor automatikusan a *l_username* mezőbe ugrik. Ezzel egy picit segítünk a felhasználónak, ráadásul magunknak is megtakarítunk egy csomó időt a tesztelés időszakában. Figyeljük meg, hogy a login form minden mezője a „l_” karaktersorozattal kezdődik. Ez megint egy segítség, legalább tudjuk, hogy ez a FORM mezője.

Jöhet a neheze! Oldjuk meg a jelszó ellenőrzését, és a felhasználó beléptetését. Egy tippet szeretnék adni, ehhez felvetek egy gyakorlati problémát: előfordulhat, hogy egy gépen több felhasználó is dolgozik. Az egyik bejelentkezik a rendszerbe a s aját nevével és jelszavával. Dolgozgat, majd kilép. Ha nem zárja be maga után a böngészőt, abból problémák keletkezhetnek. Ugyanis bizonyos belő változók, amelyeket a böngésző használ, egészen a böngésző bezárásáig élnek. Csak képzeljük el, hogy egy másik felhasználó is szeretne belépni ugyanabba a rendszerbe. Amint behozza az oldalt, a böngészőben élő változók még aktívak, így előfordulhat, hogy az előző felhasználó beállításaival kerül aktiválásra. Még csak a jelszót sem kell tudnia. Ennek is több megoldása van. Az egyik, a legegyszerűbb, hogy az oldalt és a rajta levő változókat minden betöltődés alkalmával újraértékeljük, s nem a már letárolt böngésző cache-ből szedjük az adatokat. Ehhez annyit kell tennünk, hogy megmondjuk a böngészőnek, hogy az adott oldal lejárt, azaz legyen szíves újratölteni. Ezt egy META tag-gel tudjuk megtenni, amely neve *Expires*. Használata egyszerű:

```
<META HTTP-EQUIV="Expires" CONTENT="1">
```

Ezzel megadtuk, hogy a lap, betöltődés után 1 másodperc múlva lejár, tehát mindenképpen újra kell tölteni azt, s nem szabad a cache-ből szedni, ha valaki újrakéri a lapot! Egy érdekesség: ha nullát használunk, sok esetben – a leírásoktól eltérően – sosem jár le az oldal…

Térjünk vissza az ellenőrzésre. Tegyük fel, hogy a felhasználó beírta a nevét és a jelszavát, postázta az űrlapot. Én így oldottam meg:

```
<META HTTP-EQUIV="Expires" CONTENT="1">

<? include("unit.inc");

if (!empty($l_username) & !empty($l_password))
{
	$l_password = (trim($l_password)=="") ? "":addslashes(md5(trim($l_password)));
	$l_username = addslashes(trim($l_username));

	$db = db_connect();
	$query = mysql_query("SELECT * FROM auth WHERE a_name='$l_username' AND a_password=
'$l_password'", $db) or die (mysql_error());

	if (mysql_num_rows($query) != 1)
		header("Location: error.php?error=bad_login&username=$l_username");
	else {

		// sikeres belépés!
		srand(time());
		$random_id  = rand(1000000,9999999);
		$session_id = md5($random_id);

		$result   = mysql_fetch_array($query);

		$user_id   = $result["a_id"];
		$username  = $result["a_name"];
		$oldcount  = $result["a_counter"];
		$oldvisit  = $result["a_lastvisit"];

		$counter   = $oldcount + 1;
		$lastvisit = date("Y-m-d H:i:s",time());

		mysql_query("UPDATE auth SET a_session_id='$session_id', a_counter='$counter', a_lastvisit=
'$lastvisit' WHERE a_id='$user_id'", $db);

		setcookie("username",    $username);
		setcookie("user_id",     $user_id);
		setcookie("session_id",  $session_id);
		setcookie("counter",     $oldcount);
		setcookie("lastvisit",   $oldvisit);

		header("Location: index.php");

	}
}

?>
```

Rögtön az elején egy érdekes értékadást láthatunk:

```
$l_password = (trim($l_password)=="") ? "":addslashes(md5(trim($l_password)));
```

Akármennyire furcsa, ez tényleg egy értékadás. Ha „szabványos” programozási nyelven szeretném leírni, a következőt kellene írnom:

```
If (trim($l_password==””)) $l_password=”” else $l_password= addslashes(md5(trim($l_password)));
```

Ez a feltételes értékadás tehát egy *if…then…else* szerkezet és egy értékadás kombinációja. Szintaxisa az alábbi:

```
változó = (feltétel ? változó értéke igaz esetén : hamis esetén);
```

Pár példa:

```
$a = ($b>10 ? "nagyobb, mint 10" : "nem nagyobb, mint 10");
$a = ($a==”férfi” ? 1 : 2);
```

A második példa egészen érdekes. Ha az $a változó értéke "férfi", akkor az $a (ami eddig karakteres kifejezés volt) vegye fel az egyes számot (numerikus kifejezés), egyébként pedig a kettes számot (pl. személyi szám első jegye). Itt is látható a PHP rugalmassága (mások káosznak hívják), miszerint a változóknak nincs kötött típusuk, szabadon lehet értéket adni, a megadott érték határozza meg a változó típusát.

A *mysql_query* függvényben kiválasztjuk azt a rekordot, amelyikben a felhasználó neve és jelszava (kódolt!) megegyezik az űrlapon megadottal. Ha a visszakapott rekordok száma nem 1, akkor vagy nulla, tehát nem található meg vagy a név, vagy a jelszó, vagy nem egy rekordban vannak, esetleg több rekordot kapunk vissza, ami ugye nem lehetséges, hiszen két egyforma nevű felhasználó nem létezhet. A *header* függvényt már láttuk, itt is egy weblapot hívunk meg, amelynek a neve *error.php*, amelynek két változót adunk át: az első az *error* változó, értéke „bad_login”, a *username* értéke pedig a felhasználó által beírt felhasználói név. Gyakorlásképpen készítsük el ezt a scriptet, vegye át a hiba nevét, és írja ki a következő hibaüzenetet: „Sajnálom, a beírt felhasználói név (Lajos) hibás, vagy a megadott jelszó érvénytelen!” Természetesen a Lajos csak példa, a *$username* változó fogja tartalmazni a valódi nevet.

Most jön az a rész, ami miatt az egész fejezet készült! Meghatározzuk a munkamenet azonosítót! Ne gondoljunk semmi különleges eljárásra, igazán egyszerű lesz. PHP-ben erre előre elkészített funkciók vannak. Megismerkedtünk a *session_start*, a *session_register* és a *session_id* függvényekkel. Őszintén szólva én nem kedvelem őket. A *php.ini*-ben kell meghatároznunk egy mappát, ahol a session-ökkel kapcsolatos átmeneti állományokat tárolja a szerver. Tehát minden egyes session regisztráció egy-egy fájlművelettel jár. Éppen ezért, inkább készítsünk saját session-manager scriptet. Fent már meg van oldva, lássuk a magyarázatot!

Sikeres belépés esetén első lépésként kicsit megkeverjük a véletlenszám generátort. Biztosan mindenki tudja, hogy a programozási nyelvek által generált „véletlen” számok nem igazi véletlen számok. Úgy is hívjuk őket, hogy álvéletlen számok. Egy kezdőszámból számolja ki a nyelv egy bizonyos algoritmus segítségével. Ha ugyanazt a kezdőszámot kapja meg a generátor, akkor ugyanazt az álvéletlen számot kapjuk vissza. Ezért, hogy ne lehessen kiszámítani, az aktuális időből képzünk egy számot, amely majd a véletlenszám generátor alapját fogja

képezni. Egészen kicsi a valószínűsége, hogy valaki ezredmásodpercre pontosan ugyanabban az időben hajtja végre ezt a sort, mint egy másik felhasználó.

E rövid sor és a hosszú litánia után képzünk egy véletlenszámot egymillió és tízmillió között. Azaz van 9 millió lehetőségünk. A generált számból képzünk egy *$session_id* nevű változót, amelyet az előbb megbeszélt MD5-ös formátumba konvertálunk. Ez lesz a megtalált felhasználó munkamenet azonosítója.

A következő sorokban kiolvassuk a felhasználó adatait. Megnöveljük a számlálóját, de előtte a régit eltároljuk egy *$oldcount* változóba. Hasonlóképpen járunk el az utolsó látogatás dátumával is. Mielőtt frissítjük, ezt is eltároljuk egy *$oldvisit* nevű változóban.

Egy SQL parancs következik. Frissítjük a megtalált felhasználó rekordját. Csupán a munkamenet azonosítót, a számlálót és az utolsó látogatás dátumát kell frissítenünk, a többihez nem nyúlunk. Azonosításra az előbb beolvasott *a_id* mező szolgál. Próbáljuk meg megmagyarázni, miért jó nekünk, hogy a munkamenet azonosítót is aktualizáltuk, ráadásul egy teljesen véletlenszerű értékre! A magyarázatot a következő sorok szolgáltatják. Beállítunk 5 cookie-t, azaz sütit, amelyek értékét bárhonnan elérhetjük. Ezek rendre a felhasználó neve, azonosítója, session_id-je, számlálója és a látogatás dátuma. (Az utolsó kettő a még régi adatokat tartalmazza, frissítésre csak akkor kerül sor, ha a felhasználó újra belép a rendszerbe.) Mindenre csak azért van szükség, hogy eldönthessük a weblapunk egyéb részeiben, hogy a felhasználó jogosult-e az adott oldal megtekintésére. Másik megoldás az lenne, hogy minden oldalon bekérjük az azonosítót és a jelszót, de ez nem igazán járható út. Ezzel a megoldással azt érjük el, hogy a *login.php*-ben bekérjük az azonosítás adatait, a többi lapon pedig csak azt kell megnézzük, hogy a sütiben eltárolt munkamenet azonosító megegyezik-e a t áblából kiolvasott *$user_id* session_id-jével. Ha egyezik, akkor az aktuális felhasználó be van léptetve, ha nem találtunk ilyen egyezést, akkor nincs beléptetve. Ezt nézzük meg programban is. Én a *unit.inc* scripte tettem bele egy újabb függvényt, hiszen ezt az funkciót minden védett lapon végre kell hajtani.

```
function get_user_settings()
{
	global $username, $lastvisit, $counter, $session_id;

	$settings = "";
	if (empty($username) | empty($session_id))
	{
		return $settings; // Itt ki is lépünk!
	}
	$db = db_connect();
	$query = mysql_query("SELECT * FROM auth WHERE a_name='$username' AND
a_session_id='$session_id'", $db) or die (mysql_error());


		if (mysql_num_rows($query)==1)
	{
		$settings = mysql_fetch_array($query);
			$settings["a_lastvisit"]=$lastvisit;		// Ezeket a változókat nem a rekordból olvassuk,
			$settings["a_counter"]=$counter;		// hanem az előző értékét (login.php) adjuk vissza.
	}

		return $settings;
}
```

A globális változók beimportálása után elsőként azt vizsgáljuk meg, hogy a név, vagy a munkamenet azonosító bármelyike üres-e. Ha igen, akkor itt ki is lépünk, egy üres karaktersorozat a visszaadott érték. Ha mégsem, akkor megnyitjuk a táblát, majd kiolvassuk belőle azt a rekordot, amelyben a név megegyezik a cookie-ból beolvasottal, ráadásul a táblában tárolt session_id is azonos a sütiben tárolttal. Ha egyetlen sort sikerült visszaolvasnunk, akkor jól csináltuk. Egy asszociatív tömböt adunk vissza, amely a beolvasott adatokat tartalmazza. Egy trükköt itt is meg kellett ejtenünk. Ha a táblából kiolvasott számlálót és látogatás dátumát adnánk vissza, az aktuális adatokat kapnánk. Pedig az előző adatokra van szükségünk, azokra, amelyeket a bejelentkezés folyamán a sütikben eltároltunk. A most aktuális adatok csak a következő belépés folyamán válnak szükségessé, addig felülírjuk őket a visszaadott tömbben.

Hogyan hívjuk meg ezt a függvényt? Egy értékadásos vizsgálattal:

```
if ($user=get_user_settings())
{
        // műveletek
}
```

Azaz olvassuk be az aktuális felhasználó adatait egy *$user* változóba, ha ez nem üres, akkor hajtsuk végre a műveleteket. Két legyet egy csapásra! Csak arra vigyázzunk, hogy nem egyenlőségvizsgálatot hajtunk végre, hanem értékadást. Tehát nem két, csak egy „=” jelet kell tennünk!

logout.php

Ha egyszer bejelentkeztünk, a kijelentkezést is meg kellene oldani. A kijelentkezésre két lehetőség kínálkozik: ha lezárjuk a böngészőt, vagy ha végrehajtjuk a kijelentkezési procedúrát. A böngésző lezárása egyértelmű, a másik annál kevésbé. Több dolgot is végre kell hajtani. Meg kell szüntetni a cookie-k tartalmát, ráadásul a munkamenet azonosítót is ki kell törölni a felhasználó rekordjából. Kb. ennyi. Lássuk!

```
<?

include("unit.inc");
if ($user=get_user_settings())
{
        srand(time());
        $new_random_id = rand(1000000, 9999999);
        $new_session_id = md5($new_random_id);

        $db = db_connect();
        mysql_query("update auth set a_session_id='$new_session_id' where a_name='$username'", $db);

        setcookie("username",   "");
        setcookie("user_id",    "");
        setcookie("session_id", "");
        setcookie("counter",    "");
        setcookie("lastvisit",  "");
}

header("location: index.php");
```

```
?>
```

Figyeljük meg, hogy nem nulláztam a tábla session_id mezőjét, hanem felülírtam egy véletlenszerű értékkel. Sokkal biztonságosabb megoldás, mint a nullázás, ezzel kiküszöböltünk egy lyukat a rendszerünkben. Ha készen vagyunk, újrahívjuk az *index.php*-t, amelyikbe a részprogramokat hívó linkeket helyezhetjük el.

reg.php

Már csak egy fontos része van az autentikációs programunknak. Ez pedig a regisztráció! E nélkül nem sokat ér a rendszerünk, hiszen feliratkozás nélkül nem bővül a táblánk, s nem lesz egyre bonyolultabb az egész.

Mostani scriptünk egy picit hosszabb, mint a többi. No nem azért, mert bonyolultabb, mint az előzőek. Inkább azért, mert több HTML rész lesz benne. Létre kell hoznunk egy felviteli űrlapot. Ennek az ellenőrzését kihagytam, csupán egyetlen momentumra koncentráltam, mégpedig a jelszavak azonosságának ellenőrzésére. Ezt tovább lehet gondolni, lehet figyelni a kötelező mezőkre, a bevitt mezők formátumára, stb. Csupán ötletként, definiáltam egy stílust. Ez a beviteli mezők stílusa lesz, figyeljük meg, hogyan alkalmazhatjuk őket lapjainkon. Érdemes! Az egyetlen, ami talán magyarázatra szorulhat, a felhasználói azonosító meghatározása. Ehhez a *max()* SQL függvényt használhatjuk fel. Kiolvashatjuk vele a zárójelben megadott mező maximális értékét. Ha nincs rekord, vagy csak az admin létezik, ennek az értéke 0 lesz. (Az admin azonosítója 0!) Elég eggyel megnövelni, s készen van a következő azonosító. Ennek van még egy előnye, ami egyben a hátránya is: ha törlésre kerül egy felhasználó, az általa elfoglalt és felszabadított érték többször nem kerül kiosztásra, hacsak nem az utolsó rekord került törlésre. Ez lehet jó is (nem lesz sosem két egyforma azonosítójú rekordunk), de helypazarló eljárás. Mindenki eldöntheti, s kereshet jobb megoldást. Akkor lássuk a scriptet!

```
<META HTTP-EQUIV="Expires" CONTENT="1">

<STYLE>
.inbox
{
	font:			normal 8pt Verdana;
	border-top:		#c0c0c0 1px solid;
	border-left:		#c0c0c0 1px solid;
	border-bottom:		#c0c0c0 1px solid;
	border-right:		#c0c0c0 1px solid;
}

</STYLE>
<?

require("unit.inc");

  if (!empty($submit))
  {
```

```
        $db = db_connect();
        $result  = mysql_query("select max(m_id) from members", $db);
        $row = mysql_fetch_row($result);
        if (empty($row[0]))
                $user_counter = 1;
                else
                $user_counter = ++$row[0];

        $date = date("Y-m-d H:i:s", time());
        $username = trim(addslashes($_name));
        $passwd = addslashes(md5(trim($_password)));

        mysql_query("insert into auth
        (a_id, a_name, a_password, a_fullname, a_email, a_counter, a_lastvisit, a_regdate, a_session_id)
        VALUES
        ($user_counter, '$username', '$passwd', '$_fullname', '$_email', 1, '$date', '$date', '')" , $db) or die("Nem
sikerült az adatok felvitele: ".mysql_error()."!");

        Header("Location: index.php");
   }

?>

<body bgcolor="White">
<script>
function SetFocus()
{
    document.reg._name.focus();
}

function checkForm() {

   if (document.reg._password.value==document.reg._password_re.value)
   {
   alert("Megy az űrlap...!");
   return(true);
   } else
   {
   alert("Nem egyeznek a jelszavak!");
   return(false);
   }


}
</SCRIPT>

<body bgcolor="White" onLoad="SetFocus();">

<FORM name="reg" method="post" onSubmit="return checkForm()">
<table border="0" align=center>
<tr>
     <td>

     <table align=left border=0 cellpadding=2 cellspacing=2>

       <TR>
         <TD>Becenév:</TD>
         <TD><INPUT class="inbox" size=28 type=text name="_name"></TD>
       </TR>

       <TR>
         <TD>Jelszó:</TD>
         <TD><INPUT class="inbox" size=28 type=password name="_password"></TD>
       </TR>

       <TR>
         <TD>Jelszó újra:</TD>
         <TD><INPUT class="inbox" size=28 type=password name="_password_re"></TD>
       </TR>

       <TR>
         <TD>Teljes név:</TD>
         <TD><INPUT class="inbox" size=28 type=text name="_fullname"></TD>
```

```
      </TR>

      <TR>
        <TD>Születési dátum (év-hó-nap):</TD>
        <TD><INPUT class="inbox" size=28 type=text name="_bdate"></TD>
      </TR>

      <TR>
        <TD>E-mail címed:</TD>
        <TD><INPUT class="inbox" size=28 type=text name="_email"></TD>
      </TR>

    </TABLE>

    </td>
</tr>

<tr>
    <td  colspan=4 align=center><br><input class="button" name="submit" type="submit" value="Átnéztem
mindent, elküldöm a kérdőívet!"></td>
</tr>

</table>
<br><br><br><br>
</body>
```

profil.php

Ez a programrész, illetve script csak példaként szerepel a többi között. Feladata egyszerűen annyi, hogy kiírja a bejelentkezett felhasználó adatait. Vizsgálnunk kell a jogosultságot, s példát láthatunk arra is, hogyan használhatjuk fel a *get_user_settings*-ből származó adatokat.

```
<?

    Include ("unit.inc");

        if ($user=get_user_settings())
        {
            echo "<table border=0>";
            echo "<tr><td colspan=2><font size=5>Profil:</font></td></tr>";
            echo "<tr><td>Azonosító: </td><td>".$user["a_id"]."</td></tr>";
            echo "<tr><td>Bejelentkezési név: </td><td>".$user["a_name"]."</td></tr>";
            echo "<tr><td>Jelszó (kódolt!): </td><td>".$user["a_password"]."</td></tr>";
            echo "<tr><td>Teljes név: </td><td>".$user["a_fullname"]."</td></tr>";
            echo "<tr><td>E-mail cím: </td><td>".$user["a_email"]."</td></tr>";
            echo "<tr><td>Számláló: </td><td>".$user["a_counter"]."</td></tr>";
            echo "<tr><td>Utolsó látogatás dátuma: </td><td>".$user["a_lastvisit"]."</td></tr>";
            echo "<tr><td>Regisztráció dátuma: </td><td>".$user["a_regdate"]."</td></tr>";
            echo "<tr><td>Session_ID: </td><td>".$user["a_session_id"]."</td></tr>";
            echo "</table>";
        }
?>
```

Ilyen egyszerű. A jelszót és a session_id-t fölöslegesen (és buta módon) írattam ki. Ezek sosem jelennek meg a felhasználó szeme előtt, de nem is érdemes ilyen dolgokkal fárasztani őket.

index.php

Az *index.php*-re többször is hivatkoztunk. Én egy menüt tettem bele, amely segít a navigációban. Egyszerű *„a href”* tag-ekkel dolgoztam, csak egy picit variáltam rajta. Próbáljuk kitalálni, mi történik, vagy egyszerűen próbáljuk ki!

```
<? include("unit.inc"); ?>

<a href="login.php">Belépés</a><br>
<?
  if ($user=get_user_settings())
            {
              echo 'Név: '.$username;
              echo '<a href="logout.php">Kilépés</a><br>';
              echo '<a href="profil.php">Profil</a><br>';
            }
            else
            {
              echo "Kilépés<br>";
              echo "Profil<br>";
            }

  echo '------------<br>';
  echo '<a href="reg.php">Regisztráció</a><br>';

?>
```

Akinek nem sikerült, egy piciny segítség: bizonyos menüpontokat csak akkor kell kiírni, ha a felhasználó be van jelentkezve…

Összefoglalás

Az autentikáció a munkamenet azonosítókra épül. Ezek tárolása a sütikben történik, tehát nem árt, ha engedélyezzük őket böngészés közben. Szükségünk lesz még egy felhasználói névre, egy jelszóra, amelyek egyértelműen azonosítják majd minden egyes tagunkat, akik regisztrálták magukat rendszerünkbe. Több munkafolyamatot kell megtervezzünk. Szükségünk lesz a tagok regisztrációját segítő, a beléptetést megoldó, a kilépést elvégző scriptekre, de ne feledkezzünk meg pl. a megfelelő felhasználó felület létrehozásáról sem. Egy jó felület már fél siker. Ápol, s eltakar.

Minél több weblapot fejlesztünk majd ki, annál több tapasztalatra teszünk szert. Észrevesszük hibáinkat, a lyukakat a rendszerben. Egyre több saját függvényt készítünk majd, ezeket több helyen is fel tudjuk használni. Próbáljunk modulárisan dolgozni: modulokat, függvényeket, unitokat készíteni. A stíluslapok megváltoztatásával teljesen meg tudjuk változtatni a felületet, csak a háttérben zajló folyamatok lesznek ugyanazok. Minden rutinunkat dokumentáljuk. Sok olyan függvényem van, amelyeket több éve készítettem. Ha nem lennének kommentezve a rutinok, igen nehéz helyzetbe kerülnék néha, hiszen újra át kellene gondolnom őket, ami nem hiányzik, főleg, ha időre megy a tervezés és a programozás. S ki tudja, pár éve milyen ötlettől vezérelve tettem oda be azt a változót, amely… Egy példa a :

```
function Kill_Search_Variables()
```

```
// Törli a keresés globális változóit
// Bemenő paraméterek: globális keresési változók (search_*)
// Kimenő paraméterek: nincs
{
```

Használjunk mindig a változóknak és a függvényneveknek is beszédes neveket. Nem érdemes takarékoskodni velük, hiszen az előző fejezetekben felvázolt weblap forráskódja is csupán 10 kbyte lett nálam. Egy megából egy komplett e-bolt teljes rendszerét össze lehet állítani, s ebben már a képek is benne vannak (csak ne használjunk belőlük túl sokat). Ne feledjük, amíg ki nem adjuk a kezünkből a kész munkát, addig magunknak dolgozunk!

Irodalomjegyzék

http://www.apache.org
http://www.php.net
http://www.mysql.com
http://www.netcraft.com
http://www.w3.org